RWTH Aachen

Department of Computer Science
Technical Report

WST'04
7th International Workshop on Termination

Michael Codish and Aart Middeldorp (eds.)

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2004-07

RWTH Aachen · Department of Computer Science · June 2004

The publications of the Department of Computer Science of RWTH Aachen (*Aachen University of Technology*) are in general accessible through the World Wide Web.

http://aib.informatik.rwth-aachen.de/

Preface

This report contains the proceedings of the *7th International Workshop on Termination* (WST 2004), which was held June 1 – 2, 2004 in Aachen as part of the *Federated Conference on Rewriting, Deduction, and Programming.* Previous workshops were held in St. Andrews (1993), La Bresse (1995), Ede (1997), Dagstuhl (1999), Utrecht (2001), and Valencia (2003).

This workshop delves into all aspects of termination of processes. Though the halting of computer programs is undecidable, methods of establishing termination play a fundamental role in many applications and the challenges are both practical and theoretical. From a practical point of view, proving termination is a central problem in software development and formal methods for termination analysis are essential for program verification. From a theoretical point of view, termination is central in mathematical logic and ordinal theory.

Because of the success of the exhibition/competition of termination provers at last year's workshop in Valencia, it was decided to make the workshop annual. The results of this year's competition are not known at the time of writing, so the interested reader is referred to

`http://www.lri.fr/~marche/wst2004-competition/`

We are grateful to Claude Marché for running the competition and to both Claude Marché and Albert Rubio for the preliminary work.

In addition to the regular abstracts and system descriptions selected by the program committee, the workshop features an invited talk by Danny De Schreye, surveying the last 10 years of research on termination in the logic programming community.

It is our pleasure to thank the organizing committee of RDP 2004, and in particular Jürgen Giesl, for hosting the workshop and printing the proceedings.

May 2004

Michael Codish
Aart Middeldorp

Workshop Organization

Program Committee

Michael Codish	Beer-Sheva	(co-chair)
Danny De Schreye	Leuven	
Alfons Geser	Hampton, VA	
Neil D. Jones	Copenhagen	
Claude Marché	Orsay	
Aart Middeldorp	Innsbruck	(co-chair)
Frederic Mesnard	La Reunion	
Helmut Schwichtenberg	Munich	
Harald Søndergaard	Melbourne	

Competition / Exhibition

Claude Marché	Orsay
Albert Rubio	Barcelona

Local Arrangements

Jürgen Giesl	Aachen

Contents

The Never-Ending Story: 10 years after (preliminary and limited version)

Danny De Schreye

Dept. Computer Science, K.U.Leuven, Belgium
dannyd@cs.kuleuven.ac.be

Abstract. The research area of Logic Program termination analysis has its origins from nearly 20 years ago. Exactly 10 years ago, a survey of the work in this area appeared as one of a collection of survey papers on Logic Programming in the Journal of Logic Programming, under the title: Termination of Logic Programs: the never-ending story ([1]).

This talk is in preparation of a sequel to this survey, presenting the new directions of work published over the last 10 years. The survey is in a very preliminary phase of its preparation. As a result, the goals for this talk are somewhat broader then just to survey the area.

Because the audience for this talk includes many people who may be less familiar with Logic Programming termination analysis, one important goal for the talk is present and discuss some of the issues in termination analysis that are specific for the Logic Programming context and present some proposed solutions for these issues. These issues may not correspond to publications of the last 10 years. In that sense, the talk will be broader than what the title suggests.

On the other hand, since most of the work on the survey still needs to be done, in the talk we will focus more on contributions of the Leuven group than on others. This reflects in no way a judgment of the importance of the Leuven contributions, but is a cheap trick to avoid extensive preparation in limited remaining time.

Finally, whenever possible, contributions will be placed in their historic context. to give the audience a broader and more conceptual view of the evolution within this area.

References

1. D. De Schreye and S. Decorte, Termination of Logic Programs: the never-ending story, *J. Logic Programming* 19-20, pp.199-260, 1994.

Connecting Remote Termination Tools*

María Alpuente and Salvador Lucas

DSIC, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{alpuente,slucas}@dsic.upv.es

Abstract. After more than thirty years of development of the theory of termination of rewriting, a number of tools for automatically proving termination of TRSs have recently emerged. Their application to real programming languages and systems is of course desirable and has been envisaged in many of these tools. The World Wide Web makes it possible to gain access to the different resources in a number of ways, ranging from remote downloads followed by local executions to remote execution via WWW services. Unfortunately, though, few of the existing systems and tools (including termination tools) are readily connectable. We advocate systematically considering interoperability across platforms, applications and programming languages when developing termination tools and related language processors. This is only possible if a number of common practices (and standards) are seriously put into use by the international community.

1 Introduction

The quest for a *verifying compiler* is a classic but still hot, challenging goal for both the Software Industry and the Computer Science community [4]. Of course, termination analysis would be an essential component of such a tool. As mentioned in [4], the effective development of such a system will require an incremental and cooperative effort from different work teams all around the world. Thanks to the WWW, the physical distance among those teams is becoming less and less important. However, many existing systems and tools are not easily conciliable for working together, even if they address closely related problems or rely on similar theoretical bases. The WWW technology, however, is full of possibilities for removing barriers to integration and for dramatically improving the re-usability of previous theoretical results and development efforts.

2 Termination of programs and termination tools

As a motivating example, we consider the termination analysis of programs written in programming languages such as CafeOBJ, Elan, Erlang, Maude, OBJ, etc., whose operational principle is based on term rewriting. Proofs of termination of TRSs can be used for proving the termination of programs written in these languages. A number of systems for proving termination are available on the WWW: e.g.,

Tool	WWW site (use prefix `http://`)
AProVE	`www-i2.informatik.rwth-aachen.de/AProVE`
CiME	`cime.lri.fr`
Hasta-La-Vista	`www.cs.kuleuven.ac.be/~dtai/dpl/systems-E.shtml`
Matchbox	`theo1.informatik.uni-leipzig.de/matchbox`
TALP	`bibiserv.techfak.uni-bielefeld.de/talp`
TerminWeb	`lvs.cs.bgu.ac.il/~mcodish/suexec/terminweb`
Termptation	`www.lsi.upc.es/~albert/term.tar.gz`
Tyrolean/Tsukuba Termination Tool	`cl2-informatik.uibk.ac.at`

* Work partially supported by MCyT project TIC2001-2705-C03-01, MCyT Acción Integrada HU 2003-0003 and AVCyT grant GR03/025.

Other systems, like the Mercury compiler `www.cs.mu.oz.au/research/mercury`, include a termination checker.

Unfortunately, however, it is not easy to connect independently developed analysis tools to a practical environment such as the Maude interpreter[1]: the syntax of Maude programs does not correspond to the syntax of any of existing termination tools (see [5] for descriptions of many of them); the systems have quite different interfaces (for instance, stream-based input output or graphical interfaces); the Maude interpreter is written in C++, whereas the tools are written in different languages, namely CAML, Java, SICSTUS Prolog, etc.

From a semantic point of view, there are also other relevant issues which have to be considered when trying to bridge term rewriting tools and Maude: Maude programs are syntactically richer than term rewriting systems. Maude features that are not necessarily managed by these rewriting tools include: sorts, conditional rules, pattern matching expressions, programmable evaluation strategies, associative/commutative/idempotent symbols, modules, etc. The previous tools for proving termination of rewriting hardly deal with these features, which still have to be properly managed if we want to (accurately) use the existing tools to prove properties of real programs. Eventually, this can be done by means of program transformations which preserve the focused property (e.g., termination). This is somehow related to the transformational approaches for proving termination of (well-moded) logic programs which reduce the termination problem of logic programs to that of TRSs. These findings are the basis for logic programming termination tools consisting of a front-end which implements the considered transformation and a back-end for proving termination of the generated TRS.

3 Web services for termination analysis

Interoperability (i.e., to make it possible for *a program on one system to access programs and data on another system* [1]) is a general problem in Software Engineering and a number of solutions have been devised up to now (namely, *middleware systems* [1,6]). The XML WWW services (or just WWW services) are the most recent proposal for interconnecting systems [7]. They provide a flexible architecture for achieving interoperability of loosely-coupled systems that use different internal data structures to represent the information, and that have been developed in different programming languages. The exchange of information, the implementation of the necessary calls, and the localization of services on the web is based on different XML dialects (XML for representing data, SOAP for packaging messages, WSDL for defining the available services, and UDDI for finding them on the Web [3]). Even in the case that the systems are available on the same local machine, most of these interoperability facilities could play a relevant role in gluing together different tools which are often developed in different programming languages as well as combining architectural approaches where more tightly-coupled techniques like RPC or CORBA are not viable in many cases. Innovation is moving towards the so-called WWW services choreography, which pursues new standards to support the specification of complex systems out of simpler ones.

The definition of standards is widely recognized as desirable in the industry. The scientific community would also benefit from some standardization regarding, e.g., interoperability of systems which are naturally and highly distributed. We do believe that more general frameworks are lacking, that is, abstract schemes for particular kinds of applications -abstract in the sense of isolating the common features of that class. In many framework

[1] `http://maude.cs.uiuc.edu`

designs, the applications are created by the combination of the framework with a number of application-dependent decisions so that they get permanently -and undesirably- linked to the framework. Tackling the following specific issues seems desirable:

1. Defining suitable XML-like formats for expressing the most important features of currently used (families of) programming languages (in particular, rewriting–based programming languages).
2. XML sublanguages for expressing analysis/verification requirements to existing tools (e.g., innermost termination, simple termination, etc., possibly considering information regarding special operators, types, modules, strategies, conditions, etc.). We believe that this approach deserves great interest since it does contribute to the design of abstractions which are completely independent from the target language and are highly reusable.
3. Middleware translators from the existing programming languages to the lower level languages or formalisms which underly the program analysis tools. XSLT could be used for that purpose. Also declarative languages such as CiaoProlog, Haskell, SICStus Prolog, SWI-Prolog, etc., are well-suited for that.
4. Including the termination tools into the SOAP/WSDL/UDDI framework to get systematic access to their functionality.

As a concrete starting point, we think that the current format for specifying termination problems in the WST 2004 competition

`http://www.lri.fr/~marche/wst2004-competition`

could be further developed in the following directions:

1. separate the purely syntactic components of the problem (i.e., those referring the structure of the TRS) from the proof-oriented ones (i.e., those concerning a kind of termination property or termination proof which could be required from a tool);
2. use XML to express these different aspects of the problem; and
3. making some concrete, preliminary experiences to connect 'server' tools (e.g., termination provers) and 'client' tools (e.g., compilers, interpreters, or other systems which use auxiliary termination tools).

Beyond being the current trend of the international community, we believe that the use of XML-based technology for these purposes is also positive due to its flexibility and the existence of many tools and libraries which can be reused. Of course, other approaches, such as the ASF+SDF technology [2] could also provide useful ideas and solutions.

These tasks could probably be better addressed in coordination with existing working groups in international associations: for instance,

W3 Consortium	`http://www.w3c.org`
IFIP	`http://www.ifip.org`
ERCIM	`http://www.ercim.org`

4 Conclusion

The recent development of new and powerful termination tools is encouraging for the scientific community and suggests that they could be used by more complex computational systems such as advanced compilers and integrated software development environments.

However, the current tools are not commonly used or even easily reusable. The use of remote analysis tools (where remote means not only distant but also difficult to connect) is essential to be able to integrate powerful program analysis into programming environments. Not only should correctness or efficiency be taken into account while designing new tools, environments and frameworks, but interoperability issues should also be systematically considered as well. We honestly believe that putting greater effort into these issues will contribute to being successful with the current challenge of connecting analysis tools. In our opinion, these tasks should be appropriately coordinated by existing working groups in international associations.

References

1. P.A. Bernstein. Middelware: A Model for Distributed System Services. *Comm. of the ACM*, 39(2):86-98, 2002.
2. M.G.J. van den Brand, A. van Deursen, J. Heering, H.A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P.A. Olivier, J. Scheerder, J.J. Vinju, E. Visser, and J. Visser The ASF+SDF Meta-Environment: A component-based language development environment In R. Wilhelm, editor, *Proc. of 10th International Conference on Compiler Construction CC'01*, LNCS 2027:365-370, Springer-Verlag, Berlin, 2001.
3. M. Burner. The Deliberate Revolution. Creating Connectedness with XML Web Services. *ACM Queue* 1(1):29-37, March 2003.
4. T. Hoare. The Verifying Compiler: A Grand Challenge for Computing Research. *Journal of the ACM*, 50(1):63-69, 2003.
5. A. Rubio, editor, *Proc. of 6th Int'l Workshop on Termination, WST'03*, Tech. Report DSIC II/15/03, Valencia, Spain, 2003.
6. I. Sommerville. Software Engineering (6th Edition). Addison Wesley, 2001.
7. M. Stal. Web services: Beyond Component-Based Computing. *Comm. of the ACM*, 45(10):71-76, 2002.

A Note on a Term Rewriting Characterization of PTIME

Toshiyasu Arai[1] and Georg Moser[2]

[1] Graduate School of Science and Technology
Kobe University
arai@kurt.scitec.kobe-u.ac.jp

[2] Institut für mathematische Logik und Grundlagenforschung
WWU Münster
moserg@math.uni-muenster.de

Abstract. In [2] a clever term rewriting characterization of the polytime functions is given by defining a set $R_{B'}$ of feasible rewrite rules for predicative recursion. It is shown that the derivation length function ($\mathrm{DL}_{R_{B'},f}$) of $R_{B'}$ for f is bounded by a monotone polynomial in the length of the inputs. We give a simplified proof of this result. As a consequence we obtain the stronger result that $\mathrm{DL}_{R_{B'},f}$ is bounded by a monotone polynomial in the length of the *normal* inputs, only. Thereby the specific features of $R_{B'}$ are more accurately expressed. We believe our results imply the usefulness of so-called Cichon's principle (CP) to the (worst-case) derivation length analysis of a given term rewriting system R.

1 Motivation

In [2] a *feasible* term rewriting framework for the Bellantoni Cook schemata of *predicative recursion* [3] is introduced. The latter yields a canonical definition of the polynomial time computable function (PTIME). The term rewriting characterization of PTIME is obtained by restricting the natural rewriting analogue of the schema of predicative recursion suitably. These restrictions amount essentially to the fact that rewrite rule applications are restricted to the case where all the *safe* argument are numerals. No restriction is given for the *normal* arguments. Thus the schemata of predicative recursion are feasibly represented by the below set of rewrite rules. Following [2] this rewrite system is called $R_{B'}$.

Due to space limitations, we suppose (at least nodding) familiarity with [2,3]. Furthermore we assume the notations of [2] and use the following conventions. Let k, l, k', l', r denote arbitrary natural numbers, respectively. The symbols $\bar{n}, \bar{m}$ are used to denote numerals. We write $\boldsymbol{t}$, to denote sequences of terms $t_1, \ldots, t_k$ and $\boldsymbol{g}$ to denote sequences of function symbols $g_1, \ldots, g_k$, respectively. For simplicity we write $\boldsymbol{n}$ for a sequence of numerals $\bar{n}_1, \ldots, \bar{n}_k$ and we write O for $O^{0,0}$. The set $R_{B'}$ consists of the following rules.

$$O^{k,l}(x_1, \ldots, x_k; y_1, \ldots, y_l) \to O \ .$$

for all k, l so that $k + l > 0$;

$$U_r^{k,l}(x_1, \ldots, x_k; x_{k+1}, \ldots, x_{k+l}) \to x_r \ ,$$

for any $1 \leq r \leq k + l$;

$$P^{0,1}(; O) \to O \ ,$$
$$P^{0,1}(; S_i(; y)) \to y \ ,$$

for $i \in \{1, 2\}$;

$$C^{0,3}(; O, y_1, y_2) \to y_1 \ ,$$
$$C^{0,3}(; S_i(; y), y_1, y_2) \to y_i \ ,$$

for $i \in \{1,2\}$;

$$\mathrm{SUB}^{k,l}_{k',l'}[f,\boldsymbol{g},\boldsymbol{h}](\boldsymbol{x};\boldsymbol{n}) \to f(g_1(\boldsymbol{x};),\dots,g_{k'}(\boldsymbol{x};);h_1(\boldsymbol{x};\boldsymbol{n}),\dots,h_{l'}(\boldsymbol{x};\boldsymbol{n}))\ ,$$

and finally

$$\mathrm{PREC}^{k+1,l}[g,h_1,h_2](O,\boldsymbol{x};\boldsymbol{n}) \to g(\boldsymbol{x};\boldsymbol{n})\ ,$$
$$\mathrm{PREC}^{k+1,l}[g,h_1,h_2](S_i(;x),\boldsymbol{x};\boldsymbol{n}) \to h_i(x,\boldsymbol{x};\boldsymbol{n},\mathrm{PREC}^{k+1,l}[g,h_1,h_2](x,\boldsymbol{x};\boldsymbol{n}))\ .$$

We write B ($B^{k,l}$) to denote the set of *predicative recursive functions* (with k normal terms and l safe terms as arguments).[1] The set of ground terms over B is defined as usual and denoted as $\mathcal{T}(B)$. We denote the *derivation length function* of a rewrite system R for $f \in B$ as $\mathrm{DL}_{R_{B'},f}$. The latter is defined as follows. For any $f \in B^{k,l}$ let

$$\mathrm{DL}_{R_{B'},f}(\boldsymbol{m};\boldsymbol{n}) := \max\{n \mid \exists t_1,\dots,t_n \in \mathcal{T}(B)(t_1 \to_{R_{B'}} \cdots \to_{R_{B'}} t_n) \wedge t_1 = f(\boldsymbol{m};\boldsymbol{n})\}\ .$$

One of the main results of [2] states

Theorem 1. *The rewrite system $R_{B'}$ is terminating and for any $f \in B$, $\mathrm{DL}_{R_{B'},f}$ is bounded by a monotone polynomial in the length of the inputs.*

To obtain this, Beckmann and Weiermann define a monotone interpretation $J\colon \mathcal{T}(B) \to \omega$ that normalizes $R_{B'}$. In some sense their result is non-optimal. The bounding function for $\mathrm{DL}_{R_{B'},f}$ depends on (the length of) *all* inputs, i.e. on the length of *normal* and *safe* arguments. However an inspection of the rules in $R_{B'}$ shows that a bounding function for $\mathrm{DL}_{R_{B'},f}$ need not depend on (the length of) the *safe* argument terms of f.

Below we define a monotone interpretation $\mathrm{S}\colon \mathcal{T}(B) \to \omega$ that normalizes $R_{B'}$. This interpretation is designed such that $\mathrm{DL}_{R_{B'},f}(\boldsymbol{m},\boldsymbol{n}) \leq (2 + \sum_i |\bar{m}_i|)^{\ell(f)}$ holds, where $|\bar{m}|$ denotes the *dyadic length* of the numeral $\bar{m}$. I.e. the derivation length function of $R_{B'}$ for f becomes bounded by a monotone polynomial in the length of the input of normal arguments. Thereby we obtain a somewhat stronger result. Furthermore the designed interpretation S is considerably simpler than the original given interpretation J.

2 Results

We state the definition of the interpretation $\mathrm{S}\colon \mathcal{T}(B) \to \omega$. For $f \in B$, let $\ell(f) \geq 1$ be defined as follows:

- $\ell(f) := 1$, for $f \in \{S_i^{0,1}, O^{k,l}, U_r^{k,l}, P^{0,1}, C^{0,3}\}$.
- $\ell(\mathrm{SUB}^{k,l}_{k',l'}[f,\boldsymbol{g},\boldsymbol{h}]) := 2 + \max\{\ell(f) \cdot (\max_i \ell(g_i) + k'), \max_j \ell(h_j) + l'\}$.
- $\ell(\mathrm{PREC}^{k+1,l}[g,h_1,h_2]) := 1 + \max\{\ell(g),\ell(h_1),\ell(h_2)\}$.

Let $sn(t)$ ($t \in \mathcal{T}(B)$) denote the *maximal length of numerals* occurring in a safe position in t, defined as follows: (i) $sn(\bar{n}) := |n|$ or (ii) $sn(f(\boldsymbol{t};\boldsymbol{s})) = \max_j sn(s_j)$, otherwise. For the definition of S, we simultaneously define $\mathrm{S}\colon \mathcal{T}(B) \to \omega$ and $\mathrm{N}\colon \mathcal{T}(B) \to \omega$. We set

$$\begin{aligned}
\mathrm{S}(\bar{n}) &:= 0\ ,\\
\mathrm{S}(S_i(;t)) &:= \mathrm{S}(t) + 1 \quad \text{if } t \not\equiv \bar{n}\ ,\\
\mathrm{S}(f(\boldsymbol{t};\boldsymbol{s})) &:= (2 + \sum_i \mathrm{N}(t_i))^{\ell(f)} + \sum_j \mathrm{S}(s_j) \quad \text{otherwise, and}\\
\mathrm{N}(t) &:= \mathrm{S}(t) + sn(t)\ .
\end{aligned}$$

[1] For a formal definition the reader is kindly referred to Definition 2.2 in [2].

In particular note that $\mathrm{N}(\bar{n}) = |n|$, while $\mathrm{S}(\bar{n}) = 0$. Based on this notion we obtain the following proposition.

Proposition 1. *The interpretation* S *provides a monotone interpretation of* $\mathcal{T}(B)$ *into the natural numbers. This interpretation is compatible with* $\to_{R_{B'}}$. *Thus termination of* $R_{B'}$ *follows. Furthermore let* $t = f(\boldsymbol{m};\boldsymbol{n})$; *we obtain* $\mathrm{DL}_{R_{B'},f}(\boldsymbol{m};\boldsymbol{n}) \leq \mathrm{S}(t) = (2 + \sum_i |m_i|)^{\ell(f)}$ *for any sequence of numerals* $\boldsymbol{m}$, $\boldsymbol{n}$, *respectively. Thus* $\mathrm{DL}_{R_{B'},f}$ *is bounded by a polynomial in the length of the input of the normal arguments.*

In the remainder of this abstract we (briefly) indicate the steps that led us to the final definition of S. We think these considerations are interesting to the reader as they indicate a successful application of so-called Cichon's principle (CP) [4]. We make us of the follow formulation: "*The (wort-case) complexity of a rewrite system for which termination is provable using a termination order of order type* α *is eventually dominated by a function from the* slow-growing hierarchy *along* α."

We started with an interpretation $\pi\colon \mathcal{T}(B) \to \omega^\omega$ that guarantees termination of the rewrite system $R_{B'}$. We write $\Phi(f)$ to denote the number theoretic function represented by $f \in B$, the extension of Φ to arbitrary terms in $\mathcal{T}(B)$ is defined as usual.

Let $\mathrm{lh}(f)$, $f \in B$ be defined as in [2]: (i) $\mathrm{lh}(f) := 1$, for $f \in \{S_i^{0,1}, O^{k,l}, U_r^{k,l}, P^{0,1}, C^{0,3}\}$. (ii) $\mathrm{lh}(\mathrm{SUB}^{k,l}_{k',l'}[f,\boldsymbol{g},\boldsymbol{h}]) := 1 + \mathrm{lh}(f) + \mathrm{lh}(g_1) + \cdots + \mathrm{lh}(g_{k'}) + \mathrm{lh}(h_1) + \cdots + \mathrm{lh}(h_{l'})$. (iii) $\mathrm{lh}(\mathrm{PREC}^{k+1,l}[g,h_1,h_2]) := 1 + \mathrm{lh}(g) + \mathrm{lh}(h_1) + \mathrm{lh}(h_2)$. Then set for $f(\boldsymbol{t};\boldsymbol{s}) \in \mathcal{T}(B)$

$$\begin{aligned}\pi(f(\boldsymbol{t};\boldsymbol{s})) &:= \pi(\boldsymbol{t}) \oplus \pi(\boldsymbol{s}) \oplus 1 \quad \text{if } f \in \{S_i^{0,1}, O^{k,l}, U_r^{k,l}, P^{0,1}, C^{0,3}\}\ ,\\ \pi(f(\boldsymbol{t};\boldsymbol{s})) &:= \omega^{\mathrm{lh}(f)} \cdot \pi(\boldsymbol{t}) \cdot \sum_i |\Phi(t_i)| \oplus \pi(\boldsymbol{s}) \quad \text{otherwise}\ ,\end{aligned}$$

such that $\cdot$, $\oplus$ denote ordinal product and natural sum, respectively and $\pi(\boldsymbol{t})$ abbreviates $\pi(t_1) \oplus \cdots \oplus \pi(t_k)$ if $\boldsymbol{t} = t_1,\ldots,t_k$.

Proposition 2. *The interpretation* π *provides a monotone interpretation of* $\mathcal{T}(B)$ *into the set of ordinals below* ω^ω. *As* π *is compatible with* $\to_{R_{B'}}$ *the rewrite system* $R_{B'}$ *is terminating.*

This reveals that the order type of a suitable termination order for $R_{B'}$ is in fact a rather small ordinal, namely ω^ω. An immediate consequence of the above proposition is that $\mathrm{DL}_{R_{B'},f}$ can principally be majorized in terms of the so-called slow-growing function $\mathrm{G}_{\omega^{\mathrm{lh}(f)+1}}$.[2] This can be conjectured on the assumption that CP holds for the set of rewrite rules $R_{B'}$.

To verify that this really works, i.e. that CP indeed holds, a little bit more work is necessary. We employ the following proposition from [3]

Proposition 3. *Let* $f \in B$. *There exists a monotone polynomial* q_f *such that for all sequences of numerals* $\boldsymbol{m}$, $\boldsymbol{n}$ $|\Phi(f)(\boldsymbol{m};\boldsymbol{n})| \leq q_f(|m_1|,\ldots,|m_k|) + \max_j |n_j|$ *holds.*

This proposition allows us to define a more suitable ordinal interpretation $\mathrm{ord}\colon \mathcal{T}(B) \to \omega^\omega$. Let $\otimes$ denote the natural product. We set for $f(\boldsymbol{s};\boldsymbol{t}) \in \mathcal{T}(B)$

$$\begin{aligned}\mathrm{ord}(\bar{n}) &:= 0\ ,\\ \mathrm{ord}(S_i(;t)) &:= \mathrm{ord}(t) + 1 \quad \text{if } t \not\equiv \bar{n}\ ,\\ \mathrm{ord}(f(\boldsymbol{t};\boldsymbol{s})) &:= \omega^{\mathrm{lh}(f)} \otimes (\mathrm{ord}(\boldsymbol{t}) + 1) \otimes J_f(|\Phi(\boldsymbol{t})|) \oplus \mathrm{ord}(\boldsymbol{s}) \quad \text{otherwise}\ ,\end{aligned}$$

[2] The slow-growing functions are number-theoretic functions indexed by ordinals. One defines $\mathrm{G}_0(x) := 0$, $\mathrm{G}_{\alpha+1}(x) := \mathrm{G}_\alpha(x) + 1$, and $\mathrm{G}_\alpha(x) := \mathrm{G}_{\alpha[x]}(x)$, if α is a limit; $\alpha[x]$ denotes the x^{th} branch of the fundamental sequence for α.

where for $f \in \{O^{k,l}, U^{k,l}_r, P^{0,1}, C^{0,3}\}$ we set $J_f(\boldsymbol{m}) := 1$, for $F = \mathrm{SUB}^{k,l}_{k',l'}[f, \boldsymbol{g}, \boldsymbol{h}]$ let

$$J_F(\boldsymbol{m}) := (1 + \sum_i J_{g_i}(\boldsymbol{m})) \cdot J_f(q_{g_1}(\boldsymbol{m}), \ldots, q_{g_{k'}}(\boldsymbol{m})) + \sum_j J_{h_j}(\boldsymbol{m}) + 1 ,$$

and finally for $F = \mathrm{PREC}^{k+1,l}[g, h_1, h_2]$ we set

$$J_F(m, \boldsymbol{m}) := m \cdot (J_{h_1}(m, \boldsymbol{m}) + J_{h_2}(m, \boldsymbol{m})) + J_g(\boldsymbol{m}) .$$

Proposition 4. *The interpretation* ord *provides a monotone interpretation of $\mathcal{T}(B)$ into the ordinals below ω^ω. This interpretation is compatible with $\to_{R_{B'}}$. Furthermore let $t = f(\boldsymbol{m}; \boldsymbol{n})$; for some suitable variant G_α of the slow-growing hierarchy, we obtain $\mathrm{G}_{\mathrm{ord}(t)}(c) \geq \mathrm{D}\mathrm{L}_{R_{B'},f}(\boldsymbol{m}; \boldsymbol{n})$ for any sequence of numerals $\boldsymbol{m}$, $\boldsymbol{n}$, and some small number c, respectively.*

To achieve the above proposition one replaces the usual order relation $>$ on ordinals by a pointwise order relation $>_1$. I.e. one has to make sure that for $s, t \in \mathcal{T}(B)$ $s \to_{R_{B'}} t$ implies $\mathrm{ord}(s) >_1 \mathrm{ord}(t)$ and $\mathrm{G}_{\mathrm{ord}(s)}(c) > \mathrm{G}_{\mathrm{ord}\, t}(c)$ for some effectively given $c < \omega$. A suitable definition of $>_1$ can be either extracted from [1] or from [5].

It is easy to see that the thus obtained bounding function is a polynomial in the length of inputs of the *normal* arguments. Furthermore a brief inspection of the proof of the proposition reveals that no use of the occurring infinite ordinals is made. This led us to the quest for a natural number-theoretic interpretation S, whose definition has been given above.

3 Conclusion

Based on the work of Beckmann and Weiermann in [2] we have established a new and relatively simple interpretation $\mathrm{S}\colon \mathcal{T}(B) \to \omega$ that yields a stronger bounding function of $\mathrm{D}\mathrm{L}_{R_{B'},f}$ $(f \in B)$. Our solution was based on a successful application of CP, employing the intermediate (ordinal) interpretations π and ord, respectively.

Finally, assume that $g \in B^{1,1}$, $h_i \in B^{2,2}$ and $p \in B^{2,1}$. Let f be a new function symbol of arity $2, 1$. Extend $R_{B'}$ by the following rules for f, $f(O, x_1; \bar{n}) \to g(x_1; \bar{n})$ and $f(S_i(; x), x_1; \bar{n}) \to h_i(x, x_1; \bar{n}, f(x, x_1; p(x, x_1, \bar{n}))$. The interpretation S can be extended to show that the derivation length function of the extended rewrite system for f is bounded by a monotone polynomial in the length of the input of the normal arguments. This yields that predicative *parameter* recursion is predicative recursive, cf. [2].

We believe the given simplifications make it easier to grasp the elegance of the term rewriting characterization of PTIME, presented in [2] and could perhaps ease applications of the method proposed by Beckmann and Weiermann.

References

1. T. Arai. Variations on a Theme by Weiermann. *Journal of Symbolic Logic*, 63:897–925, 1998.
2. A. Beckmann and A. Weiermann. A term rewriting characterization of the polytime functions and related complexity classes. *Archive for Mathematical Logic*, (36):11–30, 1996.
3. S. Bellantoni and S. Cook. A new recursion-theoretic characterization of the polytime functions. *Comput. Complexity*, 2(2):97–110, 1992.
4. E.A. Cichon. Termination orderings and complexity characterisations. In P. Aczel, H. Simmons, and S.S. Wainer, editors, *Proof Theory*, pages 171–193, 1992.
5. G. Moser and A. Weiermann. Relating derivation lengths with the slow-growing hierarchy directly. In *Proceedings of RTA*, volume LNCS 2706, pages 296–310. Springer Verlag, 2003.

Tree Automata that Certify Termination of Term Rewriting Systems

Alfons Geser[1*], Dieter Hofbauer[2], Johannes Waldmann[3], and Hans Zantema[4]

[1] National Institute of Aerospace, 144 Research Drive,
Hampton, Virginia 23666, USA. Email: geser@nianet.org
[2] Mühlengasse 16, D-34125 Kassel, Germany.
Email: dieter@theory.informatik.uni-kassel.de
[3] Hochschule für Technik, Wirtschaft und Kultur (FH) Leipzig
Fb IMN, PF 30 11 66, D-04251 Leipzig, Germany. Email: waldmann@imn.htwk-leipzig.de
[4] Faculteit Wiskunde en Informatica, Technische Universiteit Eindhoven
Postbus 513, 5600 MB Eindhoven, The Netherlands. Email: H.Zantema@tue.nl

Introduction

Termination of a string rewriting system can be proven automatically via match bounds [3]. This method is implemented in the tools TORPA [6] and Matchbox [5]. In the present paper, we describe how to extend it to term rewriting.

To prove that a term rewriting system (TRS) R over a signature Σ terminates on a set of terms L, we adopt the following basic plan:

1. Reduce to the problem of termination of some *enrichment* R' over some Σ' on some L'.
2. Show that although R' and Σ' may be infinite, every subset of R' over a finite signature is terminating.
3. Construct a finite tree automaton A that contains L' and is closed under R'-rewriting. Then R' is *compact* for L': every infinite R'-derivation involves only a finite signature.

In all, A certifies termination of R' on L', and therefore R terminates on L.

In the remainder of this note we elaborate on this new automated termination proof method. In particular we show how to choose R' such that proof obligation (2) holds, and we give an approximate construction of rewriting-closed automata. Finally we apply the method for L being the set $\mathrm{RFC}(R)$ of right hand sides of forward closures of R. Termination of R on $\mathrm{RFC}(R)$ implies uniform termination.

Enrichments by Height Annotations

We call a term rewriting system (TRS) R over a signature Σ an *enrichment* of a TRS R' over a signature Σ', if there is a mapping $\mathrm{base} : \mathcal{T}(\Sigma') \to \mathcal{T}(\Sigma)$ such that every R-derivation step can be lifted to an R'-derivation step: for each step $s \to_R t$, and each $s' \in \mathrm{base}^{-1}(s)$, there is some $t' \in \mathrm{base}^{-1}(t)$ with $s' \to_{R'} t'$. In this way the problem of termination of R on L is reduced to the problem of termination of R' on some language L' such that $L = \mathrm{base}(L')$.

We choose the enriched signature $\Sigma' = \Sigma \times \mathbb{N}$, and call the numbers *heights*. We define $\mathrm{base} : \Sigma' \to \Sigma$ by dropping all annotations. We obtain R' from R by annotating the function symbols in the rules. Depending on the linearity of R, we use one of the following:

- the enriched system $\mathrm{top}(R)$ is defined by all right hand side heights being one plus the height of the *top* symbol of the left hand side,

* Partly supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while this author was in residence at the NIA.

- the enriched system $\text{match}(R)$ has all heights of the right hand side of an enriched rule to be one plus the least height of *all* symbols in the left hand side.

Take $R = \{s(x) + 0 \to s(x)\}$ as an example. Then $\text{top}(R)$ contains, among others, the rule $s_1(x) +_2 0_0 \to s_3(x)$, where subscripts denote annotated heights; and $\text{match}(R)$ contains, among others, the rule $s_1(x) +_2 0_0 \to s_1(x)$.

For arbitrary R, every restriction of $\text{top}(R)$ to a finite signature is terminating, by a standard recursive path order argument. For linear R, every restriction of $\text{match}(R)$ to a finite signature is terminating, and the derivational complexity is linear. This follows by ordering the multisets of heights.

Compatible Automata

A *tree automaton* $A = (Q, \Gamma, F, T)$ over a signature Γ consists of a set Q, disjoint from Γ, of nullary symbols, called *states*; a set $F \subseteq Q$ called the *final states*; and a ground rewriting system T over $\Gamma \cup Q$, with rules of the form $f(q_1, \ldots, q_k) \to q$ or $q_0 \to q$ for $f \in \Gamma, q_0, \ldots, q_k, q \in Q$. The automaton is called *finite* if T is finite. Note that for finite T, the reachable parts of Q and Γ are finite as well. The automaton is called *deterministic* if T is non-overlapping. The *language* accepted by the automaton A is $L(A) = \{t \in \mathcal{T}(\Gamma) \mid \exists q \in F : t \to_T^* q\}$.

We call $A = (Q, \Gamma, F, T)$ *compatible* with a term rewriting system S over Γ and a language M over Γ if (1) $M \subseteq L(A)$, and (2) for each rule $(\ell \to r) \in S$, for each state $q \in Q$, and for each substitution $\sigma : \mathcal{V}(\ell) \to Q$, we have that $\ell\sigma \to_T^* q$ implies $r\sigma \to_T^* q$. Under additional restrictions, a compatible automaton is closed under rewriting: If A is compatible with S and M, and A is deterministic or S is left-linear, then $S^*(M) \subseteq L(A)$. Here $S^*(M)$ denotes the set $\{s \in \mathcal{T}(\Gamma) \mid \exists m \in M : m \to_S^* s\}$ of S-descendants of M.

We call a finite or infinite rewriting system S over a finite or infinite signature Γ *compact* for a language M if every derivation starting from M involves only a finite sub-signature of Γ. If A is a finite automaton with $S^*(M) \subseteq L(A)$, then S is compact for M.

Our approach is to construct an automaton that is compatible with a rewriting system R' and a regular tree language L' by repeatedly adding transitions and states to the automaton that accepts L'. This completion procedure for tree automata need not stop. We developed and use heuristics for re-using existing states, inspired by a similar technique for string rewriting in the tool TORPA [6]. We plan to compare our algorithm to the completion procedure given by Genet [2]. We require that R' is left linear in order to avoid determinisation of tree automata.

Right Hand Sides of Forward Closures

A theorem by Dershowitz [1] states that a right-linear TRS R is terminating if and only if it is terminating on the right hand sides of forward closures $\text{RFC}(R)$. Hence if we are able to describe $\text{RFC}(R)$ as a regular tree language, then termination of R can be certified by an automaton compatible with the system R and the language $\text{RFC}(R)$.

For the remainder of this section we assume R to be linear. We construct $\text{RFC}(R)$ while we construct the compatible automaton. We do so by adding a system $\text{border}(R)$ of rules that are created from R rules by replacing some proper non-variable subterms of the left hand side, and the corresponding variable subterms of the right hand side, with a new nullary symbol $\#$.

For example, for the rewriting system

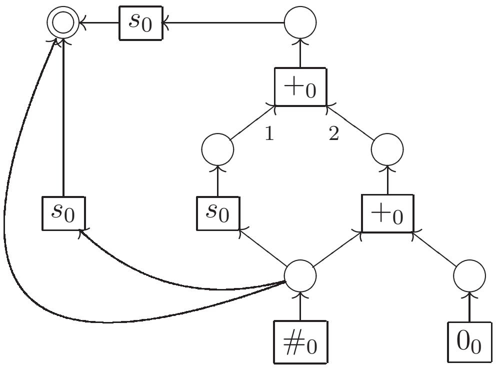
$$R = \{0 + y \to y, s(x) + 0 \to s(x), s(x) + s(y) \to s(s(x) + (y + 0))\},$$

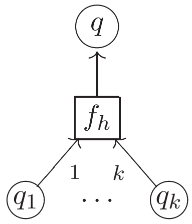
the following system border(R) is obtained:

$$\left\{ \begin{array}{rrr} & \# + 0 \to s(\#), & \# + s(y) \to s(s(\#) + (y + 0)), \\ \# + y \to y, & s(x) + \# \to s(x), & s(x) + \# \to s(s(x) + (\# + 0)), \\ & \# + \# \to s(\#), & \# + \# \to s(s(\#) + (\# + 0)) \end{array} \right\}$$

Then, $\mathrm{RFC}(R)_\# = (R \cup \mathrm{border}(R))^*(\mathrm{rhs}(R)_\#)$, where $(\cdot)_\#$ maps each variable to $\#$. In the example, $\mathrm{rhs}(R)_\# = \{\#, s(\#), s(s(\#) + (\# + 0))\}$. Now, termination of R on $\mathcal{T}(\Sigma)$ follows from termination of R on $(R \cup \mathrm{border}(R))^*(\mathrm{rhs}(R)_\#)$, which will be certified by an automaton compatible with an enriched system. As the enriched system we choose $\mathrm{match}(R) \cup \mathrm{zero}(\mathrm{border}(R))$, where zero is the enrichment that labels all right hand side symbols with 0, totally disregarding the annotations in the left hand side.

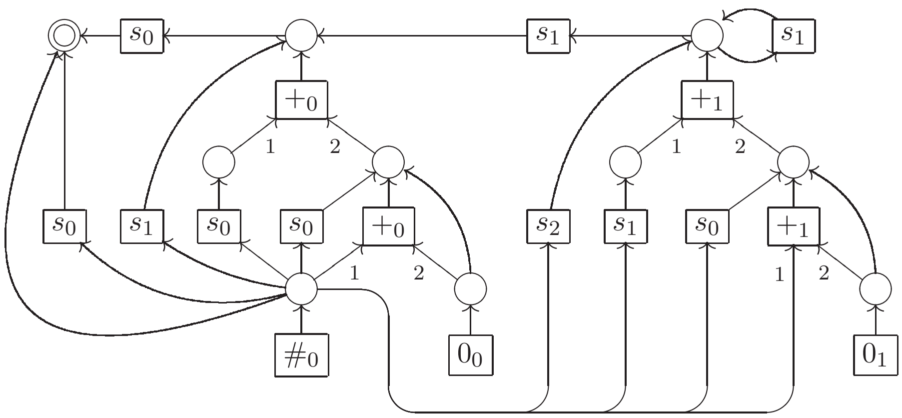
We first construct a tree automaton that accepts the language $\mathrm{lift}_0(\mathrm{rhs}(R)_\#)$, containing the terms from $\mathrm{rhs}(R)_\#$ where all symbols are labelled by 0:





Here, double circles denote final states. Squares contain function symbols with height annotations as subscripts. A transition $f_h(q_1, \ldots, q_n) \to q$ is graphically represented as in the illustration at the right. The argument order of the function symbol is indicated by numbers at the incoming arrows.

Completion yields the compatible automaton below, which establishes the upper bound 2 on match heights, and therefore proves termination of R. Note that the automaton contains a looping transition for s_1 that has been found by our "state re-use heuristics".



Conclusion

We have implemented these algorithms as extensions to the program Matchbox [5]. For linear TRSs, this is a powerful automated termination proof method, with the RFC-match-bound method for strings [3] as a special case. By using top(R) instead of match(R), and $L = \Sigma^*$, we can also prove termination of some left-linear, right-non-linear TRSs.

We use the rewriting system as it is, and try to construct an approximate automaton. Another idea is to approximate the rewriting system by a system from a class for which the set of descendants can be constructed exactly [4].

Directions for further research are: to relate the idea of height annotations to the question of preservation of regular tree languages, and to combine it with standard methods for automated termination proofs.

References

1. Nachum Dershowitz. Termination of linear rewriting systems. In S. Even and O. Kariv (Eds.), *Proc. 8th Int. Coll. Automata, Languages and Programming ICALP-81*, *Lecture Notes in Comput. Sci.* Vol. 115, pp. 448–458. Springer-Verlag, 1981.
2. Thomas Genet. Decidable approximations of sets of descendants and sets of normal forms. In T. Nipkow (Ed.), *Proc. 9th Int. Conf. Rewriting Techniques and Applications RTA-98*, *Lecture Notes in Comput. Sci.* Vol. 1379, pp. 151–165. Springer-Verlag, 1998.
3. Alfons Geser, Dieter Hofbauer and Johannes Waldmann. Match-bounded string rewriting systems. In B. Rovan and P. Vojtas (Eds.), *Proc. 28th Int. Symp. Mathematical Foundations of Computer Science MFCS-03*, *Lecture Notes in Comput. Sci.* Vol. 2747, pp. 449-459. Springer-Verlag, 2003.
4. Aart Middeldorp. Approximations for strategies and termination. In *Proc. 2nd Int. Workshop on Reduction Strategies in Rewriting and Programming*, *Electronic Notes in Theoret. Comput. Sci.* 70(6), 2002.
5. Johannes Waldmann. Matchbox: a tool for match-bounded string rewriting, In V. van Oostrom (Ed.), *Proc. 15th Int. Conf. Rewriting Techniques and Applications RTA-04*, *Lecture Notes in Comp. Sci.*, Springer-Verlag, 2004.
6. Hans Zantema. TORPA: Termination of rewriting proved automatically. In V. van Oostrom (Ed.), *Proc. 15th Int. Conf. Rewriting Techniques and Applications RTA-04*, *Lecture Notes in Comp. Sci.*, Springer-Verlag, 2004.

Reducing the Constraints of the Dependency Pair Approach

Jürgen Giesl, René Thiemann, and Peter Schneider-Kamp

LuFG Informatik II, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany
{giesl|thiemann|psk}@informatik.rwth-aachen.de

1 Introduction

The dependency pair approach [1–3] is a powerful technique for automated (innermost) termination proofs of term rewrite systems (TRSs). For any TRS, it generates inequality constraints that have to be satisfied by well-founded orders. We improve the approach by considerably reducing the number of constraints for (innermost) termination proofs.

Example 1. The following TRS [1] is not simply terminating. Therefore classical approaches for termination proofs fail, while the example is easy to handle with dependency pairs.

$$\mathsf{minus}(x,\mathsf{0}) \to x \quad (1) \qquad \mathsf{quot}(\mathsf{0},\mathsf{s}(y)) \to \mathsf{0} \quad (3)$$
$$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(x,y) \quad (2) \qquad \mathsf{quot}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x,y),\mathsf{s}(y))) \quad (4)$$

The dependency pair approach is recapitulated in Sect. 2. Then we show in Sect. 3 and 4 how to reduce the constraints for both termination and innermost termination proofs. We implemented our results in the automated termination prover AProVE [6] and evaluated them on large collections of examples. The system is available at `http://www-i2.informatik.rwth-aachen.de/AProVE`. The results of this paper are contained in [4,5,10] to which we refer for further details.

2 Dependency Pairs

We restrict ourselves to finite signatures and TRSs. For a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined symbols* $\mathcal{D}$ are the root symbols of the left-hand sides of rules and the *constructors* are $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. For every $f \in \mathcal{D}$, F is a fresh *tuple symbol* with the same arity as f. If $f(s_1,\ldots,s_n) \to r \in \mathcal{R}$ and $g(t_1,\ldots,t_m)$ is a subterm of r with $g \in \mathcal{D}$, then $F(s_1,\ldots,s_n) \to G(t_1,\ldots,t_m)$ is a *dependency pair* of $\mathcal{R}$. So the dependency pairs of the TRS in Ex. 1 are

$$\mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{MINUS}(x,y) \quad (5)$$
$$\mathsf{QUOT}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{MINUS}(x,y) \quad (6)$$
$$\mathsf{QUOT}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{QUOT}(\mathsf{minus}(x,y),\mathsf{s}(y)) \quad (7)$$

A sequence of dependency pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ is a *chain* if there exist substitutions σ_j such that $t_j\sigma_j \to_{\mathcal{R}}^* s_{j+1}\sigma_{j+1}$ for all j. A chain is an *innermost chain* if $t_j\sigma_j \xrightarrow{\text{i}}_{\mathcal{R}}^* s_{j+1}\sigma_{j+1}$ and if $s_j\sigma_j$ is a normal form for all j.

Theorem 1. *$\mathcal{R}$ is (innermost) terminating iff there is no infinite (innermost) chain.*

To estimate which dependency pairs may occur consecutively in (innermost) chains, we build an (innermost) *dependency graph*. Its nodes are the dependency pairs and there is an arc from $s \to t$ to $v \to w$ iff $s \to t$, $v \to w$ is an (innermost) chain. In Ex. 1, the (innermost) dependency graph has arcs from (5) to itself, from (6) to (5), and from (7) to (6) and to itself. So it has the cycles $\mathcal{P}_1 = \{(5)\}$ and $\mathcal{P}_2 = \{(7)\}$. Since it is undecidable whether two dependency pairs form an (innermost) chain, for automation one constructs *estimated* graphs such that all arcs in the real graph are also arcs in the estimated graph [1,5,8].

To show (innermost) termination, one can prove absence of infinite (innermost) chains separately for every cycle of the (innermost) dependency graph. To this end, one generates constraints which should be satisfied by some *reduction pair* $(\succsim, \succ)$ consisting of a quasi-rewrite order $\succsim$ (i.e., $\succsim$ is reflexive, transitive, monotonic, and stable) and a stable well-founded order $\succ$ which is compatible with $\succsim$ (i.e., $\succsim \circ \succ \subseteq \succ$ and $\succ \circ \succsim \subseteq \succ$). However, $\succ$ need not be monotonic. For that reason, before synthesizing a suitable order, some arguments of function symbols can be eliminated by an *argument filtering*.

Theorem 2. *A TRS $\mathcal{R}$ is terminating iff for every cycle $\mathcal{P}$ of the dependency graph, there is a reduction pair $(\succsim, \succ)$ and argument filtering π such that*

(a) $\pi(s) \succ \pi(t)$ for at least one pair $s \to t \in \mathcal{P}$ and $\pi(s) \succsim \pi(t)$ for all other $s \to t \in \mathcal{P}$
(b) $\pi(l) \succsim \pi(r)$ for all rules $l \to r \in \mathcal{R}$

A TRS $\mathcal{R}$ is innermost terminating if for every cycle $\mathcal{P}$ of the innermost dependency graph, there is a reduction pair $(\succsim, \succ)$ and an argument filtering π satisfying both (a) and

(c) $\pi(l) \succsim \pi(r)$ for all rules $l \to r \in \mathcal{U}(\mathcal{P})$

Here, $\mathcal{U}(\mathcal{P})$ are all rules that are *usable* for the symbols in right-hand sides of $\mathcal{P}$'s dependency pairs. For a symbol f, all f-rules are usable and if g occurs in the right-hand side of an f-rule, then all usable rules for g are also usable for f.

So in Ex. 1, we obtain the following 10 constraints for termination. Here, $(\succsim_i, \succ_i)$ is the reduction pair and π_i is the argument filtering for cycle $\mathcal{P}_i$, where $i \in \{1, 2\}$.

$$\pi_1(\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y))) \succ_1 \pi_1(\mathsf{MINUS}(x, y)) \tag{8}$$
$$\pi_2(\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y))) \succ_2 \pi_2(\mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y))) \tag{9}$$
$$\pi_i(\mathsf{minus}(x, \mathsf{0})) \succsim_i \pi_i(x) \tag{10}$$
$$\pi_i(\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y))) \succsim_i \pi_i(\mathsf{minus}(x, y)) \tag{11}$$
$$\pi_i(\mathsf{quot}(\mathsf{0}, \mathsf{s}(y))) \succsim_i \pi_i(\mathsf{0}) \tag{12}$$
$$\pi_i(\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y))) \succsim_i \pi_i(\mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y)))) \tag{13}$$

Let π_i be the argument filtering that replaces all terms $\mathsf{minus}(t_1, t_2)$ by $\mathsf{minus}(t_1)$. With this filtering, (8) – (13) are satisfied by the lexicographic path order (LPO) with the precedence $\mathsf{quot} > \mathsf{s} > \mathsf{minus}$. So termination of the TRS is proved.

For innermost termination, we only obtain the constraint (8) for the cycle $\mathcal{P}_1$, since it has no usable rules. For $\mathcal{P}_2$, the constraints (12) and (13) are not necessary, since the quot-rules are not usable for any symbol in right-hand sides of dependency pairs.

3 Reducing Constraints by $\mathcal{C}_\varepsilon$-compatibility and Dependency Graphs

Now we improve Thm. 2 for termination proofs and show that even for termination, it suffices to require $\pi(l) \succsim \pi(r)$ just for the usable rules $l \to r$ from $\mathcal{U}(\mathcal{P})$ instead of all rules from $\mathcal{R}$. So essentially, one may replace (b) by (c) in Thm. 2, provided one imposes a minor restriction on the reduction pairs $(\succsim, \succ)$ used.

To this end, we improve a result of Urbain [12] who showed how to use dependency pairs for modular termination proofs of hierarchical combinations of $\mathcal{C}_\varepsilon$-terminating TRSs. $\mathcal{R}$ is *$\mathcal{C}_\varepsilon$-terminating* iff $\mathcal{R} \cup \mathcal{C}_\varepsilon$ terminates where $\mathcal{C}_\varepsilon = \{\mathsf{c}(x,y) \to x, \mathsf{c}(x,y) \to y\}$ for a fresh symbol $\mathsf{c} \notin \mathcal{F}$. But due to the restriction to $\mathcal{C}_\varepsilon$-termination, in [12] one could not use the full power of dependency graphs. For example, recent estimations [8] detect that the dependency graph for the TRS $\mathsf{f}(\mathsf{0},\mathsf{1},x) \to \mathsf{f}(x,x,x)$ of Toyama [11] has no cycle and thus, it is terminating. But since it is not $\mathcal{C}_\varepsilon$-terminating, this example cannot be handled by the approach of [12].

Thm. 3 improves the results of [12] in two ways: First, it integrates the approach of [12] with (arbitrary estimations of) dependency graphs and combines their advantages. To this end, we remove the restriction to $\mathcal{C}_\varepsilon$-terminating TRSs in Thm. 3. Instead, one only has to restrict the reduction pairs to be $\mathcal{C}_\varepsilon$-compatible. This requirement is satisfied by virtually all reduction pairs used in practice (based on RPOS, KBO, or polynomial orders).

Moreover, [12] required a weak decrease of the form "$\pi(l) \succsim \pi(r)$" for unnecessary many rules $l \to r$. For example, to prove termination of the minus-rules (1) and (2), in [12] one obtained the constraints (8), (10), and (11). However, Thm. 3 shows that demanding (10) and (11) is not necessary to verify the termination of minus. More precisely, for each cycle $\mathcal{P}$ of the dependency graph it suffices to require $\pi(l) \succsim \pi(r)$ only for the usable rules $l \to r$ of $\mathcal{P}$. Hence, the constraints for termination are analogous to the ones for innermost termination except that for termination we have to require $\mathcal{C}_\varepsilon$-compatibility of the quasi-order $\succsim$ (i.e., $\mathsf{c}(x,y) \succsim x$ and $\mathsf{c}(x,y) \succsim y$).

Theorem 3. *A TRS $\mathcal{R}$ is terminating if for every cycle $\mathcal{P}$ of the dependency graph, there is a reduction pair $(\succsim, \succ)$ and an argument filtering π such that*

(a) $\pi(s) \succ \pi(t)$ *for at least one pair* $s \to t \in \mathcal{P}$ *and* $\pi(s) \succsim \pi(t)$ *for all other* $s \to t \in \mathcal{P}$
(b) $\pi(l) \succsim \pi(r)$ *for all rules* $l \to r \in \mathcal{U}(\mathcal{P}) \cup \mathcal{C}_\varepsilon$

Now the constraints of type (b) are reduced significantly and it becomes easier to find a reduction pair satisfying them. For instance, termination of a well-known example of [9] to compute intervals of natural numbers cannot be shown with Thm. 2 and a reduction pair based on simplification orders, whereas a proof with Thm. 3 and LPO is easy, cf. [4]. In fact, by the refinement of Thm. 3, proving termination is almost the same as proving innermost termination.

4 Reducing Constraints by Argument Filtering Before Usable Rules

Now we improve Thm. 2 for innermost termination and Thm. 3 for termination further. These approaches can be refined by applying the argument filtering π first and determining the usable rules $\mathcal{U}(\mathcal{P},\pi)$ afterwards. The advantage is that by the argument filtering, some symbols f may have been eliminated from right-hand sides of dependency pairs and rules. Then the f-rules do not have to be weakly decreasing anymore. Here, $\mathcal{U}(\mathcal{P},\pi)$ are all rules that are *π-usable* for the symbols in $\{\pi(t) \mid s \to t \in \mathcal{P}\}$. For f, all f-rules are π-usable and if g occurs in $\pi(r)$ for an f-rule $l \to r$, then all π-usable rules for g are also π-usable for f. However, this refinement is only sound for non-collapsing argument filterings. If π is collapsing, then one should instead use $\mathcal{U}(\mathcal{P},\pi')$, where π' is like π but does not remove function symbols.

Example 2. We illustrate the new definition of usable rules with the following TRS of [7].

$$\begin{array}{rlrl} \mathsf{rev}(\mathsf{nil}) & \to \mathsf{nil} & \mathsf{rev1}(x, \mathsf{nil}) & \to x \\ \mathsf{rev}(\mathsf{cons}(x,l)) & \to \mathsf{cons}(\mathsf{rev1}(x,l), \mathsf{rev2}(x,l)) & \mathsf{rev1}(x, \mathsf{cons}(y,l)) & \to \mathsf{rev1}(y,l) \\ \mathsf{rev2}(x, \mathsf{nil}) & \to \mathsf{nil} & \mathsf{rev2}(x, \mathsf{cons}(y,l)) & \to \mathsf{rev}(\mathsf{cons}(x, \mathsf{rev}(\mathsf{rev2}(y,l)))) \end{array}$$

For any cycle containing $\mathsf{REV2}(x, \mathsf{cons}(y,l)) \to \mathsf{REV}(\mathsf{cons}(x, \mathsf{rev}(\mathsf{rev2}(y,l))))$, up to now all rules were usable, since rev and $\mathsf{rev2}$ occur in the right-hand side and rev calls rev_1. However, if the argument filtering π removes the first argument of cons, then the rev_1-rules are not π-usable. The reason is that while rev and $\mathsf{rev2}$ still occur in the right-hand side of the filtered dependency pair, $\mathsf{rev1}$ no longer occurs in right-hand sides of filtered rev- or $\mathsf{rev2}$-rules.

Theorem 4. *A TRS $\mathcal{R}$ is terminating if for every cycle $\mathcal{P}$ of the dependency graph, there is a reduction pair $(\succsim, \succ)$ and an argument filtering π such that*

(a) $\pi(s) \succ \pi(t)$ for at least one pair $s \to t \in \mathcal{P}$ and $\pi(s) \succsim \pi(t)$ for all other $s \to t \in \mathcal{P}$
(b) $\pi(l) \succsim \pi(r)$ for all rules $l \to r \in \mathcal{U}(\mathcal{P}, \pi) \cup \mathcal{C}_\varepsilon$

A TRS $\mathcal{R}$ is innermost terminating if for every cycle $\mathcal{P}$ of the innermost dependency graph, there is a reduction pair $(\succsim, \succ)$ and an argument filtering π satisfying both (a) and

(c) $\pi(l) \succsim \pi(r)$ for all rules $l \to r \in \mathcal{U}(\mathcal{P}, \pi)$

The TRS from Ex. 2 shows the advantages of Thm. 4. When proving innermost termination with Thm. 2 or when proving termination with Thm. 3, for the cycle of the REV- and $\mathsf{REV2}$-dependency pairs, we obtain inequalities from the dependency pairs and $\pi(l) \succsim \pi(r)$ for all rules $l \to r$, since all rules are usable. But with standard reduction pairs based on RPOS, KBO, or polynomial orders, these constraints are not satisfied for any argument filtering. In contrast, with Thm. 4 and a suitable argument filtering, all constraints are oriented by the embedding order, cf. [4].

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Applicable Algebra in Engineering, Communication & Computing*, 12(1,2):39–72, 2001.
3. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
4. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proc. 10th LPAR*, LNAI 2850, pages 165–179, 2003.
5. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing dependency pairs. Technical Report AIB-2003-08, RWTH Aachen, Germany, 2003. `http://aib.informatik.rwth-aachen.de`.
6. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proc. 15th RTA*, LNCS, 2004. To appear.
7. G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25:239–299, 1982.
8. A. Middeldorp. Approximations for strategies and termination. In *Proc. 2nd WRS*, ENTCS 70(6), 2002.
9. J. Steinbach. Automatic termination proofs with transformation orderings. In *Proc. 6th RTA*, LNCS, pages 11–25, 1995. Full version appeared as Technical Report SR-92-23, Universität Kaiserslautern, Germany.
10. R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proc. 2nd IJCAR*, LNAI, 2004. To appear.
11. Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.
12. X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In *Proc. IJCAR 2001*, LNAI 2083, pages 485–498, 2001.

Modularity of Termination of Left-Linear Rewrite Systems Revisited

Bernhard Gramlich

Fakultät für Informatik, TU Wien, Austria
gramlich@logic.at

Abstract. We investigate modularity of termination of left-linear rewrite systems in the constructor sharing case. More precisely, we study cases where both involved system are left-linear, as well as other related cases where the conditions on the involved systems systems (like left-linearity) are not the same. Several tempting approaches to generalize the well-known deep modularity results of [21,22], [8,18] are discussed and falsified by counterexamples. By a thorough analysis of the crucial notions of *uniquely collapsing* and *consistency w.r.t. reduction* and of the role of left-linearity in *collapsing* and, more generally, *shared symbol lifting* reductions we finally show how to generalize certain modularity results for termination in the case of disjoint unions to combinations of constructor sharing systems. The problem in the symmetric case (where both systems are left-linear) remains open, whereas we are able to extend a couple of asymmetric modularity results from [4] to combinations of constructor sharing systems.

1 Background

Modularity of termination of rewrite systems has been extensively and thoroughly investigated in the past, under various types of combinations. Starting with the pioneering work of Toyama [20,19], especially with the fact that termination is in general not even modular for disjoint unions, a whole series of (positive and negative) results have been obtained so far (cf. e.g. [17], [13,14], [5,6], [2,3], [15,16], [1]). Subsequently, most of these results for disjoint unions – whether symmetric or asymmetric preservation criteria - have also been extended to more general types of non-disjoint combinations, namely *constructor sharing* systems, *composable* ones, and (certain) *hierarchical* combinations.

Somehow surprisingly, there is one major exception with the extension of basic results from the disjoint union case, namely for the case of left-linear systems. Toyama, Klop and Barendregt [21,22] have shown by a very sophisticated analysis of *collapsing reduction*:

(1) **Termination is modular for left-linear confluent systems**.

Later, independently Marchiori and Schmidt-Schauss & Panitz ([8], [18]) succeeded in giving a simplified proof of this deep result. Actually, their proof(s)[1] rely on the same basic observations (about the uniqueness of certain collapsing reductions) as the proof of [21,22]. However, the actual construction of [8], [18] for transforming an assumed minimal counterexample in the union to a smaller one is somewhat easier (to understand) than the proof in [21,22].[2] Moreover, they have shown that instead of confluence the weaker assumption of *consistency w.r.t. reduction ($CON^{\rightarrow}$)* suffices, too, i.e.:

(2) **Termination is modular for left-linear systems satisfying $\mathbf{CON}^{\rightarrow}$**.

The property $\mathrm{CON}^{\rightarrow}$ says that a reduction of a term to two distinct variables is impossible, and means that in disjoint unions the *collapsing of some mixed term* to some of its *aliens* (or *principal subterms*) is only possible in a very restricted way.

[1] which are not identical but very similar
[2] Yet, it should be noted that in both papers [8], [18] these transformations are not really formally defined, but rather sketched. A completely formal presentation would most probably be much more involved.

Strikingly, and in contrast to most of the other modularity results for termination, none of the latter results (for left-linear systems) have been extended later on to non-disjoint unions. Perhaps this is partially due to the fact that the authors in [18] mentioned

> "The proof technique in this paper appears not to be extendable to the sum of term rewriting systems where there may be common constructors, i.e., function symbols that do not appear at the top in the left-hand sides of rules ...

Furthermore they argue that there are simple counterexamples to (2) above if shared constructors are permitted, e.g. the following variant of Toyama's basic counterexample (cf. [20,19]):

$$R_1 : h(a, b, x) \to h(x, x, x) \qquad\qquad R_2 : D \to a, D \to b$$

These two systems only share the common constructors a, b. Both are left-linear, terminating, and CON$^{\to}$. Yet, their union allows a cycle:

$$h(D, D, D) \to h(a, D, D) \to h(a, b, D) \to h(D, D, D)$$

So, there seems to be no hope to extend (1), (2) above to non-disjoint combinations of rewrite systems.

The properties of being *uniquely collapsing*, *collapsing confluent* (cf. [4]) and left-linear are crucial in the analysis of the above *symmetric* case where both systems are supposed to be left-linear:

- Intuitively, a system is *uniquely collapsing* if whenever there is a reduction $s \to^* x$ from a term to a variable, then the *predecessor* of x in s is unique.
- A rewrite system is *collapsing confluent* whenever $s \to^* x$ and $s \to^* t$ imply $t \to^* x$, i.e., reduction to a variable remains always possible.

These properties are also crucial for various other *asymmetric* modularity results on termination of disjoint unions, e.g., when only one of the systems is supposed to be left-linear. Here we will consider the following known results in this area ([4]:

(3) **If one system is non-collapsing and the other one is both uniquely collapsing and collapsing confluent, then their disjoint union is terminating.**

(4) **If one system is non-collapsing and left-linear and the other one is uniquely collapsing, then their disjoint union is terminating.**

2 Our Approach

Regarding (1), (2), we will show that an extension attempt as above is too native to work. From the extension of other basic results about modularity of termination for disjoint unions to constructor sharing systems (and more general combinations) it is also well known that the additional complications caused by the increased amount of *shared information* in the union (especially the increasing possibilities of inferences between reduction steps in one system and steps in the other system) have to be taken appropriately into account.

We do this concerning (2) above, by a thorough analysis of the role of the preconditions *left-linearity*, *consistency w.r.t. reduction (CON*$^{\to}$*)* and of related properties (cf. [4]). We present a couple of tempting approaches and conjectures of how to adapt the preconditions of (2) in such a way that the extension to constructor sharing systems holds, and also provide counterexamples to most of them. Based on this analysis, we finally arrive at an extended version of (2) for systems with shared constructors. However, so far we have neither been able to prove it nor to discover a counterexample. [3] Hence, this remains an open problem.

On the positive side, we show that the concepts of *uniquely collapsing* reduction and *collapsing confluence* can be extended in a natural way to the constructor sharing case such that for instance the asymmetric results (3) and (4) above can indeed be extended to this case.

We think that our results obtained are not only interesting from the point of view of providing new termination criteria, but possibly even more regarding the structural analysis of (uniqueness of) *collapsing* or, more generally, *shared symbol lifting* reductions in combinations of systems with shared constructors.

References

1. Thomas Arts and Jürgen Giesl. Modularity of termination using dependency pairs. In Tobias Nipkow, editor, *Proc. 9th Int. Conf. on Rewriting Techniques and Applications (RTA'98)*, LNCS 1379, pages 226–240, Tsukuba, Japan, 1998. Springer-Verlag.
2. Bernhard Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994.
3. Bernhard Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.
4. Bernhard Gramlich. Modular aspects of rewrite-based specifications. In Francesco Parisi-Presicce, editor, *Recent Trends in Algebraic Development Techniques, 12th Int. Workshop, WADT 97, Tarquinia, Italy, June 1997, Selected Papers*, LNCS 1376, pages 253–268. Springer-Verlag, 1998.
5. Masahito Kurihara and Azuma Ohuchi. Modularity of simple termination of term rewriting systems with shared constructors. *Theoretical Computer Science*, 103:273–282, 1992.
6. Masahito Kurihara and Azuma Ohuchi. Modularity in noncopying term rewriting. *Theoretical Computer Science*, 152(1):139–169, December 1995.
7. Massimo Marchiori. Modularity of $\mathrm{UN}^{\rightarrow}$ for left-linear term rewriting systems. Technical Report CS-R9433, CWI (Centre for Mathematics and Computer Science), Amsterdam, May 1994.
8. Massimo Marchiori. Modularity of completeness revisited. In Jieh Hsiang, editor, *Proc. 6th Int. Conf. on Rewriting Techniques and Applications (RTA'95)*, LNCS 914, pages 2–10, Kaiserslautern, Germany, April 1995. Springer-Verlag.
9. Massimo Marchiori. On the modularity of normal forms in rewriting. *Journal of Symbolic Computation*, 22(2):143–154, 1996.
10. Massimo Marchiori. The theory of vaccines. In Pierpaolo Degano, Robert Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Proc. 24th Int. Conf. on Automata, Languages and Programming*, Lecture Notes in Computer Science, pages 660–670, Bologna, Italy, July 1997. Springer-Verlag.
11. Massimo Marchiori. Bubbles in modularity. *Theoretical Computer Science*, 192(1):31–54, 1998.
12. Aart Middeldorp. Modular aspects of properties of term rewriting systems related to normal forms. In N. Dershowitz, editor, *Proc. 3rd Int. Conf. on Rewriting Techniques and Applications*, LNCS 355, pages 263–277. Springer-Verlag, 1989.
13. Aart Middeldorp. A sufficient condition for the termination of the direct sum of term rewriting systems. In *Proceedings of the 4th IEEE Symposium on Logic in Computer Science*, pages 396–401, Pacific Grove, 1989.
14. Aart Middeldorp. *Modular Properties of Term Rewriting Systems.* PhD thesis, Free University, Amsterdam, 1990.
15. Enno Ohlebusch. On the modularity of termination of term rewriting systems. *Theoretical Computer Science*, 136:333–360, 1994.

[3] In essence, a proof in the spirit of [18] (still) seems to fail, since the *deletion part* of the *pile-and-delete* technique of [18], [22] appears not to work anymore (as argued already in [18]).

16. Enno Ohlebusch. Termination is not modular for confluent variable-preserving term rewriting systems. *Information Processing Letters*, 53:223–228, March 1995.
17. Michaël Rusinowitch. On termination of the direct sum of term rewriting systems. *Information Processing Letters*, 26:65–70, 1987.
18. Manfred Schmidt-Schauß, Massimo Marchiori, and Sven Eric Panitz. Modular termination of r-consistent and left-linear term rewriting systems. *Theoretical Computer Science*, 149(2):361–374, October 1995.
19. Yoshihito Toyama. Counterexamples to termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.
20. Yoshihito Toyama. On the Church-Rosser property for the direct sum of term rewriting systems. *Journal of the ACM*, 34(1):128–143, 1987.
21. Yoshihito Toyama, Jan Willem Klop, and Henk Pieter Barendregt. Termination for the direct sum of left-linear term rewriting systems. In N. Dershowitz, editor, *Proc. 3rd Int. Conf. on Rewriting Techniques and Applications (RTA'89)*, LNCS 355, pages 477–491. Springer-Verlag, 1989.
22. Yoshihito Toyama, Jan Willem Klop, and Henk Pieter Barendregt. Termination for direct sums of left-linear complete term rewriting systems. *Journal of the ACM*, 42(6):1275–1304, November 1995.

Natural Polynomial Interpretations

Nao Hirokawa and Aart Middeldorp

Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria
nao.hirokawa@uibk.ac.at
aart.middeldorp@uibk.ac.at

Abstract. We show that polynomial interpretations with negative constants can be used effectively to prove termination of term rewrite systems.

1 Introduction

Consider the following recursive definition

$$f(x) = \text{if } x > 0 \text{ then } f(f(x-1)) + 1 \text{ else } 0$$

from [5]. It computes the identity function over the natural numbers. Termination of the rewrite system

$$1\colon\ \mathsf{f}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{f}(\mathsf{f}(\mathsf{p}(\mathsf{s}(x))))) \qquad 2\colon\ \mathsf{f}(\mathsf{0}) \to \mathsf{0} \qquad 3\colon\ \mathsf{p}(\mathsf{s}(x)) \to x$$

obtained after the obvious translation is not easily proved. The (manual) proof in [5] relies on forward closures whereas powerful automatic tools like AProVE [6] and CiME [4] that incorporate both polynomial interpretations and the dependency pair method fail to prove termination. There are three dependency pairs:

$$4\colon\ \mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(\mathsf{f}(\mathsf{p}(\mathsf{s}(x)))) \qquad 5\colon\ \mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(\mathsf{p}(\mathsf{s}(x))) \qquad 6\colon\ \mathsf{F}(\mathsf{s}(x)) \to \mathsf{P}(\mathsf{s}(x))$$

By taking the *natural* polynomial interpretation

$$\mathsf{f}_{\mathbb{Z}}(x) = \mathsf{F}_{\mathbb{Z}}(x) = x \qquad \mathsf{s}_{\mathbb{Z}}(x) = x + 1 \qquad \mathsf{0}_{\mathbb{Z}} = 0 \qquad \mathsf{p}_{\mathbb{Z}}(x) = \mathsf{P}_{\mathbb{Z}}(x) = x - 1$$

over the integers, the rule and dependency pair constraints reduce to the following inequalities:

$$1\colon\quad x + 1 \geqslant x + 1 \qquad 2\colon\quad 0 \geqslant 0 \qquad 3\colon\quad x \geqslant x \qquad 4,5,6\colon\quad x + 1 > x$$

These constraints are obviously satisfied. The question is whether we are allowed to conclude termination at this point. We will argue that the answer is affirmative and, moreover, that the search for appropriate natural polynomial interpretations can be efficiently implemented.

The approach described in this paper is inspired by the combination of the general path order and forward closures [5] as well as semantic labeling [9].

2 Theoretical Framework

The first challenge we face is that the standard order $>$ on $\mathbb{Z}$ is not well-founded. Restricting the domain to the set $\mathbb{N}$ of natural numbers makes an interpretation like $\mathsf{p}_{\mathbb{Z}}(x) = x - 1$ ill-defined.

Definition 1. Let $\mathcal{F}$ be a signature and let $(\mathbb{Z}, \{f_{\mathbb{Z}}\}_{f \in \mathcal{F}})$ be an $\mathcal{F}$-algebra such that every interpretation function is weakly monotone in all its arguments: $f_{\mathbb{Z}}(x_1, \ldots, x_n) \geqslant f_{\mathbb{Z}}(y_1, \ldots, y_n)$ whenever $x_i \geqslant y_i$ for all $1 \leqslant i \leqslant n$. The interpretation functions of the *induced* algebra $(\mathbb{N}, \{f_{\mathbb{N}}\}_{f \in \mathcal{F}})$ are defined as follows: $f_{\mathbb{N}}(x_1, \ldots, x_n) = \max\{0, f_{\mathbb{Z}}(x_1, \ldots, x_n)\}$ for all $x_1, \ldots, x_n \in \mathbb{N}$. We write $s >_{\mathbb{N}} t$ if $[\alpha]_{\mathbb{N}}(s) > [\alpha]_{\mathbb{N}}(t)$ and $s \geqslant_{\mathbb{N}} t$ if $[\alpha]_{\mathbb{N}}(s) \geqslant [\alpha]_{\mathbb{N}}(t)$ for all assignments α of natural numbers for the variables in s and t.

It is easy to show that the interpretations functions of the induced algebra are weakly monotone in all arguments. Routine arguments reveal that the relation $>_{\mathbb{N}}$ is a well-founded order which is closed under substitutions and that $\geqslant_{\mathbb{N}}$ is a preorder closed under contexts and substitutions. Moreover, the inclusion $>_{\mathbb{N}} \circ \geqslant_{\mathbb{N}} \subseteq >_{\mathbb{N}}$ holds. These properties make the pair $(>_{\mathbb{N}}, \geqslant_{\mathbb{N}})$ suitable as basic ingredient of powerful termination methods like the dependency pair method [1] and the monotonic semantic path order [3].

It is interesting to remark that unlike usual polynomial interpretations, the relation $\geqslant_{\mathbb{N}}$ does not have the weak subterm property. For instance, with respect to the interpretations in the example of the introduction, we have $\mathsf{s}(0) >_{\mathbb{N}} \mathsf{p}(\mathsf{s}(0))$ and not $\mathsf{p}(\mathsf{s}(0)) \geqslant_{\mathbb{N}} \mathsf{s}(0)$.

Example 1. Consider TRS consisting of the following rewrite rules:

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

The function $\mathsf{half}(x)$ computes $\lceil \frac{x}{2} \rceil$ and $\mathsf{bits}(x)$ computes the number of bits that are needed to represent all numbers less than or equal to x. Termination of this TRS is proved in [2] by using the dependency pair method together with the narrowing refinement. There are three dependency pairs:

$$
\begin{array}{rlrl}
6: & \mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{HALF}(x) & 7: & \mathsf{BITS}(\mathsf{s}(x)) \to \mathsf{BITS}(\mathsf{half}(\mathsf{s}(x))) \\
 & & 8: & \mathsf{BITS}(\mathsf{s}(x)) \to \mathsf{HALF}(\mathsf{s}(x))
\end{array}
$$

By taking the interpretations $0_{\mathbb{Z}} = 0$, $\mathsf{half}_{\mathbb{Z}}(x) = x - 1$, $\mathsf{bits}_{\mathbb{Z}}(x) = \mathsf{HALF}_{\mathbb{Z}}(x) = x$, and $\mathsf{s}_{\mathbb{Z}}(x) = \mathsf{BITS}_{\mathbb{Z}}(x) = x + 1$, we obtain the following constraints over $\mathbb{N}$:

$$
\begin{array}{rlrlrl}
1,2,4: & 0 \geqslant 0 & 3: & x + 1 \geqslant \max\{0, x - 1\} + 1 & 5: & x + 1 \geqslant x + 1 \\
 & & 6: & x + 2 > x & 7,8: & x + 2 > x + 1
\end{array}
$$

These constraints are satisfied, so the TRS is terminating, but how can an inequality like $x + 1 \geqslant \max\{0, x - 1\} + 1$ be verified automatically?

3 Towards Automation

Because the inequalities resulting from interpretations with negative constants may contain the max operator, we cannot use standard techniques for comparing polynomial expressions. In order to avoid reasoning by case analysis ($x - 1 > 0$ or $x - 1 \leqslant 0$ for constraint 3 in Example 1), we approximate the evaluation function of the induced algebra.

Definition 2. Given a polynomial P with coefficients in $\mathbb{Z}$, we denote the constant part by $c(P)$ and the non-constant part $P - c(P)$ by $n(P)$. Let $(\mathbb{Z}, \{f_\mathbb{Z}\}_{f\in\mathcal{F}})$ be an $\mathcal{F}$-algebra such that every $f_\mathbb{Z}$ is a weakly monotone polynomial. With every term t we associate polynomials $P_{\mathit{left}}(t)$ and $P_{\mathit{right}}(t)$ with coefficients in $\mathbb{Z}$ and variables in t as indeterminates:

$$P_{\mathit{left}}(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ 0 & \text{if } t = f(t_1, \dots, t_n),\ n(P_1) = 0, \text{ and } c(P_1) < 0 \\ P_1 & \text{otherwise} \end{cases}$$

$$P_{\mathit{right}}(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ n(P_2) & \text{if } t = f(t_1, \dots, t_n) \text{ and } c(P_2) < 0 \\ P_2 & \text{otherwise} \end{cases}$$

Here $P_1 = f_\mathbb{Z}(P_{\mathit{left}}(t_1), \dots, P_{\mathit{left}}(t_n))$ and $P_2 = f_\mathbb{Z}(P_{\mathit{right}}(t_1), \dots, P_{\mathit{right}}(t_n))$. Let $\alpha\colon \mathcal{V} \to \mathbb{N}$ be an assignment. The result of evaluating $P_{\mathit{left}}(t)$ and $P_{\mathit{right}}(t)$ under α is denoted by $[\alpha]^l_\mathbb{Z}(t)$ and $[\alpha]^r_\mathbb{Z}(t)$. Furthermore, the result of evaluating a polynomial P under α is denoted by $\alpha(P)$.

Lemma 1. *Let $(\mathbb{Z}, \{f_\mathbb{Z}\}_{f\in\mathcal{F}})$ be an $\mathcal{F}$-algebra such that every $f_\mathbb{Z}$ is a weakly monotone polynomial. Let t be a term. For every assignment $\alpha\colon \mathcal{V} \to \mathbb{N}$ we have $[\alpha]^r_\mathbb{Z}(t) \geqslant [\alpha]_\mathbb{N}(t) \geqslant [\alpha]^l_\mathbb{Z}(t)$.*

Proof. By induction on the structure of t. If $t \in \mathcal{V}$ then $[\alpha]^r_\mathbb{Z}(t) = [\alpha]^l_\mathbb{Z}(t) = \alpha(t) = [\alpha]_\mathbb{N}(t)$. Suppose $t = f(t_1, \dots, t_n)$. According to the induction hypothesis, $[\alpha]^r_\mathbb{Z}(t_i) \geqslant [\alpha]_\mathbb{N}(t_i) \geqslant [\alpha]^l_\mathbb{Z}(t_i)$ for all i. Since $f_\mathbb{Z}$ is weakly monotone,

$$f_\mathbb{Z}([\alpha]^r_\mathbb{Z}(t_1), \dots, [\alpha]^r_\mathbb{Z}(t_n)) \geqslant f_\mathbb{Z}([\alpha]_\mathbb{N}(t_1), \dots, [\alpha]_\mathbb{N}(t_n)) \geqslant f_\mathbb{Z}([\alpha]^l_\mathbb{Z}(t_1), \dots, [\alpha]^l_\mathbb{Z}(t_n))$$

By applying the weakly monotone function $\max\{0, \cdot\}$ we obtain $\max\{0, \alpha(P_2)\} \geqslant [\alpha]_\mathbb{N}(t) \geqslant \max\{0, \alpha(P_1)\}$ where $P_1 = f_\mathbb{Z}(P_{\mathit{left}}(t_1), \dots, P_{\mathit{left}}(t_n))$ and $P_2 = f_\mathbb{Z}(P_{\mathit{right}}(t_1), \dots, P_{\mathit{right}}(t_n))$. We have

$$[\alpha]^l_\mathbb{Z}(t) = \begin{cases} 0 & \text{if } n(P_1) = 0 \text{ and } c(P_1) < 0 \\ \alpha(P_1) & \text{otherwise} \end{cases}$$

and thus $[\alpha]^l_\mathbb{Z}(t) \leqslant \max\{0, \alpha(P_1)\}$. Likewise,

$$[\alpha]^r_\mathbb{Z}(t) = \begin{cases} \alpha(n(P_2)) & \text{if } c(P_2) < 0 \\ \alpha(P_2) & \text{otherwise} \end{cases}$$

In the former case, $\alpha(n(P_2)) = \alpha(P_2) - c(P_2) > \alpha(P_2)$ and $\alpha(n(P_2)) \geqslant 0$. In the latter case $\alpha(P_2) \geqslant 0$. So in both cases we have $[\alpha]^r_\mathbb{Z}(t) \geqslant \max\{0, \alpha(P_2)\}$. Hence we obtain the desired inequalities. □

An immediate consequence of the preceding lemma is that $[\alpha]^l_\mathbb{Z}(s) \geqslant [\alpha]^r_\mathbb{Z}(t)$ ($[\alpha]^l_\mathbb{Z}(s) > [\alpha]^r_\mathbb{Z}(t)$) is a sufficient condition for $[\alpha]_\mathbb{N}(s) \geqslant [\alpha]_\mathbb{N}(t)$ ($[\alpha]_\mathbb{N}(s) > [\alpha]_\mathbb{N}(t)$) and hence we can safely replace a constraint $l \geqslant_\mathbb{N} r$ ($l >_\mathbb{N} r$) by $P_{\mathit{left}}(l) \geqslant P_{\mathit{right}}(r)$ ($P_{\mathit{left}}(l) > P_{\mathit{right}}(r)$).

Example 2. Consider again the TRS of Example 1. By applying P_{left} to the left-hand sides and P_{right} to the right-hand sides of the rewrite rules and the dependency pairs, the following ordering constraints are obtained:

$1\colon\ 0 \geqslant 0$	$3\colon\ x+1 \geqslant x+1$	$5\colon\ x+1 \geqslant x+1$	$7\colon\ x+2 > x+1$
$2\colon\ 0 \geqslant 0$	$4\colon\ \quad 0 \geqslant 0$	$6\colon\ x+2 > x$	$8\colon\ x+2 > x+1$

The only difference with the constraints in Example 1 is the interpretation of the term $\mathsf{s}(\mathsf{half}(x))$ on the right-hand side of rule 3. We have $P_{right}(\mathsf{half}(x)) = n(x-1) = x$ and thus $P_{right}(\mathsf{s}(\mathsf{half}(x))) = x+1$. Although $x+1$ is less precise than $\max\{0, x-1\}+1$, it is accurate enough to solve the ordering constraint resulting from rule 3.

So once the interpretations $f_{\mathbb{Z}}$ are determined, we transform a rule $l \to r$ into the polynomial $P_{left}(l) - P_{right}(r)$. Standard techniques can then be used to test whether this polynomial is positive (or non-negative) for all values in $\mathbb{N}$ for the variables. The remaining question is how to find suitable interpretations for the function symbols.

4 Finding Appropriate Interpretations

The idea is to take the *natural* interpretation for certain function symbols that appear in many example TRSs: $\mathsf{0}_{\mathbb{Z}} = 0$, $\mathsf{1}_{\mathbb{Z}} = 1$, $\mathsf{2}_{\mathbb{Z}} = 2$, $\ldots$, $\mathsf{s}_{\mathbb{Z}}(x) = x+1$, $\mathsf{p}_{\mathbb{Z}}(x) = x-1$, $x +_{\mathbb{Z}} y = x+y$, and $x \times_{\mathbb{Z}} y = xy$. For other function symbols f we take linear interpretations: $f_{\mathbb{Z}}(x_1, \ldots, x_n) = a_1x_1 + \cdots + a_nx_n + b$ with $a_1, \ldots, a_n \in \{0, 1\}$ and $b \in \{-1, 0, 1\}$. By disallowing negative coefficients for $a_1, \ldots, a_n$ weak monotonicity is obtained for free. Determining appropriate coefficients can be done by a straightforward but inefficient "generate and test" algorithm. We are currently implementing a more involved algorithm in the Tyrolean Termination Tool [8]. The initial experiments are very promising. If we use the most basic version of the dependency pair method, termination of the example in the introduction is proved in 0.01 seconds. With the recursive SCC algorithm of [7] for the dependency graph but without any other refinements, 50 of the 66 TRSs in Section 3 of [2] are handled (with an average time of just 0.08 seconds).[1]

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, 2001.
3. C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 346–364, 2000.
4. E. Contejean, C. Marché, B. Monate, and X. Urbain. C*i*ME version 2, 2000. Available at `http://cime.lri.fr/`.
5. N. Dershowitz and C. Hoot. Natural termination. *Theoretical Computer Science*, 142(2):179–207, 1995.
6. 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*, Lecture Notes in Computer Science, 2004. To appear.
7. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. In *Proceedings of the 19th International Conference on Automated Deduction*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 32–46, 2003.
8. N. Hirokawa and A. Middeldorp. Tyrolean termination tool, 2004. System description submitted to WST'04.
9. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.

[1] On a PC equipped with a 2.20 GHz Mobile Intel Pentium 4 Processor - M and 512 MB of memory.

Erasure and Termination in Higher-Order Rewriting

Jeroen Ketema and Femke van Raamsdonk

Faculty of Sciences, Vrije Universiteit, Amsterdam, The Netherlands
jketema@cs.vu.nl
femke@cs.vu.nl

Abstract. Two applications of the Erasure Lemma for first-order orthogonal term rewriting systems are: weak innermost termination implies termination, and weak normalization implies strong normalization for non-erasing systems. We discuss these two results in the setting of higher-order rewriting.

1 Introduction

The Erasure Lemma for orthogonal term rewriting systems (TRSs) states that if we lose the possibility of performing an infinite reduction by reducing s to t, then a subterm that admits an infinite reduction is erased. Two well-known results can be seen as applications of the Erasure Lemma. The first, due to O'Donnell [8], states that weak innermost normalization implies termination for orthogonal TRSs. We give an example showing that O'Donnell's result cannot directly be generalized to the higher-order case, and we propose a restricted version for the higher-order setting. The second result originates from Church, and states that weak and strong normalization coincide for orthogonal and non-erasing TRSs. Klop extends this to second-order rewriting. We discuss the definitions of non-erasing and propose one that permits to extend Church's result to (general) higher-order rewriting.

2 Notation

We work with higher-order rewriting systems (HRSs) as defined by Nipkow [7]; we also refer to combinatory reduction systems (CRSs) defined by Klop [4].

A term s is *terminating* or *strongly normalizing*, notation $\mathsf{SN}(s)$, if all rewrite sequences starting in s are finite. It is *weakly normalizing*, notation $\mathsf{WN}(s)$, if there is a (finite) sequence $s \to^* n$ from s to a normal form n. We use the notation $\infty(s)$ to indicate that the term s admits an infinite rewrite sequence.

A rewrite step $s \to t$ is *critical* if $\infty(s)$ but not $\infty(t)$. A rewrite step $s \to t$ that is not critical is *perpetual*; in that case we have $\infty(s) \Rightarrow \infty(t)$. A rewrite step $s \to t$ is *innermost*, denoted $s \to_i t$, if no proper subterm of the contracted redex is itself a redex. A term s is *weakly innermost normalizing*, notation $\mathsf{WIN}(s)$, if there is an innermost reduction $s \to_i^* n$ to a normal form n. It is *strongly innermost normalizing*, notation $\mathsf{SIN}(s)$, if all innermost reductions starting in s are finite.

A term s has the *conservation property* if every step from s is perpetual. A term s is *uniformly normalizing* if $\mathsf{WN}(s) \Rightarrow \mathsf{SN}(s)$.

A set of terms satisfies a property if all terms in the set satisfy that property. We then also use the notations as above without the term argument. For instance, we write that a TRS is SN. As remarked in [6], uniform normalization implies the conservation property, but the reverse implication only holds for sets of terms that are closed under reduction. For instance, in the ARS $\{a \to a, a \to b, b \to b, b \to c\}$ the set $\{a\}$ has the conservation property because every step from a is perpetual, but $\{a\}$ is not uniformly normalizing.

3 O'Donnell

The Erasure Lemma (Proposition 4.8.4 and Lemma 9.3.27 in [10]) for first-order orthogonal TRSs states that in a critical step a sub-term u with $\infty(u)$ is erased. O'Donnell [8] shows that for orthogonal TRSs we have $\mathsf{WIN} \Rightarrow \mathsf{SN}$. This can be shown using the Erasure Lemma. The following example shows that the result by O'Donnell cannot directly be generalized to the higher-order case.

Example 1. Consider the HRS defined by the following rewrite rules:

$$\begin{array}{ll} f(X) & \to a \\ g(\lambda x.F(x)) & \to F(g(\lambda x.f(x))) \end{array}$$

This is a second-order orthogonal fully extended HRS. It is not SN because we have the following infinite reduction:

$$g(\lambda x.\, f(x)) \to f(g(\lambda x.\, f(x))) \to f(f(g(\lambda x.\, f(x)))) \to \dots$$

However, the term $g(\lambda x.\, f(x))$ has an innermost reduction to normal form:

$$g(\lambda x.\, f(x)) \to_i g(\lambda x.\, a) \to_i a$$

The HRS is WIN; it is even SIN. This can be shown by giving a measure on terms that strictly decreases with each innermost reduction step. Hence the implications $\mathsf{WIN} \Rightarrow \mathsf{SN}$ and $\mathsf{SIN} \Rightarrow \mathsf{SN}$ do not hold for second-order orthogonal fully extended HRSs.

One might wonder where the proof of $\mathsf{WIN} \Rightarrow \mathsf{SN}$ breaks down in the higher-order case. First we consider the proof using the Erasure Lemma. The problem here is that the Erasure Lemma does not hold for higher-order rewriting. Indeed, the Erasure Lemma for second-order rewriting, given in [3], states the following: In a critical step a subterm u is erased, that can descend, by reduction steps that do not overlap with u, to a subterm u' with $\infty(u')$. In the example, the critical step $g(\lambda x.\, f(x)) \to g(\lambda x.\, a)$ erases the subterm x which itself does not admit an infinite reduction. However, its descendant $g(\lambda x.\, f(x))$ does; note that it is obtained by contracting the g-redex that does not overlap with x.

An alternative proof of $\mathsf{WIN} \Rightarrow \mathsf{SN}$ is given in [9], by showing the sequence of implications $\mathsf{WIN} \Rightarrow \mathsf{SIN} \Rightarrow \mathsf{SN}$ for non-overlapping TRSs. The generalization of O'Donnell's result to the case of non-overlapping instead of orthogonal TRSs is due to Gramlich [1,2]. The proof given in [9] uses besides [1,2] also [5]. The implication $\mathsf{WIN} \Rightarrow \mathsf{SIN}$ holds for non-overlapping HRSs; the proof directly carries over from the first-order case. However, Example 1 shows that the implication $\mathsf{SIN} \Rightarrow \mathsf{SN}$ does not hold for second-order orthogonal fully-extended HRSs. The proof of $\mathsf{SIN} \Rightarrow \mathsf{SN}$ for the first-order case uses the following: if in a reduction step $s \to t$ a redex is contracted that is not convergent (i.e. both confluent and SN), then $\phi(s) \to^+ \phi(t)$. Here ϕ computes the result of reducing all maximal convergent subterms to their (unique) normal form. This is Lemma 5.6.6 in [9]. Example 1 shows that this statement is not true for orthogonal second-order fully extended HRSs: In the step $g(\lambda x.\, f(x)) \to f(g(\lambda x.\, f(x)))$ the non-terminating g-redex is contracted. If we compute in both terms the normal forms of the maximal convergent subterms, then we find $g(\lambda x.\, a)$ and $f(g(\lambda x.\, a))$. There is no reduction sequence consisting of one or more steps from $g(\lambda x.\, a)$ to $f(g(\lambda x.\, a))$.

Is it possible to impose additional restrictions such that the implication $\mathsf{WIN} \Rightarrow \mathsf{SN}$ holds for the higher-order case? One possibility is to restrict attention to *bounded* orthogonal fully extended HRSs. Using the generalization of [10, Lemma 9.2.28] to the higher-order case, which states that for orthogonal fully extended HRSs, WN implies acyclicity, we conclude SN from the fact that the length of the reducts of all terms are bounded. Another possibility is to restrict attention to non-erasing rules which actually brings us to the second application of the Erasure Lemma discussed in the next section.

To conclude, note that for orthogonal TRSs the (stronger) local version of O'Donnell's result, the statement $\forall t.\, [\mathsf{WIN}(t) \Rightarrow \mathsf{SN}(t)]$, also follows from the Erasure Lemma. Klop [4, Remark 5.9.8.1(ii)] remarks that the local version of O'Donnell's result does not hold for orthogonal CRSs. For instance, the λ-term $(\lambda x.(\lambda y.z)(xx))(\lambda u.uu)$ is SIN but not SN. (This is a solution to [10, Exercise 4.8.13].)

4 Church

Church proved that weak and strong normalization coincide in the λI-calculus, where in an abstraction $\lambda x.\, M$ there is at least one free occurrence of x in M. The crucial property that makes the λI-calculus uniformly normalizing is the fact that it is *non-erasing*. That is, intuitively, a reduction step cannot erase a subterm.

A first-order rewrite rule $l \to r$ is said to be non-erasing if all variables in l also occur in r. A rewrite step is non-erasing if the applied rule is non-erasing. Orthogonal and non-erasing TRSs are uniformly normalizing (see [10, Theorem 4.8.5] and [9, Theorem 5.6.10(2)]).

For second-order rewriting, this requirement on the rewrite rule does not guarantee that the induced step is non-erasing. Clearly the step $(\lambda x.\, y)\, z \to y$ is erasing although in the β-reduction rule all variables in the left-hand side also occur in the right-hand side. Klop [4] therefore defines a CRS (which is a second-order rewriting system) to be non-erasing if for every rewrite step $s \to t$ the terms s and t have the same free variables. He then proves uniform normalization for orthogonal and non-erasing CRSs. This generalizes the result by Church because the λI-calculus is an example of an orthogonal and non-erasing CRS.

For third-order rewriting, this second-order notion of non-erasing is not sufficient for proving uniform normalization, as shown in the following example.

Example 2. Consider the HRS defined by the following rules:

$$\begin{aligned} f(\lambda x.\, F(x)) &\to F(f(\lambda x.\, F(x))) \\ g(\lambda y.\, G(y)) &\to G(\lambda u.\, a) \end{aligned}$$

This is a third-order orthogonal fully extended HRS. (In fact for the left-hand side of the g-rule we should write $g(\lambda y.\, G(\lambda z.\, y(z)))$.) Consider the term $g(\lambda y.\, f(\lambda x.\, y(x)))$. It is a redex with respect to the g-rule using the substitution $\{G \mapsto \lambda u.\, f(\lambda x.\, u(x))\}$. It contains a redex with respect to the f-rule using the substitution $\{F \mapsto \lambda u.y(u)\}$. If first the g-redex is contracted, we obtain a reduction to normal form:

$$g(\lambda y.\, f(\lambda x.\, y(x))) \to f(\lambda x.\, a) \to a$$

Note that both steps are non-erasing in the sense of Klop because all free variables (none) are preserved. But repeatedly contracting the f-redexes yields an infinite rewrite sequence:

$$g(\lambda y.\, f(\lambda x.\, y(x))) \to g(\lambda y.\, y(f(\lambda x.\, y(x)))) \to \ldots$$

Hence in a third-order orthogonal fully extended HRS a non-erasing step is not necessarily perpetual. This is already remarked in [3] where another example is given.

The question is how to adapt the definition of non-erasure such that it is useful for proving uniform normalization. Note that a rewrite step in a HRS is obtained in two stages:

$$l\sigma\downarrow_\beta \longrightarrow_R r\sigma \longrightarrow^*_\beta r\sigma\downarrow_\beta$$

Intuitively, a rewrite step is non-erasing if both stages are non-erasing.

In a first-order HRS, which is in fact a TRS, the second stage, the β-reduction to normal form, does not play a role. In that case, indeed the requirement that the applied rewrite rule is non-erasing guarantees that the rewrite step is non-erasing.

In a second-order HRS, the second stage may play a role. There is no erasure in the β-reduction to normal form if this is a λI-reduction. Observe that all abstractions in the second stage originate from the substitution σ. This motivates the following alternative to the second-order definition of non-erasure due to Klop: a rewrite step in a second-order HRS is non-erasing if all variables in the left-hand side of the applied rule also occur in the right-hand side, and in addition the substitution used to instantiate the rewrite rule is non-erasing in the sense that the terms assigned to free variables are λI-terms. This is a more restrictive definition than the one by Klop: the step in the example $f(\lambda x.\, a) \to a$ does not remove free variables, but it uses the erasing substitution $\{F \mapsto \lambda x.a\}$.

In a third-order HRS, the abstractions that occur in the second stage may originate from the substitution σ, but also from the arguments of the free variables in the rewrite rule. In the example, the free variable G in the g-rule has as argument the abstraction $\lambda u.\, a$. Note that this is not a λI-term. In a second-order HRS, free variables cannot have abstractions as argument. The second stage is a λI-reduction if both the abstractions originating from σ and the abstractions originating from the rewrite rule are λI-terms.

To summarize, a rewrite step in a HRS is non-erasing if first, all variables in the left-hand side of the applied rewrite rule also occur in the right-hand side, and second, the applied substitution assigns λI-terms to free variables, and third, the arguments of the free variables in the applied rewrite rule are λI-terms. We claim that for orthogonal HRSs weak and strong normalization coincide if all steps are non-erasing, and that non-erasing steps are perpetual in orthogonal fully extended HRSs.

References

1. B. Gramlich. Relating innermost, weak, uniform and modular termination of term rewriting systems. In Andrei Voronkov, editor, *Proceedings of the Third International Conference on Logic Programming and Automated Reasoning (LPAR'92)*, volume 624 of *Lecture Notes in Artificial Intelligence*, pages 285–296. Springer-Verlag, July 1992.
2. B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.
3. Z. Khasidashvili, M. Ogawa, and V. van Oostrom. Perpetuality and uniform normalization in orthogonal rewrite systems. *Information and Computation*, 164:118–151, 2001.
4. J.W. Klop. *Combinatory Reduction Systems.* PhD thesis, Rijksuniversiteit Utrecht, 1980.
5. A. Middeldorp. A simple proof to a result of Bernhard Gramlich. Distributed at the 5th Term Rewriting Meeting in Tsukuba, February 1994.
6. P. Møller Neergaard and M.H. Sørensen. Conservation and uniform normalization in lambda calculi with erasing reductions. *Information and Computation*, 178(1):149–179, 2002.
7. T. Nipkow. Higher-order critical pairs. In *Proceedings of the 6th annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 342–349, Amsterdam, The Netherlands, July 1991.
8. M.J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *Lecture Notes in Computer Science*. Springer-Verlag, 1977.
9. E. Ohlebusch. *Advanced Topics in Term Rewriting.* Springer-Verlag, 2002.
10. Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 2003.

Proof-Theoretic Analysis of Lexicographic Path Orders

Jan Willem Klop[1], Vincent van Oostrom[2], and Roel de Vrijer[1]

[1] Department of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands
jwk@cs.vu.nl, rdv@cs.vu.nl
[2] Department of Philosophy, Universiteit Utrecht, The Netherlands
oostrom@phil.uu.nl

Abstract. We relate Kamin and Lévy's original presentation of *lexicographic path orders* (LPO), using an inductive definition, to a presentation, which we will refer to as *iterative lexicographic path orders* (ILPO), based on Bergstra and Klop's definition of recursive path orders by way of an auxiliary term rewriting system.

1 Introduction

In his seminal paper [4], Dershowitz introduced the recursive-path-order method to prove termination of a first-order term rewrite system (TRS) $\mathcal{T}$. The method is based on extending a well-quasi-order (WQO) $\preceq$ on the signature of a TRS to a WQO $\preceq_{rpo}$ on the set of (open) terms over the signature. Termination of the TRS follows if $l \succ_{rpo} r$ holds for every rule $l \to r$ of $\mathcal{T}$. Several variants of this technique have been studied. Here we restrict our attention to the so-called lexicographic path order (LPO), due to Kamin and Lévy [6].

In Bergstra and Klop [2] an alternative definition of RPO is presented, which we call the *iterative* path order (IPO), the name stressing the way it is generated. It is operational in the sense that it is itself defined by means of an (auxiliary) term rewriting system, the rules of which depend only on the given well-quasi order $\preceq$. The iterative approach can easily be adapted to the case of LPO, yielding the *iterative, lexicographic* path order ($\preceq_{ilpo}$).

The auxiliary rewrite rules for ILPO as given here Definition 1 were essentially formulated for the first time in the PhD-thesis of Geser [5] and, in a somewhat restricted form, in Klop [7]. It should be noted though that no proofs are given in [5] and [7]. What also has been lacking until now is an understanding of the exact relationship between a path order and its iterative variant. That is the main subject of our investigation, here for the case of LPO versus ILPO.

2 The iterative lexicographic path order

A path order is a way to extend a terminating relation on a signature to a reduction order on the terms over the signature. Here by a reduction order we mean a transitive and terminating relation on terms which is closed under substitution and contexts. ILPO is a particular path order.

As a running example to illustrate ILPO, we take the terminating relation R given by $\mathtt{M}\ R\ \mathtt{A}$ and $\mathtt{A}\ R\ \mathtt{S}$ on the signature of the TRS $\mathcal{M}\mathsf{ul}$ of addition and multiplication on natural numbers with rewrite rules: $\{\mathtt{A}(x,\mathtt{0}) \to x,\ \mathtt{A}(x,\mathtt{S}(y)) \to \mathtt{S}(\mathtt{A}(x,y)),\ \mathtt{M}(x,\mathtt{0}) \to \mathtt{0},\ \mathtt{M}(x,\mathtt{S}(y)) \to \mathtt{A}(x,\mathtt{M}(x,y))\}$. Clearly, the relation R is terminating and ILPO will extend it to a reduction order R_{ilpo}, in such a way that termination of the TRS $\mathcal{M}\mathsf{ul}$ can be concluded.

The definition of R_{ilpo} proceeds in two steps. First, a term rewriting system $\mathcal{L}\mathsf{ex}$ (depending on R) over the signature extended with *control* symbols, is defined. The relation

R_{ilpo} is then obtained by restricting the transitive closure of $\rightarrow_{\mathcal{L}\mathsf{ex}}$ to terms over the original signature.

Definition 1. Let R be a relation on a signature Σ, and let V be a set of variables, disjoint from Σ. The TRS $\mathcal{L}\mathsf{ex}_R$ is $\langle \Sigma \uplus \Sigma^* \uplus V, \mathcal{R}\rangle$:

1. The signature Σ^* of *control* symbols is a copy of Σ, i.e. for each function symbol $f \in \Sigma$, Σ^* contains a fresh symbol f^* having the same arity f has.
2. The rules $\mathcal{R}$ are given in the table, for arbitrary function symbols f, g in Σ, with $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$ disjoint vectors of pairwise disjoint variables of appropriate lengths.

$$\boxed{\begin{array}{rll} f(\boldsymbol{x}) & \rightarrow_{\text{put}} & f^*(\boldsymbol{x}) \\ f^*(\boldsymbol{x}) & \rightarrow_{\text{select}} & x_i \quad (1 \leq i \leq |\boldsymbol{x}|) \\ f^*(\boldsymbol{x}) & \rightarrow_{\text{copy}} & g(f^*(\boldsymbol{x}), \ldots, f^*(\boldsymbol{x})) \quad (f\ R\ g) \\ f^*(\boldsymbol{x}, g(\boldsymbol{y}), \boldsymbol{z}) & \rightarrow_{\text{lex}} & f(\boldsymbol{x}, g^*(\boldsymbol{y}), l, \ldots, l) \quad (l = f^*(\boldsymbol{x}, g(\boldsymbol{y}), \boldsymbol{z})) \end{array}}$$

The idea of $\mathcal{L}\mathsf{ex}$ is that marking the head symbol of a term, by means of the put-rule, corresponds to the obligation to make that term smaller. The other rules correspond to *atomic* ways in which this can be brought about. For our running example $\mathcal{M}\mathsf{ul}$, the reduction $\mathtt{A}(x, \mathtt{0}) \rightarrow_{\text{put}} \mathtt{A}^*(x, \mathtt{0}) \rightarrow_{\text{select}} x$ in $\mathcal{L}\mathsf{ex}$, is a decomposition of the first rule into atomic $\mathcal{L}\mathsf{ex}$-steps. This also holds for the other rules of $\mathcal{M}\mathsf{ul}$. E.g. in case of the fourth rule:

$$\begin{tikzcd}
\mathtt{M}(x, \mathtt{S}(y)) \arrow[r, "\text{put}"] \arrow[ddd, "\mathcal{L}\mathsf{ex}"', "+"] & \mathtt{M}^*(x, \mathtt{S}(y)) \arrow[d, "\text{copy}"'] \\
& \mathtt{A}(\mathtt{M}^*(x, \mathtt{S}(y)), \mathtt{M}^*(x, \mathtt{S}(y))) \arrow[d, "\text{select}"'] \\
& \mathtt{A}(x, \mathtt{M}^*(x, \mathtt{S}(y))) \arrow[d, "\text{lex}"'] \\
\mathtt{A}(x, \mathtt{M}(x, y)) & \mathtt{A}(x, \mathtt{M}(x, \mathtt{S}^*(y))) \arrow[l, "\text{select}"]
\end{tikzcd}$$

Definition 2. The *iterative lexicographic path order* R_{ilpo} of a relation R on a signature Σ is the restriction of $\rightarrow^+_{\mathcal{L}\mathsf{ex}_R}$ to $T(\Sigma \uplus V)$. A TRS is *ILPO*-terminating if its rules are contained in R_{ilpo}, for some terminating relation R.

Since $l \rightarrow^+_{\mathcal{L}\mathsf{ex}} r$ for each rule $l \rightarrow r$ of the TRS $\mathcal{M}\mathsf{ul}$, it is ILPO-terminating. Note that transitivity and closure under contexts and substitutions of R_{ilpo} are 'built in' into our definition. In order to show R_{ilpo} is a reduction order, it remains to show it is terminating.

3 ILPO-termination implies termination

We show that if a relation R is terminating, then R_{ilpo} is terminating as well. Since the rewrite rules of R_{ilpo} are given by the restriction of $\rightarrow^+_{\mathcal{L}\mathsf{ex}}$ to $T(\Sigma \uplus V)$, termination of the TRS $\mathcal{L}\mathsf{ex}$ would be sufficient for termination of R_{ilpo}. However, $\mathcal{L}\mathsf{ex}$ is in general *not* terminating, e.g. in case of the running example we have, despite R being terminating:

$$\mathtt{A}(x, y) \rightarrow_{\text{put}} \mathtt{A}^*(x, y) \rightarrow_{\text{copy}} \mathtt{S}(\mathtt{A}^*(x, y)) \rightarrow_{\text{copy}} \mathtt{S}(\mathtt{S}(\mathtt{A}^*(x, y))) \rightarrow_{\text{copy}} \cdots$$

Non-termination is 'caused' by the left-hand side of the copy-rule being a subterm of its right-hand side. Observe that the marked symbol $\mathtt{A}^*$ is 'used' infinitely often. We will show that this is necessary in *any* infinite reduction; more precisely, that fixing how often a marked symbol can be used in advance in the copy- and lex-rules yields a terminating TRS $\mathcal{L}\mathsf{ex}^{\omega}$. Since in any given atomic decomposition $l \to^{+}_{\mathcal{L}\mathsf{ex}} r$ of a rule $l \to r$, any marked symbol is only used finitely often (certainly not more often than the length of the decomposition), the relations $\to^{+}_{\mathcal{L}\mathsf{ex}}$ and $\to^{+}_{\mathcal{L}\mathsf{ex}^{\omega}}$ coincide on the unmarked terms. We will exploit this fact, by showing that ILPO-termination implies termination *via* termination of $\mathcal{L}\mathsf{ex}^{\omega}$.

Definition 3. Let R be a relation on a signature Σ, and let V be a signature of nullary symbols disjoint from Σ. The TRS $\mathcal{L}\mathsf{ex}^{\omega}_{R}$ is $\langle \Sigma \uplus \Sigma^{\omega} \uplus V, \mathcal{R}^{\omega}_{R}\rangle$:

1. The signature Σ^{ω} of ω-control symbols consists of ω copies of Σ, i.e. for each symbol $f \in \Sigma$ and natural number n, Σ^{ω} contains a fresh symbol f^{n} having the arity f has.
2. The rules $\mathcal{R}^{\omega}_{R}$ are given in the table, for arbitrary symbols f, g in Σ and natural number n, with $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$ disjoint vectors of pairwise disjoint variables of appropriate lengths.

$$\begin{array}{rll} f(\boldsymbol{x}) & \to_{\text{put}} & f^{n}(\boldsymbol{x}) \\ f^{n}(\boldsymbol{x}) & \to_{\text{select}} & x_i \quad (1 \le i \le |\boldsymbol{x}|) \\ f^{n+1}(\boldsymbol{x}) & \to_{\text{copy}} & g(f^{n}(\boldsymbol{x}),\ldots,f^{n}(\boldsymbol{x})) \quad (f\ R\ g) \\ f^{n+1}(\boldsymbol{x}, g(\boldsymbol{y}), \boldsymbol{z}) & \to_{\text{lex}} & f(\boldsymbol{x}, g^{n}(\boldsymbol{y}), l, \ldots, l) \quad (l = f^{n}(\boldsymbol{x}, g(\boldsymbol{y}), \boldsymbol{z})) \end{array}$$

The TRS $\mathcal{L}\mathsf{ex}_{R}$ (Definition 1) is seen to be a homomorphic image of $\mathcal{L}\mathsf{ex}^{\omega}_{R}$, by mapping f^{n} to f^{*}, for any natural number n. Vice versa, reductions in $\mathcal{L}\mathsf{ex}_{R}$ can be 'lifted' to $\mathcal{L}\mathsf{ex}^{\omega}_{R}$.

Lemma 1. *1.* $\to^{+}_{\mathcal{L}\mathsf{ex}_{R}}$ *and* $\to^{+}_{\mathcal{L}\mathsf{ex}^{\omega}_{R}}$ *coincide as relations restricted to* $T(\Sigma \uplus V)$.
2. If R *is terminating, then* $\mathcal{L}\mathsf{ex}^{\omega}_{R}$ *is terminating.*

The latter item is proven by employing the proof technique of [3]. It follows that R_{ilpo} is terminating, if R is, hence ILPO-termination implies termination. So $\mathcal{M}\mathsf{ul}$ is terminating.

4 Equivalence of ILPO with the recursive lexicographic path order

We show that ILPO is at least as powerful as the recursively defined lexicographic path order found in the literature, and is equivalent to it for transitive relations. In the following we refer verbatim to the recursive definition of $>_{lpo}$ as it is given in [1, Definition 5.4.12], starting from a strict order $>$ on Σ. It is easy to see that this is still a correct recursive definition for $>$ being an arbitrary relation R, yielding R_{lpo}. We call a TRS $\mathcal{T} = \langle \Sigma, \mathcal{R}\rangle$ *LPO*-terminating for a terminating relation R, if $\mathcal{R} \subseteq R_{lpo}$.

Theorem 1. $R_{lpo} \subseteq R_{ilpo}$*, for any relation* R.

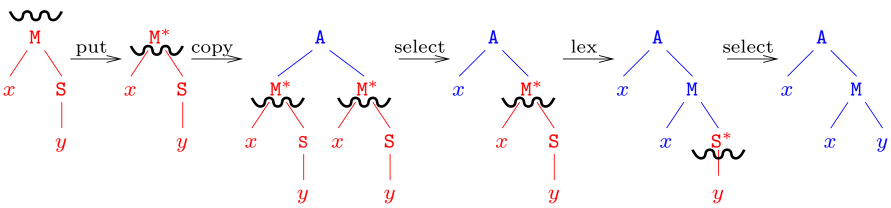
Proof. We show by induction on the definition of $s\ R_{lpo}\ t$ that $s^{*} \twoheadrightarrow_{\mathcal{L}\mathsf{ex}_{R}} t$, where $(f(\boldsymbol{s}))^{*} = f^{*}(\boldsymbol{s})$. This suffices, since $s \to_{\text{put}} s^{*}$ and s, t are not marked.

(LPO1) If $t \in \mathcal{V}ar(s)$ and $s \neq t$, then the result follows by repeatedly selecting the subterm on a path to an occurrence of t in s.
(LPO2) Otherwise, let $s = f(s_1, \ldots, s_m)$, $t = g(t_1, \ldots, t_n)$.
(LPO2a) Suppose there exists i, $1 \le i \le m$, with either $s_i = t$ or $s_i\ R_{lpo}\ t$. In the former case, the result follows by a single application of the select-rule for index i. In the latter case, this step is followed by an application of the put-rule after which the result follows by the IH.

(LPO2b) Suppose $f\ R\ g$ and $s\ R_{lpo}\ t_j$ for all j, $1 \leq j \leq n$. Then the result follows by a single application of the copy-rule and n applications of the IH.

(LPO2c) Suppose $f = g$, $s\ R_{lpo}\ t_j$ for j, $1 \leq j \leq n$, and there exists i, $1 \leq i \leq m$, such that $s_1 = t_1, \ldots, s_{i-1} = t_{i-1}$ and $s_i\ R_{lpo}\ t_i$. Then the result follows by a single application of the lex-rule, selecting the ith argument, and the IH for $s_i\ R_{lpo}\ t_i$ and $s\ R_{lpo}\ t_j$ for j, $i < j \leq n$.

We call the $\mathcal{L}$ex-strategy defined by this proof the *wave* strategy. The idea is that the marked positions in a term represent the wave front, which moves downwards in the term tree, i.e. from the root in the direction of the subterms. This is visualised for the $\mathcal{L}$ex-reduction from $\mathtt{M}(x, \mathtt{S}(y))$ to $\mathtt{A}(x, \mathtt{M}(x, y))$ above as:



We prove a converse to Lemma 1 by a detailed proof-theoretic analysis, showing that any $\mathcal{L}$ex-reduction can be transformed, into a wave reduction.

Theorem 2. $R_{ilpo} = (R_{lpo})^+$.

As a corollary we have that ILPO is equivalent with LPO for any strict order, and that R_{ilpo} is decidable, in case R is a terminating relation for which reachability is decidable: simply 'try all waves up to the size of the right-hand side'.

Note that $(R^+)_{ilpo}$ may differ from R_{ilpo}. Consider the signature consisting of nullary symbols a, b, and unary symbols f, g, with precedence relation $f\ R\ b\ R\ g$. Then the one-rule TRS $f(a) \to g(a)$ is not ILPO-terminating, but it is ILPO-terminating for R^+.

5 Conclusion

We have shown that our iterative set-up of ILPO can serve as an independent alternative to the classic, recursive, treatment of LPO. It was obtained by a proof-theoretic analysis of the usual inductive definition of LPO, giving prominence to the *steps* hidden in the inductive clauses. From this perspective it is only natural that we have taken an arbitrary terminating relation (instead of order) on the signature as our starting point, so one could speak, in the spirit of Persson's presentation of recursive path relations [8], of iterative lexicographic path *relations*. Finally, we note that the correspondence between recursive and iterative ways of specifying path orders, also exists for variants of LPO like the embedding relation and the recursive path order.

References

1. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
2. J.A. Bergstra and J.W. Klop. Algebra of communicating processes. *TCS*, 37(1):171–199, 1985.
3. W. Buchholz. Proof-theoretic analysis of termination proofs. *APAL*, 75(1-2):57–65, 1995.
4. N. Dershowitz. Orderings for term rewriting systems. *TCS*, 17(3):279–301, 1982.
5. Alfons Geser. *Relative termination*. Dissertation, Fakultät für Mathematik und Informatik, Universität Passau, Germany, 1990. 105 pages. Also available as: Report 91-03, Ulmer Informatik-Berichte, Universität Ulm, 1991.

6. S. Kamin and J.-J. Lévy. Two generalizations of the recursive path ordering. University of Illinois, 1980.
7. J.W. Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 1–116. Oxford University Press, 1992.
8. H. Persson. *Type Theory and the Integrated Logic of Programs*. PhD thesis, Chalmers, 1999.

Polynomials over the Reals in Proofs of Termination*

Salvador Lucas

DSIC, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
slucas@dsic.upv.es

Abstract. Polynomials over the real numbers were proposed as an alternative to polynomials over the naturals in termination proofs. We have recently shown how to use an arbitrary polynomial interpretation over the reals to generate well-founded and stable term orderings. Monotonicity can, then, be gradually introduced in the interpretations to deal with different applications. The first one is the generation of reduction orderings. We can also take advantage of non-fully monotonic polynomial interpretations in some remarkable cases. The dependency pairs method for proving termination of rewriting is an interesting one.

1 Introduction

Polynomials over the real numbers were proposed by Dershowitz [5] as an alternative to Lankford's polynomials over the naturals [6]. In contrast to Lankford's, well-foundedness has to be explicitly ensured by further requiring the subterm property; on the other hand, comparisons of terms by using the orderings induced by such polynomials are decidable.The automatic generation of such polynomials, however, has been hardly explored to date.

In [8], we have described how to associate a well-founded and stable ordering to an arbitrary polynomial interpretation over the reals, i.e., a collection $\{[f] \mid f \in \mathcal{F}\}$ of polynomials, where $[f]$ is a polynomial in $ar(f)$ variables whose coefficients are real numbers: $[f] \in \mathbb{R}[x_1, \ldots, x_{ar(f)}]$. For the purpose of this paper, we assume that $[f](x_1, \ldots, x_{ar(f)}) \geq 0$ for all $x_1, \ldots, x_{ar(f)}$ and $f \in \mathcal{F}$. Given a positive real number $\delta \in \mathbb{R}_{>0}$, a well-founded and stable (strict) ordering $>_\delta$ on terms is defined as follows: for all $t, s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $t >_\delta s$ if and only if $[t] - [s] \geq_{\mathbb{R}} \delta$, where $[t]$ is the polynomial which is inductively obtained by interpreting each symbol f in t as $[f]$ and each variable $x \in \mathcal{V}ar(t)$ as a variable x ranging in $\mathbb{R}$. Monotonicity is ensured for each argument $i \in \{1, \ldots, ar(f)\}$ of each symbol $f \in \mathcal{F}$ if $\frac{\partial [f](x_1, \ldots, x_i, \ldots, x_{ar(f)})}{\partial x_i} \geq 1$.

Polynomial interpretations are well-suited to mechanize the proofs of termination. A proof of termination of a TRS is transformed into the problem of solving a set of constraints over the coefficients of a polynomial interpretation for the symbols of the TRS [6]. For practical reasons, we consider polynomials using nonnegative, *rational* coefficients. Thus, $[f] \in \mathbb{Q}_{\geq 0}[x_1, \ldots, x_k]$ for each k-ary symbol $f \in \mathcal{F}$. The tool MU-TERM[1] implements the previous approach to automatically generate μ-reduction orderings based on polynomial interpretations over the rationals. We use the C*i*ME system [4] to solve the set of constraints that we obtain. C*i*ME solves Diophantine inequations and yields non-negative integers as solutions. The use of rational numbers is easily made compatible with this limitation: given a Diophantine constraint $e_1 \geq e_2$ containing an occurrence of $\frac{p}{q}$ in e_1 (or e_2), we obtain an equivalent constraint $q \cdot e_1 \geq q \cdot e_2$. Then, we propagate the multiplication of q inside the members of e_1 and e_2 thus removing occurrences of q as a denominator. We repeat this process to remove all rational coefficients.

* Work partially supported by MCyT project TIC2001-2705-C03-01, MCyT Acción Integrada HU 2003-0003 and AVCyT grant GR03/025.

[1] See http://www.dsic.upv.es/~slucas/csr/termination/muterm.

The previous framework is well-suited for context-sensitive rewriting (*CSR* [7]). In *CSR*, a *replacement map* μ discriminates, for each symbol f of the signature, the argument positions $\mu(f)$ on which replacements are allowed. This can improve the termination behavior by pruning (all) infinite rewrite sequences[2]. Termination of *CSR* is fully captured by the so-called *μ-reduction orderings* [11], i.e., well-founded, stable orderings $>$ which are *μ-monotonic*, i.e., for all $f \in \mathcal{F}$ and $i \in \mu(f)$, $>$ is monotonic in the i-th argument of f. Term rewriting is a particular case of *CSR* where the replacement map $\mu_\top(f) = \{1, \ldots, ar(f)\}$, for all $f \in \mathcal{F}$ is used. Thus, polynomial $\mu_\top$-reduction orderings can also be used in proofs of termination. We will also see that more general μ-reduction orderings can also be useful in proofs of termination of rewriting.

2 Proofs of polynomial termination of TRSs

We do not know whether polynomial interpretations over the rationals (or reals) are actually more powerful than polynomial interpretations over the naturals. As far as the author knows, this is an open problem. In practice, however, they can be helpful:

Example 1. Consider the TRS $\mathcal{R}$:

```
a -> b          c -> d          b -> c
```

Most constraint solvers use some finite domain to give value to the unknowns. For instance, a system using a domain[3] $\{0, 1, 2\}$ would be unable to prove the termination of $\mathcal{R}$ by using polynomials over the naturals. However, the polynomial interpretation

```
[a] = 2        [b] = 1        [c] = 1/2        [d] = 0
```

(computed by MU-TERM for $\delta = \frac{1}{10}$) proves termination of $\mathcal{R}$.

Although this example looks somehow artificial, these chains of symbols naturally arise in some applications. For instance, proofs of termination of a Conditional TRS (CTRS) $\mathcal{R}$ are usually attempted by first transforming it into a TRS $\mathcal{U}(\mathcal{R})$ and then proving termination of $\mathcal{U}(\mathcal{R})$ (see [9] for an overview of these methods). In this setting, it is usual to introduce n new symbols $U_1, U_2, \ldots, U_n$ for each conditional rule $l \to r \Leftarrow s_1 = t_1, s_2 = t_2, \ldots, s_n = t_n$ which are related in the transformed system as in the previous example.

Example 2. Consider the following CTRS $\mathcal{R}$ [10, Example 3.4]:

```
f(x) -> g(y) <= x -> h(y), i(x) -> y        a -> b <= c -> d
i(x) -> a                                    c -> d
```

By using the transformation of [9, Definition 7.2.48], we get $\mathcal{U}(\mathcal{R})$:

```
f(X) -> u1(X,X)                 i(X) -> a     a -> u(c)
u1(h(Y),X) -> u2(i(X),X,Y)      c -> d        u(d) -> b
u2(Y,X,Y) -> g(Y)
```

which can be proved terminating by the following polynomial interpretation (computed by MU-TERM with $\delta = \frac{1}{10}$):

```
[f](X) = 3.X + 1                    [h](X) = X + 3      [a] = 1        [b] = 0
[u1](X1,X2) = X1 + 2.X2             [i](X) = X + 3/2    [g](X) = X     [d] = 0
[u2](X1,X2,X3) = X1 + X2 + X3 + 1   [u](X) = X + 1/2    [c] = 1/3
```

No polynomial interpretation over the naturals using coefficients below 4 can directly prove termination of $\mathcal{U}(\mathcal{R})$.

---

[2] See http://www.dsic.upv.es/~slucas/csr/termination/examples.

[3] This is the current default domain for C*i*ME and AProVE (see http://www-i2.informatik.rwth-aachen.de/AProVE).

3 Termination of TRSs using dependency pairs

A reduction pair $(\gtrsim, \sqsupset)$ consists of a reflexive, transitive, stable, and weakly monotonic relation $\gtrsim$ and a stable and well-founded ordering $\sqsupset$ satisfying either $\gtrsim \circ \sqsupset \subseteq \sqsupset$ or $\sqsupset \circ \gtrsim \subseteq \sqsupset$. Note that *monotonicity is not required* for $\sqsupset$. When using the dependency pairs method [1], we can prove termination by showing that the *lhs* and *rhs* of each rule of the TRS are comparable by using $\gtrsim$ whereas the components of each dependency pair are comparable by using $\sqsupset$ [2].

Example 3. Consider the TRS $\mathcal{R}$ which is part of the TPDB[4]:

```
f(f(X)) -> f(g(f(g(f(X)))))
f(g(f(X))) -> f(g(X))
```

Termination of $\mathcal{R}$ can be proved by finding a reduction pair $(\gtrsim, \sqsupset)$ such that:

`f(f(X))`	$\gtrsim$	`f(g(f(g(f(X)))))`	`F(f(X))`	$\sqsupset$	`F(g(f(g(f(X)))))`
`f(g(f(X)))`	$\gtrsim$	`f(g(X))`	`F(f(X))`	$\sqsupset$	`F(g(f(X)))`
			`F(f(X))`	$\sqsupset$	`F(X)`
			`F(g(f(X)))`	$\sqsupset$	`F(g(X))`

where `F` is introduced to define the dependency pairs (see [1]).

Given a polynomial interpretation over the reals, the relation $t \gtrsim s$ iff $[t] - [s] \geq 0$, is a quasi-ordering; this quasi-ordering is weakly monotonic in all arguments of all symbols provided that only nonnegative coefficients are used in the polynomials. Given a polynomial interpretation and $\delta > 0$, we have $\gtrsim \circ >_\delta \subseteq >_\delta$ (if there is u such that $[t] - [u] \geq 0$ and $[u] - [s] \geq \delta$, then $[t] - [u] + [u] - [s] = [t] - [s] \geq 0 + \delta = \delta$); thus $(\gtrsim, >_\delta)$ is a *reduction pair*. We have implemented the generation of such reduction pairs in MU-TERM: we just give a TRS $\mathcal{R}$ the *least* replacement map $\mu_\perp(f) = \emptyset$ for all $f \in \mathcal{F}$. Since $\mu_\perp$ expresses *no* monotonicity requirements for $>_\delta$, this is the most flexible choice we can do (although it is not the only one). MU-TERM (tries) to compute the interpretation which makes $\gtrsim$ and $>_\delta$ compatible with the rules and the dependency pairs as above.

Example 4. The following polynomial interpretation:

```
[f](X) = X + 1      [g](X) = 1/2.X      [nF_f](X) = X
```

(where `nF_f` is the MU-TERM representation of symbol `F` in Example 3) defines a reduction pair $(\gtrsim, >_\delta)$ (with $\delta = \frac{1}{10}$) which proves termination of $\mathcal{R}$ in Example 3.

In fact, Arts and Giesl already noticed that the polynomials used with dependency pairs *do not necessarily depend on all their arguments.*

Example 5. Consider the TRS $\mathcal{R}$ [1, Example 2]:

```
minus(X,0) -> X          minus(s(X),s(Y)) -> minus(X,Y)
quot(0,s(Y)) -> 0        quot(s(X),s(Y)) -> s(quot(minus(X,Y),s(Y)))
```

The following polynomial interpretation (computed by MU-TERM for $\delta = \frac{1}{10}$):

```
[minus](X1,X2) = X1      [s](X) = X + 1          [nF_minus](X1,X2) = X1
[0] = 0                  [quot](X1,X2) = X1      [nF_quot](X1,X2) = X1
```

defines a reduction pair $(\gtrsim, >_\delta)$ which proves termination of $\mathcal{R}$ in Example 3. The interpretation computed by MU-TERM exactly corresponds to the ad-hoc polynomial proof given by Arts and Giesl [1, page 142].

[4] Termination Problems Data Base, see `http://www.lsi.upc.es/~albert/tpdb.html` and also `http://www.lri.fr/~marche/wst2004-competition/tpdb/Rubio/aoto.trs`.

This nicely corresponds to polynomial μ-reduction orderings. Arts and Giesl, however, do not consider polynomials over the rationals in their proofs of termination. It is worth to note that the use of rational coefficients (between 0 and 1) to introduce non-monotonicity in the corresponding term orderings makes a difference which cannot be simulated by just using polynomials over the naturals (see [8, Section 4] for a deeper discussion in this respect). In the proof of termination of Example 3 above, the difference is noticeable in that the proof which uses polynomials over the rationals is pretty simple and automatic (Example 4), but it becomes difficult or impossible when more traditional base orderings are used in combination with the dependency pairs approach (e.g., RPOS or polynomials over the naturals).

4 Conclusion

The use of μ-reduction orderings based on polynomial interpretations over the real or rational numbers can play a role in proofs of termination of term rewriting. Moreover, we stress that μ-reduction orderings provide a more general framework and, in fact, other μ-reduction orderings (e.g., the context-sensitive recursive path ordering, CSRPO [3]) could also be suitable for implementing the necessary comparisons.

There are, however, many theoretical and practical issues that deserve further investigation. An exciting one is: *are the polynomial interpretations over the reals (or rationals) more powerful than polynomial interpretations over the naturals?* As remarked above, regarding the generation of term orderings which are *not* fully monotonic, the answer is yes (already for the rationals). Regarding the generation of monotonic term orderings, the answer is not clear yet. In particular, we do not know of any TRS which can be proved terminating by using a reduction ordering based on a polynomial interpretation over the reals (or rationals) but cannot be proved terminating by using a polynomial interpretation over the naturals.

References

1. T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs *Theoretical Computer Science*, 236:133-178, 2000.
2. T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical report, AIB-2001-09, RWTH Aachen, Germany, 2001.
3. C. Borralleras, S. Lucas, and A. Rubio. Recursive Path Orderings can be Context-Sensitive. In *Proc. of CADE'02*, LNAI 2392:314-331,
4. E. Contejean and C. Marché. C*i*ME: Completion Modulo *E*. In *Proc. of RTA'96*, LNCS 1103:416-419, Springer-Verlag, Berlin, 1996.
5. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17(3):279–301, 1982.
6. D.S. Lankford. On proving term rewriting systems are noetherian. Technical Report, Louisiana Technological University, Ruston, LA, 1979.
7. S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):293-343, 2002.
8. S. Lucas. Polynomials for proving termination of context-sensitive rewriting. In *Proc. of FOSSACS'04*, LNCS 2987:318-332, Springer-Verlag, Berlin, 2004.
9. E. Ohlebusch. *Advanced Topics in Term Rewriting*. Springer-Verlag, Apr. 2002.
10. T. Suzuki, A. Middeldorp, and T. Ida. Level-Confluence of Conditional Rewrite Systems with Extra Variables in Right-Hand Sides. In *Proc. of RTA'95*, LNCS 914:179-193, Springer-Verlag, Berlin, 1995.
11. H. Zantema. Termination of Context-Sensitive Rewriting. In *Proc. of RTA'97*, LNCS 1232:172-186, Springer-Verlag, Berlin, 1997.

A Practical Approach to Proving Termination of Recursive Programs in Theorema

Nikolaj Popov, Tudor Jebelean*

Research Institute for Symbolic Computation, Linz, A–4232 Hagenberg, Austria
popov@risc.uni-linz.ac.at

Abstract. We report work in progress concerning the theoretical basis and the implementation in the Theorema system of a methodology for the generation of verification conditions for recursive procedures, with the aim of practical verification of recursive programs. Proving total correctness is achieved by proving separately partial correctness and then termination. In contrast to other approaches, which use a special theory describing the behavior of programs, we use such a theory only "in the background", for developing a general rule for generating verification conditions, while the conditions themselves are presented (and provable) using the theories relevant to the program text only. This is very important for automatic proving, since it reduces significantly the effort of the provers. We performed practical experiments in which various programs are completely verified using the verification condition generator and the provers of the Theorema system.

Introduction. While proving [partial] correctness of non-recursive procedural programs is quite well understood, for instance by using Hoare Logic [3], [6], there are relatively few approaches to recursive procedures (see e.g. [8] Chap. 2).

We discuss here a practical approach to automatic generation of verification conditions for functional recursive programs, partially based on Scott induction in the fixpoint theory of programs [13,11,7,9] and the implementation of this approach. The implementation is part of the *Theorema* system, and complements the research performed in the *Theorema* group on verification and synthesis of functional algorithms based on logic principles [1,2,4].

The *Theorema* system (`www.theorema.org`, [12]) aims at realization of a computer aided assistant for the working mathematicians and engineers, which integrates automatic reasoning, algebraic computing, and equational solving. The system provides an uniform environment in natural logico-mathematical language for defining, testing, and proving properties of algorithms, and in general for creating and investigating mathematical models.

We consider the correctness problem expressed as follows: *given* the program (by its source text) which computes the function F and given its specification by a precondition on the input $I_F[x]$ and a postcondition on the input and the output $O_F[x, y]$, *generate* the verification conditions which are [minimally] sufficient for the program to satisfy the specification.

For simplifying the presentation, we consider here the "homogeneous" case: all functions and predicates are interpreted over the same domain. Proving the verification conditions will require the *specific theory* relevant to this domain and to the auxiliary functions and predicates which occur in the program.

* The program verification project in the frame of e-Austria Timişoara is supported by BMBWK (Austrian Ministry of Education, Science and Culture), BMWA (Austrian Ministry of Economy and Work) and MEC (Romanian Ministry of Education and Research). The Theorema project is supported by FWF (Austrian National Science Foundation) – SFB project P1302. Additional support comes from the EU project CALCULEMUS (HPRN-CT-2000-00102).

The functional program of F can be interpreted as a set of predicate logic formulae, and the correctness of the program can be expressed as:

$$\forall x \; I_F[x] : O_F[x, F[x]], \tag{1}$$

which we will call the *correctness formula* of F. In order for the program to be correct, the correctness formula (1) must be a logical consequence the formulae corresponding to the definition of the function (and the specific theory). This approach was previously used by other authors and is also experimented in the Theorema system [1]. However, the proof of such one-single theorem may be difficult, because the prover has to find the appropriate induction principle and has to find out how to use the properties of the auxiliary functions present in the program.

The method presented in this paper generates several verification conditions, which are easier to prove. In particular, only the termination condition needs an inductive proof, and this termination condition is "reusable", because it basically expresses an induction principle which may be useful for several programs. This is important for automatic verification embedded in a practical verification system, because it leads to early detection of bugs (when proofs of simpler verification conditions fail).

Moreover, the verification conditions are provable in the frame of predicate logic, without using any theoretical model for program semantics or program execution, but only using the theories relevant to the predicates and functions present in the program text. This is again important for the automatic verification, because any additional theory present in the system will significantly increase the proving effort.

We start by developing a set of rules for generating verification conditions, for programs having a particular structure. The rules for partial correctness are developed using Scott induction and the fixpoint theory of programs, however the verification conditions themselves do not refer to this theory, they only state facts about the predicates and functions present in the program text. In particular, the termination condition consists in a property of a certain simplified version of the original program. By inspecting the shape of these rules for several program structures, it is possible to derive a more general rule for the derivation of verification conditions, such that the correctness formula (see above) **is a logical consequence of these verification conditions** in the frame of predicate logic, without using any model of computation.

We approach the correctness problem by splitting it into two parts: *partial correctness* (prove that the program satisfies the specification provided it terminates), and *termination* (prove that the program always terminates). Proving *partial correctness* may be achieved by Scott's induction [13,11,7,9] – a detailed description of the method for a certain class of functional programs is presented in [10].

Example: Simple Recursive Programs. Let be the program:

$$F[x] = \textbf{ If } Q[x] \textbf{ then } S[x] \textbf{ else } C[x, F[R[x]]],$$

where Q is a predicate and S, C, and R are auxiliary functions whose total correctness is assumed (S is a "simple" function, C is a "combinator" function, and R is a "reduction" function). Note that the program above can be seen as an abbreviated notation for the logical formulae:

$$\forall x \; (Q[x] \implies F[x] = S[x])$$

$$\forall x\ (\neg Q[x] \implies F[x] = C[x, F[R[x]]]).$$

Using Scott induction in the fixpoint theory of functions, one obtains the following verification conditions for the *partial correctness* of the program of F :

$$\forall x\ I_F[x]:\ (Q[x] \implies O_F[x, S[x]])$$

$$\forall x, y\ I_F[x]:\ ((\neg Q[x] \wedge O_F[R[x], y]) \implies O_F[x, C[x, y]])$$

(In fact, the conditions can be further decomposed using the preconditions and the post-conditions of the auxiliary functions, see [5].)

The condition for the *termination* of the program can be expressed using a simplified version of the initial function:

$$F'[x] = \textbf{ If } Q[x] \textbf{ then } 0 \textbf{ else } F'[R[x]],$$

which only depends on Q and R. Namely, the verification condition is

$$\forall x\ I_F[x] : F'[x] = 0, \tag{2}$$

which must be proven based on the logical formulae corresponding to the definition of F' (and the local theory relevant to the program). The condition follows from the equivalence of the termination properties of F and F', which can be proven in the fixpoint theory of functions e. g. by using induction on the number of recursion steps (see [14]).

The termination condition can be further refined to:

$$\forall x\ I_F[x] : ((Q[x] \Longrightarrow P[x]) \wedge (\neg Q[x] \Longrightarrow (P[R[x]] \Longrightarrow P[x])) \implies \forall x P[x]),$$

where P is a new predicate symbol, which is a proof in second order logic of the formula above for any P. This condition defines in fact an induction principle. By taking $P[x] \Longleftrightarrow (F'[x] = 0)$, one can obtain a proof of (2). By taking $P[x] \Longleftrightarrow O_F[x, F[x]]$, one can obtain a proof of the correctness formula (1).

Note that both conditions only depend on Q and R, thus they abstract some part of the function definition. In practical programming, these Q and R correspond to typical algorithm schemes (take e. g. $Q[x] \Longleftrightarrow (x = 0)$ and $R[x] = (x - 1)$), thus the termination condition (whose proof usually involves induction) is usable for several programs.

Generalization. By applying the same method to other recursion schemes, one notes that the verification conditions for the partial correctness can be generated directly (without using an additional theory) by applying few basic principles:

- check the input conditions when calling subroutines;
- accumulate all reasonable assumptions (coming from the input condition of the main function, from **if–then–else** statements and from the correctness formulae of the subroutines),
- try to finally obtain the correctness property for the output of F.

Furthermore, the termination condition can be generated either:

- as the "identical zero" property of the simplified function, or

- as the appropriate induction principle which makes the correctness formula a logical consequence the partial correctness conditions.

Implementation and experiments. The methods described above are implemented in the *Theorema* system and we are studying various recursive schemes and several test cases in order to improve the power of the verification condition generator. Furthermore, the concrete proof problems are used as test cases for the provers of *Theorema* and for experimenting with the organization and management of the mathematical knowledge. Currently we are able to generate the verification conditions and to prove them automatically in the *Theorema* system for many concrete programs.

References

1. A. Craciun, B. Buchberger. Functional Program Verification with Theorema. In *Computer Aided Verification of Information Systems (CAVIS-04)*, 2003. Technical Report 03-05, Institute e-Austria Timisoara (`www.ieat.ro`).
2. B. Buchberger. Verified Algorithm Development by Lazy Thinking. In *International Mathematica Symposium (IMS 2003)*, Imperial College, London, July 2003.
3. C. A. R. Hoare. *An axiomatic basis for computer programming. Comm. ACM*, 12, 1969.
4. T. Jebelean, L. Kovacs, N. Popov. Verification of Imperative Programs in Theorema. In *1st South-East European Workshop in Formal Methods (SEEFM03)*, Thessaloniki, Greece, 20 November 2003.
5. T. Jebelean. Forward Verification of Recursive Programs. In *Computer Aided Verification of Information Systems (CAVIS-04)*, 2004. Technical report 04-01, Institute e-Austria Timisoara (`www.ieat.ro`).
6. B. Buchberger, F. Lichtenberger. *Mathematics for Computer Science I - The Method of Mathematics. (in German)* Springer, 2nd edition, 1981.
7. Z. Manna. *Mathematical Theory of Computation.* McGraw-Hill Inc., 1974.
8. M. C. Paull. *Algorithm Design. A recursion transformation framework.* Wiley, 1987.
9. N. Popov. Operators in Recursion Theory. Technical Report 03-06, RISC-Linz, Austria, 2003.
10. N. Popov, T. Jebelean. A Practical Approach to Verification of Recursive Programs in Theorema. In T. Jebelean and V. Negru, editors, *Proceedings of the 5th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2003)*, Timisoara, Romania, 1–4 October 2003.
11. J. Loeckx, K. Sieber. *The Foundations of Program Verification.* Teubner, second edition, 1987.
12. B. Buchberger, C. Dupre, T. Jebelean, F. Kriftner, K. Nakagawa, D. Vasaru, W. Windsteiger. Theorema: A progress report. In *International Symposium on Integrating Computation and deduction (Calculemus 2000)*, St.Andrews, Scotland, 2000.
13. J. W. de Bakker, D. Scott. A Theory of Programs. In *IBM Seminar*, Vienna, Austria, 1969.
14. N. Popov, T. Jebelean. Verification of Simple Recursive Programs: Sufficient Conditions. Technical Report 04-06, RISC-Linz, Austria, 2004.

Abstract Partial Evaluation for Termination Analysis

Lior Tamary and Michael Codish

Department of Computer Science, Ben-Gurion University of the Negev, Israel
(tamaryl|mcodish)@cs.bgu.ac.il

1 Introduction

Current techniques for proving termination of logic programs focus on size and instantiation information. Structure information is ignored and for many examples this leads to imprecision in the analysis. There are two approaches to consider structure information: either to enhance the abstract domain over which the analysis is performed; or to transform the program so that structure in the concrete program is considered before abstract interpretation is applied. In [5], the authors propose a technique to transform a program so that the predicate names are adorned to reflect different structure patterns. In this approach, the adorned program is equivalent to the original program and termination analysis of the adorned program is more precise.

In this paper we present an alternative approach based on well-studied techniques of partial evaluation (unfolding) and abstract interpretation. Partial evaluation is a natural choice when the intention is to pre-evaluate just so much as to consider the structure present in the program. The advantage of this approach is that it relies completely on well-studied formal techniques.

In addition to partial evaluation we also abstract the program, maintaining structure, but possibly adding computations. This process preserves nontermination and hence is safe for termination analysis. This often simplifies the transformation.

We obtain results similar to those illustrated in [5]. We also compare our results to those obtained using ECCE [3] which is an off-the-shelf partial evaluator. Our results are more precise and analyses are considerably faster than those obtained using ECCE.

2 Partial Evaluation for Termination Analysis

When considering partial evaluation as a preprocessing for termination analysis it is important to keep in mind that unfolding does not preserve non-termination in general. Hence a preprocessed program might appear to terminate while the original does not. To illustrate the problem consider the following example.

Example 1. The program below is nonterminating for the query `?- r(A,B)`. It is intended to find two natural numbers the product and sum of which each are 0.

```
r(X,Y) :- times(X,Y,0), plus(X,Y,0).        times(0,X,0).
plus(0, X, X).                               times(s(X),Y,Z) :-
plus(s(X), Y, s(Z)) :- plus(X,Y,Z).            times(X,Y,W), plus(W,Y,Z).
```

Partial evaluation (using ECCE) results in the program consisting of the single fact `r(0,0)` which is clearly terminating for the query `?- r(A,B)`.

The ECCE [3] partial evaluator comes with a predefined setting `‘‘t’’` which is non-termination preserving and hence safe for termination analysis. This setting is used in all of our experiments.

3 Our method

Our method consists of three simple program transformations: **(a)** Binarisation — A program P is transformed to a binary program P' — this phase involves an abstraction, however termination of P' implies termination of P; **(b)** Binary unfolding — Unfolding a binary program is a simple transformation which preserves termination (and nontermination) — The main problem is to decide how much to unfold; and **(c)** Predicate renaming — This is a simple transformation along the lines of adorning [5] in which structural information is “encoded” in the predicate names. We illustrate that for the examples illustrated in [5] (and others) the TerminWeb analyzer [6] is able to prove termination after the application of the above transformations.

Binarisation: For a logic program P, the following (possibly infinite program) has the same termination behaviour (and makes observable the same answers and calls) as P [1].

$$bin(P) = \left\{ (h \leftarrow b_i)\theta \;\middle|\; \begin{array}{l} h \leftarrow b_1, \ldots, b_n \in P,\ 1 \leq i \leq n, \\ \theta \in ans_P(b_1, \ldots, b_{i-1}) \end{array} \right\}$$

where $ans_P(G)$ denotes the answer set for goal G with program P. As $bin(P)$ cannot be computed, we approximate it by taking an approximation of $ans_P(G)$. For the results of this paper we look at the approximation where $ans_P(G)$ maps all goals to the singleton $\{\epsilon\}$ (the empty substitution). We denote by $bin^a(P) = \left\{ h \leftarrow b_i \;\middle|\; h \leftarrow b_1, \ldots, b_n \in P,\ 1 \leq i \leq n \right\}$.

Example 2. The following Prolog program, quoted from [5], rewrites an expression in variable x by repeatedly applying laws of distributivity. The original statement of the problem is given in terms of rewrite systems and is described in [2].

```
dist(x,x).                              % (1)
dist(x*x,x*x).                          % (2)
dist(X+Y,U+V) :- dist(X,U), dist(Y,V).  % (3)
dist(X*(Y+Z),T) :- dist(X*Y+X*Z,T).     % (4)
dist((X+Y)*Z,T) :- dist(X*Z+Y*Z,T).     % (5)
```

The program $bin^a(P)$ is obtained by replacing the clause `(3)` by the two clauses

```
dist(X+Y,U+V) :- dist(X,U).  % (3a)        dist(X+Y,U+V) :- dist(Y,V).  % (3b)
```

Theorem 1. *Termination of $bin^a(P)$ implies termination of P.*

Proof. The result follows because P and $bin(P)$ have the same termination behaviour and because $bin^a(P)$ is more general than $bin(P)$.

Binary unfolding: Binary unfolding, in contrast to general unfolding, is guaranteed to preserve termination and nontermination [1](theorem 4.3). A binary clause $a \leftarrow b$ in a binary program P can be replaced by the set of binary clauses

$$unf_P(a \leftarrow b) = \left\{ (a \leftarrow c)\theta \;\middle|\; b' \leftarrow c \in P, mgu(b, b') = \theta \right\}.$$

The unfolding operation can be applied repeatedly, however, in case the call graph of the program contains cycles, this process may not terminate. In practice, whenever it is possible to unfold a clause along a cyclic path, we do so only if the size of the clause body gets smaller according to a selected norm (we currently apply the termsize norm by default).

Example 3. Unfolding the clause (4) with (3a) from the binarised program $bin^a(P)$ of Example 2 yields: `dist(A*(B+C),D+E) :- dist(A*B,D).`

This demonstrates our strategy: Even though clause (3a) is cyclic, the body of the unfolded clause decreases when compared to (4) (in at least one of its arguments), hence we choose to follow this path for unfolding.
Unfolding $bin^a(P)$ results in the following program (where facts have been omitted as they do not contribute to non-termination).

```
dist(A+B,D+E) :- dist(A,D).          % (1')
dist(A+B,D+E) :- dist(B,E).          % (2')
dist(A*(B+C),D+E) :- dist(A*B,D).    % (3')
dist(A*(B+C),D+E) :- dist(A*C,E).    % (4')
dist((A+B)*C,D+E) :- dist(A*C,D).    % (5')
dist((A+B)*C,D+E) :- dist(B*C,E).    % (6')
```

Clauses (3'), (4'), (5') and (6') are obtained as a result of unfolding, whereas (1') and (2') remain unchanged. TerminWeb succeeds to prove termination of this program, where with the original it fails.

Predicate renaming: This is a simple program transformation to remove from the program calls to clauses with incompatible structure. Once program analysis will be applied these inconsistencies will disappear and precision will be lost.

Example 4. The program on the left is presented in [5]. We first rename each predicate occurrence to obtain the program on the right.

```
loop(0,X,s(Y)) :- loop(0,s(X),Y).            loop1(0,X,s(Y)) :- loop2(0,s(X),Y).
loop(0,X,Y) :- loop(s(0),X,Y).               loop3(0,X,Y) :- loop4(s(0),X,Y).
loop(s(0),s(X),Y) :- loop(s(0),X,s(Y)).      loop5(s(0),s(X),Y) :- loop6(s(0),X,s(Y)).
```

We then add: (a) entry points of the form $loop$`(A,B,C) :-` $loop_j$`(A,B,C)` corresponding to the new head atoms; and (b) call graph edge clauses of the form $loop_i$`(A,B,C) :-` $loop_j$`(A,B,C)` if the body atom $loop_i(\ldots)$ is unifiable with the head atom $loop_j(\ldots)$.

```
loop(A,B,C) :- loop1(A,B,C).
loop(A,B,C) :- loop3(A,B,C).
loop(A,B,C) :- loop5(A,B,C).
loop2(A,B,C) :- loop1(A,B,C).
loop2(A,B,C) :- loop3(A,B,C).
loop4(A,B,C) :- loop5(A,B,C).
loop6(A,B,C) :- loop5(A,B,C).
```

TerminWeb can prove the termination of the transformed program (but not the original).

4 Experimentation

We have applied the approach to a set of benchmarks which can be found at `http://www.cs.bgu.ac.il/~mcodish/TerminWeb/` (under "Examples"). All of the transformations applied are fully implemented. Table 1 summarizes the results. The rows in the table correspond to the test programs. The columns indicate: `program` - the program, `query` - the query pattern ('b' means bound and 'f' means free), `*.pl` - termination analysis for the original program, `*.pe` - termination analysis after applying ECCE (non-termination preserving mode), `*.bin` - termination analysis after applying binarisation, `*.bin_pe` - termination analysis after applying ECCE to the binarised program, and `*.bin_unf` - termination analysis after applying binarisation and binary unfolding (as described in Section 3).

For each application of the termination analyzer the table indicates the total termination analysis time (in milliseconds) if the analysis was successful, or "no" if termination was not

[1] Selected norm is "termsize edges".
[2] The analyzer timed-out.
[3] The original program is binary to start with.

Program	Query	*.pl	*.pe	*.bin	*.bin_pe	*.bin_unf
dist	dist(b,f)	no	no	no	3040	10
	dist(f,b)	no	650	no	3490	10
d	d(b,f,f)	no	no	no	1280	20
	d(f,f,b)	no	no	no	1280	20
thrm1[1]	false(b)	no	no	no	no	20
thrm2[1]	true(b)	no	1010	no	450	20
balance	balance(b,f)	75780	∞^2	63250	43290	2880
plus	reduce(b,f)	no	1570[3]	no[3]	1570	20

Table 1. Results

proven. The analyses were performed through the web interface to TerminWeb using the default options of the analyzer unless indicated otherwise. We have focused on the cost of the termination analysis of the transformed programs, ignoring for the time being the cost of the transformations themselves. We have not compared the cost of unsuccessful analyses (also to get a "no" has a cost).

The results indicate that off-the-shelf partial evaluation is not suitable for termination analysis. This is apparent from the columns `*.pe` and `*.bin_pe`, illustrating that the analyses of the partial evaluated programs are less precise and are more costly than those obtained with simple unfolding. The purpose of the column `*.bin` is to illustrate that binarisation on its own does not solve the problem.

5 Conclusions and Ongoing Work

We illustrate that simple techniques of partial evaluation can be applied to improve the precision of termination analysis. Most likely, partial evaluators can be better tuned to perform this task. Meanwhile we have implemented straightforward transformations to illustrate the point.

An advantage in comparison to the adornments approach is that all is based on simple and well-studied transformations. A disadvantage is that it is not clear how to generalize ours for numeric computation.

We note that unfolding has been recognized important for improving the precision of termination analysis already in [4]. We note also that TerminWeb can analyse the mergesort program directly without applying unfolding.

References

1. M. Codish and C. Taboch. A semantic basis for the termination analysis of logic programs. *The Journal of Logic Programming*, 41(1):103–123, 1999.
2. N. Dershowitz. 33 examples of termination. In H. Comon and J.-P. Jouannaud, editors, *French Spring School of Theoretical Computer Science Advanced Course on Term Rewriting (Font Romeux, France, May 1993)*, volume 909, pages 16–26, Berlin, 1995. Springer-Verlag.
3. M. Leuschel. The ecce partial deduction system. `http://www.ecs.soton.ac.uk/~mal/systems/`.
4. N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. Unfolding the mystery of mergesort. In N.E. Fuchs, editor, *Proceedings of the 7th International Workshop on Logic Program Synthesis and Transformation*, volume 1463 of *Lecture Notes in Computer Science*. Springer-Verlag Heidelberg, July 1997.
5. A. Serebrenik and D . De Schreye. Proving termination with adornments. In M. Bruynooghe, editor, *Pre-proceedings of International Symposium on Logic-based Program Synthesis and Transformation*, pages 139–154, 2003. `http://www.cs.kuleuven.ac.be/publicaties/rapporten/cw/CW365.pdf`.
6. C. Taboch, S. Genaim, and M. Codish. Terminweb: Semantic based termination analyser for logic programs, 2002. `http://www.cs.bgu.ac.il/~mcodish/TerminWeb`.

Relative Termination in Term Rewriting

Hans Zantema

Department of Computer Science, TU Eindhoven
P.O. Box 513, 5600 MB, Eindhoven, The Netherlands
h.zantema@tue.nl

Abstract. We investigate

- how current techniques for proving termination for term rewriting systems (TRSs) essentially use relative termination, although not always stated explicitly,
- how using relative termination is helpful for proving termination in a modular way,
- which techniques for proving termination easily extend to proving relative termination,
- how for some applications relative termination is the natural underlying concept, justifying studying relative termination in itself.

For abstract reduction systems $\rightarrow_R$ and $\rightarrow_S$ we write $\rightarrow_R / \rightarrow_S = \rightarrow_S^* \cdot \rightarrow_R \cdot \rightarrow_S^*$. We say that $\rightarrow_R$ is *terminating relative to* $\rightarrow_S$ (notation: $\mathsf{SN}(\rightarrow_R / \rightarrow_S)$) if $\rightarrow_R / \rightarrow_S$ is terminating. Equivalently, $\mathsf{SN}(\rightarrow_R / \rightarrow_S)$ holds if and only if no infinite sequence $t_1, t_2, t_3, \ldots$ exists such that $t_i \rightarrow_{R \cup S} t_{i+1}$ for all $i = 1, 2, 3, \ldots$, and $t_i \rightarrow_R t_{i+1}$ for infinitely many values of i.

In case R and S are term rewriting systems with corresponding rewrite relations $\rightarrow_R$ and $\rightarrow_S$ we shortly write $\mathsf{SN}(R/S)$ for $\mathsf{SN}(\rightarrow_R / \rightarrow_S)$. Clearly termination of R coincides with $\mathsf{SN}(R/\emptyset)$, hence termination is a special case of relative termination.

Relative termination was investigated by Alfons Geser in his PhD thesis in 1990, [2]. It is thoroughly used for string rewriting in TORPA, [8].

We start by a characterization of relative termination by compatible orders.

Theorem 1. *Let R, S be TRSs. Then $\mathsf{SN}(R/S)$ holds if and only if a strict order (transitive and irreflexive) $>$ and a quasi-order (transitive and reflexive) $\geq$ on terms exist such that*

- *$R \subseteq >$ and $S \subseteq \geq$,*
- *$>$ and $\geq$ are closed under contexts and substitutions,*
- *$>$ is well-founded, and*
- *$\geq \cdot > \cdot \geq \subseteq >$.*

Here we avoid to require that $>$ is the strict part of $\geq$ since it may occur that $\geq$ is closed under contexts and substitutions but its strict part is not.

The next property may be used for proving (relative) termination in a stepwise way.

Theorem 2. *Let R, S, R' and S' be TRSs for which*

- *$R \cup S = R' \cup S'$,*
- *$\mathsf{SN}(R'/S')$, and*
- *$\mathsf{SN}((R \cap S')/(S \cap S'))$.*

Then $\mathsf{SN}(R/S)$.

Variants of these basic properties Theorems 1 and 2 were already given in [2]. Theorem 2 may be used as follows. If we want to prove $\mathsf{SN}(R/S)$ then we try to split up $R \cup S$ into two disjoint parts R' and S' for which $R' \neq \emptyset$ and $\mathsf{SN}(R'/S')$. Typically this is done by searching for a compatible order $>$ for $R \cup S$. This may partially succeed: for some rules $l \to r$ we have $l > r$, but for the others we have $l \geq r$, for $>, \geq$ satisfying the requirements of Theorem 1. Then we choose R' to consist of the rules $l \to r$ satisfying $l > r$, and S' the rest, then $\mathsf{SN}(R'/S')$ holds by Theorem 1. Hence by Theorem 2 we may weaken the proof obligation $\mathsf{SN}(R/S)$ to $\mathsf{SN}((R \cap S')/(S \cap S'))$, i.e., all rules from R' may be removed. This process may be repeated as long as it is applicable. If after a number of steps $R \cap S' = \emptyset$ then $\mathsf{SN}((R \cap S')/(S \cap S'))$ trivially holds and the desired proof has been given.

As a very simple example we prove termination of the TRS consisting of the following three rules:

$$a(b(x)) \to b(a(x)),\ b(c(x)) \to c(b(x)),\ c(a(x)) \to a(c(x)).$$

Doing this in one step is not that easy, but proving $\mathsf{SN}(R'/S')$ is very simple for R' consisting of the first rule and S' consisting of the rest: choose the polynomial order where a, b, c are interpreted as $\lambda x \cdot 2x$, $\lambda x \cdot x+1$ and the identity, respectively. Hence by Theorem 2 it remains to prove termination of the last two rules, which is immediate by recursive path order using the precedence $b > c > a$.

Dependency pairs

For a TRS R over an alphabet Σ let Σ_D be the set of *defined symbols* of R, i.e., the set of symbols occurring as the root of the left hand side of a rule in R. For every defined symbol $f \in \Sigma_D$ we introduce a fresh symbol $\overline{f}$, usually written as the capitalized version of f if f is a lower case letter. The TRS $DP(R)$ over $\overline{\Sigma} = \Sigma \cup \{\overline{f} \mid f \in \Sigma_D\}$ is defined to consist of all rules of the shape

$$\overline{f}(t_1, \ldots, t_n) \ \to \ \overline{g}(u_1, \ldots, u_m)$$

for which $f(t_1, \ldots, t_n) = l$ and $r = C[g(u_1, \ldots, u_m)]$ for some rule $l \to r$ in R and $f, g \in \Sigma_D$. Rules of $DP(R)$ are called *dependency pairs*. In our view the main theorem of dependency pairs from [1] is the following.

Theorem 3. *Let R be any TRS. Then* $\mathsf{SN}(R)$ *if and only if* $\mathsf{SN}(DP(R)/R)$.

Hence proving termination of a TRS can be done by proving relative termination using dependency pairs. In [1] this is done by analyzing chains of dependency pairs, essentially being reductions of $R \cup DP(R)$ in which $DP(R)$-steps are only applied on the root. It has to be proved that only finitely many $DP(R)$-steps occur in such a $R \cup DP(R)$-reduction, corresponding to relative termination. On a first glance it seems that in our approach the requirement that the $DP(R)$-steps are only applied on the root is lost. However, by type elimination / type introduction as introduced in [5] this aspect is easily brought in separately.

It is a natural question whether dependency pairs can be used for relative termination rather than termination. More precisely, we wonder whether $\mathsf{SN}(R/S)$ can be concluded from $\mathsf{SN}(DP(R)/R')$ for some R' like $R' = R \cup S \cup DP(S)$. This is not the case: let R consist of the rule $a \to b$ and S of the rule $f(b) \to f(a)$. Then clearly $\mathsf{SN}(R/S)$ does not hold, while $\mathsf{SN}(DP(R)/\cdots)$ holds since $DP(R) = \emptyset$. We expect that in some cases $\mathsf{SN}(R/S)$ may be concluded from $\mathsf{SN}(DP(E(R))/R \cup S \cup DP(S))$ for a TRS $E(R)$ consisting of rules of the shape $C[l] \to C[r]$ for $l \to r \in R$, along the lines of [3].

Generalized argument filtering

Argument filtering is a technique used in the dependency pair method in [1] to deal with combinations of strict and non-strict inequalities, hence essentially for relative termination. The idea is that terms are reduced by filtering arguments: for one or more function symbols arguments may be removed or the symbol itself is removed and one argument is chosen. Here we generalize this to recursive program schemes.

A *recursive program scheme (RPS)* is defined to be a TRS in which all left hand sides of the rules have distinct root symbols, and all of these left hand sides are of the shape $f(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are distinct variables. By definition an RPS is orthogonal, hence confluent. If an RPS S is terminating then we write $S(t)$ for the unique normal form of a term t, and $S(R) = \{S(l) \to S(r) \mid l \to r \in R\}$ for a TRS R.

Theorem 4. *Let R, R' be TRSs satisfying $\mathsf{SN}(S(R)/S(R'))$ for some non-erasing terminating RPS S. Then $\mathsf{SN}(R/R')$ holds.*

The proof of this theorem is easily given using Lemma's 6.5.4 and 6.5.5 from [7]. Unfortunately, for this theorem the requirement of non-erasingness is essential: for R consisting of the rule $a \to f(a)$ and $R' = \emptyset$ and S consisting of the rule $f(x) \to b$ we do have $\mathsf{SN}(S(R)/S(R'))$ but not $\mathsf{SN}(R/R')$. However, for dependency pairs the requirement of non-erasingness is not essential:

Theorem 5. *Let R be a TRS satisfying $\mathsf{SN}(S(DP(R))/S(R))$ for some terminating RPS S. Then $\mathsf{SN}(R)$ holds.*

Semantic labelling

The theory of semantic labelling ([6,7]) directly extends to relative termination:

Theorem 6. *Let $\overline{R}$ and $\overline{R'}$ be the labelled versions of TRS R, R' with respect to some quasi-model, and let Decr be the corresponding set of decreasing rules, as defined in [6,7]. Then $\mathsf{SN}(R/R')$ if and only if $\mathsf{SN}(\overline{R})/(\overline{R'} \cup \mathsf{Decr}))$.*

This theorem turned out to be extremely fruitful in TORPA ([8]).

Relative termination in liveness and fairness

In [4] it was investigated how termination of term rewriting may be applied for proving liveness properties: proving that some desired property eventually will hold in some computation. For many instances of such problems fairness is involved: an infinite computation is only considered valid if some particular steps are applied infinitely often. As a very simple example consider a waiting line in which on the one end new clients may enter and on the other end clients may be served. Introducing a constant serve for the serving end of the line, a unary symbol top for the end where new clients may enter, and unary symbols $\mathsf{old}, \mathsf{new}$ for old and new clients, respectively, the behavior of the waiting line may be described by the three rules $\mathsf{top}(x) \to \mathsf{top}(\mathsf{new}(x))$, $\mathsf{new}(\mathsf{serve}) \to \mathsf{serve}$, $\mathsf{old}(\mathsf{serve}) \to \mathsf{serve}$.

For this system we want to prove the liveness property that eventually all old clients will be served. According to the method described in [4] this can be done by considering the transformed TRS obtained by replacing $\mathsf{top}(x) \to \mathsf{top}(\mathsf{new}(x))$ by $\mathsf{top}(x) \to \mathsf{top}(\mathsf{c}(\mathsf{new}(x)))$ and adding the rules $\mathsf{c}(\mathsf{new}(x)) \to \mathsf{new}(\mathsf{c}(x))$, $\mathsf{c}(\mathsf{old}(x)) \to \mathsf{old}(\mathsf{c}(x))$, $\mathsf{c}(\mathsf{old}(x)) \to \mathsf{old}(x)$.

Without a fairness requirement the liveness property does not hold: if the line contains an old client, the serve rules are never applied and infinitely often a new client enters, then an infinite computation is obtained in which the old client remains for ever in the waiting line. However, if we require that in the infinite computation infinitely often clients are served, then the liveness property holds. In terms of relative termination this means that R consists of the serving rules $\mathsf{new}(\mathsf{serve}) \to \mathsf{serve}$, $\mathsf{old}(\mathsf{serve}) \to \mathsf{serve}$ and S consists of the other rules

$$\mathsf{top}(x) \to \mathsf{top}(\mathsf{new}(x)),\quad \mathsf{c}(\mathsf{new}(x)) \to \mathsf{new}(\mathsf{c}(x)),\quad \mathsf{c}(\mathsf{old}(x)) \to \mathsf{old}(\mathsf{c}(x)),\quad \mathsf{c}(\mathsf{old}(x)) \to \mathsf{old}(x)$$

By the observations given in [4] the liveness property under the above fairness restriction can be concluded from $\mathsf{SN}(R/S)$. This may be proved using semantic labelling; by TORPA this is done automatically.

This example is very simple, but we believe that this approach is applicable for much more involved examples, and that relative termination is the natural property to be considered for proving liveness properties as soon as fairness properties come in.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. A. Geser. *Relative Termination*. PhD thesis, Universität Passau, Germany, 1990.
3. J. Giesl and D. Kapur. Dependency pairs for equational rewriting. In A. Middeldorp, editor, *Proceedings of the 12th Conference on Rewriting Techniques and Applications (RTA)*, volume 2051 of *Lecture Notes in Computer Science*, pages 93–107. Springer, 2001.
4. J. Giesl and H. Zantema. Liveness in rewriting. In R. Nieuwenhuis, editor, *Proceedings of the 14th Conference on Rewriting Techniques and Applications (RTA)*, volume 2706 of *Lecture Notes in Computer Science*, pages 321–336. Springer, 2003.
5. H. Zantema. Termination of term rewriting: Interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.
6. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.
7. H. Zantema. Termination. In *Term Rewriting Systems, by Terese*, pages 181–259. Cambridge University Press, 2003.
8. H. Zantema. TORPA: Termination of Rewriting Proved Automatically. In *Proceedings of the 15th Conference on Rewriting Techniques and Applications (RTA)*, *Lecture Notes in Computer Science*, Springer, 2004.

Proving Termination with AProVE

Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke

LuFG Informatik II, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany
{giesl|thiemann|psk}@informatik.rwth-aachen.de,spf@i2.informatik.rwth-aachen.de

1 Introduction

The system AProVE (Automated Program Verification Environment) offers a variety of techniques for automated (innermost) termination proofs of (possibly conditional) TRSs, logic programs, and first-order functional programs. Besides efficient implementations of classical simplification orders (Sect. 2), it offers the *dependency pair* technique [2,11] which allows the application of classical orders to examples where they would fail otherwise (Sect. 3). In contrast to most other implementations, we integrated numerous refinements such as *narrowing*, *rewriting*, and *instantiation* of dependency pairs [2,10,12,13], recent improvements to reduce the constraints generated by the dependency pair technique [12,13,23], etc. Therefore, AProVE succeeds on many examples where other currently available systems for automated termination proofs fail. AProVE also features the *size-change principle* [20] and it is possible to combine this principle with dependency pairs [22]. The tool is written in Java and proofs can be performed both in a fully automated or in an interactive mode via a graphical user interface. Sect. 4 compares AProVE with related tools.

2 Direct Termination Proofs

In direct termination proofs, one tries to find a reduction order where all rules are decreasing. AProVE contains *recursive path orders* (RPOS) with multiset or lexicographic status for each function symbol as well as several restrictions of RPOS (RPO, LPOS, LPO, EMB, etc.). AProVE also offers *Knuth-Bendix orders* (KBO) using the polynomial-time algorithm of [18].

Finally, AProVE also features *polynomial orders* (POLO) where every function symbol is associated with a polynomial with natural coefficients. The user can specify the degree of the polynomials and the range of the coefficients. One can also provide individual polynomials for some function symbols manually. To prove termination, AProVE generates a set of polynomial inequalities stating that left-hand sides of rules should be greater than the corresponding right-hand sides. By the method of partial derivation [9,19], these inequalities are transformed into inequalities only containing coefficients, but no variables anymore. Finally, a search algorithm determines coefficients satisfying the resulting inequalities. The user can choose between brute force search, greedy search, a genetic algorithm, and a constraint-based method based on interval arithmetic, which is preferable in most examples.

To improve power and efficiency, one can apply a pre-processing step to remove rules from the TRS that do not influence the termination behavior. Here, AProVE tries to find a monotonic order $\succ$ such that the rules of the TRS $\mathcal{R}$ are at least weakly decreasing (i.e., $l \succsim r$ for all $l \to r \in \mathcal{R}$). Then those rules which are strictly decreasing can be removed, i.e., it suffices to prove termination of $\mathcal{R} \setminus \{l \to r \mid l \succ r\}$. This extends related approaches to remove rules [5,15,19,25] which were restricted to certain classes of orders or TRSs.

For this pre-processing, we use linear polynomial interpretations with coefficients from $\{0, 1\}$. AProVE's algorithm for polynomial orders solves constraints where some inequalities are strictly decreasing and all others are weakly decreasing in just one search attempt without backtracking [13]. Thus, removal of rules can be done very efficiently and it is repeated until no rule can be removed anymore.

3 Termination Proofs with Dependency Pairs

The *dependency pair* approach [2,11] increases the power of automated termination analysis significantly. For every TRS, it generates sets of inequality constraints. If there exist well-founded (quasi-)orders satisfying these constraints, then termination is proved. In AProVE, one can select whether to use the dependency pair approach for termination or for innermost termination proofs. The system can also check if a TRS is non-overlapping (because then innermost termination implies termination). To search for suitable orders, one can select any base order from Sect. 2.

Argument Filtering

However, most of these orders are *strongly* monotonic, while the dependency pair approach only requires *weak* monotonicity. (For polynomial orders, a weakly monotonic variant is obtained by permitting the coefficient 0. But `LPO(S)`, `RPO(S)`, and `KBO` are always strongly monotonic.) Thus, before searching for an order, some of the arguments of the function symbols in the constraints can be eliminated by an *argument filtering* [2]. Moreover, we developed an improvement by first applying the argument filtering and determining the constraints of the dependency pair approach afterwards [12,23].

Since there are exponentially many argument filterings, a crucial problem is to explore this search space efficiently. AProVE uses a depth-first algorithm [12] which starts with the set of argument filterings possibly satisfying the first constraint. Here we use the idea of [16] to keep argument filterings as "undefined" as possible. Then this set is reduced further to those filterings which can possibly satisfy the second constraint as well. This procedure is repeated until all constraints are investigated. By inspecting the constraints in a suitable order (instead of treating them separately as in [16]), already after the first constraint the set of possible argument filterings is rather small and one only inspects a small subset of all potential argument filterings.

Heuristics

To improve performance, AProVE offers several heuristics. For example, one can use heuristics to restrict the set of possible argument filterings. The most successful of these heuristics ("`Type`") only regards argument filterings where for every symbol f, either no argument position is eliminated or all non-eliminated argument positions are of the same type. Here, we use a (monomorphic) type inference algorithm to transform a TRS into a sorted TRS.

Another heuristic to increase the efficiency ("`EMB for DPs`") is to only use the very simple embedding order for orienting constraints which come from dependency pairs. Only for constraints from rules, one may apply more complicated orders like `LPO`, `RPO(S)`, etc. Since our depth-first algorithm to determine argument filterings starts with the dependency pairs, this reduces the search space significantly without compromising power very much.

Dependency Graph

To perform (innermost) termination proofs in a modular way, one constructs an estimated (innermost) dependency graph and regards its cycles separately [2,11]. One can select between standard [2] and more powerful recent estimations (EDG* / EIDG**) [13,16].

To benefit from all refinements on modularity of dependency pairs, we developed and implemented an improved technique which combines recent results on modularity of $\mathcal{C}_\varepsilon$-terminating TRSs [24] with arbitrary estimations of dependency graphs, cf. [12,23].

Dependency Pair Transformations

To increase power, a dependency pair can be transformed into several new pairs by *narrowing*, *rewriting*, and *instantiation* [2,10,12,13]. In contrast to [10,12], AProVE can instantiate dependency pairs both w.r.t. the pairs before and behind it in chains (the latter is called *forward* instantiation) [13]. The user can select which of these transformations should be used when applicable. Usually, all transformations should be enabled, since they are often crucial for the success of the proof and they can never "harm": if the termination proof succeeds without transformations, then it also succeeds when performing transformations [13], but not vice versa. However, the problem is when to use these transformations, since they may be applicable infinitely often. Moreover, transformations may increase runtime by producing a large number of similar constraints. AProVE performs transformations in "safe" cases where their application is guaranteed to terminate [12].

4 Comparison with Other Tools

Compared with other recent termination provers for TRSs (Arts [1], Cariboo [8], CiME [6], Termptation [4], TTT [17]), AProVE is the only system with improvements like automated dependency pair transformations, applying argument filterings before determining the constraints, and combining modularity results based on $\mathcal{C}_\varepsilon$-termination with recent dependency graph estimations. Moreover, it offers more base orders than any other system, it can also handle conditional TRSs, and integrates the size-change principle. Finally, AProVE's design permits the combination of powerful heuristics and different termination techniques.

The following experiments compare AProVE 1.0 (using its "Meta Combination" algorithm) with the only other tools currently available on the web (CiME and Termptation). They were tested on the collections of [3,7,21] (130 TRSs for termination, 151 TRSs for innermost termination). To show that the techniques described in [17] are a substantial restriction, in the last row we ran AProVE in a mode where we switched off all improvements and only used the methods available in [17]. Since [17] has several base orders and argument filtering heuristics, we took the ones which gave the best overall result on this collection.

	Termination		Innermost Term.	
System	Power	Time	Power	Time
AProVE	95.4 %	26.2 s	98.0 %	34.3 s
CiME	71.5 %	660.7 s	—	—
Termptation	65.4 %	521.8 s	72.8 %	681.7 s
AProVE with techniques of [17]	52.3 %	679.1 s	—	—

The "Power" column contains the percentage of those examples in the collection where the proof attempt was successful. The "Time" column gives the overall time for running the system on all examples of the collection (also on the ones where the proof attempt failed). For each example we used a time-out of 60 seconds on a Pentium IV with 2.4 GHz and 1 GB memory.

For more details on the above experiments and to download AProVE, the reader is referred to `http://www-i2.informatik.rwth-aachen.de/AProVE`. A more detailed description of the system can be found in [14].

References

1. T. Arts. System description: The dependency pair method. In *Proc. 11th RTA*, LNCS 1833, pages 261–264, 2000.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
3. T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09[1], RWTH Aachen, Germany, 2001.
4. C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In *Proc. 17th CADE*, LNAI 1831, pages 346–364, 2000.
5. A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137–159, 1987.
6. E. Contejean, C. Marché, B. Monate, and X. Urbain. CiME version 2, 2000. Available from `http://cime.lri.fr`.
7. N. Dershowitz. 33 examples of termination. In *Proc. French Spring School of Theoretical Computer Science*, LNCS 909, pages 16–26, 1995.
8. O. Fissore, I. Gnaedig, and H. Kirchner. Cariboo: An induction based proof tool for termination with strategies. In *Proc. 4th PPDP*, pages 62–73. ACM, 2002.
9. J. Giesl. Generating polynomial orderings for termination proofs. In *Proc. 6th RTA*, LNCS 914, pages 426–431, 1995.
10. J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1,2):39–72, 2001.
11. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
12. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proc. 10th LPAR*, LNAI 2850, pages 165–179, 2003.
13. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing dependency pairs. Technical Report AIB-2003-08[1], RWTH Aachen, Germany, 2003.
14. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proc. 15th RTA*, LNCS, 2004. To appear.
15. J. Giesl and H. Zantema. Liveness in rewriting. In *Proc. 14th RTA*, LNCS 2706, pages 321–336, 2003.
16. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. In *Proc. 19th CADE*, LNAI 2741, 2003.
17. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proc. 14th RTA*, LNCS 2706, pages 311–320, 2003.
18. K. Korovin and A. Voronkov. Verifying orientability of rewrite rules using the Knuth-Bendix order. In *Proc. 10th RTA*, LNCS 2051, pages 137–153, 2001.
19. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
20. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proc. POPL '01*, pages 81–92, 2001.
21. J. Steinbach. Automatic termination proofs with transformation orderings. In *Proc. 6th RTA*, LNCS 914, pages 11–25, 1995. Full version appeared as Technical Report SR-92-23, Universität Kaiserslautern, Germany.
22. R. Thiemann and J. Giesl. Size-change termination for term rewriting. In *Proc. 14th RTA*, LNCS 2706, pages 264–278, 2003.
23. R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proc. 2nd IJCAR*, LNAI, 2004. To appear.
24. X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In *Proc. IJCAR 2001*, LNAI 2083, pages 485–498, 2001.
25. H. Zantema. TORPA: termination of rewriting proved automatically. In *Proc. 15th RTA*, LNCS, 2004. To appear.

[1] Available from `http://aib.informatik.rwth-aachen.de`

Tyrolean Termination Tool

Nao Hirokawa and Aart Middeldorp

Institute of Computer Science, University of Innsbruck, 6020 Innsbruck, Austria
nao.hirokawa@uibk.ac.at
aart.middeldorp@uibk.ac.at

1 Introduction

This contribution describes the Tyrolean Termination Tool, the successor of the Tsukuba Termination Tool [8]. We describe the differences between the two and explain the new features in some detail. The Tyrolean Termination Tool (T$_{\text{T}}$T in the sequel) is a tool for automatically proving termination of first-order rewrite systems based on the dependency pair method of Arts and Giesl [1]. It produces high-quality output and has a convenient web interface. The tool is available at

`http://cl2-informatik.uibk.ac.at/ttt`

Figure 1 shows the web interface. T$_{\text{T}}$T incorporates several new improvements to the dependency pair method. In addition, it is now possible to run the tool in *fully automatic mode* on a *collection* of rewrite systems.

In the next section we describe the differences between the semi automatic mode and the Tsukuba Termination Tool. Section 3 describes the fully automatic mode. In Section 4

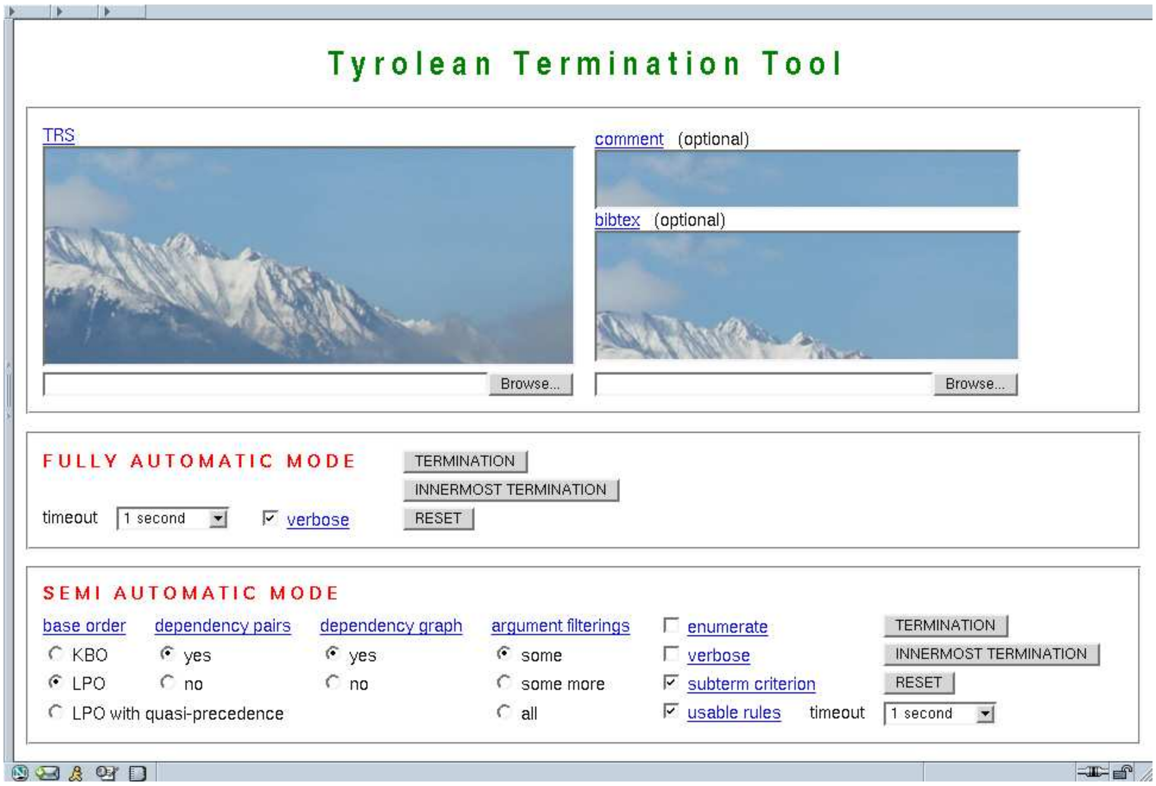


Fig. 1. A screen shot of the web interface of T$_{\text{T}}$T.

we describe how to input a collection of rewrite systems and how to interpret the resulting output.

2 Semi Automatic Mode

This menu corresponds to the options that were available in the Tsukuba Termination Tool. A first difference is that we now support the dependency pair method for innermost termination [1]. A second difference is that dependency pairs that are covered by the subterm criterion of Dershowitz [3] are excluded. The other differences are described in the following paragraphs.

First of all, when approximating the (innermost) dependency graph the original estimations of [1] are no longer available since the approximations described in [6] generally produce smaller graphs while the computational overhead is neglectable.

Secondly, the user can no longer select the cycle analysis method (all cycles separately, all strongly connected components separately, or the recursive SCC algorithm of Hirokawa and Middeldorp [7]). Extensive experiments reveal that the latter method outperforms the other two, so this is now the only supported method in T$_\mathsf{T}$T.

Finally, the default method to search for appropriate argument filterings has been changed from *enumeration* to the *divide and conquer* algorithm of [7]. By using dynamic programming techniques, the divide and conquer method has been improved (cf. the full version of [7]) to the extent that for most examples it is more efficient than the straightforward enumeration method. Still, there are TRSs where enumeration is more effective, so the user has the option to change the search strategy (by clicking the enumerate box).

New features include (1) a very useful criterion based on the subterm relation to discard SCCs of the dependency graph without considering any rewrite rules and (2) a very powerful modularity criterion for termination inspired by the *usable rules* of [1] for innermost termination. The second criterion[1] is stronger than previous modularity results in connection with the dependency pair method ([4,10,12]). These new features are described in detail in [9]. The first one is selected by clicking the *subterm criterion* box and the second by clicking the *usable rules* box.

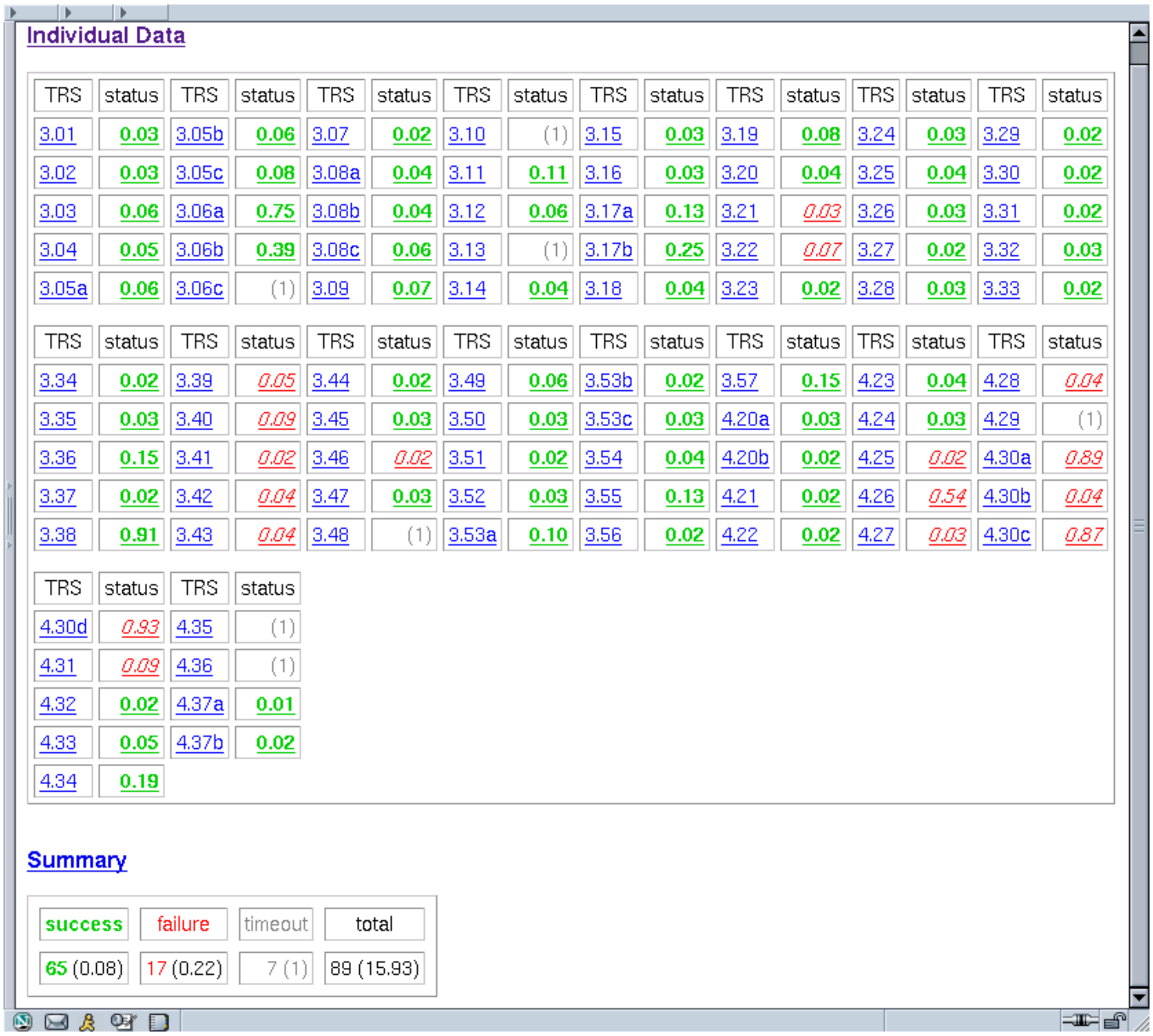
3 Fully Automatic Mode

In this mode T$_\mathsf{T}$T uses a simple strategy to (recursively) solve the ordering constraints for each SCC of the approximated dependency graph. The strategy is based on the new features described in the previous section and uses LPO (with strict precedence) with *some* argument filterings and linear polynomial interpretations with coefficients from $\{0, 1\}$ as base orders. The usefulness of the latter has been observed first in [5].

After computing the SCCs of the approximated (innermost) dependency graph, the strategy subjects each SCC to the following algorithm:

1. First we check whether the new subterm criterion is applicable.
2. If the subterm criterion was unsuccessful, we compute the usable rules (using the new modularity criterion for termination).
3. The resulting (usable rules and dependency pairs) constraints are subjected to the divide and conquer algorithm for computing suitable argument filterings with respect to the *some* heuristic and LPO with strict precedence.

[1] The criterion was independently obtained by Thiemann *et al.* [11].



Individual Data

TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status
3.01	0.03	3.05b	0.06	3.07	0.02	3.10	(1)	3.15	0.03	3.19	0.08	3.24	0.03	3.29	0.02
3.02	0.03	3.05c	0.08	3.08a	0.04	3.11	0.11	3.16	0.03	3.20	0.04	3.25	0.04	3.30	0.02
3.03	0.06	3.06a	0.75	3.08b	0.04	3.12	0.06	3.17a	0.13	3.21	*0.03*	3.26	0.03	3.31	0.02
3.04	0.05	3.06b	0.39	3.08c	0.06	3.13	(1)	3.17b	0.25	3.22	*0.07*	3.27	0.02	3.32	0.03
3.05a	0.06	3.06c	(1)	3.09	0.07	3.14	0.04	3.18	0.04	3.23	0.02	3.28	0.03	3.33	0.02

TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status
3.34	0.02	3.39	*0.05*	3.44	0.02	3.49	0.06	3.53b	0.02	3.57	0.15	4.23	0.04	4.28	*0.04*
3.35	0.03	3.40	*0.09*	3.45	0.03	3.50	0.03	3.53c	0.03	4.20a	0.03	4.24	0.03	4.29	(1)
3.36	0.15	3.41	*0.02*	3.46	*0.02*	3.51	0.02	3.54	0.04	4.20b	0.02	4.25	*0.02*	4.30a	*0.89*
3.37	0.02	3.42	*0.04*	3.47	0.03	3.52	0.03	3.55	0.13	4.21	0.02	4.26	*0.54*	4.30b	*0.04*
3.38	0.91	3.43	*0.04*	3.48	(1)	3.53a	0.10	3.56	0.02	4.22	0.02	4.27	*0.03*	4.30c	*0.87*

TRS	status	TRS	status
4.30d	*0.93*	4.35	(1)
4.31	*0.09*	4.36	(1)
4.32	0.02	4.37a	0.01
4.33	0.05	4.37b	0.02
4.34	0.19		

Summary

success	failure	timeout	total
65 (0.08)	17 (0.22)	7 (1)	89 (15.93)

Fig. 2. Output produced by TTT.

4. If the constraints could not be solved in step 3, we try linear polynomials with coefficients from $\{0, 1\}$.

If only part of an SCC could be handled, we subject the resulting new SCCs recursively to the same algorithm.

The effectiveness of the automatic strategy can be seen from the data presented in Figure 2, which were obtained by running TTT in fully automatic mode on the 89 terminating TRSs (66 in Section 3 and 23 in Section 4) of [2]. An explanation of the data is given in the next section.

4 A Collection of Rewrite Systems as Input

In addition to inputting a single TRS by typing the rules into the upper left text area or by uploading a file via the browse button, the user can upload a zip archive. All files ending in `.trs` are extracted from the archive and the termination prover runs on each of these files in turn. The results are collected and presented in two tables. The first table lists for each

TRS the execution time in seconds together with the status: **bold green** indicates success, *red italics* indicates failure, and gray indicates timeout. By clicking **green** (*red*) entries the user can view the termination proof (attempt) in HTML or high-quality Postscript format. The second table gives the number of successes and failures, both with the average time spent on each TRS, the number of timeouts, and the total number of TRSs extracted from the zip archive together with the total execution time. Figure 2 shows the two tables for the 66 TRSs in Section 3 of [2]. Here we used T⊤T's fully automatic mode with a timeout of 1 second (for each TRS). The experiment was performed on a PC equipped with a 2.20 GHz Mobile Intel Pentium 4 Processor - M and 512 MB of memory.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, 2001.
3. N. Dershowitz. Termination dependencies. In *Proceedings of the 6th International Workshop on Termination*, Technical Report DSIC-II/15/03, Universidad Politécnica de Valencia, pages 27–30, 2003.
4. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2850 of *Lecture Notes in Artificial Intelligence*, pages 165–179, 2003.
5. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing dependency pairs. Technical Report AIB-2003-08, RWTH Aachen, Germany, 2003.
6. N. Hirokawa and A. Middeldorp. Approximating dependency graphs without using tree automata techniques. In *Proceedings of the 6th International Workshop on Termination*, Technical Report DSIC-II/15/03, Universidad Politécnica de Valencia, pages 9–11, 2003.
7. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. In *Proceedings of the 19th International Conference on Automated Deduction*, volume 2741 of *Lecture Notes in Artificial Intelligence*, pages 32–46, 2003. Extended version submitted for publication.
8. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 311–320, 2003.
9. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, 2004. To appear.
10. E. Ohlebusch. Hierarchical termination revisited. *Information Processing Letters*, 84(4):207–214, 2002.
11. R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, 2004. To appear.
12. X. Urbain. Modular & incremental automated termination proofs. *Journal of Automated Reasoning*, 2004. To appear.

MU-TERM: A Tool for Proving Termination of Rewriting with Replacement Restrictions*

Salvador Lucas[1]

DSIC, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain.
slucas@dsic.upv.es

Abstract. This paper describes MU-TERM, a tool which can be used to automatically prove termination of computational restrictions of rewriting such as context-sensitive rewriting and lazy rewriting. The tool can also be used to prove termination of rewriting. In this sense, MU-TERM provides the first implementation of reduction orderings based on polynomial interpretations over the rational numbers.

1 Introduction

Restrictions of rewriting can eventually *achieve* termination of rewriting computations by pruning all infinite rewrite sequences issued from every term. However, such kind of improvements can be difficult to prove. Context-sensitive rewriting (*CSR* [6]) is a restriction of rewriting which is useful for describing semantic aspects of programming languages (e.g., Maude, OBJ2, OBJ3, or CafeOBJ) and analyzing termination of the corresponding programs. In *CSR*, a *replacement map* μ discriminates, for each symbol of the signature, the argument positions $\mu(f)$ on which replacements are allowed. Although several methods have been developed for proving termination of *CSR*, no tool for doing it has been reported to date. Our tool, MU-TERM, is intended to fill this gap. MU-TERM is written in Haskell[1] and makes use of the graphical library wxHaskell[2]. The MU-TERM system is available at

`http://www.dsic.upv.es/~slucas/csr/termination/muterm`

There are two main approaches to prove termination of *CSR*: *direct proofs* use adapted versions of simplification orderings such as RPOs and polynomial orderings to compare the left- and right-hand sides of the rules [4]; and *transformations* which convert the problem of proving termination of *CSR* into a proof of termination of rewriting [5]. Our tool implements both approaches. The modular analysis of termination of *CSR* described in [3] has also been implemented. We have used MU-TERM for developing the experiments reported in

`http://www.dsic.upv.es/~slucas/csr/termination/examples`

where we collect almost all published examples of TRSs which can be proved μ-terminating for concrete replacement maps μ.

The tool can also be used for proving *termination* of rewriting. Polynomials over the rationals [8] are used to generate appropriate reduction orderings (nowadays MU-TERM is the only tool which uses such kind of polynomials). On the other hand, μ-reduction orderings [10] based on such polynomial interpretations are also used together with the dependency pairs approach [1] to prove termination of rewriting.

* Work partially supported by MCyT project TIC2001-2705-C03-01, MCyT Acción Integrada HU 2003-0003 and AVCyT grant GR03/025.

[1] See `http://haskell.org/`.

[2] See `http://wxhaskell.sourceforge.net`.

2 Interface and functionality

The tool has a quite intuitive graphical user interface. TRSs can be loaded from files containing the rules in the 'simple format' l `->` r, where l and r are terms in the usual prefix syntax. The system is also able to deal with modules following a subset of the full OBJ / Maude grammar. The advantage is that, in contrast to the simple format, we are able to directly specify the *replacement map* by means of the OBJ / Maude *strategy annotations*. For instance, the following module

```
obj ExNatsOddsPairs is
  sort S .
  op 0 : -> S .
  op s : S -> S .
  op nil : -> S .
  op cons : S S -> S [strat (1 0)] .
  op nats : -> S .
  op pairs : -> S .
  op odds : -> S .
  op incr : S -> S .
  op tail : S -> S .
  vars X XS : S .
  eq nats = cons(0,incr(nats)) .
  eq pairs = cons(0,incr(odds)) .
  eq odds = incr(pairs) .
  eq incr(cons(X,XS)) = cons(s(X),incr(XS)) .
  eq tail(cons(X,XS)) = XS .
endo
```

describes the generation of some infinite lists of natural numbers. The strategy annotation (1 0) for `cons` is interpreted by MU-TERM as follows: $\mu(\texttt{cons}) = \{1\}$. The very recent TPDB (Termination Problems Data Base) format[3] can also be used to fully specify a TRS together with a replacement map.

TRSs are introduced in the system from text files via menu `File`; after successfully reading the file, the TRS becomes the *current TRS*. MU-TERM uses the *current TRS* to perform most actions selected by the user: prove termination, transform, etc.

Panel Termination of CSR (direct proof). The tool implements the techniques described in [8]. A proof of μ-termination of a TRS is transformed into the problem of solving a set of constraints over the coefficients of a polynomial interpretation for the symbols of the TRS. An interesting feature of our technique is that we generate polynomial interpretations with *rational* coefficients. We use C*i*ME [2] as an *external* tool to solve the constraints generated by the system. If C*i*ME is directly available (on the path of the OS), then the constraints are automatically generated and sent to C*i*ME; the answer is automatically received and processed to show the corresponding polynomial interpretation. For instance, for `ExNatsOddsPairs` we get:

```
[0] = 0        [nats] = 1     [cons](X1,X2) = X1 + 1/5.X2
[s](X) = X     [pairs] = 1    [incr](X) = X + 1
[nil] = 0      [odds] = 3     [tail](X) = 5.X + 1
```

which proves termination of *CSR* for the system. In this case, the use of rational numbers below one (e.g., $\frac{1}{5}$ in the polynomial interpreting `cons`) plays a crucial role for achieving the proof of termination (see [8, Section 4] for a deeper discussion in this respect). If the

[3] See `http://www.lri.fr/~marche/wst2004-competition/format.html`.

modular proofs are activated, then MU-TERM computes a maximal *safe* decomposition of the current system (according to the results in [3]) and separately proves termination of each module.

Panel Transformations. This option permits to apply different transformations for proving termination of *CSR* (see [5] for an overview). Each of them transforms the *current* TRS which remains unchanged (the transformed system is added to the internal list of TRSs). Again, if the *modular proofs* are activated, then MU-TERM uses the computed maximal decomposition of the current system to separately apply the transformations. Eventually, the transformed systems will also be separately proved terminating.

We also include a transformation from CS-TRSs into CS-TRSs (rather than TRSs) that permits to prove termination of *lazy rewriting* as termination of *CSR* (see [7]). However, since the results in [3] only concern *CSR*, no modular treatment is given with this transformation.

Panel Termination of rewriting. Term rewriting is a particular case of *CSR* where the replacement map $\mu_\top(f) = \{1, \ldots, ar(f)\}$, for all $f \in \mathcal{F}$ is used. Thus, polynomial orderings described above can also be used in proofs of termination of rewriting. On the other hand, Arts and Giesl discuss the use of weakly monotonic and non-monotonic orderings for proving termination of TRSs in combination with the dependency pairs approach [1]. We have implemented the use of polynomial interpretations over the rationals to generate such orderings and we use them together with the dependency pairs approach for proving termination of rewriting (see [9] for further details). In particular, we can use this technique, for instance, to prove termination of the TRSs which are obtained from the previous transformations.

References

1. T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs *Theoretical Computer Science*, 236:133-178, 2000.
2. E. Contejean and C. Marché. C*i*ME: Completion Modulo *E*. In *Proc. of RTA'96*, LNCS 1103:416-419, Springer-Verlag, Berlin, 1996.
3. B. Gramlich and S. Lucas. Modular termination of context-sensitive rewriting. In *Proc. of PPDP'02*, pages 50-61, ACM Press, New York, 2002.
4. B. Gramlich and S. Lucas. Simple termination of context-sensitive rewriting. In *Proc. of RULE'02*, pages 29-41, ACM Press, New York, 2002.
5. J. Giesl and A. Middeldorp. Transformation Techniques for Context-Sensitive Rewrite Systems. *Journal of Functional Programming*, to appear, 2004.
6. S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):293-343, 2002.
7. S. Lucas. Lazy Rewriting and Context-Sensitive Rewriting. Electronic Notes in Theoretical Computer Science, volume 64. Elsevier Sciences, 2002.
8. S. Lucas. Polynomials for proving termination of context-sensitive rewriting. In *Proc. of FOSSACS'04*, LNCS 2987:318-332, Springer-Verlag, Berlin, 2004.
9. S. Lucas. Polynomials over the rationals in proofs of termination. In *Proc. of WST'04*, this volume, 2004.
10. H. Zantema. Termination of Context-Sensitive Rewriting. In *Proc. of RTA'97*, LNCS 1232:172-186, Springer-Verlag, Berlin, 1997.

Aachener Informatik-Berichte

This is a list of recent technical reports. To obtain copies of technical reports please consult `http://aib.informatik.rwth-aachen.de/` **or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email:** `biblio@informatik.rwth-aachen.de`

95-11 *	M. Staudt / K. von Thadden: Subsumption Checking in Knowledge Bases
95-12 *	G.V. Zemanek / H.W. Nissen / H. Hubert / M. Jarke: Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modeling Technology
95-13 *	M. Staudt / M. Jarke: Incremental Maintenance of Externally Materialized Views
95-14 *	P. Peters / P. Szczurko / M. Jeusfeld: Business Process Oriented Information Management: Conceptual Models at Work
95-15 *	S. Rams / M. Jarke: Proceedings of the Fifth Annual Workshop on Information Technologies & Systems
95-16 *	W. Hans / St. Winkler / F. Sáenz: Distributed Execution in Functional Logic Programming
96-1 *	Jahresbericht 1995
96-2	M. Hanus / Chr. Prehofer: Higher-Order Narrowing with Definitional Trees
96-3 *	W. Scheufele / G. Moerkotte: Optimal Ordering of Selections and Joins in Acyclic Queries with Expensive Predicates
96-4	K. Pohl: PRO-ART: Enabling Requirements Pre-Traceability
96-5	K. Pohl: Requirements Engineering: An Overview
96-6 *	M. Jarke / W. Marquardt: Design and Evaluation of Computer–Aided Process Modelling Tools
96-7	O. Chitil: The ς-Semantics: A Comprehensive Semantics for Functional Programs
96-8 *	S. Sripada: On Entropy and the Limitations of the Second Law of Thermodynamics
96-9	M. Hanus (Ed.): Proceedings of the Poster Session of ALP'96 — Fifth International Conference on Algebraic and Logic Programming
96-10	R. Conradi / B. Westfechtel: Version Models for Software Configuration Management
96-11 *	C. Weise / D. Lenzkes: A Fast Decision Algorithm for Timed Refinement
96-12 *	R. Dömges / K. Pohl / M. Jarke / B. Lohmann / W. Marquardt: PRO-ART/CE* — An Environment for Managing the Evolution of Chemical Process Simulation Models
96-13 *	K. Pohl / R. Klamma / K. Weidenhaupt / R. Dömges / P. Haumer / M. Jarke: A Framework for Process-Integrated Tools
96-14 *	R. Gallersdörfer / K. Klabunde / A. Stolz / M. Eßmajor: INDIA — Intelligent Networks as a Data Intensive Application, Final Project Report, June 1996
96-15 *	H. Schimpe / M. Staudt: VAREX: An Environment for Validating and Refining Rule Bases

96-16 * M. Jarke / M. Gebhardt, S. Jacobs, H. Nissen: Conflict Analysis Across Heterogeneous Viewpoints: Formalization and Visualization

96-17 M. Jeusfeld / T. X. Bui: Decision Support Components on the Internet

96-18 M. Jeusfeld / M. Papazoglou: Information Brokering: Design, Search and Transformation

96-19 * P. Peters / M. Jarke: Simulating the impact of information flows in networked organizations

96-20 M. Jarke / P. Peters / M. Jeusfeld: Model-driven planning and design of cooperative information systems

96-21 * G. de Michelis / E. Dubois / M. Jarke / F. Matthes / J. Mylopoulos / K. Pohl / J. Schmidt / C. Woo / E. Yu: Cooperative information systems: a manifesto

96-22 * S. Jacobs / M. Gebhardt, S. Kethers, W. Rzasa: Filling HTML forms simultaneously: CoWeb architecture and functionality

96-23 * M. Gebhardt / S. Jacobs: Conflict Management in Design

97-01 Jahresbericht 1996

97-02 J. Faassen: Using full parallel Boltzmann Machines for Optimization

97-03 A. Winter / A. Schürr: Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems

97-04 M. Mohnen / S. Tobies: Implementing Context Patterns in the Glasgow Haskell Compiler

97-05 * S. Gruner: Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge

97-06 M. Nicola / M. Jarke: Design and Evaluation of Wireless Health Care Information Systems in Developing Countries

97-07 P. Hofstedt: Taskparallele Skelette für irregulär strukturierte Probleme in deklarativen Sprachen

97-08 D. Blostein / A. Schürr: Computing with Graphs and Graph Rewriting

97-09 C.-A. Krapp / B. Westfechtel: Feedback Handling in Dynamic Task Nets

97-10 M. Nicola / M. Jarke: Integrating Replication and Communication in Performance Models of Distributed Databases

97-13 M. Mohnen: Optimising the Memory Management of Higher-Order Functional Programs

97-14 R. Baumann: Client/Server Distribution in a Structure-Oriented Database Management System

97-15 G. H. Botorog: High-Level Parallel Programming and the Efficient Implementation of Numerical Algorithms

98-01 * Jahresbericht 1997

98-02 S. Gruner/ M. Nagel / A. Schürr: Fine-grained and Structure-oriented Integration Tools are Needed for Product Development Processes

98-03 S. Gruner: Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr

98-04 * O. Kubitz: Mobile Robots in Dynamic Environments

98-05 M. Leucker / St. Tobies: Truth — A Verification Platform for Distributed Systems

98-07 M. Arnold / M. Erdmann / M. Glinz / P. Haumer / R. Knoll / B. Paech / K. Pohl / J. Ryser / R. Studer / K. Weidenhaupt: Survey on the Scenario Use in Twelve Selected Industrial Projects

98-08 * H. Aust: Sprachverstehen und Dialogmodellierung in natürlichsprachlichen Informationssystemen

98-09 * Th. Lehmann: Geometrische Ausrichtung medizinischer Bilder am Beispiel intraoraler Radiographien

98-10 * M. Nicola / M. Jarke: Performance Modeling of Distributed and Replicated Databases

98-11 * A. Schleicher / B. Westfechtel / D. Jäger: Modeling Dynamic Software Processes in UML

98-12 * W. Appelt / M. Jarke: Interoperable Tools for Cooperation Support using the World Wide Web

98-13 K. Indermark: Semantik rekursiver Funktionsdefinitionen mit Striktheitsinformation

99-01 * Jahresbericht 1998

99-02 * F. Huch: Verifcation of Erlang Programs using Abstract Interpretation and Model Checking — Extended Version

99-03 * R. Gallersdörfer / M. Jarke / M. Nicola: The ADR Replication Manager

99-04 M. Alpuente / M. Hanus / S. Lucas / G. Vidal: Specialization of Functional Logic Programs Based on Needed Narrowing

99-07 Th. Wilke: CTL+ is exponentially more succinct than CTL

99-08 O. Matz: Dot-Depth and Monadic Quantifier Alternation over Pictures

2000-01 * Jahresbericht 1999

2000-02 Jens Vöge / Marcin Jurdzinski: A Discrete Strategy Improvement Algorithm for Solving Parity Games

2000-04 Andreas Becks / Stefan Sklorz / Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach

2000-05 Mareike Schoop: Cooperative Document Management

2000-06 Mareike Schoop / Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling

2000-07 * Markus Mohnen / Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages

2000-08 Thomas Arts / Thomas Noll: Verifying Generic Erlang Client-Server Implementations

2001-01 * Jahresbericht 2000

2001-02 Benedikt Bollig / Martin Leucker: Deciding LTL over Mazurkiewicz Traces

2001-03 Thierry Cachat: The power of one-letter rational languages

2001-04 Benedikt Bollig / Martin Leucker / Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus

2001-05 Benedikt Bollig / Martin Leucker / Thomas Noll: Regular MSC Languages

2001-06 Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic

2001-07 Martin Grohe / Stefan Wöhrle: An Existential Locality Theorem

2001-08 Mareike Schoop / James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling

2001-09 Thomas Arts / Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs

2001-10 Achim Blumensath: Axiomatising Tree-interpretable Structures

2001-11 Klaus Indermark / Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung

2002-01 * Jahresbericht 2001

2002-02 Jürgen Giesl / Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems

2002-03 Benedikt Bollig / Martin Leucker / Thomas Noll: Generalised Regular MSC Languages

2002-04 Jürgen Giesl / Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting

2002-05 Horst Lichter / Thomas von der Maßen / Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines

2002-06 Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata

2002-07 Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities

2002-08 Markus Mohnen: An Open Framework for Data-Flow Analysis in Java

2002-09 Markus Mohnen: Interfaces with Default Implementations in Java

2002-10 Martin Leucker: Logics for Mazurkiewicz traces

2002-11 Jürgen Giesl / Hans Zantema: Liveness in Rewriting

2003-01 * Jahresbericht 2002

2003-02 Jürgen Giesl / Ren Thiemann: Size-Change Termination for Term Rewriting

2003-03 Jürgen Giesl / Deepak Kapur: Deciding Inductive Validity of Equations

2003-04 Jürgen Giesl / René Thiemann / Peter Schneider-Kamp / Stephan Falke: Improving Dependency Pairs

2003-05 Christof Löding / Philipp Rohde: Solving the Sabotage Game is PSPACE-hard

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.