20th International Workshop on Termination

WST 2025, September 3–4, 2025, HTWK Leipzig, Germany

Edited by

Dieter Hofbauer

Johannes Waldmann

Editors

Dieter Hofbauer
ASW Saarland

Johannes Waldmann
HTWK Leipzig

Digital Object Identifier: `https://doi.org/10.33968/2025.48`

Contents

Preface

This report contains the proceedings of the 20th International Workshop on Termination (WST 2025), which was held in Leipzig, Germany, during September 3–4, co-located with the 14th International Workshop on Confluence (IWC 2025), September 2–3.

The Workshop on Termination traditionally brings together, in an informal setting, researchers interested in all aspects of termination, whether this interest be practical or theoretical, primary or derived. The workshop also provides a ground for cross-fertilization of ideas from the different communities interested in termination (e.g., working on computational mechanisms, programming languages, software engineering, constraint solving, etc.). The friendly atmosphere enables fruitful exchanges leading to joint research and subsequent publications. Previous workshops were held in St. Andrews (1993), La Bresse (1995), Ede (1997), Dagstuhl (1999), Utrecht (2001), Valencia (2003), Aachen (2004), Seattle (2006), Paris (2007), Leipzig (2009), Edinburgh (2010), Obergurgl (2012), Bertinoro (2013), Vienna (2014), Obergurgl (2016), Oxford (2018), virtually (2021), Haifa (2022), and Obergurgl (2023).

The program included a joint invited talk for WST and IWC by Aart Middeldorp on *Termination and Confluence: Remembering Hans Zantema* and a report on the 2025 Termination Competition by Florian Frohn. This competition ran live during the workshop and the results are available at `https://termcomp.github.io/Y2025/`.

WST 2025 received 17 submissions. After light reviewing the program committee decided to accept 16 submissions, contained in these proceedings.

We would like to thank the program committee members and the external reviewers for their dedication and effort, and Kerstin Höcherl, Ines Rohrbach, Enrico Ruge and Katja Stumpf for the invaluable help in the organization.

Kassel, Leipzig, September 2025 Dieter Hofbauer, Johannes Waldmann

Organization

Program Committee

Carsten Fuhs	Birkbeck, University of London
Jürgen Giesl	RWTH Aachen University
Dieter Hofbauer (co-chair)	ASW Saarland
Pierre Lescanne	PLUME, ENS de Lyon
Salvador Lucas	DSIC & VRAIN, Universitat Politècnica de València
Aart Middeldorp	University of Innsbruck
Johannes Waldmann (co-chair)	HTWK Leipzig
Akihisa Yamada	AIST, Japan

External Reviewers

Nachum Dershowitz	Tel Aviv University
Benjamin Kaminski	Universität des Saarlandes

Local Organization

Johannes Waldmann	HTWK Leipzig

Termination Competition Organization

Florian Frohn	RWTH Aachen University

Termination and Confluence: Remembering Hans Zantema

Aart Middeldorp

University of Innsbruck, Austria
aart.middeldorp@uibk.ac.at

In this presentation I give an incomplete overview of the many contributions of Hans Zantema[1] to termination and confluence (including [3–6]). Several of these were presented at earlier workshops on termination[2] and confluence,[3] and I include a biased overview of the development of these workshops and associated competitions [1, 2].

References

[1] Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The Termination and Complexity Competition. *Proc. 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 11429 of *LNCS*, pages 156–166, 2019. doi: 10.1007/978-3-030-17502-3_10.

[2] Aart Middeldorp, Julian Nagele, and Kiraku Shintani. CoCo 2019: Report on the Eighth Confluence Competition. *International Journal on Software Tools for Technology Transfer*, 2021. doi: 10.1007/s10009-021-00620-4.

[3] Hans Zantema. Termination of Term Rewriting: Interpretation and Type Elimination. *Journal of Symbolic Computation*, 17:23–50, 1994. doi: 10.1006/jsco.1994.1003.

[4] Hans Zantema. Termination of Term Rewriting by Semantic Labelling. *Fundamenta Informaticae*, 24(1-2):89–105, 1995. doi: 10.3233/FI-1995-24124.

[5] Hans Zantema. Termination of String Rewriting Proved Automatically. *Journal of Automated Reasoning*, 34:105–139, 2005. doi: 10.1007/s10817-005-6545-0.

[6] Hans Zantema. Finding Small Counterexamples for Abstract Rewriting Properties. *Mathematical Structures in Computer Science*, 28:1485–1505, 2018. doi: 10.1017/S0960129518000221.

[1] https://hzantema.win.tue.nl/
[2] https://termination-portal.org/wiki/WST
[3] http://cl-informatik.uibk.ac.at/iwc/

Core matrix interpretations for proving termination of term rewrite systems

Ulysse Le Huitouze
University of Rennes, France

René Thiemann
University of Innsbruck, Austria

Abstract

Matrix interpretations are powerful techniques for proving termination of term rewrite systems. Among them, the original paper that introduced the matrix interpretation technique, originally aimed at string rewriting, also described sets of matrices that inherently provide a well-founded relation and its required monotonic properties, simplifying proofs. To the extent of our knowledge, these special sets of matrices have not yet explicitly been used in a term rewriting setting. We report on the generalisation of these sets of matrices to a term rewriting setting. Several results have been formalised in Isabelle/HOL and are integrated in the CeTA certifier.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Logic and verification

Keywords and phrases Term rewriting, core matrix interpretations, formalisation, Isabelle/HOL

Acknowledgements We want to thank Johannes Waldmann for providing us with a version of Matchbox that produces core matrix interpretation proofs, and we thank the WST reviewers for their helpful comments.

1 Introduction

In this paper we present the core matrix interpretations termination technique, originally designed for string rewrite systems (SRSs) [6] for integer matrices.

▶ **Example 1.** Consider the following SRS (TPDB Waldmann19/SRS_Standard/random-246).

$$baaa \to bbaa \qquad abbb \to bbba \qquad abbb \to bbaa$$

During termCOMP 2024, the tool Multum-non-Multa of Hofbauer [5] generated the following termination proof by a core matrix interpretation.

$$\alpha(a) = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \qquad \alpha(b) = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

This proof could not be certified by CeTA [10] in year 2024.

We generalise the core matrix interpretation method to ordered rings so that in particular it can be used for matrices over rational numbers using δ-orders, similarly to [7, 8]. We further generalise the result from SRSs to term rewrite systems (TRSs). Both generalisations have been formalised using Isabelle/HOL [9] and they are fully integrated into the formalised rewriting framework IsaFoR/CeTA as of version 3.6. Consequently, proofs such as the one in Example 1 are now accepted by CeTA.

In addition to the aforementioned generalisations, during the formalisation process, a proof in [6] was found to be flawed. Whilst not required to deduce the end results of the termination technique, we provide a reconstructed proof nevertheless.

Our conditions on matrix interpretations are incomparable to other variants of monotone matrix interpretations from the literature [4, 3, 8]. Still, we show that every matrix interpretation of [4, 8] can be transformed into a core matrix interpretation. Moreover, the integration of core matrix interpretations has already successfully been tested: the Matchbox tool of Waldmann [11] generates certificates for termination proofs using core matrix interpretations, and all proofs are certified with CeTA version 3.6.

2 Preliminaries

We assume familiarity with term rewriting [1] and string rewriting and recall important notions and notations. $\mathcal{T}(\mathcal{F},\mathcal{V})$ will be the set of first-order terms that can be constructed from function symbols in $\mathcal{F}$ and variables in $\mathcal{V}$. Then, a term rewriting system (abbreviated TRS) is a binary relation in $\mathcal{T}(\mathcal{F},\mathcal{V}) \times \mathcal{T}(\mathcal{F},\mathcal{V})$. Given a rule in a TRS $\mathcal{R}$ of the form $\ell \to r$, a rewriting operation, denoted $\to_{\mathcal{R}}$, using such a rule consists of rewriting $C[\ell\sigma]$ to $C[r\sigma]$, where C is a term with exactly one special variable $\Box$ that will be replaced with $\ell\sigma$ (resp. $r\sigma$), and where $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F},\mathcal{V})$ is a substitution. A string rewriting system (abbreviated SRS) is a binary relation in $\Sigma^* \times \Sigma^*$ given an alphabet Σ. Rewriting using a rule $\ell \to r$ in an SRS $\mathcal{R}$ is denoted by $\to_{\mathcal{R}}$ and consists in rewriting $x\ell y$ to xry for any $x, y \in \Sigma^*$. Termination of a TRS/SRS $\mathcal{R}$ is denoted by $SN(\to_{\mathcal{R}})$ and refers to the non-existence of an infinite sequence of rewrite operations. Termination of a TRS (resp. SRS) $\mathcal{R}$ *relative* to a TRS (resp. SRS) $\mathcal{S}$ is the property $SN(\to_{\mathcal{S}}{}^* \circ \to_{\mathcal{R}} \circ \to_{\mathcal{S}}{}^*)$ and it is abbreviated by $SN(\mathcal{R}/\mathcal{S})$.

Throughout this paper, only square matrices of size n (often left implicit) are considered. Then, we need a few notations: The zero matrix shall be denoted by $\mathbf{0}$, whilst the identity matrix shall be denoted by $\mathbf{1}$; The usual notation $A_{i,j}$ will be used to denote the component of A at the i-th row and j-th column; The non-strict comparison on matrices $\geq$ shall correspond to a weak decrease in coefficients component-wise; On the other hand, the strict comparison on matrices $>$ shall correspond to a weak decrease component-wise *in addition* to a strict decrease component-wise for at least one coefficient in the matrices; Then, N shall refer to the set of non-negative matrices, i.e. $\{A \mid A \geq \mathbf{0}\}$, while P shall refer to the set of non-negative and non-null matrices, i.e. $\{A \mid A > \mathbf{0}\}$; Moreover, given two sets of matrices S and T, the set ST shall denote the set $\{A_1 \times A_2 \mid A_1 \in S, A_2 \in T\}$; Following that, the set T^k shall refer to $TT\ldots T$ k-times, T^* shall denote the set $\bigcup_k T^k$ and T^0 shall be $\{\mathbf{1}\}$; Finally, the notation ST shall be extended to individual matrices, i.e. given a matrix d, the notation dT shall denote $\{d \times A \mid A \in T\}$.

3 Core matrix interpretations

In this section, we present the "core" matrix interpretation technique in an SRS setting, as it is described in [6], before moving to its generalisation from integers to further ordered rings and before moving to a TRS setting. Roughly speaking, the termination technique consists in turning the string rewrite setting into a matrix setting, where every possible rewrite operation would be mapped to a well-founded relation on integer matrices, such as the strict comparison on N, ensuring termination.

Concretely, if we consider the SRS $\mathcal{R}$, if we call α the interpretation of strings (i.e. the map from strings to matrices), if $\alpha(s) \in N$ holds for any string s, and if $\alpha(s) > \alpha(t)$ holds for any strings $s \to_{\mathcal{R}} t$, then $\mathcal{R}$ is strongly-normalising. To achieve relative termination, e.g. $SN(\to_{\mathcal{S}}{}^* \circ \to_{\mathcal{R}} \circ \to_{\mathcal{S}}{}^*)$, in addition to the previous criteria, satisfying the non-strict comparison on N for $\mathcal{S}$-rewrite operations is sufficient, i.e. $\alpha(s) \geq \alpha(t)$ for any strings $s \to_{\mathcal{S}} t$. This criterion is sufficient since $\geq^* \circ > \circ \geq^* \subseteq >$, meaning $\to_{\mathcal{S}}{}^* \circ \to_{\mathcal{R}} \circ \to_{\mathcal{S}}{}^*$ would map to $>$ which terminates.

Since our goal is first and foremost to implement this termination technique into IsaFoR/CeTA to be able to certify proofs by "core" matrix interpretations, the process of verifying the required criteria (namely $\forall s, t,\ s \to_{\mathcal{R}} t \implies \alpha(s) > \alpha(t); s \to_{\mathcal{S}} t \implies \alpha(s) \geq \alpha(t)$ in our case; let us call this criteria **Property 1**) *must* be computable. However, this is obviously not the case for **Property 1**, due to the quantification over all possible strings. Thus, the "core" matrix interpretation approach aims to reformulate this criteria in such a way that it becomes computable. Since s has been rewritten to t in **Property 1**, s can be split into a left context, the place where a rewrite rule was applied, and a right context, making **Property 1** equivalent to

$$\left.\begin{array}{llll} \forall x, y, \ell, r,\ (\ell \to r) & \in \mathcal{R} & \implies & \alpha(x\ell y) > \alpha(xry) \\ \forall x, y, \ell, r,\ (\ell \to r) & \in \mathcal{S} & \implies & \alpha(x\ell y) \geq \alpha(xry) \end{array}\right\} \textbf{Property 2}$$

While **Property 2** is easily lifted to TRSs by simply changing the domain and the definitions of left and right context, to further reformulate it we need to take into account SRS-specific properties. In particular, in [6], the interpretation of the string ab is defined as $\alpha(a) \times \alpha(b)$ considering $\{a, b\} \subseteq \Sigma$. Thus, **Property 2** can be reformulated as

$$\left.\begin{array}{llll} \forall x, y, \ell, r,\ (\ell \to r) & \in \mathcal{R} & \implies & \alpha(x)(\alpha(\ell) - \alpha(r))\alpha(y) > \mathbf{0} \\ \forall x, y, \ell, r,\ (\ell \to r) & \in \mathcal{S} & \implies & \alpha(x)(\alpha(\ell) - \alpha(r))\alpha(y) \geq \mathbf{0} \end{array}\right\} \textbf{Property 3}$$

Now, assume (1) $\alpha(a) \in S$ for any letter $a \in \Sigma$, (2) $\alpha(\ell) - \alpha(r) \in T$ for all $l \to r \in \mathcal{R}$, and (3) $S^*TS^* \subseteq P$ for some sets S and T, then the first line of **Property 3** can be deduced. Furthermore, if (4) $\alpha(\ell) - \alpha(r) \in U$ for all $\ell \to r \in \mathcal{S}$, and (5) $S^*US^* \subseteq N$ for some other set U, then the second line can be deduced. The latter property is usually given for free if we choose $S \subseteq N$ and $U \subseteq N$, since N is closed under multiplication.

$$\left.\begin{array}{rl} \mathit{range}(\alpha) \subseteq S;\ S \subseteq N; \forall(\ell \to r) \in \mathcal{R}, & \alpha(l) - \alpha(r) \in T;\ S^*TS^* \subseteq P \\ \forall(\ell \to r) \in \mathcal{S}, & \alpha(l) - \alpha(r) \in N \end{array}\right\} \textbf{Property 4}$$

In the original paper, the first line of **Property 4** is presented using the definition of "core of a set of matrices", namely $\mathit{core}(\mathcal{A}) = \{d \in N \mid \mathcal{A}^* d \mathcal{A}^* \subseteq P\}$. Then, if for some $\mathcal{A}$, $\mathit{range}(\alpha) \subseteq \mathcal{A}$ and $\forall(\ell \to r) \in \mathcal{R},\ \alpha(\ell) - \alpha(r) \in \mathit{core}(\mathcal{A})$ hold, $\mathcal{R}$ is strongly normalising. If additionally $\mathcal{A} \subseteq N$ and $\forall(\ell \to r) \in \mathcal{S},\ \alpha(\ell) - \alpha(r) \in N$ hold, $\mathcal{R}$ is terminating *relative* to $\mathcal{S}$.

We now introduce concrete sets of matrices, presented in the original paper, which exhibit the desired properties stated above. Here, I is a subset of matrix-indices. Note that checking whether a matrix is in such a set is computable.

$$E_I = \{d \in N \mid \forall i \in I,\ d_{i,i} > 0\} \qquad M_I = \{d \in N \mid \forall i \in I, \exists j \in I,\ d_{i,j} > 0\}$$
$$P_I = \{d \in N \mid \exists i, j \in I,\ d_{i,j} > 0\}$$

▶ **Lemma 2** ([6, Lemma 4]). $P_I = \mathit{core}(E_I), \quad M_I = \mathit{core}(M_I)$

▶ **Corollary 3.** *Let $\mathcal{R}$ and $\mathcal{S}$ be SRSs over signature Σ and let $\alpha : \Sigma \to \mathbb{Z}^{n \times n}$ be a matrix interpretation and $\emptyset \subset I \subseteq \{1, \ldots, n\}$. Let $\mathcal{S}$ satisfy $\alpha(\ell) - \alpha(r) \in N$ for all $\ell \to r \in \mathcal{S}$. Then relative termination $SN(\mathcal{R}/\mathcal{S})$ is ensured if one of the following conditions is satisfied.*

- *$\mathit{range}(\alpha) \subseteq E_I$ and $\alpha(\ell) - \alpha(r) \in P_I$ for all $\ell \to r \in \mathcal{R}$, or*
- *$\mathit{range}(\alpha) \subseteq M_I$ and $\alpha(\ell) - \alpha(r) \in M_I$ for all $\ell \to r \in \mathcal{R}$.*

Note that only the inclusions from left to right in Lemma 2 (e.g. $P_I \subseteq \mathit{core}(E_I)$) are required to get Corollary 3. Nevertheless, the original proofs of the inclusions from right to left were found to be flawed during our Isabelle formalisation. We present an instance contradicting the original proof for each inclusion before providing a reconstructed proof.

1. Original proof of $core(E_I) \subseteq P_I$ (directly quoted from [6]): For showing the inverse inclusion $P/E_I \subseteq P_I$, assume the existence of a matrix d with $dE_I \subseteq P$ and $d \notin P_I$, so $d_{i,j} = 0$ for $i, j \in I$. Define $e \in E_I$ by $e_{i,j} = 1$ for $i = j \in I$ and $e_{i,j} = 0$ otherwise. Then $de = \mathbf{0} \notin P$, a contradiction.

   Instance: Consider the set of indices $I := \{1\}$ and the matrix $d := \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$. Then, $d \notin P_I$ holds and finally, observe that $de = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \in P$.
2. Similarly for $core(M_I) \subseteq M_I$: $I := \{1\}$ and $d := \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$.
3. Reconstructed proofs: For both inclusions, the only change needed is that, instead of proving $de = \mathbf{0}$ (resp. $md = \mathbf{0}$) which is not true, we prove that $ede = \mathbf{0}$ (resp. $mdm = \mathbf{0}$) which is a contradiction with respect to the definition of *core*.

3.1 Arbitrary ordered rings

We now consider a generalisation of using integer matrices to allow matrices over other ordered rings with a strongly normalising order $\succ$ on the set of non-negative numbers. In particular we consider δ-orders [7, 8]. Here, some parameter $\delta > 0$ is fixed, and $\succ_\delta$ is defined as $x \succ_\delta y$ iff $x - y \geq \delta$. By choosing $\delta = 1$ and the integers, we obtain the original setting of [6], but we may also choose a different value of δ and the ring of rational or real numbers.

The definitions of E_I, P_I and M_I are adjusted as follows to the new setting with δ-orders, where we also add the new set $L_{I,\delta}$: for the integers with $\delta = 1$ we have $M_I = L_{I,\delta}$, so in the original setting no differentiation between these sets is required, but in the general case we gain from a distinction between these sets.

$$\begin{aligned} E_I &= \{d \in N \mid \forall i \in I,\ d_{i,i} \geq 1\} & M_I &= \{d \in N \mid \forall i \in I, \exists j \in I,\ d_{i,j} \geq 1\} \\ P_{I,\delta} &= \{d \in N \mid \exists i, j \in I,\ d_{i,j} \succ_\delta 0\} & L_{I,\delta} &= \{d \in N \mid \forall i \in I, \exists j \in I,\ d_{i,j} \succ_\delta 0\} \end{aligned}$$

The results of [6] generalise as follows: we define P_δ as the set of non-negative matrices d for which at least one entry $d_{i,j}$ satisfies $d_{i,j} \succ_\delta 0$; the definition of the core is generalised to $core_\delta(\mathcal{A}) = \{d \in N \mid \mathcal{A}^* d \mathcal{A}^* \subseteq P_\delta\}$.

We obtain the important direction of Lemma 2 in a more general setting.

▶ **Lemma 4.** $P_{I,\delta} \subseteq core_\delta(E_I)$, $\quad L_{I,\delta} \subseteq core_\delta(M_I)$.

The proof of the lemma is similar to the integer setting. Here, we only mention why it was required to change the definitions of E_I and M_I. Multiplication with some number $d_{i,j}$ is monotone for δ-orders: if $x \succ_\delta y$ and $d_{i,j} \geq 1$ then also $d_{i,j}x \succ_\delta d_{i,j}y$. But only requiring $d_{i,j} \succ_\delta 0$ instead of $d_{i,j} \geq 1$ at this point is not sufficient anymore to guarantee $d_{i,j}x \succ_\delta d_{i,j}y$.

▶ **Corollary 5.** *Let $\mathcal{R}$ and $\mathcal{S}$ be SRSs over signature Σ and let $\alpha : \Sigma \to D^{n \times n}$ be a matrix interpretation over domain $D \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$ and $\emptyset \subset I \subseteq \{1, \ldots, n\}$. Let $\delta \in D$ satisfy $0 < \delta$. Let $\mathcal{S}$ satisfy $\alpha(\ell) - \alpha(r) \in N$ for all $\ell \to r \in \mathcal{S}$. Then $SN(\mathcal{R}/\mathcal{S})$ is ensured if*

- *$range(\alpha) \subseteq E_I$ and $\alpha(\ell) - \alpha(r) \in P_{I,\delta}$ for all $\ell \to r \in \mathcal{R}$, or*
- *$range(\alpha) \subseteq M_I$ and $\alpha(\ell) - \alpha(r) \in L_{I,\delta}$ for all $\ell \to r \in \mathcal{R}$.*

3.2 Term rewrite setting

The main difference between matrix interpretations for SRSs and TRSs is that in the SRS version only matrix multiplication is used, whereas for TRSs we see both addition and multiplication. Given some n-ary function symbol f, the interpretation of f is of the form

$$\alpha(f)(x_1, \ldots, x_k) = f_0 + f_1 \cdot x_1 + \ldots + f_k \cdot x_k$$

where $f_1, \ldots, f_k$ are matrices, and f_0 is a vector (as in [4, 8]) or a matrix (as in [3]). We consider the "f_0 is a matrix" setting as it subsumes the vector setting: one can always enlarge the vector to a matrix by filling the remaining columns with zero entries. Hence, in this setting a matrix interpretation is a linear polynomial interpretation with matrices as domain.

Consequently, the interpretation of a term t leads to a linear polynomial $\alpha(t)$ with matrix coefficients, and orienting a rewrite step is similar to **Property 2**, where left-contexts are replaced by term contexts C and right-contexts are replaced by substitutions σ:

$$\forall C, \sigma,\ (\ell \to r) \in \mathcal{R} \implies \alpha(C[\ell\sigma]) - \alpha(C[r\sigma]) > \mathbf{0}$$

Since contexts C are just terms with a special variable $\Box$, the hole, we can rewrite $\alpha(C[\ell\sigma]) - \alpha(C[r\sigma])$ to $\alpha(C)\{\Box/\alpha(\ell\sigma) - \alpha(r\sigma)\}$, and by a substitution lemma the previous condition can be rewritten to a variant of **Property 3**.

$$\forall C, \vec{x},\ (\ell \to r) \in \mathcal{R} \implies \alpha(C)\{\Box/\alpha(\ell) - \alpha(r)\} > \mathbf{0}$$

where $\alpha(C)\{\Box/\alpha(\ell) - \alpha(r)\}$ is a linear polynomial over variables $\vec{x}$ that range over the domain of the matrix interpretation.

We now again want to use $core_\delta(\mathcal{A}) = \{d \in N \mid \mathcal{A}^* d \mathcal{A}^* \subseteq P_\delta\}$. In order to get rid of the context C in comparisons, the interpretation needs be chosen such that being in the core propagates via $\alpha(C)$ for all possible contexts C. This is done by demanding $f_1 \in \mathcal{A}, \ldots, f_k \in \mathcal{A}$ for each k-ary symbol f. Moreover, we identify the domain of the matrix interpretation with $\mathcal{A}$. Hence, in order to satisfy these conditions we further require the following two conditions: $f_0 \in N$ for every k-ary symbol f, and whenever f is a constant, then $f_0 \in \mathcal{A}$.

By enforcing these conditions a strict decrease of a rewrite step is now ensured by the orientation condition $\alpha(\ell) - \alpha(r) \in core_\delta(\mathcal{A})$. Note that $\alpha(\ell) - \alpha(r)$ is a linear polynomial $c_0 + c_1 x_1 + \ldots c_m x_m$ with matrix coefficients $c_0, \ldots, c_m$ over variables $x_1, \ldots, x_m$ that occur in ℓ and that range over $\mathcal{A}$. So formally, for some set of matrices M we write $\alpha(\ell) - \alpha(r) \in M$ as an abbreviation for $\forall x_1, \ldots, x_m \in \mathcal{A}.\ \alpha(\ell) - \alpha(r) \in M$, and we call $c_0, \ldots, c_m$ the coefficients of $\alpha(\ell) - \alpha(r)$. We next instantiate the abstract setting with $\mathcal{A}$ and $core_\delta(\mathcal{A})$ by E_I and $P_{I,\delta}$ (resp. M_I and $L_{I,\delta}$), i.e., the sets of matrices that are known from Section 3.1.

▶ **Theorem 6** (Core matrix interpretations for TRSs). *Let $\mathcal{R}$ and $\mathcal{S}$ be TRSs over signature $\mathcal{F}$ and let i be a matrix interpretation over domain $D \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$ of dimension n and let $\emptyset \subset I \subseteq \{1, \ldots, n\}$. Let every k-ary symbol f be interpreted as $\alpha(f)(x_1, \ldots, x_k) = f_0 + f_1 \cdot x_1 + \ldots + f_k \cdot x_k$. Let $\delta \in D$ satisfy $0 < \delta$. Relative termination $SN(\mathcal{R}/\mathcal{S})$ is ensured if all of the following conditions are satisfied.*

1. *$f_i \in E_I$ (resp. M_I) for all $1 \leq i \leq k$ and k-ary symbols $f \in \mathcal{F}$*
2. *$\alpha(f)(x_1, \ldots, x_k) \in E_I$ (resp. M_i) whenever all $x_1, \ldots, x_k \in E_I$ (resp. M_I); this condition is ensured by demanding—in combination with condition 1—that*
   - **a.** *$f_0 \in N$ for each $f \in \mathcal{F}$, and*
   - **b.** *$f_0 \in E_I$ (resp. M_I) for each constant $f \in \mathcal{F}$*
3. *$\alpha(\ell) - \alpha(r) \in N$ for $\ell \to r \in \mathcal{R} \cup \mathcal{S}$; this condition is ensured by demanding that each coefficient of $\alpha(\ell) - \alpha(r)$ is in N*

4. *$\alpha(\ell) - \alpha(r) \in P_{I,\delta}$ (resp. $L_{I,\delta}$) for each $\ell \to r \in \mathcal{R}$; this condition is ensured by demanding—in combination with condition 3—that some coefficient of $\alpha(\ell) - \alpha(r)$ is in $P_{I,\delta}$ (resp. $L_{I,\delta}$).*

Theorem 6 is the main result of this paper and it strictly generalises the SRS setting: if every unary symbol f is interpreted by $\alpha(f)(x_1) = f_1 \cdot x_1$ then Theorem 6 is equivalent to Corollary 5, and choosing $D = \mathbb{Z}$ and $\delta = 1$ we arrive at Corollary 3.

▶ Remark 7. If one does not want to prove relative termination directly, but instead requires a reduction pair—e.g., when using dependency pairs—then condition 1 can be weakened to $f_i \in N$.

The conditions in Theorem 6 are incomparable to the monotone matrix interpretations of [4, 3, 8]. Our version has the advantage in condition 4: we allow a strict decrease for any coefficient of the linear polynomial $\alpha(\ell) - \alpha(r)$ whereas the strict decrease must happen in the constant part of $\alpha(\ell) - \alpha(r)$ in [4, 3, 8]. On the other hand, condition 2b is a new requirement in Theorem 6 which is not present in [4, 3, 8]. Roughly speaking, we require that constants must be interpreted by positive matrices, in particular they cannot be interpreted by the zero matrix as in [4, 3, 8]. Note that condition 2b cannot be dropped from Theorem 6. Consider $\mathcal{R} = \{f(x) \to g(x)\}$ and $\mathcal{S} = \{g(a) \to f(a)\}$. Clearly, $SN(\mathcal{R}/\mathcal{S})$ does not hold, but one can find a matrix interpretation that satisfies all conditions except for condition 2b, namely: $D = \mathbb{Z}$, $n = 1$, $I = \{1\}$, $\delta = 1$, $\alpha(f)(x) = 2x$, $\alpha(g)(x) = x$, and $\alpha(a) = 0$.

In the remainder of this paper, we show how Theorem 6 can be strengthened even further, i.e., by using more relaxed sufficient criteria to ensure conditions 1–4. It will turn out, that with these improved conditions, Theorem 6 even subsumes the monotone matrix interpretations of [4, 8].

First of all, we can improve the criterion for condition 4 in the M_I setting:

▶ **Lemma 8.** *Condition 4 in Theorem 6 is satisfied in the M_I setting, if condition 3 is satisfied and $\forall i \in I.\ \exists c \in C.\ \exists j \in I.\ c_{ij} \succ_\delta 0$, where C is the set of coefficients of $\alpha(\ell) - \alpha(r)$.*

This new condition is more powerful than the one of Theorem 6, since there the sufficient criterion to ensure condition 4 is equivalent to $\exists c \in C.\ \forall i \in I.\ \exists j \in I.\ c_{ij} \succ_\delta 0$ where one cannot choose different coefficients c for each row i.

For the E_I setting, we further improve the sufficient criteria for conditions 2–4. Here we use a transformation similar to [2, Section 3.4] that is known for polynomial interpretations over integers. In our setting, we basically switch from the carrier E_I to N. To this end, we define 1_I to be the matrix which is always 0, except that $(1_I)_{i,i} = 1$ whenever $i \in I$. It is easy to see that $x \in E_I$ iff $x = 1_I + y$ for some $y \in N$. We now just substitute each variable x that ranges over E_I by a variable y that ranges over N.

▶ **Lemma 9.** 2. *$\forall x_1, \ldots, x_k \in E_I.\ \alpha(f)(x_1, \ldots, x_k) \in E_I$ iff $\forall y_1, \ldots, y_k \in N.\ \alpha(f)(1_I + y_1, \ldots, 1_I + y_k) \in E_I$.*

3. *$\alpha(\ell) - \alpha(r) \in N$ (with variables $x_1, \ldots, x_m$ ranging over E_i) iff $(\alpha(\ell) - \alpha(r))\{x_1/1_I + y_1, \ldots, x_m/1_I + y_m\} \in N$ (with variables $y_1, \ldots, y_m$ ranging over N.)*
4. *$\alpha(\ell) - \alpha(r) \in P_{I,\delta}$ (with variables $x_1, \ldots, x_m$ ranging over E_i) iff $(\alpha(\ell) - \alpha(r))\{x_1/1_I + y_1, \ldots, x_m/1_I + y_m\} \in P_{I,\delta}$ (with variables $y_1, \ldots, y_m$ ranging over N.)*

The advantage of the switch to N via Lemma 9 is that the criteria are equivalences, and the new universally quantified conditions over N can easily be decided as follows.

▶ **Lemma 10.** *Let p be a linear polynomial over variables $y_1, \ldots, y_k$.*

- *$\forall y_1, \dots, y_k \in N.\ p(y_1, \dots, y_k) \in N$ iff all coefficients of p are in N.*
- *$\forall y_1, \dots, y_k \in N.\ p(y_1, \dots, y_k) \in E_I$ iff $\forall y_1, \dots, y_k \in N.\ p(y_1, \dots, y_k) - 1_I \in N$.*
- *$\forall y_1, \dots, y_k \in N.\ p(y_1, \dots, y_k) \in P_{I,\delta}$ iff all coefficients of p are in N and the constant part of p is in $P_{I,\delta}$.*

▶ **Example 11.** The E_I interpretation $\alpha(f)(x) = 2x-1$ for $n = 1$ and $I = \{1\}$ is not accepted by the sufficient criterion for condition 2 in Theorem 6, but it is accepted via Lemmas 9 and 10, since $\alpha(f)(1 + y) = 2(1 + y) - 1 = 2y + 1$ is clearly in E_I when y ranges over N.

With the help of Lemmas 9 and 10 we are able to show that Theorem 6 fully subsumes matrix interpretations as they are defined by Endrullis et al. [4] and Neurauter et al. [8].

▶ **Theorem 12.** *If there is some relative termination proof via a matrix interpretation following [4, 8], then there also is a relative termination proof using Theorem 6.*

Proof. Given a strictly monotone interpretation in the style of [4, 8] with $\alpha'(f)(y_1, \dots, y_k) = f_0 + f_1 y_1 + \dots + f_k y_k$—where f_0 is interpreted as matrix—we convert it into the E_I interpretation α with $I = \{1\}$ and $\alpha(f)(x_1, \dots, x_k) := \alpha'(f)(x_1 - 1_I, \dots, x_k - 1_I) + 1_I$. We get the relationship $\alpha(t) = \alpha'(t) + 1_I$ for all ground terms t, and both interpretations define the same ordering $((\alpha(\ell) - \alpha(r))\{z_1/1_I + z_1, \dots, z_m/1_I + z_m\}) = \alpha'(\ell) - \alpha'(r))$. Moreover, all conditions of Theorem 6 are satisfied for α by using the criteria in Lemmas 9 and 10 whenever α' satisfies the criteria for a relative termination proof in [4, 8]. ◀

We briefly show that Lemma 9 is crucial for this subsumption result: given $\alpha'(f)(y) = 2y$, it is transformed into $\alpha(f)(x) = \alpha'(f)(x - 1) + 1 = 2(x - 1) + 1 = 2x - 1$, and accepting this interpretation α by Theorem 6 requires Lemma 9, cf. Example 11.

Note that the very same transformation can also be applied in the other direction, so core matrix interpretations can be turned into the ones of [4, 8] for the $E_{\{1\}}$ setting.

We do not know, whether similar transformations are possible for the M_I setting. Here, the complication arises that we cannot add or subtract a matrix to switch between M_I and N, since there is not a unique minimal element such as 1_I in the E_I case, but there are $|I|^{|I|}$ many of those: a matrix A is minimal in M_I if for every $i \in I$ there is exactly one $j \in I$ such that $A_{ij} = 1$, and all other entries are 0.

We leave it as future work, to include further sets than E_I and M_I into the formalization and into CeTA. Such an addition can be triggered if there is demand from the tool author side.

References

1 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

2 Evelyne Contejean, Claude Marché, Ana Paula Tomás, and Xavier Urbain. Mechanically proving termination using polynomial interpretations. *J. Autom. Reason.*, 34(4):325–363, 2005. URL: `https://doi.org/10.1007/s10817-005-9022-x`, `doi:10.1007/S10817-005-9022-X`.

3 Pierre Courtieu, Gladys Gbedo, and Olivier Pons. Improved matrix interpretation. In Jan van Leeuwen, Anca Muscholl, David Peleg, Jaroslav Pokorný, and Bernhard Rumpe, editors, *SOFSEM 2010: Theory and Practice of Computer Science, 36th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 23-29, 2010. Proceedings*, volume 5901 of *Lecture Notes in Computer Science*, pages 283–295. Springer, 2010. `doi:10.1007/978-3-642-11266-9\_24`.

4 Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. *J. Autom. Reason.*, 40(2-3):195–220, 2008. URL: `https://doi.org/10.1007/s10817-007-9087-9`, `doi:10.1007/S10817-007-9087-9`.

5 Dieter Hofbauer. MultumNonMulta at TermComp 2018. In Salvador Lucas, editor, *Proceedings of the 16th International Workshop on Termination, WST-18*, page 80, Oxford, U. K, 2018.

6 Dieter Hofbauer and Johannes Waldmann. Termination of string rewriting with matrix interpretations. In Frank Pfenning, editor, *Term Rewriting and Applications, 17th International Conference, RTA 2006, Seattle, WA, USA, August 12-14, 2006, Proceedings*, volume 4098 of *Lecture Notes in Computer Science*, pages 328–342. Springer, 2006. `doi:10.1007/11805618\_25`.

7 Salvador Lucas. Polynomials over the reals in proofs of termination: from theory to practice. *RAIRO Theor. Informatics Appl.*, 39(3):547–586, 2005. URL: `https://doi.org/10.1051/ita:2005029`, `doi:10.1051/ITA:2005029`.

8 Friedrich Neurauter and Aart Middeldorp. On the domain and dimension hierarchy of matrix interpretations. In Nikolaj S. Bjørner and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings*, volume 7180 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2012. `doi:10.1007/978-3-642-28717-6\_25`.

9 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002. `doi:10.1007/3-540-45949-9`.

10 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009. `doi:10.1007/978-3-642-03359-9\_31`.

11 Johannes Waldmann. Matchbox: A tool for match-bounded string rewriting. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 85–94. Springer, 2004. `doi:10.1007/978-3-540-25979-4\_6`.

Off with the head: termination provers and the word problem for 1-relation monoids

Reinis Cirpons
University of St Andrews, United Kingdom

James D. Mitchell
University of St Andrews, United Kingdom

Finn L. Smith
University of St Andrews, United Kingdom

Abstract

In this article we discuss the application of termination provers to the decidability of the word problem for 1-relation monoids. In particular, we describe how Adian's algorithm $\mathfrak{A}$ for a particular left cycle-free 2-generated 1-relation monoid M can be used to produce a string rewriting system whose termination implies that the word problem for M is decidable. Such monoids are among the only cases of 1-relation monoids where it is not known whether or not the word problem is decidable. Our findings show that this new class of SRS is not only theoretically significant, but is also challenging for existing termination provers.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics; Mathematics of computing → Combinatorics on words; Mathematics of computing → Mathematical software; Security and privacy → Logic and verification

Keywords and phrases termination of string rewriting, word problem in monoids, one-relation monoid

Funding *Reinis Cirpons*: This work was supported by the Engineering and Physical Sciences Research Council grant number EP/V520123/1

Acknowledgements We would like to thank Johannes Waldmann and Dieter Hofbauer for assistance with the verification of the more difficult SRS termination instances arising from our research, as well as their encouragement, which lead to this short article. We also thank our collaborators on our wider 1-relation monoid projects: Florent Hivert, Assia Mahboubi, and Guillaume Melquiond.

1 Introduction

The decidability of the word problem for 1-relation monoids (WP1M) has been one of the most prominent open problems in combinatorial algebra for at least the past 100 years.

If A is a non-empty set, then the set A^* consists of the finite (possibly empty) sequences of elements of A, we write ε for the empty sequence and $A^+ = A^* \setminus \{\varepsilon\}$. The set A is called an *alphabet*, its elements are *letters*, and elements of A^* are *words*. The set A^* is a monoid (called the *free monoid*) when endowed with the binary operation of juxtaposition. A *monoid presentation* is a pair $\langle A \mid R\rangle$ for some non-empty set A and some set $R \subseteq A^* \times A^*$. For example, $\langle a, b \mid baaba = a\rangle$ is a monoid presentation (omitting superfluous brackets). Formally, a presentation *defines* the quotient monoid of A^* by the least congruence containing R. Informally, we may think of such a monoid M as consisting of words in A where two words $u, v \in A^*$ represent the same element of M, written $u \leftrightarrow_R^* v$, if there is a sequence of relations in R transforming u to v. For example, in the presentation given above

$$baab \leftrightarrow_R^* ba(baaba)b = ba(baab)ab \leftrightarrow_R^* \cdots \leftrightarrow_R^* (ba)^k baab(ab)^k, \quad k \geq 0.$$

If A and R are both finite, then the monoid defined by $\langle A \mid R\rangle$ is *finitely presented.*

The *word problem* for a monoid M defined by a presentation $\langle A \mid R\rangle$ is the decision problem with input $u, v \in A^*$, that outputs "yes" if the words u and v represent the same element in M and "no" if they do not. The word problem for monoids is undecidable in general. The literature is replete with the search for minimal examples, in one sense or another, of finitely presented monoids with undecidable word problem. If "minimal" means "with the fewest possible relations", then the best example to date is that given in [7] of a 2-generator and 3-relation presentation defining a monoid with undecidable word problem. On the other hand, the word problem for groups defined by (group) presentations with 1 relation were shown to be decidable by Magnus [6]. Although intensively studied for many decades, the question of whether or not there exists a monoid defined by a 1- or 2-relation presentation with undecidable word problem is open.

Recently there has been something of a renaissance in interest in the decidability of the WP1M; see [9] and the references therein for more details. Despite, or perhaps precisely due to, the wide variety of results proving decidability of the WP1M for various subclasses of 1-relation monoid presentations, there is no simple answer to the seemingly obvious question:

> *what is the smallest known instance of a* 1*-relation monoid presentation where the decidability of its word problem is unknown?*

This question motivated the authors of the current paper to explore to what extent, and by what means, the word problem for 1-relation monoids can be solved using existing computational tools and known mathematical results. In order to make our findings easily accessible and reproducible, we are in the process of collating our computational findings into a database containing formal ROCQ [12] proofs of decidability, which we intend to distribute as part of an accompanying website, the "Online Encyclopedia of 1-relation Monoids".

Adian and Oganesian [1] proved that the word problem is decidable for all 1-relation monoids if it is decidable for all presentations of the form:

$$\langle a, b \mid bua = ava\rangle \quad \text{or} \quad \langle a, b \mid bua = a\rangle \tag{1}$$

for some $u, v \in \{a, b\}^*$. In particular, given any 1-relation monoid M it is straightforward to construct a presentation of the form given in (1) defining a monoid M' with the property that if the word problem is decidable for M', then it is decidable for M also. So, from this point on we will restrict our attention to presentations of the form given in (1).

An overwhelming majority of instances of the WP1M that we have considered were readily tackled by known mathematical results or the Knuth-Bendix algorithm [5]. For other, more difficult instances, however, we were able to reduce the WP1M instance to an instance of the string rewriting system (SRS) termination problem, some of which we could solve using termination provers such as matchbox [13] and MultumNonMulta [4]. The aim of this paper is: to describe how we constructed SRS termination instances from the WP1M; and to provide some examples.

2 A rewriting system formulation of Adian's algorithm

A monoid defined by a 2-generator 1-relation presentation $\mathcal{P} = \langle a, b \mid u = v\rangle$ satisfies the *left cycle-free* condition if and only if $\mathcal{P}$ is in one of the forms given in (1), see [9, Section 2.2]. The *relations words* of $\mathcal{P}$ are the words u and v. As a consequence of the left cycle-free condition, every word $w \in \{a, b\}^+$ has a unique factorization $w = p_1p_2 \cdots p_nht$ where $n \geq 0$, each $p_i \in \{a, b\}^+$ is a proper and non-empty prefix of either u or v, $h \in \{u, v, \varepsilon\}$ and $t \in \{a, b\}^*$ are such that:

(i) for each i, if p' is a longer (possibly improper) prefix of the same relation word as p_i, then $p_1 \cdots p_{i-1}p'$ is not a prefix of w,

(ii) $h \neq \varepsilon$ if and only if $p_1 \cdots p_n \neq w$ and there does not exist a proper and non-empty prefix of a relation word p' such that $p_1 \cdots p_n p'$ is a prefix of w.

We call this factorization the *prefix decomposition* of w, the factor h is called the *head* of the decomposition. If $h = \varepsilon$ we call the prefix decomposition *headless*. We denote the prefix decomposition visually by separating the factors with bars | and highlighting the head in bold, if it is present. For example, with respect to the presentation $\langle a, b \mid baabbaa = aba\rangle$, the word $babbaababaabbaa$ has the prefix decomposition $ba|b|baab|\mathbf{aba}|abbaa$ with head aba. The word $abbaabab$ has the prefix decomposition $ab|baab|ab$, which is headless.

Adian's algorithm $\mathfrak{A}$ for a left cycle-free presentation $\mathcal{P} = \langle a, b \mid u = v\rangle$ takes as input a letter $x \in \{a, b\}$ and a word $w \in \{a, b\}^+$ and proceeds as follows:

1. If the first letter of w is x or if the prefix decomposition of w is headless, then return w;
2. If the head of the prefix decomposition of w equals u, then replace the head by v in w and go to step 1;
3. If the head of the prefix decomposition of w equals v, then replace the head by u in w and go to step 1.

For example, a run of Adian's algorithm for the presentation $\langle a, b \mid baabbaa = aba\rangle$ with input $x = a$, $w = bbaabbabaababa$ produces the following sequence of prefix decompositions:

$$b|baabba|baab|\mathbf{aba} \to b|baabba|\mathbf{baabbaa}|bbaa \to b|\mathbf{baabbaa}|babbaa \to ba|ba|ba|b|baa$$

hence the $\mathfrak{A}(a, bbaabbabaababa) = babababbaa$.

Note that Adian's algorithm does not necessarily terminate on all inputs, e.g. in the monoid given by the presentation $\langle a, b \mid baabbaa = a\rangle$, running $\mathfrak{A}(a, bbaaa)$ results in the following sequence of rewrites:

$$b|baa|\mathbf{a} \to b|baab|\mathbf{a}|abbaa \to b|\mathbf{baabbaa}|bbaaabbaa \to ba|b|baa|\mathbf{a}|bbaa \to \cdots$$

The decomposition $ba|b|baa|\mathbf{a}|bbaa$ contains the decomposition $|b|baa|\mathbf{a}|$ of the initial word, hence $\mathfrak{A}(a, bbaaa)$ will not terminate. In fact, understanding termination of Adian's algorithm leads to a solution of the word problem for the monoid defined by the underlying presentation.

▶ **Theorem 1** (c.f. [9, Theorem 4.3]). *Let $\mathcal{P}$ be a 2-generated 1-relation left cycle-free presentation. If there is an algorithm which, given $x \in \{a, b\}$ and $w \in \{a, b\}^+$, decides whether or not $\mathfrak{A}(x, w)$ terminates, then the word problem for the monoid defined by $\mathcal{P}$ is decidable.*

When Adian's algorithm terminates on all inputs we get Corollary 2, which has been used in several articles to prove the decidability of the word problem for subclasses of 1-relation monoids, see [9, Section 4.3].

▶ **Corollary 2.** *Let $\mathcal{P}$ be a 2-generated 1-relation left cycle-free presentation. If Adian's algorithm terminates on all inputs, then the word problem in the monoid defined by $\mathcal{P}$ is decidable.*

The analysis of termination of Adian's algorithm is complicated somewhat by its description. Despite being written as a general algorithm, it only consists of repeated rewrites on a word depending on the prefix decomposition. Furthermore, the process of prefix decomposition itself can be formulated as a rewriting system, which produces an ever longer factorization until it either finds a head or factorizes the whole word. In order to broaden the termination proving methods available to us, we use these observations to produce, for every cycle-free presentation $\mathcal{P}$, a string rewriting system $\mathcal{A}_{\mathcal{P}}$ for which Theorem 3 holds.

▶ **Theorem 3.** *Let $\mathcal{P}$ be a left cycle-free presentation. Then $\mathcal{A}_{\mathcal{P}}$ is terminating if and only if $\mathfrak{A}$ terminates on all inputs.*

We will only describe our construction as it applies to 2-generated 1-relation left cycle-free presentation, the general construction which applies for all left cycle-free presentation is given in Appendix A. A similar construction has been considered in [2].

▶ **Definition 4.** *Let $\mathcal{P} = \langle a, b \,|\, u = v \rangle$ be a left cycle-free presentation and let*

$$P = \{\alpha \in \{a,b\}^+ : \exists \beta \in \{a,b\}^* \text{ s.t. } \alpha\beta \in \{u,v\}\}$$

be the set of all non-empty prefixes of relation words (including u and v themselves). Let $B = \{q_p \,:\, p \in P\}$ be a set of symbols disjoint from $\{a,b\}$ and let

$$\begin{aligned}
\mathcal{B} &= \{q_p x \to q_{px} \,:\, p \in P, x \in \{a,b\} \text{ s.t. } px \in P\}\\
\mathcal{C} &= \{q_p x \to q_p q_x \,:\, p \in P, x \in \{a,b\} \text{ s.t. } p \notin \{u,v\} \text{ and } px \notin P\}\\
\mathcal{D} &= \{q_u \to v, q_v \to u\}.
\end{aligned}$$

Then Adian's string rewriting system associated to $\mathcal{P}$ *is the string rewriting system $\mathcal{A}_{\mathcal{P}}$ on $\{a,b\} \cup B$ given by the union $\mathcal{A}_{\mathcal{P}} = \mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$.*

The proof of Theorem 3 is rather technical and we won't reproduce it here in full, but we give some indication of its correctness. Let $w \in \{a,b\}^+$, $x \in \{a,b\}$ and $w' \in \{a,b\}^*$ be such that $w = xw'$. We claim that the rewriting system consisting of $\mathcal{B} \cup \mathcal{C}$ when applied to $q_x w'$ computes the prefix decomposition of w. Indeed, after every application, the word will be of the form $q_{p_1} q_{p_2} \dots q_{p_k} t$ for some prefixes $p_1, \dots, p_k \in P$ and $t \in A^*$ such that $w = p_1 \cdots p_k t$. Rules in $\mathcal{B}$ can be used to extend the rightmost prefix p_k in the factorization, if this leads to a longer prefix, including either relation word u and v. Rules in $\mathcal{B}$ apply if the rightmost prefix p_k is not a whole relation word and the letter following p_k does not extend it to a longer prefix. In this case, a new prefix starting at the following letter is added to the factorization. If $p_k \in \{u,v\}$, then no rewrite rule from $\mathcal{B} \cup \mathcal{C}$ applies and p_k is the head of the decomposition. Rules in $\mathcal{D}$ simulate the substitution of the head performed by Adian's algorithm. Let $y \in \{a,b\} \setminus \{x\}$. Then it follows that the output of $\mathfrak{A}(y,w)$ terminates if and only if $\mathcal{A}_{\mathcal{P}}$ terminates when applied to $q_x w'$, and the result of $\mathfrak{A}(y,w)$ can be recovered from the resulting word. This is justification for the forward implication of Theorem 3. The reverse implication follows with some extra work from the fact that the rules in $\mathcal{A}_{\mathcal{P}}$ all commute.

▶ **Example 5.** Let $\mathcal{P} = \langle a, b \mid baabaa = aba \rangle$. Then the associated Adian's SRS is $\mathcal{A}_{\mathcal{P}}$ is as follows:

$$\begin{array}{cccc}
q_a a \to q_a q_a, & q_a b \to q_{ab}, & q_{ab} a \to q_{aba}, & q_{ab} b \to q_{ab} q_b,\\
q_b a \to q_{ba}, & q_b b \to q_b q_b, & q_{ba} a \to q_{baa}, & q_{ba} b \to q_{ba} q_b,\\
q_{baa} a \to q_{baa} q_a, & q_{baa} b \to q_{baab}, & q_{baab} a \to q_{baaba}, & q_{baab} b \to q_{baab} q_b,\\
q_{baaba} a \to q_{baabaa}, & q_{baaba} b \to q_{baaba} q_b, & q_{aba} \to baabaa, & q_{baabaa} \to aba.
\end{array}$$

The SRS $\mathcal{A}_{\mathcal{P}}$ is terminating, which can be established e.g. with the help of the matchbox termination prover. It follows that the monoid defined by the presentation $\mathcal{P} = \langle a, b \mid baabaa = aba \rangle$ has decidable word problem.

Method	Count
Known mathematical results	222786
Knuth-Bendix	37624
Knuth-Bendix backtrack	502
Adian's SRS terminates (certified)	460
Adian's SRS terminates (no certificate)	135
Unsolved (Adian's SRS non-terminating)	75
Unsolved (Other)	50
Total	261632

Table 1 Distribution of solutions to the word problem by the method used for monoids defined by a left cycle-free 2-generated 1-relation presentation $\mathcal{P} = \langle a, b \mid u = v \rangle$, where $|u|, |v| \leq 10$. See Section 3 for more details.

3 Results

We attempted to solve the word problem for all monoids defined by a left cycle-free 2-generated 1-relation presentation $\mathcal{P} = \langle a, b \mid u = v \rangle$, where $|u|, |v| \leq 10$. The only previous published attempt at doing so appears in [10] for presentations with $|u|, |v| \leq 6$. The results of our efforts are collected in Table 1. The rows of the table are as follows:

- "Known mathematical results" refers to presentations which are resolved by results described in [9, Section 2, Section 4.3 and Section 5.3] and generally involves criteria on the presentation which can be checked in polynomial time.
- "Knuth-Bendix" refers to presentations for which we could find complete rewriting systems using the Knuth-Bendix algorithm as it is implemented in libsemigroups [8]. The rewriting systems were obtained by exhaustively checking all possible shortlex orderings, as well as subword substitutions via Tietze transformations of depth up to 5.
- "Knuth-Bendix backtrack" refers to presentations for which complete rewriting systems could be found using a variation of the Knuth-Bendix algorithm which explores all possible word orderings in a backtracking fashion, akin to [15, 14]. This process produces a locally confluent SRS, whose termination we prove using the matchbox termination prover.
- "Adian's SRS terminates (certified)" refers to presentations for which a proof and CPF [11] certificate of termination of Adian's SRS was found using matchbox.
- "Adian's SRS terminates (no certificate)" refers to presentations for which a proof of termination of Adian's SRS was found using either matchbox or MultumNonMulta, but no CPF certificate of termination could be produced. This is mainly because RFC-based [3] proof methods cannot be certified at the moment.
- "Unsolved (Adian's SRS non-terminating)" refers to presentations for which we did not find a solution to the word problem, but for which a proof of non-termination of Adian's SRS was found by either matchbox or MultumNonMulta.
- "Unsolved (Other)" refers to presentations for which we did not find a solution and we do not know whether Adian's SRS terminates on all inputs.

Our results highlight the utility of termination provers for solving the WP1M and the difficulty of proving and certifying termination of the Adian's SRS instances arising from Adian's algorithm. We intend to make the SRS instances arising from Adian's algorithm publicly available by adding them to the Termination Problems Data Base (`https://termination-portal.org/wiki/TPDB`).

References

1 S. I. Adian and G. U. Oganesian. On the word and divisibility problems for semigroups with one relation. *Math. Notes*, 41:235–240, 1987. [Mat. Zametki 41, 3 (1987)].

2 Janet Brugh Bouwsma. *Semigroups presented by a single relation.* PhD thesis, The Pennsylvania State University, 1993.

3 Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting systems. *Applicable Algebra in Engineering, Communication and Computing*, 15(3–4):149–171, October 2004. URL: `http://dx.doi.org/10.1007/s00200-004-0162-8`, `doi:10.1007/s00200-004-0162-8`.

4 Dieter Hofbauer. System description: MultumNonMulta. In Aart Middeldorp and René Thiemann, editors, *WST*, page 90, 2016.

5 D. E. Knuth and P. B. Bendix. *Simple Word Problems in Universal Algebras*, pages 342–376. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983. `doi:10.1007/978-3-642-81955-1_23`.

6 W. Magnus. Das Identitätsproblem für Gruppen mit einer definierenden Relation. *Mathematische Annalen*, 106(1):295–307, December 1932. URL: `http://dx.doi.org/10.1007/BF01455888`, `doi:10.1007/bf01455888`.

7 Yu. V. Matiyasevich. Simple examples of undecidable associative calculi. *Sov. Math., Dokl.*, 8:555–557, 1967.

8 James Mitchell, Michael Young, Finn Smith, Reinis Cirpons, Florent Hivert, Jerry James, Nicolas M. Thiéry, Joe Edwards, Wilf Wilson, Dima Pasechnik, Isuru Fernando, Jan Engelhardt, Luke Elliott, and Max Horn. libsemigroups/libsemigroups: 2.7.0, 2023. URL: `https://zenodo.org/record/7761055`, `doi:10.5281/ZENODO.7761055`.

9 Carl-Fredrik Nyberg-Brodda. The word problem for one-relation monoids: a survey. *Semigroup Forum*, 103(2):297–355, August 2021. `doi:10.1007/s00233-021-10216-8`.

10 John Pedersen. Morphocompletion for one-relation monoids. In Nachum Dershowitz, editor, *Rewriting Techniques and Applications*, pages 574–578, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

11 Christian Sternagel and René Thiemann. The certification problem format. In Christoph Benzmüller and Bruno Woltzenlogel Paleo, editors, Proceedings Eleventh Workshop on *User Interfaces for Theorem Provers,* Vienna, Austria, 17th July 2014, volume 167 of *Electronic Proceedings in Theoretical Computer Science*, pages 61–72. Open Publishing Association, 2014. `doi:10.4204/EPTCS.167.8`.

12 The Rocq Development Team. The rocq prover, April 2025. `doi:10.5281/zenodo.15149629`.

13 Johannes Waldmann. Matchbox: A tool for match-bounded string rewriting. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications*, pages 85–94, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

14 Ian Wehrman, Aaron Stump, and Edwin Westbrook. Slothrop: Knuth-Bendix Completion with a Modern Termination Checker. In Frank Pfenning, editor, *Term Rewriting and Applications*, Lecture Notes in Computer Science, pages 287–296, Berlin, Heidelberg, 2006. Springer. `doi:10.1007/11805618_22`.

15 Sarah Winkler, Haruhiko Sato, Aart Middeldorp, and Masahito Kurihara. Multi-Completion with Termination Tools. *Journal of Automated Reasoning*, 50(3):317–354, March 2013. `doi:10.1007/s10817-012-9249-2`.

A The construction for general left cycle-free presentations

During our experimentation we found that making certain substitutions to a left cycle-free presentation which increase then number of relations, but keep the presentation left cycle-free could sometimes lead to string rewriting systems which were easier to prove termination of. Similar phenomena for the Knuth-Bendix procedure are well known. In order to apply such transformations, however, we need to use a slightly more general construction than the one given in Definition 4. We do so in this appendix.

For a set X we define the symmetric closure of a relation $R \subseteq X \times X$ to be $R^{\text{Sym}} = R \cup \{(\beta, \alpha) : (\alpha, \beta) \in R\}$. For a partial function f, we write $f(x) = \bot$ to indicate that f is not defined on input x. For a non-empty word $w \in A^+$ we will denote the first letter of w by $\texttt{first}(w) \in A$ and write $\texttt{tail}(w)$ for the remainder after removing the first letter, so that $w = \texttt{first}(w)\texttt{tail}(w)$. Let M be a monoid defined by the presentation $\mathcal{P} = \langle A \mid R\rangle$. We assume that that $\alpha, \beta \in A^+$ are non-empty for all relations $(\alpha, \beta) \in R$.

We define $\mathcal{L}(M)$, the *left graph* of M with respect to $\mathcal{P}$, to be the undirected multigraph with vertex set A and an edge between letters $a, b \in A$ for *every* occurrence $(\alpha, \beta) \in R$ such that $a = \texttt{first}(\alpha)$ and $b = \texttt{first}(\beta)$. We say that the monoid M is *left cycle-free* if $\mathcal{L}(M)$ is acyclic with respect to some presentation $\mathcal{P}$.

Note that in a left cycle-free monoid, for every pair of adjacent letters a, b in $\mathcal{L}(M)$ we can find a unique relation $(\alpha, \beta) \in R^{\text{Sym}}$ such that $a = \texttt{first}(\alpha)$ and $b = \texttt{first}(\beta)$. With this in mind, we define the partial functions $\sigma, \tau : A \times A \to A^+$ as follows. Given $a, b \in A$,

- if $a = b$ or a and b are not not connected in the graph $\mathcal{L}(M)$, then let $\sigma(a, b) = \tau(a, b) = \bot$;
- if $a \neq b$ are adjacent in $\mathcal{L}(M)$ then let $\sigma(a, b) = \alpha$ and $\tau(a, b) = \beta$ where $(\alpha, \beta) \in R^{\text{Sym}}$ is the unique relation such that $a = \texttt{first}(\alpha)$ and $b = \texttt{first}(\beta)$;
- if $a \neq b$ are connected but not adjacent in $\mathcal{L}(M)$, then let $\sigma(a, b) = \sigma(a, v_2)$ and $\tau(a, b) = \tau(a, v_2)$, where $a = v_1, v_2, \ldots, v_n = b$ is the unique path connecting a, b in $\mathcal{L}(M)$. Note that $v_2 \neq b$ as a, b are not adjacent.

It follows from the construction that $(\sigma(a, b), \tau(a, b)) \in R^{\text{Sym}}$ is a relation whenever $\sigma(a, b)$ and $\tau(a, b)$ are defined. Note also that $\texttt{first}(\sigma(a, b)) = a$ whenever $\sigma(a, b) \neq \bot$.

▶ **Definition 6.** *Let $\mathcal{P} = \langle A \,|\, R\rangle$ be a left cycle-free presentation defining a monoid M. For every pair $(a, b) \in A \times A$ let $P_{(a,b)} = \{\alpha \in A^+ : \exists \beta \in A^* \text{ s.t. } \alpha\beta = \sigma(a, b)\}$ be the set of non-empty prefixes of $\sigma(a, b)$. Let B be set disjoint from A given by*

$$B = \{q_{(a,b)} : \forall a, b \in A\} \cup \bigcup_{(a,b) \in A \times A} \{q_{(a,\alpha)} : \alpha \in P_{(b,a)}\},$$

and let

$$\begin{aligned}
\mathcal{T}_M &= \{q_{(a,a)} \to a : a \in A\}\\
\mathcal{B}_M &= \{q_{(a,\gamma)} b \to q_{(a,\gamma b)} : a, b \in A, \gamma \in A^+, \gamma, \gamma b \in P_{(\mathit{first}(\gamma),a)}\}\\
\mathcal{C}_M &= \{q_{(a,\gamma)} b \to q_{(a,\gamma)} q_{(c,b)} : a, b, c \in A, \gamma \in A^+, \gamma, \gamma c \in P_{(\mathit{first}(\gamma),a)}, \gamma b \notin P_{(\mathit{first}(\gamma),a)}\}\\
\mathcal{D}_M &= \{q_{(a,\sigma(d,a))} \to q_{(a,\mathit{first}(\tau(d,a)))}\, \mathit{tail}(\tau(d, a)) : a, d \in A, \delta \in A^+, \sigma(d, a) \neq \bot\}.
\end{aligned}$$

The Adian rewriting system *is the rewriting system on $C = A \cup B$ given by the union $\mathcal{A}_{\mathcal{P}} = \mathcal{T} \cup \mathcal{B} \cup \mathcal{C} \cup \mathcal{D}$.*

Theorem 3 holds for $\mathcal{A}_{\mathcal{P}}$ given by Definition 6.

Dependency Pairs for Innermost Constrained Higher-Order Rewriting

Carsten Fuhs
Birkbeck, University of London, United Kingdom

Liye Guo
Radboud University, Nijmegen, The Netherlands

Cynthia Kop
Radboud University, Nijmegen, The Netherlands

Abstract

Logically constrained simply-typed term rewriting systems (LCSTRSs) provide a form of rewriting geared towards analysis of programs with higher-order features and both algebraic and primitive data types. Termination of LCSTRSs has been studied for full rewriting without strategy assumptions. This extended abstract adapts the higher-order constrained Dependency Pair framework for innermost termination, which implies termination under call-by-value evaluation. The DP framework for innermost termination effectively handles universal computability. This provides a foundation for open-world termination analysis of programs using call-by-value evaluation via LCSTRSs.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Logic and verification

Keywords and phrases Higher-order term rewriting, Logically constrained term rewriting, Innermost termination, Open-world termination, Call-by-value, Dependency pair framework.

Related Version A full version of this paper has been published at FSCD 2025 [5].

1 Introduction

In this extended abstract, we sketch techniques for proving *termination* of *Logically Constrained Simply-typed Term Rewriting Systems* (LCSTRSs) [14] for *call-by-value* evaluation. LCSTRSs are a higher-order functional intermediate verification language, designed as a compilation target for static analysis of programs with higher-order types. Our long-term goal is for LCSTRSs to be the cornerstone for a two-stage approach to static program analysis:

1. The *frontend* of the analysis: Given a program P written in (a practically relevant fragment of) the programming language L, translate P to an intermediate representation as an LCSTRS $\mathcal{R}_P$. This translation must (provably) preserve the properties of interest for our static analysis.
2. The *backend* of the analysis: Analyze $\mathcal{R}_P$ using dedicated techniques for static analysis of LCSTRSs. The answer carries over to the original property of P.

This approach has been applied successfully across paradigms to multiple programming languages, such as Prolog [23, 10], Haskell [9], Scala [20], Java [21], and C [7], using various flavors of (constrained or unconstrained) term rewriting instead of LCSTRSs.

Classic term rewriting without any pre-defined data types and operations is known to be Turing-complete, which – in principle – makes it a sufficiently expressive intermediate language for this methodology. However, for *automated* static analysis, being able to represent programming language features directly, without cumbersome encodings, can make the difference between a program analysis tool that quickly finds a proof of the desired property and a tool that returns with an inconclusive result or times out.

We believe that LCSTRSs fill a "sweet spot" for such an intermediate representation. LCSTRSs integrate two strands of automated reasoning that have both proven useful for program analysis: term rewriting – here with support for higher-order functions – and Satisfiability Modulo Theories (SMT) solving.

In this extended abstract, we focus on step 2 of the above program analysis methodology. We analyze *termination* of the *call-by-value* rewrite relation and the (slightly more permissive) *innermost* rewrite relation of LCSTRSs. Call-by-value is a common evaluation strategy for many programming languages (e.g., OCaml, Scala, ...), so analyzing LCSTRSs directly for this rewrite strategy (or its generalization innermost rewriting) is a natural choice. Even for languages with *lazy* evaluation strategies, such as Haskell, past work [9] has shown how a frontend based on a form of abstract interpretation [3] can produce problems whose *innermost* termination implies termination of the lazy evaluation relation of the original program.

As is standard for termination analysis of rewriting, we build on *Dependency Pairs (DPs)* [1] and the *DP framework* [11, 12, 15]. The idea behind DPs is to prove termination of each function call separately. The DP framework allows us to combine different termination proving techniques, commonly referred to as *DP processors*, to simplify and decompose the termination proof obligation at hand. The DP framework was recently adapted to LCSTRSs [13]. However, this adaptation addresses termination for arbitrary rewrite strategies ("full termination"). While a termination proof in this setting is correct also for call-by-value evaluation, some proof methods that are sound for call-by-value or innermost evaluation are not applicable to full termination. Specifically, the analysis of *open-world* termination [13] – intuitively, termination of a set of rules also in the context of an arbitrary program known to be terminating on its own – is limited in power: The reduction pair processor [11, 12, 15] as the workhorse for termination proofs in the DP framework would be unsound.

In this extended abstract, we sketch an adaptation of the DP framework to proving call-by-value and innermost termination of LCSTRSs. We assume familiarity with term rewriting (see, e.g., [2]) and with the DP framework [11, 12, 15]. We use an informal presentation style for the introduced concepts, deliberately eliding some technical details. For formal definitions, theorems, proofs, as well as a discussion of experiments and related work, we refer to the full version of the paper published at FSCD 2025 [5].

2 Background

In this section, we recapitulate LCSTRSs [14] with the help of examples.

LCSTRSs are essentially an extension of (first-order) *Logically Constrained Term Rewriting Systems (LCTRSs)* [16] to a higher-order setting (without λ-abstractions). All variables and function symbols in an LCSTRS are *typed*, using curried simple types. The sorts underlying the type systems used for LCSTRSs represent either user-defined algebraic data types (as for many-sorted classic term rewriting) or pre-defined theory types such as integers or bitvectors taken from a logical theory (in the sense of SMT). The former correspond to user-defined data types in a programming language, and the latter represent primitive types as provided by many practically used programming languages. Types are then constructed inductively using a binary arrow constructor, which also allows for higher types for function arguments.

▶ **Example 1.** Consider the function declarations $\mathsf{gcdlist} : \mathsf{intlist} \to \mathsf{int}$, $\mathsf{fold} : (\mathsf{int} \to \mathsf{int} \to \mathsf{int}) \to \mathsf{int} \to \mathsf{intlist} \to \mathsf{int}$ and $\mathsf{gcd} : \mathsf{int} \to \mathsf{int} \to \mathsf{int}$. Here int is a theory type from integer arithmetic, whereas $\mathsf{intlist}$ is a user-defined data type with constructors $\mathsf{nil} : \mathsf{intlist}$ and $\mathsf{cons} : \mathsf{int} \to \mathsf{intlist} \to \mathsf{intlist}$. For our examples, we use integer arithmetic as the background theory, so we include all values from $\mathbb{Z}$ and the boolean values $\mathfrak{t}$ and $\mathfrak{f}$ as constructors, as well as standard arithmetic operations $(+, -, \ldots)$, in the underlying signature of our LCSTRS.

We can now define the constrained rewrite rules of an LCSTRS $\mathcal{R}$ to compute the greatest common divisor of a list of integers, where we omit constraints of the form $[\mathfrak{t}]$:

$$\begin{array}{llll} \mathsf{gcdlist} \to \mathsf{fold}\ \mathsf{gcd}\ 0 & \mathsf{fold}\ f\ y\ \mathsf{nil} \to y & \mathsf{fold}\ f\ y\ (\mathsf{cons}\ x\ l) \to f\ x\ (\mathsf{fold}\ f\ y\ l) & \\ \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ (-m)\ n\quad [m < 0] & \mathsf{gcd}\ m\ 0 \to m & & [m \geq 0] \\ \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ m\ (-n)\quad [n < 0] & \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ n\ (m \bmod n) & & [m \geq 0 \wedge n > 0] \end{array}$$

To see how an LCSTRS works, consider the following example evaluation, where the used redex is underlined: $\underline{\mathsf{gcdlist}}\ (\mathsf{cons}\ (1+1)\ \mathsf{nil}) \to_{\mathcal{R}} \underline{\mathsf{fold}\ \mathsf{gcd}\ 0\ (\mathsf{cons}\ (1+1)\ \mathsf{nil})} \to_{\mathcal{R}} \mathsf{gcd}\ (1+1)\ \underline{(\mathsf{fold}\ \mathsf{gcd}\ 0\ \mathsf{nil})} \to_{\mathcal{R}} \mathsf{gcd}\ \underline{(1+1)}\ 0 \to_{\mathcal{R}} \underline{\mathsf{gcd}\ 2\ 0} \to_{\mathcal{R}} 2$. The fourth step is a calculation step: $(1+1) \to_{\mathcal{R}} 2$. In the last step, $2 \geq 0$ holds, and we use the top-right gcd rule. Note that this evaluation is neither a call-by-value evaluation ("$\xrightarrow{\mathsf{v}}$": proper subterms of the used redex must be values; in our setting, constructor ground terms) nor an innermost evaluation ("$\xrightarrow{\mathsf{i}}$": proper subterms of the used redex must be in normal form). The reason is that in the second step, we did not rewrite the innermost redex $(1+1)$, which is not a value.

The following is an evaluation sequence that is both call-by-value and thus also innermost: $\underline{\mathsf{gcdlist}}\ (\mathsf{cons}\ (1+1)\ \mathsf{nil}) \xrightarrow{\mathsf{v}}_{\mathcal{R}} \mathsf{fold}\ \mathsf{gcd}\ 0\ (\mathsf{cons}\ \underline{(1+1)}\ \mathsf{nil}) \xrightarrow{\mathsf{v}}_{\mathcal{R}} \underline{\mathsf{fold}\ \mathsf{gcd}\ 0\ (\mathsf{cons}\ 2\ \mathsf{nil})} \xrightarrow{\mathsf{v}}_{\mathcal{R}} \mathsf{gcd}\ 2\ \underline{(\mathsf{fold}\ \mathsf{gcd}\ 0\ \mathsf{nil})} \xrightarrow{\mathsf{v}}_{\mathcal{R}} \underline{\mathsf{gcd}\ 2\ 0} \xrightarrow{\mathsf{v}}_{\mathcal{R}} 2$. Note that in the first step, $(1+1)$ is *not* a subterm of the used redex.

▶ **Example 2.** To see the difference between call-by-value and innermost rewriting, consider the LCSTRS $\mathcal{R} = \{\mathsf{hd}\ (\mathsf{cons}\ x\ l) \to x,\ \mathsf{tl}\ (\mathsf{cons}\ x\ l) \to l\}$. Then $\mathsf{hd}\ (\mathsf{cons}\ 42\ (\mathsf{tl}\ \mathsf{nil})) \xrightarrow{\mathsf{i}}_{\mathcal{R}} 42$ is innermost but not call-by-value. The reason is that the subterm $\mathsf{tl}\ \mathsf{nil}$ of the redex is in normal form, but not a value: Innermost rewriting lets us rewrite above function calls that are "stuck" on their arguments; in OCaml, the computation would abort with an error.

▶ **Example 3** (Ex. 1 continued)**.** Our goal is to prove that call-by-value evaluation with $\mathcal{R}$ from Ex. 1 is terminating. However, termination of *innermost* rewriting, which is slightly more permissive than call-by-value evaluation, is more commonly considered in the term rewriting community and has been studied extensively for first-order term rewriting. Rather than reinvent the wheel, we formulate our techniques for innermost rewriting and provide a transformation to carry over (some) information about the intended call-by-value strategy.

The idea of the transformation is that in practice we may consider theory sorts to be *inextensible*: programmers are not allowed to add new constructors to pre-defined types like int. So we are interested only in instantiations of variables of inextensible theory types like int by their theory values. Now, for LCSTRSs, variables occurring in the constraint of a rule may be instantiated only by theory values during matching (and not, e.g., by normal forms that result from a function call "getting stuck", analogous to Ex. 2). Thus, we add variables of inextensible theory sorts in rewrite rules to their constraints, writing $\varphi, x_1, \ldots, x_n$ for $\varphi \wedge x_1 \equiv x_1 \wedge \cdots \wedge x_n \equiv x_n$. The operation $\equiv$ corresponds to semantic equality in the underlying theory, so the only effect is to prevent undesired variable instantiation by non-values. For $\mathcal{R}$, the transformation adds variables to the constraints of four rules, where variables are added to a constraint only if they do not already occur there, e.g., $n < 0$ in the bottom-right rule already enforces that n may be instantiated only by values from $\mathbb{Z}$:

$$\begin{array}{rlrl} \mathsf{fold}\ f\ y\ \mathsf{nil} \to y & [\mathfrak{t}, y] & \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ (-m)\ n & [m < 0, n] \\ \mathsf{fold}\ f\ y\ (\mathsf{cons}\ x\ l) \to f\ x\ (\mathsf{fold}\ f\ y\ l) & [\mathfrak{t}, x, y] & \mathsf{gcd}\ m\ n \to \mathsf{gcd}\ m\ (-n) & [n < 0, m] \end{array}$$

We will refer to the result of the transformation as $\mathcal{R}_{\mathsf{gcd}}$, and our goal will be to prove innermost termination of this LCSTRS.

Note that this approach of restricting innermost evaluation to call-by-value evaluation affects only variables of *theory* sorts like int. As terms of non-theory types like $\mathsf{intlist}$ are not allowed to occur in constraints of LCSTRS rules, the variable l in the bottom-left rule cannot be added to the constraint. This means that a term like $\mathsf{fold}\ \mathsf{gcd}\ 0\ (\mathsf{cons}\ 1\ (\mathsf{tl}\ \mathsf{nil}))$ can still be rewritten innermost using the bottom-left rule, even though this rewrite step is not call-by-value. Still, innermost termination of the LCSTRS produced by the transformation *implies* call-by-value termination of the input LCSTRS (but in general not vice versa).

3 Dependency Pairs for Innermost Termination of LCSTRSs

DPs [1] are a standard technique for proving termination of term rewriting systems. *Static DPs (SDPs)* [18, 19, 17, 6] are a common adaptation of DPs to the higher-order setting. While SDPs have soundness requirements based on computability (see, e.g., [6] for a discussion), SDPs tend to be applicable to LCSTRSs corresponding to real-world programs.

▶ **Example 4** (Ex. 3 cont'd)**.** The set $\mathrm{SDP}(\mathcal{R}_{\mathsf{gcd}})$ of SDPs for $\mathcal{R}_{\mathsf{gcd}}$ consists of these SDPs:

(1) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{gcd}^\sharp\ m'\ n'$
(2) $\mathsf{gcdlist}^\sharp\ l' \Rightarrow \mathsf{fold}^\sharp\ \mathsf{gcd}\ 0\ l'$
(3) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ (-m)\ n\ [m < 0, n]$
(4) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ m\ (-n)\ [n < 0, m]$
(5) $\mathsf{gcd}^\sharp\ m\ n \Rightarrow \mathsf{gcd}^\sharp\ n\ (m \bmod n)\ [m{\geq}0{\wedge}n{>}0]$
(6) $\mathsf{fold}^\sharp\ f\ y\ (\mathsf{cons}\ x\ l) \Rightarrow \mathsf{fold}^\sharp\ f\ y\ l\ [\mathsf{t}, x, y]$

As usual in the DP setting, "$\sharp$" marks head symbols of potentially non-terminating function calls. Note that rules are implicitly flattened for the calculation of DPs so that variables are added as arguments until the rules have base type; hence the extra argument l' in the left-hand side of DP (1) and in both sides of DP (2). In contrast to the first-order setting, the right-hand side of a DP may contain variables that do not occur in the left-hand side, such as m' and n' in DP (1). The reason is that in the corresponding rewrite rule, the defined symbol gcd occurs in the right-hand side with fewer arguments than required for a step using a gcd-rule. At this point we do not know to which arguments gcd may eventually be applied. This is why we introduce fresh variables for the missing arguments, which in an innermost DP chain are instantiated by arbitrary (well-typed) normal forms. As we do not use SDPs to prove *non*-termination, correctness is not affected.

Our proof obligations are called *DP problems*, pairs $(\mathcal{P}, \mathcal{R})$ with $\mathcal{P}$ a set of DPs and $\mathcal{R}$ a set of rewrite rules. The initial DP problem for $\mathcal{R}_{\mathsf{gcd}}$ is $(\mathrm{SDP}(\mathcal{R}_{\mathsf{gcd}}), \mathcal{R}_{\mathsf{gcd}})$. We must prove absence of infinite $(\mathcal{P}, \mathcal{R})$-chains, analogously to the unconstrained first-order setting. For a DP problem $(\emptyset, \mathcal{R})$, this holds trivially. As in the first-order innermost DP framework [11, 22], we keep track of the rewrite rules of the initial DP problem in a set $\mathcal{Q}$ to have a faithful representation of the rewrite strategy (both components of a DP problem may be modified).

Proof techniques to simplify and decompose DP problems towards reaching DP problems of the shape $(\emptyset, \mathcal{R})$ are called *DP processors*. Many standard DP processors for (constrained and unconstrained) term rewriting can be adapted to the setting of full termination of LCSTRSs [13] and carry over to our innermost setting. These include the Dependency Graph processor, which decomposes the *dependency graph* of potentially consecutive DPs in a chain into its non-trivial strongly connected components (roughly: splits the call graph into mutually recursive calls, deletes other DPs), the subterm criterion and integer mappings (both delete DPs that make arguments smaller). These processors already suffice to prove innermost termination of $\mathcal{R}_{\mathsf{gcd}}$ (see [5] for details).

In the innermost setting, we can provide further processors that are practically relevant for program analysis. Rewrite systems that were obtained from an automatic translation by a frontend tend to contain many rewrite rules that make only small changes to the program state – each instruction is translated separately. Thus, information that is needed for making progress in a termination proof may be spread over different rewrite rules or DPs.

▶ **Example 5.** Consider the below imperative program [7] (here in a Python-like syntax) on the left and the set $\mathcal{P}$ of SDPs generated from this program on the right:

```
def fact(x):
  z = 1           # fact
  i = 1           # u1
  while i <= x:   # u2
    z = z * i     # u3
    i = i + 1     # u4
                  # u5
```

$$\begin{array}{rcll}
\mathsf{fact}^\sharp\ x & \Rightarrow & \mathsf{u}_1^\sharp\ x\ 1 & [\mathsf{t}, x] \\
\mathsf{u}_1^\sharp\ x\ z & \Rightarrow & \mathsf{u}_2^\sharp\ x\ z\ 1 & [\mathsf{t}, x, z] \\
\mathsf{u}_2^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u}_3^\sharp\ x\ z\ i & [i \leq x, z] \\
\mathsf{u}_3^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u}_4^\sharp\ x\ (z * i)\ i & [\mathsf{t}, x, z, i] \\
\mathsf{u}_2^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u}_5^\sharp\ x\ z & [\neg(i \leq x), z] \\
\mathsf{u}_4^\sharp\ x\ z\ i & \Rightarrow & \mathsf{u}_2^\sharp\ x\ z\ (i+1) & [\mathsf{t}, x, z, i]
\end{array}$$

The only loop in the control-flow graph of the imperative program goes from program point `u2` via `u3` and `u4` back to `u2`. Intuitively, the loop terminates because each time the instruction `i = i + 1` at program point `u4` is executed, the value of `i` gets one step closer to the (constant) value of `x`, which is checked at program point `u2`. This means that the value of `x - i` decreases against the bound `0` in each iteration of the loop.

Unfortunately, the above set of DPs does not have the information "decrease" and "against a bound" together in a single DP. Processors like the integer mapping processor or standard reduction pair processors generally need to identify a decrease against a bound within the same DP. This is where *chaining processors* come to the rescue. They allow for merging consecutive DPs such that the resulting DPs carry out the operations of both original DPs.

In our example, chaining processors can iteratively remove the occurrences of $\mathsf{u}_1^\sharp$, $\mathsf{u}_3^\sharp$ and $\mathsf{u}_4^\sharp$, and end with (1) $\mathsf{fact}^\sharp\ x \Rightarrow \mathsf{u}_2^\sharp\ x\ 1\ 1\ [\mathsf{t}, x]$, (2) $\mathsf{u}_2^\sharp\ x\ z\ i \Rightarrow \mathsf{u}_2^\sharp\ x\ (z * i)\ (i + 1)\ [i \leq x, z]$, and (3) $\mathsf{u}_2^\sharp\ x\ z\ i \Rightarrow \mathsf{u}_5^\sharp\ x\ z\ [\neg(i \leq x), z]$. Now both the increase of i and the bound $i \leq x$ are represented in the single DP (2), and termination is easily proved using an integer mapping corresponding to our intuition from the imperative program.

Chaining processors occur in earlier work on constrained rewriting, e.g., for $\mathcal{PA}$-based TRSs [4] and for int-TRSs [8], which both correspond to constrained DPs with constraints over the integers and without nested function calls. Our chaining processor for the DP framework for LCSTRSs does not have the restriction on nested function calls.

Similar to chaining, also DP processors for *DP transformations* [1, 11, 12, 22] from the first-order DP framework modify the DPs themselves. Narrowing and rewriting require defined symbols from $\mathcal{R}$ below the root of a right-hand side of a DP, so they would not be applicable in our example. A future adaptation of (forward) instantiation to the constrained setting *should* be able to propagate constraints between consecutive DPs, although it would not reduce the number of DPs like the chaining processor.

Further processors for the innermost DP framework for LCSTRSs include the usable rules processor, which deletes all rules from $\mathcal{R}$ that are not called directly or indirectly from a DP, and the *reduction pair processor* using argument filterings for temporary removal of higher-order arguments to make reduction pair processors from the first-order world applicable (deletes DPs that make arguments smaller, can also handle arguments with function calls inside DPs). These processors are applicable even for compositional *open-world* termination analysis, where our LCSTRS is just a part of a larger (and growing) code base.

4 Conclusion

We have given an introduction to call-by-value and innermost evaluation using LCSTRSs, and we have sketched a framework for proving innermost termination of LCSTRSs using Dependency Pairs. For further details, please consider the full version of the paper [5].

References

1 T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 236(1–2):133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

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

3 P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In R. M. Graham, M. A. Harrison, and R. Sethi, editors, *Proc. POPL*, pages 238–252, 1977. doi:10.1145/512950.512973.

4 S. Falke and D. Kapur. A term rewriting approach to the automated termination analysis of imperative programs. In R. A. Schmidt, editor, *Proc. CADE*, pages 277–293, 2009. doi:10.1007/978-3-642-02959-2_22.

5 C. Fuhs, L. Guo, and C. Kop. An innermost DP framework for constrained higher-order rewriting. In M. Fernández, editor, *Proc. FSCD*, pages 20:1–20:24, 2025. doi:10.4230/LIPIcs.FSCD.2025.20.

6 C. Fuhs and C. Kop. A static higher-order dependency pair framework. In L. Caires, editor, *Proc. ESOP*, pages 752–782, 2019. doi:10.1007/978-3-030-17184-1_27.

7 C. Fuhs, C. Kop, and N. Nishida. Verifying procedural programs via constrained rewriting induction. *ACM TOCL*, 18(2):14:1–14:50, 2017. doi:10.1145/3060143.

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

9 J. Giesl, M. Raffelsieper, P. Schneider-Kamp, S. Swiderski, and R. Thiemann. Automated termination proofs for Haskell by term rewriting. *ACM TOPLAS*, 33(2):7:1–7:39, 2011. doi:10.1145/1890028.1890030.

10 J. Giesl, T. Ströder, P. Schneider-Kamp, F. Emmes, and C. Fuhs. Symbolic evaluation graphs and term rewriting: a general methodology for analyzing logic programs. In A. King, editor, *Proc. PPDP*, pages 1–12, 2012. doi:10.1145/2370776.2370778.

11 J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: combining techniques for automated termination proofs. In F. Baader and A. Voronkov, editors, *Proc. LPAR*, pages 301–331, 2005. doi:10.1007/978-3-540-32275-7_21.

12 J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *JAR*, 37(3):155–203, 2006. doi:10.1007/s10817-006-9057-7.

13 L. Guo, K. Hagens, C. Kop, and D. Vale. Higher-order constrained dependency pairs for (universal) computability. In R. Královič and A. Kučera, editors, *Proc. MFCS*, pages 57:1–57:15, 2024. doi:10.4230/LIPIcs.MFCS.2024.57.

14 L. Guo and C. Kop. Higher-order LCTRSs and their termination. In S. Weirich, editor, *Proc. ESOP*, pages 331–357, 2024. doi:10.1007/978-3-031-57267-8_13.

15 N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *IC*, 199(1-2):172–199, 2005. doi:10.1016/J.IC.2004.10.004.

16 C. Kop and N. Nishida. Term rewriting with logical constraints. In P. Fontaine, C. Ringeissen, and R. A. Schmidt, editors, *Proc. FroCoS*, pages 343–358, 2013. doi:10.1007/978-3-642-40885-4_24.

17 K. Kusakari, Y. Isogai, M. Sakai, and F. Blanqui. Static dependency pair method based on strong computability for higher-order rewrite systems. *IEICE Trans. Inf. Syst.*, E92.D(10):2007–2015, 2009. doi:10.1587/transinf.E92.D.2007.

18 K. Kusakari and M. Sakai. Enhancing dependency pair method using strong computability in simply-typed term rewriting. *AAECC*, 18(5):407–431, 2007. doi:10.1007/s00200-007-0046-9.

19 K. Kusakari and M. Sakai. Static dependency pair method for simply-typed term rewriting and related techniques. *IEICE Trans. Inf. Syst.*, E92.D(2):235–247, 2009. doi:10.1587/transinf.E92.D.235.

20 D. Milovančević, C. Fuhs, and V. Kunčak. Proving termination of Scala programs by constrained term rewriting. In C. Kop and J. Voigtländer, editors, *Informal Proc. WPTE*, 2025. URL: `https://wpte2025.github.io/pre-proceedings.pdf`.

21 C. Otto, M. Brockschmidt, C. von Essen, and J. Giesl. Automated termination analysis of Java bytecode by term rewriting. In C. Lynch, editor, *Proc. RTA*, pages 259–276, 2010. `doi:10.4230/LIPIcs.RTA.2010.259`.

22 R. Thiemann. *The DP framework for proving termination of term rewriting.* PhD thesis, RWTH Aachen University, Germany, 2007. URL: `https://publications.rwth-aachen.de/record/62510`.

23 F. van Raamsdonk. Translating logic programs into conditional rewriting systems. In L. Naish, editor, *Proc. ICLP*, pages 168–182, 1997. `doi:10.7551/mitpress/4299.003.0018`.

NCPO goes Beta-Eta-Long Normal Form

Johannes Niederhauser
University of Innsbruck, Innsbruck, Austria

Aart Middeldorp
University of Innsbruck, Innsbruck, Austria

Abstract

We adapt our recently introduced higher-order reduction order NCPO to $\beta\eta$-long normal forms which makes it directly applicable to Nipkow's higher-order rewrite systems.

2012 ACM Subject Classification Theory of computation $\to$ Equational logic and rewriting; Theory of computation $\to$ Lambda calculus

Keywords and phrases higher-order rewriting, termination, computability path order

1 Introduction

We recently introduced NCPO [8,9], a reduction order for proving termination of higher-order rewriting on $\beta\eta$-normal forms where matching modulo $\beta\eta$ is employed. This formalism is very similar to Nipkow's higher-order rewrite systems (HRSs) [7] which work with $\beta\eta$-long normal forms instead. As higher-order recursive path orders like NCPO usually have a hard time dealing with lambda abstractions, we defined it for $\beta\eta$-normal forms which typically contain fewer lambda abstractions than $\beta\eta$-long normal forms. However, this design choice excludes the direct applicability of NCPO to HRSs. For NCPO's cousin NHORPO [5], this was resolved by putting it into a suitable wrapper which makes it applicable to any orientation of η. However, there are benefits of directly working on $\beta\eta$-long normal forms: For both NCPO and NHORPO, termination can only be guaranteed for terms where each function symbol is applied to as many arguments as required by its arity. This is not a problem for $\beta\eta$-long normal forms since function symbols (and variables) are always maximally applied. Moreover, choosing the largest possible arity for each function symbol potentially enhances the power of NCPO and NHORPO. Finally, $\beta\eta$-long normal forms facilitate an easier definition of the important concept of nonversatile terms. Hence, we will directly adapt the definition of NCPO to $\beta\eta$-long normal forms despite the potentially increased number of lambda abstractions which the order has to deal with. Our preliminary results suggest that the resulting reduction order is a powerful and lightweight termination method for HRSs.

2 Preliminaries

In this paper, we consider higher-order rewriting on simply-typed lambda terms [1,3]. Given a set $\mathcal{S}$ of sorts, we define the set of simple types $\mathcal{T}$ as usual and follow the convention that the function space constructor $\to$ is right-associative, i.e., $a \to b \to c$ denotes $a \to (b \to c)$. Throughout this text, lowercase letters $a, b, c, \ldots$ denote sorts while upper case letters $T, U, V, \ldots$ denote arbitrary types. For each type $U \in \mathcal{T}$ we consider an infinite set of variables $\mathcal{V}_U$ as well as a set of function symbols $\mathcal{F}_U$ where $\mathcal{V}_U \cap \mathcal{F}_U = \varnothing$ and $\mathcal{V}_U \cap \mathcal{V}_V = \mathcal{F}_U \cap \mathcal{F}_V = \varnothing$ for $V \neq U$. We denote the set of all variables and function symbols by $\mathcal{V} = \bigcup\{\mathcal{V}_U \mid U \in \mathcal{T}\}$ and $\mathcal{F} = \bigcup\{\mathcal{F}_U \mid U \in \mathcal{T}\}$, respectively. The set of simply-typed lambda terms of a type U (Λ_U) is defined as follows: $\mathcal{V}_U \cup \mathcal{F}_U \subseteq \Lambda_U$, if $x \in \mathcal{V}_U$ and $s \in \Lambda_V$ then $\lambda x.s \in \Lambda_{U \to V}$, and if $s \in \Lambda_{U \to V}$ and $t \in \Lambda_U$ then $st \in \Lambda_V$. We follow the convention that application is left-associative, i.e., stu denotes $(st)u$. A *term* is a member of the set $\Lambda = \bigcup\{\Lambda_U \mid U \in \mathcal{T}\}$ and we define the function $\tau\colon \Lambda \to \mathcal{T}$ as $\tau(s) = U$ if $s \in \Lambda_U$.

We write $\mathsf{FV}(s)$ for the set of free variables of a term s. The term $s[x/t]$ denotes the term where all free occurrences of x have been replaced by t without capturing the free variables of t (*capture-avoiding substitution*). Due to the fact that infinitely many variables are available for each type, this can always be done by renaming the bound variables accordingly (*α-renaming*). In the remainder, we abstract away from this technicality by viewing lambda terms as equivalence classes modulo α-renaming. Every term s has a unique β-normal form which we denote by $s{\downarrow_\beta}$. Given a β-normal form s, we write $s{\uparrow^\eta}$ for its unique η-expanded form. Put together, every term s has a unique $\beta\eta$-long normal form $s{\updownarrow_\beta^\eta} = s{\downarrow_\beta}{\uparrow^\eta}$. Terms in $\beta\eta$-long normal form do not contain partial applications, so we can use syntactic sugar where we assign the maximal possible arity to each $h \in \mathcal{F} \cup \mathcal{V}$ and write $h(t_1, \dots, t_n)$ instead of $ht_1 \dots t_n$. Rewriting to β-normal form and the set of $\beta\eta$-long normal forms are denoted by $\to_\beta^!$ and $\mathsf{NF}(\beta\eta^{-1}) \subseteq \Lambda$, respectively.

A $\beta\eta$-long normal substitution σ is a mapping from variables to $\beta\eta$-long normal forms of the same type where $\mathcal{D}\mathsf{om}(\sigma) = \{x \mid \sigma(x) \neq x{\uparrow^\eta}\}$ is finite. We often write σ as a set of variable bindings. Given a substitution $\sigma = \{x_1 \mapsto t_1, \dots, x_n \mapsto t_n\}$, we define $s\sigma$ as the simultaneous capture-avoiding substitution $s[x_1/t_1, \dots, x_n/t_n]$. The free variables of a substitution are defined as follows: $\mathsf{FV}(\sigma) = \bigcup\{\mathsf{FV}(\sigma(x)) \mid x \in \mathcal{D}\mathsf{om}(\sigma)\}$. A substitution is said to be *away from* a finite set of variables X if $(\mathcal{D}\mathsf{om}(\sigma) \cup \mathsf{FV}(\sigma)) \cap X = \varnothing$.

Contexts C are lambda terms which contain exactly one occurrence of the special symbol $\Box$ which can assume any type. We write $C[s]$ for the lambda term C where $\Box$ is replaced by s without employing capture-avoiding substitution. A binary relation $R \subseteq \Lambda \times \Lambda$ is *monotone* if $s\ R\ t$ implies $C[s]\ R\ C[t]$ for every context C. Furthermore, we say that a term t is a *subterm* of s ($s \trianglerighteq t$) if there exists a context C such that $s = C[t]$ and define the proper subterm relation $s \triangleright t$ as $s \trianglerighteq t$ and $s \neq t$. Since we view lambda terms as equivalence classes modulo α-renaming, the subterm relation is also defined modulo α-renaming. Hence, we have e.g. $\lambda x.t \triangleright t[x/z]$ for a fresh variable z.

A pair of terms $\ell \to r$ with $\ell, r \in \mathsf{NF}(\beta\eta^{-1})$ and $\tau(\ell) = \tau(r) \in \mathcal{S}$ constitutes a *rule* if $\mathsf{FV}(r) \subseteq \mathsf{FV}(\ell)$ and ℓ is not of the form $x(t_1, \dots, t_n)$. A *higher-order rewrite system* (HRS) is a set of rules. Given an HRS $\mathcal{R}$, its *rewrite relation* $\to_\mathcal{R}$ is defined as follows: There is a *rewrite step* $s \to_\mathcal{R} t$ if there exist a rule $\ell \to r \in \mathcal{R}$, a $\beta\eta$-long normal substitution σ and a context C such that $s = C[\ell\sigma{\downarrow_\beta}] \in \mathsf{NF}(\beta\eta^{-1})$ and $t = C[r\theta{\downarrow_\beta}]$. Note that our definition of HRSs is equivalent to the original one given in [7]. In particular, $\to_\mathcal{R} \subseteq \mathsf{NF}(\beta\eta^{-1}) \times \mathsf{NF}(\beta\eta^{-1})$. Hence, both rules and rewrite steps only consider terms in their unique $\beta\eta$-long normal form whereas matching is performed modulo $\beta\eta$. We say that an HRS $\mathcal{R}$ is *terminating* if $\to_\mathcal{R}$ is well-founded. We now define the notion of $\beta\eta$-long normal higher-order reduction orders.

▶ **Definition 1.** *A $\beta\eta$-*long normal higher-order reduction order *is a pair $(>, \sqsupset)$ which satisfies the following conditions: $\sqsupset \subseteq \Lambda \times \Lambda$ is a well-founded relation, $\sqsupset$ is monotone, $\to_\beta \subseteq \sqsupset$, and $s > t$ implies $s\sigma{\downarrow_\beta} \sqsupset^+ t\sigma{\downarrow_\beta}$ for all $s, t \in \mathsf{NF}(\beta\eta^{-1})$ and $\beta\eta$-long normal substitutions σ. We often refer to the last condition as $\beta\eta$-*long normal stability*. An* HRS *$\mathcal{R}$ is* compatible *with a $\beta\eta$-long normal higher-order reduction order $(>, \sqsupset)$ if $\ell >^+ r$ for all $\ell \to r \in \mathcal{R}$.*

As in [5,9], the intuition behind this definition is that $>$ will be used to orient the rules of HRSs while relying on the termination proof of its plain variant $\sqsupset$. Despite calling $(>, \sqsupset)$ an order, we do not demand transitivity of any of its components. In the context of higher-order rewriting, this is standard as $\sqsupset$ contains β-reduction which is not transitive. By taking the identity substitution, we can see that $\beta\eta$-long normal stability implies $> \subseteq \sqsupset^+$.

▶ **Theorem 2.** *If an* HRS *$\mathcal{R}$ is compatible with a $\beta\eta$-long normal higher-order reduction order $(>, \sqsupset)$, then $\mathcal{R}$ is terminating.*

Proof. For a proof by contradiction, consider an infinite rewrite sequence $s_1 \to_{\mathcal{R}} s_2 \to_{\mathcal{R}} \cdots$. By definition, $s_1 = C[\ell\sigma{\downarrow_\beta}]$ and $s_2 = C[r\sigma{\downarrow_\beta}]$ for some $\ell \to r \in \mathcal{R}$ where σ is a $\beta\eta$-long normal substitution and C is a context such that $C[\ell\sigma{\downarrow_\beta}] \in \mathsf{NF}(\beta\eta^{-1})$. By assumption, $\ell >^+ r$, so $\ell\sigma{\downarrow_\beta} \sqsupset^+ r\sigma{\downarrow_\beta}$ follows from $\beta\eta$-long normal stability and we obtain $C[\ell\sigma{\downarrow_\beta}] \sqsupset^+ C[r\sigma{\downarrow_\beta}]$ by monotonicity of $\sqsupset$. Hence, we can transform the infinite sequence $s_1 \to_{\mathcal{R}} s_2 \cdots$ into the infinite descending sequence $s_1 \sqsupset^+ s_2 \cdots$ which contradicts well-foundedness of $\sqsupset$. Thus, $\mathcal{R}$ is terminating. ◀

3 The Beta-Eta-Long-Normal Computability Path Order

We start by adapting the notion of nonversatile terms from [5, 9] to $\beta\eta$-long normal forms. Intuitively, a term in $\beta\eta$-long normal form is nonversatile if the application of any $\beta\eta$-long normal substitution together with the subsequent β-normalization only affects its proper subterms. As this is needed in the inductive proof of $\beta\eta$-long normal stability, a comparison $s > t$ in our order is only defined for nonversatile s. As opposed to [5, 9], we can directly define the class of nonversatile terms since we work on $\beta\eta$-long normal forms: Fully applied function symbols and lambda abstractions are *nonversatile* while fully applied variables are *versatile*. The set of nonversatile terms is denoted by $\Lambda_{\mathsf{nv}} \subseteq \mathsf{NF}(\beta\eta^{-1})$.

Besides the usual inference rules on terms, reduction orders derived from HORPO [4] require appropriate orders on types. The following definition recalls the notion of admissible type orders from CPO [2].

▶ **Definition 3.** *We define the left (right) argument relation on types $\rhd_l$ ($\rhd_r$) as follows: $T \to U \rhd_l T$ ($T \to U \rhd_r U$). An order $\succ_{\mathcal{T}}$ on types is admissible if $\rhd_r \subseteq \succ_{\mathcal{T}}$, $\gtrdot = (\succ_{\mathcal{T}} \cup \rhd_l)^+$ is well-founded, and if $T \to U \succ_{\mathcal{T}} V$ then $U \succeq_{\mathcal{T}} V$ or $V = T \to U'$ with $U \succ_{\mathcal{T}} U'$. Given a type T and $a \in \mathcal{S}$ we write $a \gtrdot_{\mathcal{S}} T$ ($a \geqslant_{\mathcal{S}} T$) if $a \gtrdot b$ ($a \geqslant b$) for every $b \in \mathcal{S}$ occurring in T.*

▶ **Lemma 4** (Lemma 2.3 from [2])**.** *Given a well-founded order $\succ_{\mathcal{S}}$ on sorts, let $\succ_{\mathcal{T}}$ be the smallest order on types containing $\succ_{\mathcal{S}}$ and $\rhd_r$ such that $V \succ_{\mathcal{T}} V'$ implies $U \to V \succ_{\mathcal{T}} U \to V'$ for all U, V and V'. Then, $>_{\mathcal{T}}$ is admissible.*

Unlike their first-order versions, higher-order recursive path orders do not enjoy the subterm property. Thus, if we want to recursively define $f(\bar{t}) > v$ by $t_i \geqslant v$ for some i, we usually have to check whether $\tau(t_i) \succeq_{\mathcal{T}} \tau(v)$ holds for the given admissible type order $\succ_{\mathcal{T}}$. In particular, this means that establishing $s > t$ by choosing some $s \rhd s'$ and showing $s' \geqslant t$ is only possible if we check that there is a weak decrease in the admissible type order for all intermediate terms in the recursive definition of $\rhd$. CPO extends HORPO by allowing these checks to be dismissed for the special case of accessible subterms which are determined by type-related properties. To this end, we use the sets $\mathcal{P}\mathsf{os}^+(T)$ and $\mathcal{P}\mathsf{os}_a(T)$ of *positive sort positions* and *positions of* $a \in \mathcal{S}$ in a type $T \in \mathcal{T}$ as defined in [2, Definition 7.2] in the following definition of accessible arguments of function symbols and basic sorts as given in [2, Definitions 7.3 & 7.4].

▶ **Definition 5.** *With every $f \in \mathcal{F}_{T_1 \to \cdots \to T_n \to a}$ we associate a set $\mathsf{Acc}(f)$ of* accessible *arguments of f such that $i \in \mathsf{Acc}(f)$ implies $a \geqslant_{\mathcal{S}} T_i$ and $\mathcal{P}\mathsf{os}_a(T_i) \subseteq \mathcal{P}\mathsf{os}^+(T_i)$ for all $1 \leqslant i \leqslant n$. Furthermore, we say that $a \in \mathcal{S}$ is* basic *if the following conditions hold: $T \prec_{\mathcal{T}} a$ implies that T is a basic sort, and for all $f \in \mathcal{F}_{T_1 \to \cdots \to T_n \to a}$ and $i \in \mathsf{Acc}(f)$, $T_i = a$ or T_i is a basic sort.*

Note that the condition $T \prec_{\mathcal{T}} a$ is straightforward to check with the admissible type order from Lemma 4 as only sorts can be smaller than sorts. Next, we define the notion of

$$
\begin{array}{rl}
\langle\mathcal{F}\rhd\rangle & f(\bar{t}) >^{b,X} v \text{ if } f \in \mathcal{F} \text{ and } t_i \trianglerighteq_{\mathsf{b}}^{\mathsf{nv}} \cdot \trianglerighteq_{\mathsf{a}} \cdot \geqslant_{\tau}^{b} v \text{ for some } i \\
\langle\mathcal{F}{=}_{\mathsf{mul}}\rangle & f(\bar{t}) >^{b,X} g(\bar{u}) \text{ if } f \simeq_{\mathcal{F}} g,\ \mathsf{stat}(f) = \mathsf{mul} \text{ and } \bar{t}\ (>_{\tau}^{b} \cup \rhd_{@}^{X} \cdot \rightarrow_{\beta}^{!})_{\mathsf{mul}}\ \bar{u} \\
\langle\mathcal{F}{=}_{\mathsf{lex}}\rangle & f(\bar{t}) >^{b,X} g(\bar{u}) \text{ if } f \simeq_{\mathcal{F}} g,\ \mathsf{stat}(f) = \mathsf{lex} \text{ and} \\
 & \quad \exists i.\, t_i >_{\tau}^{b} \cup \rhd_{@}^{X} \cdot \rightarrow_{\beta}^{!} u_i \text{ and } \forall j < i.\, t_j = u_j \text{ and } \forall j > i.\, f(\bar{t}) >^{b,X} t_j \\
\langle\mathcal{F}\succ\rangle & f(\bar{t}) >^{b,X} g(\bar{u}) \text{ if } f \in \mathcal{F},\ f \succ_{\mathcal{F}} g \text{ and } f(\bar{t}) >^{b,X} u_i \text{ for all } i \\
\langle\mathcal{F}\mathcal{V}\rangle & f(\bar{t}) >^{1,X} y(\bar{u}) \text{ if } f \in \mathcal{F},\ f(\bar{t}) >^{0,X} y{\uparrow^{\eta}} \text{ and } f(\bar{t}) >^{1,X} u_i \text{ for all } i \\
\langle\mathcal{F}\lambda\rangle & f(\bar{t}) >^{b,X} \lambda y.v \text{ if } f \in \mathcal{F},\ f(\bar{t}) >^{b,X\cup\{z\}} v[y/z],\ \tau(y) = \tau(z) \text{ and } z \text{ fresh} \\
\langle\mathcal{F}\mathcal{X}\rangle & f(\bar{t}) >^{b,X} y{\uparrow^{\eta}} \text{ if } f \in \mathcal{F} \text{ and } y \in X \\
\langle\lambda\rhd\rangle & \lambda x.t >^{b,X} v \text{ if } t[x/z] \geqslant_{\tau}^{b,X} v,\ \tau(x) = \tau(z) \text{ and } z \text{ fresh} \\
\langle\lambda{=}\rangle & \lambda x.t >^{b,X} \lambda y.v \text{ if } t[x/z] >^{b,X} v[y/z],\ \tau(x) = \tau(y) = \tau(z) \text{ and } z \text{ fresh} \\
\langle\lambda{\neq}\rangle & \lambda x.t >^{b,X} \lambda y.v \text{ if } \lambda x.t >^{b,X} v[y/z],\ \tau(x) \neq \tau(y) = \tau(z) \text{ and } z \text{ fresh}
\end{array}
$$

Figure 1 Rules of NCPO-LNF.

nonversatilely accessible subterms. The definition closely follows the one given in [2, Definition 7.5] but with appropriate modifications regarding nonversatility and $\beta\eta$-long normal forms.

▶ **Definition 6.** *We write $s \rhd_{\mathsf{b}}^{\mathsf{nv}} t$ if t is a subterm of $s \in \Lambda_{\mathsf{nv}}$ reachable by nonversatile intermediate terms in the recursive definition of $\rhd$, t is of basic sort and $\mathsf{FV}(t) \subseteq \mathsf{FV}(s)$, i.e., no bound variables have become free. Furthermore, $s \rhd_{\mathsf{a}} t$ if $s = f(s_1, \ldots, s_n)$ and $s_j \trianglerighteq_{\mathsf{a}} t$ for $j \in \mathsf{Acc}(f)$. Note that $\rhd_{\mathsf{b}}^{\mathsf{nv}}, \rhd_{\mathsf{a}} \subseteq \mathsf{NF}(\beta\eta^{-1}) \times \mathsf{NF}(\beta\eta^{-1})$. A term t is* nonversatilely accessible *in a nonversatile term s if $s \rhd_{\mathsf{b}}^{\mathsf{nv}} t$ or $s \rhd_{\mathsf{a}} t$.*

The following definition, adapted from [2, Definition 7.8], introduces the notion of structurally smaller terms with respect to a set of variables X. It is an important ingredient of CPO with accessible subterms as it facilitates a way of using the set X with which the order is parameterized in places where it would otherwise lead to non-termination.

▶ **Definition 7.** *Let X be a finite set of variables. We say that a term t is* structurally smaller *than a term $s \in \mathsf{NF}(\beta\eta^{-1})$, written $s \rhd_{@}^{X} t$, if there are $a \in \mathcal{S}$, $x_1, \ldots, x_n \in X$ and $u \in \Lambda_{T_1 \to \cdots \to T_n \to a}$ such that $t = u x_1 \cdots x_n$, $s \rhd_{\mathsf{a}} u$, $\tau(s) = a$, $\tau(x_i) = T_i$,and $\mathcal{P}\mathsf{os}_a(T_i) = \varnothing$ for all $1 \leqslant i \leqslant n$. Here, $\rhd_{@}^{X} \subseteq \mathsf{NF}(\beta\eta^{-1}) \times \Lambda$.*

In the following, let $\succ_{\mathcal{T}}$ be an admissible order on types and $\succsim_{\mathcal{F}}$ a preorder on $\mathcal{F}$ called *precedence* with a well-founded strict part $\succ_{\mathcal{F}} = \succsim_{\mathcal{F}} \setminus \precsim_{\mathcal{F}}$ and the equivalence relation $\simeq_{\mathcal{F}} = \succsim_{\mathcal{F}} \cap \precsim_{\mathcal{F}}$. Furthermore, for every $f \in \mathcal{F}$ we fix a status $\mathsf{stat}(f) \in \{\mathsf{mul}, \mathsf{lex}\}$ such that symbols equivalent in $\simeq_{\mathcal{F}}$ have the same status. We are now able to lift CPO with accessible subterms and small symbols [2] to an appropriate component of a $\beta\eta$-long normal higher-order reduction order.

▶ **Definition 8.** *Given a finite set X of variables and $b \in \{0, 1\}$, the order $>^{b,X} \subseteq \mathsf{NF}(\beta\eta^{-1}) \times \mathsf{NF}(\beta\eta^{-1})$ is inductively defined in Figure 1. Furthermore, $s >_{\tau}^{b,X} t$ if $s >^{b,X} t$ and $\tau(s) \succeq_{\mathcal{T}} \tau(t)$, and $>^{b}$ ($>_{\tau}^{b}$) is a shorthand for $>^{b,\varnothing}$ ($>_{\tau}^{b,\varnothing}$).*

Note that $s >^{b,X} t$ is well-defined by induction on (b, s, t) with respect to the well-founded order $(>_{\mathbb{N}}, \rhd, \rhd)_{\mathsf{lex}}$. We refer to $>_{\tau}^{1}$ as the $\beta\eta$-long normal computability path order (NCPO-LNF) and use the symbol $\sqsupset$ with the same decorations as $>$ (except for b) to denote CPO with accessible subterms. The definition of $>^{X}$ is an adaption of the corresponding definition for NCPO to $\beta\eta$-long normal forms. In particular, the modifications of the original rules

Table 1 Experimental results.

problem	NCPO-LNF	NCPO	WANDA
[9, Example 6] (extended)	✓	×	✓
[9, Example 7]	✓	✓	✓
[9, Example 8]	✓	✓	✓
[2, Example 5.2]	✓	✓	✓
[2, Example 7.1]	✓	✓	×
[2, Example 8.19]	✓	✓	✓
[5, Example 7.1]	✓	✓	✓
[5, Example 7.2]	×	×	✓
[5, Example 7.3]	✓	✓	✓
`neutr.p`	×	×	✓
`neutrN.p`	✓	✓	✓

$\langle \mathcal{F}@ \rangle$ and $\langle \mathcal{F}\mathcal{V} \rangle$ justify renaming them to $\langle \mathcal{F}\mathcal{V} \rangle$ and $\langle \mathcal{F}\mathcal{X} \rangle$, respectively. Moreover, the rule $\langle \lambda \rhd \eta \rangle$ of NCPO has been removed and we do not need cases for applications on the left-hand side as terms of the form $y(\bar{t})$ are versatile. Furthermore, we removed the rule $\langle \lambda \mathcal{V} \rangle$ because for variables of functional type, $\lambda x.t >^X y{\uparrow^\eta}$ cannot be directly simulated in CPO which we need for the proof of $\beta\eta$-long normal stability. We also removed the extension of CPO to small symbols: Since terms are in $\beta\eta$-long normal form, applicative uncurried systems, the main motivation behind small symbols in [2], do not have to be considered anymore. Without applications we would still have the $\langle \lambda \mathcal{F}_\mathsf{s} \rangle$ rule which allows us to discard small symbols when comparing to an abstraction in NCPO, but we could not find an example where this is needed the setting of $\beta\eta$-long normal forms. More generally, $\lambda x.t >^X g(\bar{u})$ can only be a recursive comparison of $f(\lambda x.t_1, t_2, \ldots, t_n) >^X g(\bar{u})$ for some $f \in \mathcal{F}$ as $\tau(g(\bar{u})) \in \mathcal{S}$. If $f \succsim_\mathcal{F} g$ then g can be discarded anyway, so we need $g \succ_\mathcal{F} f$ which means that f is also a small symbol. However, $g \succ_\mathcal{F} f$ can only be enforced by a goal $g(\bar{v}) > f(\bar{s})$ resolved with CPO's $\langle \mathcal{F}_\mathsf{s}{=} \rangle$ which requires $g(\bar{v}) >_\mathcal{T} s_1$. Since $\tau(g(\bar{v})) \in \mathcal{S}$ and $\tau(s_1) = U \to V$, this is impossible with the admissible type ordering from Lemma 4 which we implemented.

▶ **Theorem 9.** *The pair* $(>^1_\tau, \sqsupset_\tau)$ *is a* $\beta\eta$*-long normal higher-order reduction order.*

A prototype implementation of NCPO-LNF is available at GitHub.[1] Table 1 compares NCPO-LNF with NCPO and the state of the art higher-order termination tool WANDA [6] on the small set of problems used in our original NCPO paper [9]. Here, ✓ stands for a successful termination proof and × denotes that termination could not be established. Note that NCPO-LNF proves termination of the corresponding HRS while WANDA's soundness is limited to PRSs and NCPO operates on a $\beta\eta$-normal flavor of HRSs. Since HRSs are restricted to rules operating on sorts, we have adapted [9, Example 6] accordingly which explains why NCPO cannot handle it in contrast to our results reported in [9]. Overall, our experimental results give evidence that NCPO-LNF performs slightly better than NCPO for some interesting HRSs. In order to strengthen this evidence, NCPO and NCPO-LNF should be evaluated on larger problem databases such as TPDB and COPS[2] in the future.

[1] `https://github.com/niedjoh/hrsterm`
[2] `https://ari-cops.uibk.ac.at/COPS`

References

1 Henk P. Barendregt. Lambda calculi with types. In Samson Abramsky, Dov M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992. doi:10.1093/oso/9780198537618.003.0002.

2 Frédéric Blanqui, Jean-Pierre Jouannaud, and Albert Rubio. The computability path ordering. *Logical Methods in Computer Science*, 11(4):3:1–3:45, 2015. doi:10.2168/LMCS-11(4:3)2015.

3 Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(2):56–68, 1940. doi:10.2307/2266170.

4 Jean-Pierre Jouannaud and Albert Rubio. Polymorphic higher-order recursive path orderings. *Journal of the ACM*, 54(1):1–48, 2007. doi:10.1145/1206035.1206037.

5 Jean-Pierre Jouannaud and Albert Rubio. Normal higher-order termination. *ACM Transactions on Computational Logic*, 16(2):13:1–13:38, 2015. doi:10.1145/2699913.

6 Cynthia Kop. WANDA - a higher order termination tool (system description). In Zena M. Ariola, editor, *Proc. 5th International Conference on Formal Structures for Computation and Deduction*, volume 167 of *Leibniz International Proceedings in Informatics*, pages 36:1–36:19, 2020. doi:10.4230/LIPIcs.FSCD.2020.36.

7 Richard Mayr and Tobias Nipkow. Higher-order rewrite systems and their confluence. *Theoretical Computer Science*, 192(1):3–29, 1998. doi:10.1016/S0304-3975(97)00143-6.

8 Johannes Niederhauser and Aart Middeldorp. The computability path order for beta-eta-normal higher-order-rewriting (full version). *CoRR*, abs/2505.20121, 2025. doi:10.48550/arXiv.2505.20121.

9 Johannes Niederhauser and Aart Middeldorp. The computabiliy path order for beta-eta-normal higher-order rewriting. In *Proc. 30th International Conference on Automated Deduction*, 2025. To appear.

Disjunctive Termination for Affluent Families

Vincent van Oostrom
University of Sussex, Brighton, United Kingdom

Abstract

We introduce *affluence*, a condition on composition that entails a finite *family* of rewrite relations is terminating iff its *union* is (disjunctive termination), and relate it to *jumping*. Our proofs transform infinite reductions in the family union into such in family members, by induction on family size.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Rewriting, disjunctive termination, affluence, modularity

Acknowledgements We thank Carsten Fuhs, Thomas Powell, reviewers for comments, Nachum Dershowitz for feedback and Georg Moser, Julian Parsert for work on application. CC by 4.0 🅭 🅯.
This note evolved in 4 acts: (I) Preponement (2006) introduced the *proof-by-picture* in Fig. 1, obtaining Geser's Cor. 5(2) (*transitivity*) and Doornbos and von Karger's Cor. 12(1) (*jumping*) via the *contrapositive* of that *non-termination* of a doubleton family *transfers* to one of its members; (II) Preponement (2011) recast the proof as a *transformation* into *progressive* reductions (Def. 3); (III) More Modular Termination (2023) showed Podelski and Rybalchenko's Cor. 8(3) and Dawson, Dershowitz and Goré's Cor. 14(1) follow from that reductions in *finite* families can be *upgraded*, *inductively* using *promotion* for *doubleton* families (Def. 3), and introduced *affluence* (betwixt transitivity and jumping; Sec. 2). It showed how to *blend* promotion results, *modularising* old and suggesting new blends (Cor. 16); (IV) Here, we make the proof of promotion uniform, dependent on *restricting* rewrite relations to the objects *in* ($\restriction$; Def. 2) and *along* ($|$; Def. 9) a reduction.

1 Introduction

We model program execution by means of rewrite systems, where we leave the objects abstract in order to not commit to any particular execution model. Suppose a program P comprises a finite family $(P)_I$ of program modules P_i for $i \in I$, such that the rewrite relation $\to$ modelling P is the union $\bigcup(\to)_I$ of the finite family $(\to)_I$ of rewrite relations modelling $(P)_I$. It is good science to aim for modularity, here, to try to show termination of P as a consequence of termination of (each module in) the family $(P)_I$. Naïvely taken, this fails: both $a \triangleright b$ and $b \blacktriangleright a$ are terminating, but their union $\to := \triangleright \cup \blacktriangleright$ is not, allowing the reduction cycle $a \to b \to a$. The example exhibits feature interaction: composing steps from the different modules $\triangleright$ and $\blacktriangleright$ was not accounted for, and indeed led to non-termination. Therefore, to account for that executions of P arise by composing executions of its modules in $(P)_I$, it is natural to make additional *assumptions on compositions*. We discuss two such assumptions, *affluence* and *jumping*, showing statements for them of shape: for a finite family $(\to)_I$ and $\to := \bigcup(\to)_I$, if there's a $\to$-reduction γ having property $\mathcal{P}$, then there's a $\to_i$-reduction δ having property $\mathcal{Q}$ for *some* $i \in I$. Our proofs *transform* γ into δ and are by *induction* on the family size. Letting both $\mathcal{P}$ and $\mathcal{Q}$ express that the reduction is infinite, it then follows by contraposition that $\to$ is terminating if $(\to)_I$ is, giving our main applications.

We use arrow-like notations $\to, \blacktriangleright, \triangleright, \ldots$ to denote *rewrite* relations, binary endorelations, and $\gamma, \delta, \epsilon, \ldots$ to range over *reductions*, sequences of consecutive steps of such, denoted by repeated-arrows like $\twoheadrightarrow, \blacktriangleright\!\!\blacktriangleright, \triangleright\!\!\triangleright, \ldots$. See the literature, *e.g.*, [8], for more on rewriting.

2 Affluent families

▶ **Definition 1** ([5, Def. 2.3]). $\blacktriangleright, \triangleright$ *is* affluent *if* $\triangleright \cdot \blacktriangleright \subseteq \triangleright \cup \blacktriangleright$.

Enforcing affluence bars the monster in Sec. 1 since $a \not\twoheadrightarrow a$ though $a \triangleright b \blacktriangleright a$. Affluence is *rich*: (i) If $\blacktriangleright = \triangleright$ it expresses *transitivy*; (ii) confluence (*flowing together*) $\twoheadleftarrow \cdot \twoheadrightarrow \subseteq \twoheadrightarrow \cdot \twoheadleftarrow$ is strengthened by affluence (*flowing toward*, a river being a tributary), $\twoheadleftarrow \cdot \twoheadrightarrow \subseteq \twoheadleftarrow \cup \twoheadrightarrow$, when instantiating $\triangleright$ and $\blacktriangleright$ with inverse reductions $\twoheadleftarrow$ respectively reductions $\twoheadrightarrow$; (iii) For less-than-or-equal $\leq$ on the natural numbers the assumption $n \geq \cdot \leq m$ holds for *any* n, m as $0 \leq n, m$, so affluence expresses *totality*: $n \geq m$ or $n \leq m$; (iv) The prefix order $\sqsubseteq$ on finite $\rightarrow$-reductions is affluent, *i.e.* $\sqsupseteq, \sqsubseteq$ is, iff $\rightarrow$ is *deterministic* (*cf.* the CompCert formalisation). The intuition is that affluence affords to compress consecutive out-of-order steps to yield a reduction that is progressive ($\blacktriangleright$-steps occur before $\triangleright$-steps in the reduction) and preferential (given an object in the reduction, $\blacktriangleright$-steps from it are preferred over $\triangleright$-steps). To generalise that from two rewrite relations $\blacktriangleright$ and $\triangleright$ to finite families later, we relativize affluence.

▶ **Definition 2.** *Given a $\rightarrow$-reduction γ, $\restriction\gamma$ denotes restricting a rewrite relation to objects in γ and steps to be co-initial to some step in γ, and $\blacktriangleright, \triangleright$ is* affluent *for a $\rightarrow$-reduction γ if $\blacktriangleright\restriction\gamma, \triangleright\restriction\gamma$ is affluent, where $\rightarrow = \blacktriangleright \cup \triangleright$.*

Affluence entails affluence for any reduction γ: if $a\ ((\triangleright\restriction\gamma) \cdot (\blacktriangleright\restriction\gamma))\ b$, then a, b in γ and $a\ (\triangleright \cdot \blacktriangleright)\ b$ by definition of restriction so $a\ (\triangleright \cup \blacktriangleright)\ b$ by assumption, so $a\ ((\triangleright\restriction\gamma) \cup (\blacktriangleright\restriction\gamma))\ b$ by definition. Observe that if $\rightarrow\restriction\gamma = \rightarrow$ then $\rightarrow$ has at most one normal form, which if it exists is the target of γ and thus, if γ is infinite any maximal reduction[1] is infinite too.

▶ **Definition 3.** *A reduction δ is* progressive *if $\blacktriangleright$-steps precede $\triangleright$-steps in δ except possibly for an infinite $\blacktriangleright$-tail, and* preferential *if $\triangleright$-steps in δ are from $\blacktriangleright$-normal forms. For $\rightarrow\restriction\gamma = \rightarrow = \blacktriangleright \cup \triangleright$ a reduction γ* upgrades *to δ, denoted by $\gamma\nearrow\delta$ (*promotes *to δ, denoted by $\gamma\Nearrow\delta$) if δ is co-initial to γ, maximal and progressive (and preferential).*

Observe if $\gamma\nearrow\delta$ then δ has shape $\blacktriangleright\!\!\blacktriangleright \cdot \triangleright^{\alpha}$ or $\blacktriangleright\!\!\blacktriangleright \cdot \triangleright\!\!\triangleright \cdot \blacktriangleright^{\omega}$ for $\alpha \leq \omega$ with $\alpha = \omega$ if γ is infinite.

▶ **Lemma 4.** *For a reduction γ with $\rightarrow\restriction\gamma = \rightarrow = \blacktriangleright \cup \triangleright$ and $\blacktriangleright, \triangleright$ affluent, $\gamma\Nearrow\hat{\gamma}$ for some $\hat{\gamma}$.*

Proof. Under the assumptions, let δ be a maximal $\blacktriangleright$-reduction co-initial to γ. If δ is infinite, then $\gamma\Nearrow\hat{\gamma}$ for $\hat{\gamma} := \delta$. Otherwise, we construct a $\triangleright$-reduction ϵ by initially setting it to the empty reduction on the target of δ and repeating as long as its target is not that of γ and in $\blacktriangleright$-normal form, to append to ϵ some $\triangleright$-step to an object that is either a $\blacktriangleright$-normal form or non-$\blacktriangleright$-terminating, which we claim exists. ▬ If ϵ is infinite, then per construction $\gamma\Nearrow\delta \cdot \epsilon$, so we set $\hat{\gamma}$ to $\delta \cdot \epsilon$, as visualised in Fig. 1 for reductions δ and ϵ, with $\lightning$ marking $\blacktriangleright$-normal forms in γ; ▬ If ϵ is finite, its target is either non-$\blacktriangleright$-terminating or the target of γ and we have $\gamma\Nearrow\hat{\gamma}$ when setting $\hat{\gamma}$ to $\delta \cdot \epsilon$, in the former case followed by any infinite $\blacktriangleright$-reduction.

To prove the claim, note an object $\hat{b}$ being in $\blacktriangleright$-normal form and not the target of γ, has a step $\hat{b} \triangleright c$ for some c. If c is in $\blacktriangleright$-normal form we return the $\triangleright$-step to it. Otherwise, $c \blacktriangleright c'$ for some c'. By $\hat{b} \triangleright c \blacktriangleright c'$ and affluence, we have $\hat{b} \triangleright c'$ as we cannot have the other disjunct[2] by $\hat{b}$ being in $\blacktriangleright$-normal form. Repeating this for c' instead of c, we eventually end up in the first case or find an infinite reduction $c \blacktriangleright c' \blacktriangleright \ldots$ so return the $\triangleright$-step to c. ◀

Assuming a non-terminating $\rightarrow$-reduction γ were to exist, restricting affluent $\blacktriangleright, \triangleright$ to γ allows one to conclude to non-termination of $\blacktriangleright$ or $\triangleright$ by Lem. 4 (using the observations), so:

▶ **Corollary 5.** *Let $\rightarrow := \blacktriangleright \cup \triangleright$. $\blacktriangleright, \triangleright$ are terminating iff $\rightarrow$ is, if:* **1.** *$\blacktriangleright, \triangleright$ is affluent [2, 1];* **2.** *$\rightarrow \cdot \rightarrow \subseteq \rightarrow$ (transitivity) [3, pp. 31,32][2, 8, 6, 1].*

[1] A reduction is *maximal* if it is infinite or ends in a normal form [8]. *Computations* in [6] are maximal.
[2] That is, we cannot have the other *operand of the union* in the definition of affluence, Def. 1 here $\hat{b} \blacktriangleright \ldots$.

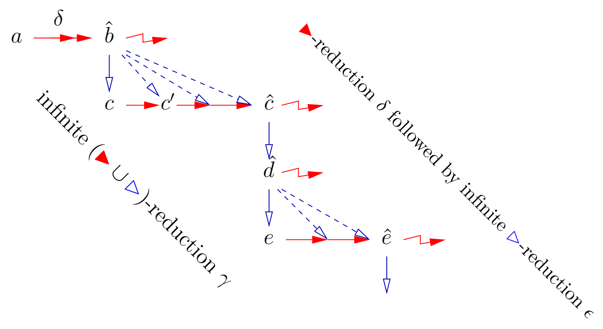
a
δ
$\hat{b}$
c
c'
$\hat{c}$
$\hat{d}$
e
$\hat{e}$
infinite ($\blacktriangleright \cup \triangleright$)-reduction γ
$\blacktriangleright$-reduction δ followed by infinite $\triangleright$-reduction ϵ

Figure 1 Transformation of γ into $\delta \cdot \epsilon$ of shape $\twoheadrightarrow_{\blacktriangleright} \cdot \triangleright^{\omega}$ in the proof of Lem. 4 with $\gamma \,/\!\!/\!\!\nearrow \delta \cdot \epsilon$

We extend the above to finite families $(\to)_I$ of rewrite relations with I ordered by $<$.

▶ **Definition 6.** *For $\mathcal{F} := (\to)_I$ a family of rewrite relations, $\mathcal{F}$ is* affluent *if $\to_{>i} \cdot \to_i \subseteq \to := \bigcup \mathcal{F}$ for $i \in I$, and* affluent for *a reduction γ if $(\to\restriction\gamma)_I$ is affluent.*

Affluence entails affluence for any reduction γ and $\bigcup(\to\restriction\gamma)_I = (\bigcup \mathcal{F})\restriction\gamma$, by distributivity. From now on we let index-sets range over intervals $[\ell, n]$ of natural numbers. A reduction δ is *progressive* if $\to_i$-steps precede $\to_j$-steps in δ for $i < j$, except possibly for an infinite $\to_k$-tail for some k, so has shape $\twoheadrightarrow_\ell \cdot \ldots \cdot \twoheadrightarrow_n (\cdot \to_k^\omega)$ with the last part optional, and *preferential* if from any source of a $\to_j$-step in δ there's no $\to_i$-step, for $i < j$.

▶ **Theorem 7.** *Let γ be a reduction with $\to\restriction\gamma = \to = \bigcup \mathcal{F}$ with $\mathcal{F} := (\to)_{[\ell,n]}$ and let $\nearrow$ be defined as in Def. 3 under these assumptions. If $\mathcal{F}$ is affluent, then $\gamma \nearrow \epsilon$ for some ϵ.*

Proof. By induction on $n \dot{-} \ell$ with base case Lem. 4. Otherwise, the $\to$-reduction γ can be seen as a $(\blacktriangleright \cup \triangleright)$-reduction for $\blacktriangleright := \to_\ell$ and $\triangleright := \bigcup(\to)_J$ with $J := [\ell+1, n]$. Then $\blacktriangleright, \triangleright$ is affluent, so $\gamma \,/\!\!/\!\!\nearrow \hat{\gamma}$ by Lem. 4 with $\hat{\gamma}$ of shape either $\twoheadrightarrow_{\blacktriangleright} \cdot \twoheadrightarrow_{\triangleright} \cdot \blacktriangleright^\omega$ or $\twoheadrightarrow_{\blacktriangleright} \cdot \triangleright^\alpha$ for $\alpha \leq \omega$. In either case let δ be the $\triangleright$-subreduction of $\hat{\gamma}$. The family $(\to\restriction\delta)_J$ is affluent for δ since not only are the objects in δ objects in γ by $\gamma \,/\!\!/\!\!\nearrow \hat{\gamma}$, they are (when source of a step in δ) in $\blacktriangleright\restriction\gamma$-normal form, entailing $(\to\restriction\delta)_J$-steps compose to other such, cannot compose to $\blacktriangleright$-steps (*Cf.* the proof of the claim in Lem. 4, footnote 2). Hence the IH applies to δ yielding $\delta \nearrow \hat{\delta}$ with $\hat{\delta}$ of shape $\twoheadrightarrow_{\ell+1} \cdot \ldots \cdot \twoheadrightarrow_n (\cdot \to_k^\omega)$ with the last infinite part (for some $\ell + 1 \leq k \leq n$) optional. Setting ϵ to the reduction obtained by substituting[3] $\hat{\delta}$ for δ in $\hat{\gamma}$ we conclude to $\gamma \nearrow \epsilon$. ◀

▶ **Corollary 8.** *Let $\mathcal{F} := (\to)_I$ and $\to := \bigcup \mathcal{F}$.* **1.** *$\mathcal{F}$ is terminating iff $\to$ is, for affluent $\mathcal{F}$;* **2.** *$\mathcal{F} = (\blacktriangleright, \triangleright, \gg)$ is terminating iff $\to$ is, if $(\triangleright \cdot \blacktriangleright) \cup (\gg \cdot \blacktriangleright) \cup (\gg \cdot \triangleright) \subseteq \to$ [1, Thm. 2];* **3.** *$\mathcal{F}$ is terminating iff $\to$ is, if $\to \cdot \to \subseteq \to$ (transitivity; disjunctive termination) [6].*

Whereas Cor. 8(3) cannot be proven directly from Cor. 5(2) **by induction** on family size, as argued on [7, p. 1218], it does not follow as implied there that one can't proceed by induction: we showed Thm. 7 from Lem. 4 **inductively**, having the former as consequences.

3 Jumping families

Sec. 2 was written such that its development (Def. 2,Def. 3,Lem. 4,Def. 6,Thm. 7) is preserved when **replacing** *affluence* and $\restriction$ everywhere by *jumping* and $|$, both defined next, where $|$ relaxes restriction $\restriction$ (Def. 2) to accommodate that jumping is *weaker*[4] than affluence.

[3] We use the convention that concatenating to an infinite reduction yields the infinite reduction.
[4] It can be further weakened to *yumping*, by adding $\blacktriangleright \cdot \to^\omega$ as a third disjunct to its right-hand side.

▶ **Definition 9.** $\blacktriangleright, \triangleright$ *is* jumping *if* $\triangleright \cdot \blacktriangleright \subseteq \triangleright \cup (\blacktriangleright \cdot \twoheadrightarrow)$ *for* $\to := \blacktriangleright \cup \triangleright$ *[2, 1]. For* $\to$*-reduction* γ*,* $|\gamma$ *restricts a rewrite relation to objects* c along γ*,* i.e. *to objects* c *such that the source of* γ *reduces to* c *and* c *reduces to the target of* γ *(if any) or is not terminating.*

Jumping entails jumping for any reduction γ using that if $\delta : a \twoheadrightarrow b$ for a, b along γ then all c in δ are along γ. If $\to|\gamma = \to$, only the target of γ (if any) can be a normal form, since if c is non-terminating, then *some* d with $c \to d$ is non-terminating too.

▶ **Lemma 10.** *For reduction* γ *with* $\to|\gamma = \to = \blacktriangleright \cup \triangleright$ *and* $\blacktriangleright, \triangleright$ *jumping,* $\gamma \nearrow\!\!\!\nearrow \hat{\gamma}$ *for some* $\hat{\gamma}$*.*

Proof. With the above replacements, the proof *is* that of Lem. 4 including it being illustrated by Fig. 1, where $\lightning$ now marks $\blacktriangleright$-normal forms *along* γ, not necessarily *in* γ, *cf.* [5, Fig. 7]. ◀

▶ **Example 11.** $\blacktriangleright, \triangleright$ given by $a' \triangleright a$ and $\gamma : a \triangleright b \blacktriangleright c$ and $\epsilon : a \blacktriangleright a' \blacktriangleright a' \blacktriangleright \ldots$, is jumping but not affluent: γ promotes (only) to ϵ with objects of γ all *along* γ, not all *in* γ (a' isn't).

▶ **Corollary 12.** *Let* $\to := \blacktriangleright \cup \triangleright$*.* **1.** $\blacktriangleright, \triangleright$ *are terminating iff* $\to$ *is, for jumping* $\blacktriangleright, \triangleright$ *[2];* **2.** $a \blacktriangleright\!\!\blacktriangleright \cdot \triangleright^{\omega}$ *if* a *is* $\blacktriangleright$*-terminating but not* $\to$*-terminating, and* $\triangleright \cdot \blacktriangleright \subseteq \blacktriangleright \cdot \triangleright$ *(*$\diamond$*) [4, Lem. 51].*

Proof. 2. If $a \to^{\omega} \nearrow\!\!\!\nearrow \hat{\gamma}$ then $\hat{\gamma}$ is not of shape $a \blacktriangleright\!\!\blacktriangleright \cdot \triangleright\!\!\triangleright \cdot \blacktriangleright^{\omega}$ as $\diamond$ entails $\triangleright\!\!\triangleright \cdot \blacktriangleright \subseteq \blacktriangleright \cdot \triangleright\!\!\triangleright$. ◀

Call a family $\mathcal{F}$ *jumping* [1] if $\to_{>i} \cdot \to_i \subseteq \to_{>i} \cup (\to_i \cdot \twoheadrightarrow_{\geq i})$ for $i \in I$.[5]

▶ **Theorem 13.** *Let* γ *be a reduction with* $\to|\gamma = \to = \bigcup \mathcal{F}$ *with* $\mathcal{F} := (\to)_{[\ell,n]}$ *and let* $\nearrow$ *be defined as in Def. 3 under these assumptions. If* $\mathcal{F}$ *is jumping, then* $\gamma \nearrow \epsilon$ *for some* ϵ*.*

Proof. With the above replacements, the proof *is* that of Thm. 7, using Lem. 10 instead of 4 with the IH applicable since $(\to|\delta)_J$-steps compose to other such by J being a *suffix* of I. ◀

▶ **Corollary 14.** *Let* $\mathcal{F} := (\to)_I$*,* $\to := \bigcup \mathcal{F}$*.* **1.** $\mathcal{F}$ *is terminating iff* $\to$ *is, for jumping* $\mathcal{F}$ *[1];* **2.** $\mathcal{F} = (\blacktriangleright, \triangleright, \gg)$ *is terminating iff* $\to$ *is, if* $(\triangleright \cup \gg) \cdot \blacktriangleright \subseteq (\triangleright \cup \gg) \cup (\blacktriangleright \cdot \to^*)$ *and* $\gg \cdot \triangleright \subseteq \gg \cup (\triangleright \cdot (\triangleright \cup \gg)^*)$ *[1, Thm. 8].*

4 Blending Affine and Jumping Families

Blending families [1] is limited only by (correctnes of) one's illusion. We give examples.

▶ **Lemma 15.** *Let* γ *be an* $\mathcal{F}$*-reduction for* $\mathcal{F} := (\to)_I$ *and* $I := [\ell, n]$*, and let* $\to := \bigcup \mathcal{F}$*.* $\gamma \nearrow \epsilon$ *for some* ϵ*, if:* **1.** $\to_{>i} \cdot \to_i \subseteq \to_{>\ell} \cup (\to_\ell \cdot \twoheadrightarrow)$ *for* $\ell \leq i \leq n$ *(*affluence$_{\curvearrowright}$*); or* **2.** $\blacktriangleright_i, \to_{>i}$ *is affluent for all* i *(*partite*), for* ϵ *a* $\mathcal{G}$*-reduction* ϵ*,*[6] *with* $\mathcal{G} := (\blacktriangleright)_{[\ell,n]}$ *and* $\blacktriangleright_i := \to_i^+$ *for* $i < n$ *and* $\blacktriangleright_n := \to_n$*; or* **3.** $\to_{>i} \cdot \to_i \subseteq \to_{>i} \cup \to_i^+ \cup (\to_\ell \cdot \twoheadrightarrow)$ *for* $\ell \leq i < n$ *(*partite$_{\curvearrowright}$*).*

Proof. **1.** γ can be seen as a $(\to_\ell|\gamma, \to_{>\ell}|\gamma)$-reduction. By assumption and Lem. 10, $\gamma \nearrow\!\!\!\nearrow \hat{\gamma}$ for some $\hat{\gamma}$. Let δ be the $\to_{>\ell}|\gamma$-subreduction of $\hat{\gamma}$. Viewed as a $(\to_{>\ell}|\gamma){\restriction}\delta$-reduction, Thm. 7 yields $\delta \nearrow \hat{\delta}$ for some $\hat{\delta}$, and we conclude by setting ϵ to $\hat{\gamma}$ in which δ is replaced by $\hat{\delta}$. Thm. 7 is applicable to δ since the second disjunct of affluence$_{\curvearrowright}$ cannot hold, reducing it to affluence: if $a \to_\ell c \twoheadrightarrow b$ for a, b in δ, with a the source of some $(\to_{>\ell}|\gamma)$-step in δ, then a, b and hence c would be along γ by δ being part of $\hat{\gamma}$, contradicting that a be in $\to_\ell|\gamma$-normal form per $\gamma \nearrow\!\!\!\nearrow \hat{\gamma}$. **2.** Adaptating the proof of Thm. 7 using *bait and switch* in the induction: a $\to_{\geq i}$-reduction γ (the bait) can be seen as a $(\blacktriangleright_i{\restriction}\gamma, \to_{>i}{\restriction}\gamma)$-reduction (the switch) since $\to_i \subseteq \to_i^+$. By assumption and Lem. 4 that promotes to some $\hat{\gamma}$, from which we conclude by applying the IH to its $\to_{>i}{\restriction}\gamma$-subreduction, yielding ϵ; **3.** As for 1 but using 2 instead of Thm. 7. ◀

[5] We based jumping of $(\to)_{[\ell,n]}$ on that of $(\to)_{[\ell+1,n]}$. For basing it on $(\to)_{[\ell,n\dot{-}1]}$ see [1, Thm. 7, Cor. 20].
[6] Though objects in ϵ must be in γ, this no longer holds if we unfold its $\to_i^+$-steps into single $\to_i$-steps.

▶ **Corollary 16.** *$\mathcal{F} := (\to)_I$ is terminating iff $\to := \bigcup \mathcal{F}$ is, if:* **1.** *affluence$_{\curvearrowright}$; or* **2.** *partite; or* **3.** *partite$_{\curvearrowright}$ [1, Thm. 22]; or* **4.** *$\mathcal{F} = (\blacktriangleright, \triangleright, \gg)$ and $(\triangleright \cup \gg) \cdot \blacktriangleright \subseteq \triangleright \cup \gg \cup (\blacktriangleright \cdot (\blacktriangleright \cup \triangleright \cup \gg)^*)$ and $\gg \cdot \triangleright \subseteq \gg \cup \triangleright^+ \cup (\blacktriangleright \cdot (\blacktriangleright \cup \triangleright \cup \gg)^*)$ (jumping$_3$) [1, Thm. 4]; or* **5.** *for some $k \leq n$, $\to_{>i} \cdot \to_i \subseteq \to_{>0} \cup (\to_0 \cdot \twoheadrightarrow)$ for $i < k$, and $\to_{>i} \cdot \to_i \subseteq \to_{>i} \cup (\to_i \cdot \twoheadrightarrow_{\geq i})$ for $k \leq i < n$ (affluence$^{\curvearrowright}_{\curvearrowright}$); or* **6.** *for some $k \leq n$, $\to_{>i} \cdot \to_i \subseteq \to_{>i} \cup \to_i^+ \cup (\to_0 \cdot \twoheadrightarrow)$ for $i < k$, and $\to_{>i} \cdot \to_i \subseteq \to_{>i} \cup (\to_i \cdot \twoheadrightarrow_{\geq i})$ for $k \leq i < n$ (partite$^{\curvearrowright}_{\curvearrowright}$) [1, Thm. 28].*

▶ **Example 17.** $\to_{>i} \cdot \to_i \subseteq \to \cup (\to_i \cdot \twoheadrightarrow_{\geq i})$ for all i, may seem a harmless blend of affluence and jumping, but though it holds for the terminating family $\mathcal{F} := (\blacktriangleright, \triangleright, \gg)$ [1, Ex. 9(a)] given by $b \blacktriangleright d$, $c \triangleright d \triangleright a \triangleright b$, and $a \gg d$, $b \gg c$, $\bigcup \mathcal{F}$ is non-terminating: $a \gg d \triangleright a$.[7]

5 Conclusions

By modularising (factoring through promotion Def. 3) and unifying (transitivity, affluence, jumping via restriction (Defs. 2 and 9)) proofs, we related disjointed results on disjunctive termination [3, 2, 6, 1] and improved upon them (*e.g.*, Cor. 8(1)). Ideas for further research: (I) Recast using 2-*rewriting* for *transducing* (infinite) reductions; (II) Exploit progressiveness and sharpen promotion to (re)gain *quantitative* results, like [7] and *quasi-commutation*; (III) Automate results in *tools* (to handle, *e.g.*, Ex. 18) and *formalise* them (*axiomatically?*).

▶ **Example 18.** **1.** [6, Fig. 2 (**CHOICE**)] presents a program having transition relation R given by relating pairs of natural numbers $\langle x, y\rangle$ and $\langle x', y'\rangle$ if the latter is either $\langle x \dot{-} 1, x\rangle$ or $\langle y \dot{-} 2, x + 1\rangle$, assuming $x, y > 0$, and a doubleton family $\mathcal{F} := (\blacktriangleright, \triangleright)$ of relations $\blacktriangleright := \neg P(x, y) \land (Q \lor P(x', y'))$ and $\triangleright := P(x, y) \land Q \land P(x', y')$ for $Q := x + y > x' + y'$ and $P(n, m) := m \dot{-} 2 \leq n \leq m \dot{-} 1$. Then $R \subseteq \to := \bigcup \mathcal{F}$, $\triangleright \cdot \blacktriangleright = \emptyset$, and $\mathcal{F}$ is terminating since Q is and since P and $\neg P$ do not compose, yielding affluence of $\blacktriangleright, \triangleright$ hence termination of $\to$ by Cor. 5(1), so R is terminating. **2.** [7, Ex. 6.1] presents a program having transition relation R given by $\langle x, y\rangle \to_{x>y\&x>0\&y>0} \langle y, 2^{x+y}\rangle$ and $\langle x, y\rangle \to_{\neg(x>y)\&x>0\&y>0} \langle x, y - 1\rangle$, and a doubleton family $\mathcal{F} := (\blacktriangleright, \triangleright)$ of terminating relations $\blacktriangleright := \{(\langle x, y\rangle, \langle x', y'\rangle) \mid x > 0 \& x > x'\}$ and $\triangleright := \{(\langle x, y\rangle, \langle x', y'\rangle) \mid y > 0 \;\&\; y > y'\}$. Then $R \subseteq \bigcup \mathcal{F}$ but affluence of $\mathcal{F}$ fails. Restricting $\triangleright$ by $\{(\langle x, y\rangle, \langle x', y'\rangle) \mid x \geq x'\}$ guided by R, both hold and Cor. 5(1) applies.

References

1 J.E. Dawson, N. Dershowitz, and R. Goré. Well-founded unions. In *IJCAR*, volume 10900 of *LNCS*, pages 117–133. Springer, 2018. doi:10.1007/978-3-319-94205-6_9.

2 H. Doornbos and B. von Karger. On the union of well-founded relations. *Log. J. IGPL*, 6(2):195–201, 1998. doi:10.1093/jigpal/6.2.195.

3 A. Geser. *Relative Termination.* PhD thesis, Universität Passau, 1990.

4 A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Inform. Comput.*, 205(4):512–534, 2007.

5 V. van Oostrom and H. Zantema. Triangulation in rewriting. In *RTA*, volume 15 of *LIPIcs*, pages 240–255, 2012. doi:10.4230/LIPIcs.RTA.2012.240.

6 A. Podelski and A. Rybalchenko. Transition invariants. In *LICS*, pages 32–41, 2004. (Winner of a LICS Test of Time Award 2024). doi:10.1109/LICS.2004.1319598.

7 S. Steila and K. Yokoyama. Reverse mathematical bounds for the termination theorem. *APAL*, 167(12):1213–1241, 2016. doi:10.1016/j.apal.2016.06.001.

8 Terese. *Term Rewriting Systems.* CUP, 2003.

[7] Such (very) finite counterexamples can be found automatically, *e.g.*, by Zantema's tool Carpa.

Non-Termination of Term Rewrite Systems Using Pattern Unfolding

Etienne Payet
LIM - Université de la Réunion, France

Abstract

We present a revisit, based on a new unfolding technique, of an approach introduced in term rewriting for the automatic detection of infinite non-looping derivations from patterns of rules.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Program analysis

Keywords and phrases Non-Termination, Non-Looping, Unfolding, Term Rewriting

1 Introduction

A derivation w.r.t. a term rewrite system (TRS) is called *non-looping* if it does not contain any loop, *i.e.*, any finite rewrite sequence where an instance of the starting term re-occurs as a subterm of the last term. In this paper, we present a work in progress on the automatic detection of infinite non-looping derivations w.r.t. a given TRS. We describe a reformulation of the approach of [2]: our contribution is the replacement of the nine inference rules of [2] for producing *pattern rules*, together with the strategy for their automated application, by a new unfolding technique. All we can say at the moment is that it provides a more compact presentation than that of [2], but we still need to compare the two approaches more precisely.

2 Preliminaries

$\mathbb{N}$ denotes the set of natural numbers. For all binary relations $\Rightarrow$ on a set A, a $\Rightarrow$-*chain* is a (possibly infinite) sequence $a_0 \Rightarrow a_1 \Rightarrow \cdots$ and $\Rightarrow^+$ (resp. $\Rightarrow^*$) denotes the transitive (resp. reflexive and transitive) closure of $\Rightarrow$.

2.1 Terms

We use the same definitions as [1] for terms. From now on, we fix a signature Σ and a set $H = \{\square_n \mid n \in \mathbb{N} \setminus \{0\}\}$ of constant symbols (called *holes*) disjoint from Σ. We also fix an infinite countable set X of variables disjoint from $\Sigma \cup H$. We let $S(\Sigma, X)$ denote the set of all substitutions from X to $T(\Sigma, X)$. We let $mgu(s,t)$ denote the set of most general unifiers of the terms (or sequences of terms) s and t. We denote function symbols by words in the *sans serif* font, *e.g.*, f, $\mathsf{0}$, while... We use the superscript notation to denote several successive applications of a unary function symbol, *e.g.*, $\mathsf{s}^2(\mathsf{0})$ stands for $\mathsf{s}(\mathsf{s}(\mathsf{0}))$ and $\mathsf{s}^0(\mathsf{0}) = \mathsf{0}$.

Let n be a positive integer. An *n-context* is an element of $T(\Sigma \cup H, X)$ that contains occurrences of $\square_1, \ldots, \square_n$ but no occurrence of another hole. For all n-contexts c and all $s_1, \ldots, s_n \in T(\Sigma \cup H, X)$, we let $c(s_1, \cdots, s_n)$ denote the element of $T(\Sigma \cup H, X)$ obtained from c by replacing all the occurrences of $\square_i$ by s_i, for all $1 \leq i \leq n$. We use the superscript notation for denoting several successive embeddings of a 1-context c into itself: $c^0 = \square_1$ and, for all $n \in \mathbb{N}$, $c^{n+1} = c(c^n)$. For all $n \in \mathbb{N}$, we denote by $\chi^{(n)}$ the set of n-contexts that contain exactly one occurrence of $\square_1, \ldots, \square_n$.

2.2 Term Rewriting

A *rule* is an element of $T(\Sigma, X)^2$ and a *term rewrite system (TRS)* is a set of rules. Given a rule (u, v), we let $[(u, v)] = \{(u\gamma, v\gamma) \mid \gamma \text{ is a renaming}\}$ denote its *equivalence class modulo renaming.* Moreover, for all TRSs $\mathcal{R}$, we let $[\mathcal{R}] = \bigcup_{r\in\mathcal{R}}[r]$. The rules of a TRS allow one to rewrite terms. This is formalised by the following binary relation $\to_{\mathcal{R}}$.

▶ **Definition 1.** *Let $\mathcal{R}$ be a TRS. We let $\to_{\mathcal{R}} = \bigcup\{\to_r \mid r \in \mathcal{R}\}$ where, for all $r = (u, v) \in \mathcal{R}$, $\to_r = \{(c(u\theta), c(v\theta)) \in T(\Sigma, X)^2 \mid c \in \chi^{(1)},\ \theta \in S(\Sigma, X)\}$. A rule (u, v) is* correct *w.r.t. $\mathcal{R}$ if $u \to^+_{\mathcal{R}} v$ holds. A TRS is correct w.r.t. $\mathcal{R}$ if all its rules are. We say that $\mathcal{R}$ is* non-terminating *(or* does not terminate*) if there exists an infinite $\to_{\mathcal{R}}$-chain.*

A *loop* in a TRS $\mathcal{R}$ is a finite $\to_{\mathcal{R}}$-chain of the form $s \to_{r_1} \cdots \to_{r_n} c(s\theta)$ where $s \in T(\Sigma, X)$, $r_1, \dots, r_n \in \mathcal{R}$, $c \in \chi^{(1)}$ and $\theta \in S(\Sigma, X)$. It gives rise to an infinite $\to_{\mathcal{R}}$-chain $s \to_{r_1} \cdots \to_{r_n} c(s\theta) \to_{r_1} \cdots \to_{r_n} c(c\theta(s\theta^2)) \to_{r_1} \cdots$. We say that a $\to_{\mathcal{R}}$-chain is *non-looping* if it does not contain any loop.

We unfold TRSs as follows. For the sake of readability, we write $[(u, v) \mid \dots]$ instead of $[\{(u, v) \mid \dots\}]$. Moreover, $(r_1, \dots, r_n) \ll_r [R]$ means that $(r_1, \dots, r_n)$ is a sequence of elements of $[R]$ variable disjoint from r and from each other.

▶ **Definition 2** (Unfolding). *For all TRSs $\mathcal{R}$ and R, we let*

$$U_{\mathcal{R}}(R) = \left[(u\theta, c(v_1, \dots, v_n)\theta) \;\middle|\; \begin{array}{l} r = (u, c(s_1, \dots, s_n)) \in \mathcal{R},\ c \in \chi^{(n)} \\ ((u_1, v_1), \dots, (u_n, v_n)) \ll_r [R] \\ \theta = mgu((s_1, \dots, s_n), (u_1, \dots, u_n)) \end{array} \right]$$

The unfolding of $\mathcal{R}$ from R *is the set $unf(\mathcal{R}, R) = (U_{\mathcal{R}})^*(R)$.*

▶ **Proposition 3.** *Let $\mathcal{R}$ and R be TRSs such that R is correct w.r.t. $\mathcal{R}$. Then, for all $n \in \mathbb{N}$, $U^n_{\mathcal{R}}(R)$ is correct w.r.t. $\mathcal{R}$, which implies that $unf(\mathcal{R}, R)$ also is.*

▶ **Example 4.** Let $\mathcal{R}$ be the TRS which consists of the rules

$$\begin{array}{ll} r_1 = (\mathsf{while}(\mathsf{true}, x, y), \mathsf{while}(\mathsf{gt}(x, y), \mathsf{add}(x, y), \mathsf{s}(y))) & r_4 = (\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{gt}(x, y)) \\ r_2 = (\mathsf{gt}(\mathsf{s}(x), \mathsf{0}), \mathsf{true}) & r_5 = (\mathsf{add}(x, \mathsf{0}), x) \\ r_3 = (\mathsf{gt}(\mathsf{0}, y), \mathsf{false}) & r_6 = (\mathsf{add}(x, \mathsf{s}(y)), \mathsf{s}(\mathsf{add}(x, y))) \end{array}$$

and which corresponds to the imperative program fragment

```
while (x > y) { x = x + y; y = y + 1; }
```

Note that this fragment does not terminate if it is run from integers x, y such that $x > y > 0$. Moreover, for all $n > m > 0$, we have the infinite $\to_{\mathcal{R}}$-chain

$$\mathsf{while}(\mathsf{true}, \mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0})) \left(\underset{r_1}{\to} \circ \overset{m}{\underset{r_4}{\to}} \circ \underset{r_2}{\to} \circ \overset{m}{\underset{r_6}{\to}} \circ \underset{r_5}{\to}\right) \mathsf{while}(\mathsf{true}, \mathsf{s}^{n+m}(\mathsf{0}), \mathsf{s}^{m+1}(\mathsf{0}))$$
$$\left(\underset{r_1}{\to} \circ \overset{m+1}{\underset{r_4}{\to}} \circ \underset{r_2}{\to} \circ \overset{m+1}{\underset{r_6}{\to}} \circ \underset{r_5}{\to}\right) \cdots$$

It is non-looping because the number of applications of r_4 and r_6 gradually increases. Let us compute some elements of $unf(\mathcal{R}, \mathcal{R})$ by applying Def. 2.

- We have $r_4 = (\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y)), c(\mathsf{gt}(x, y))) \in \mathcal{R}$ where $c = \square_1 \in \chi^{(1)}$. Moreover, we have $(\mathsf{gt}(\mathsf{s}(x_1), \mathsf{0}), \mathsf{true}) \ll_{r_4} [\mathcal{R}]$ and $\theta = \{x \mapsto \mathsf{s}(x_1), y \mapsto \mathsf{0}\} = mgu(\mathsf{gt}(x, y), \mathsf{gt}(\mathsf{s}(x_1), \mathsf{0}))$. So, $r^{\mathsf{gt}}_1 = (\mathsf{gt}(\mathsf{s}(x), \mathsf{s}(y))\theta, c(\mathsf{true})\theta) \in U_{\mathcal{R}}(\mathcal{R})$ with $r^{\mathsf{gt}}_1 = (\mathsf{gt}(\mathsf{s}^2(x_1), \mathsf{s}(\mathsf{0})), \mathsf{true})$.

- More generally, for all $n \in \mathbb{N}$, $[r_n^{\mathsf{gt}}] \subseteq U_{\mathcal{R}}^n(\mathcal{R})$ with $r_n^{\mathsf{gt}} = \big(\mathsf{gt}(\mathsf{s}^{n+1}(x), \mathsf{s}^n(0)), \mathsf{true}\big)$.
- Identically, from r_6 and r_5 one gets: for all $n \in \mathbb{N}$, $[r_n^{\mathsf{add}}] \subseteq U_{\mathcal{R}}^n(\mathcal{R})$ with $r_n^{\mathsf{add}} = \big(\mathsf{add}(x, \mathsf{s}^n(0)), \mathsf{s}^n(x)\big)$.
- Let $n \in \mathbb{N}$. From r_1, r_n^{gt} and r_n^{add} one gets: $[r_n^{\mathsf{while}}] \subseteq U_{\mathcal{R}}^{n+1}(\mathcal{R})$ where

$$r_n^{\mathsf{while}} = (\mathsf{while}(\mathsf{true}, \mathsf{s}^{n+1}(x), \mathsf{s}^n(0)), \mathsf{while}(\mathsf{true}, \mathsf{s}^{2n+1}(x), \mathsf{s}^{n+1}(0)))$$

  As $\mathcal{R}$ is correct w.r.t. $\mathcal{R}$, $U_{\mathcal{R}}^{n+1}(\mathcal{R})$ also is (Prop. 3), *i.e.*, r_n^{while} also is. So, we have

$$\mathsf{while}(\mathsf{true}, \mathsf{s}^{n+1}(x), \mathsf{s}^n(0)) \xrightarrow[\mathcal{R}]{+} \mathsf{while}(\mathsf{true}, \mathsf{s}^{2n+1}(x), \mathsf{s}^{n+1}(0))$$

For all $n > m > 0$, we also have the infinite $\rightarrow_{unf(\mathcal{R},\mathcal{R})}$-chain

$$\mathsf{while}(\mathsf{true}, \mathsf{s}^n(0), \mathsf{s}^m(0)) \underset{r_m^{\mathsf{while}}}{\rightarrow} \mathsf{while}(\mathsf{true}, \mathsf{s}^{n+m}(x_1), \mathsf{s}^{m+1}(0)) \underset{r_{m+1}^{\mathsf{while}}}{\rightarrow} \cdots$$

It is non-looping because a new rule (not occurring before) is used at each step.

3 Pattern Unfolding

Now, we describe a reformulation, based on unfolding, of the pattern approach of [2].

▶ **Definition 5.** *A* pattern substitution *is a pair $\theta = (\sigma, \mu)$ of elements of $S(\Sigma, X)$. We rather denote it as $\sigma \star \mu$. For all $n \in \mathbb{N}$, we let $\theta(n) = \sigma^n\mu$. A* pattern term *is a pair $p = (s, \theta)$ where $s \in T(\Sigma, X)$ and θ is a pattern substitution. We denote it as $s \star \sigma \star \mu$ if $\theta = \sigma \star \mu$. For all $n \in \mathbb{N}$, we let $p(n) = s\theta(n)$. For all $s \in T(\Sigma, X)$, we let $s^\star = s \star \emptyset \star \emptyset$.*

For instance, if $\sigma = \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{s}(y)\}$ and $\mu = \{x \mapsto \mathsf{s}(x), y \mapsto 0\}$ then $\theta = \sigma \star \mu$ is a pattern substitution and $p = \mathsf{gt}(x, y) \star \sigma \star \mu$ is a pattern term. For all $n \in \mathbb{N}$, we have $\theta(n) = \sigma^n\mu = \{x \mapsto \mathsf{s}^{n+1}(x), y \mapsto \mathsf{s}^n(0)\}$ and $p(n) = \mathsf{gt}(x, y)\sigma^n\mu = \mathsf{gt}(\mathsf{s}^{n+1}(x), \mathsf{s}^n(0))$.

From pattern terms one can define pattern rules.

▶ **Definition 6.** *A* pattern rule *is a pair $r = (p, q)$ of pattern terms. It describes the set $rules(r) = \{(p(n), q(n)) \mid n \in \mathbb{N}\} \subseteq T(\Sigma, X)^2$.*

▶ **Example 7** (Ex. 4 continued)**.** Let $u = \mathsf{while}(\mathsf{true}, x, y)$ be the left-hand side of r_1, $\sigma = \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{s}(y)\}$, $\sigma' = \{x \mapsto \mathsf{s}(x)\}$ and $\mu = \{x \mapsto \mathsf{s}(x), y \mapsto 0\}$. The pattern terms

$$\begin{aligned} p &= u\sigma \star \sigma \star \mu = \mathsf{while}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) \star \sigma \star \mu \\ q &= u\sigma^2 \star \sigma\sigma' \star \mu = \mathsf{while}(\mathsf{true}, \mathsf{s}^2(x), \mathsf{s}^2(y)) \star \{x \mapsto \mathsf{s}^2(x), y \mapsto \mathsf{s}(y)\} \star \mu \end{aligned}$$

respectively describe the sets of terms $\big\{p(n) = \mathsf{while}\big(\mathsf{true}, \mathsf{s}^{n+2}(x), \mathsf{s}^{n+1}(0)\big) \bigm| n \in \mathbb{N}\big\}$ and $\big\{q(n) = \mathsf{while}\big(\mathsf{true}, \mathsf{s}^{2n+3}(x), \mathsf{s}^{n+2}(0)\big) \bigm| n \in \mathbb{N}\big\}$. Moreover,

$$\begin{aligned} rules((p, q)) &= \big\{\big(\mathsf{while}(\mathsf{true}, \mathsf{s}^{n+2}(x), \mathsf{s}^{n+1}(0)), \mathsf{while}(\mathsf{true}, \mathsf{s}^{2n+3}(x), \mathsf{s}^{n+2}(0))\big) \bigm| n \in \mathbb{N}\big\} \\ &= \big\{r_n^{\mathsf{while}} \mid n > 0\big\} \subseteq \big\{r_n^{\mathsf{while}} \mid n \in \mathbb{N}\big\} \subseteq unf(\mathcal{R}, \mathcal{R}) \text{ (see Ex. 4)} \end{aligned}$$

We adapt the notion of correctness (Def. 1) to pattern rules (and we get the notion defined in [2]).

▶ **Definition 8.** *Let $\mathcal{R}$ be a TRS. A pattern rule r is* correct *w.r.t. $\mathcal{R}$ if $rules(r)$ is. A set of pattern rules is correct w.r.t. $\mathcal{R}$ if all its elements are.*

So, if a pattern rule (p, q) is correct w.r.t. $\mathcal{R}$ then $p(n) \to^+_{\mathcal{R}} q(n)$ holds for all $n \in \mathbb{N}$. In Ex. 7, we have $\mathit{rules}((p, q)) \subseteq \mathit{unf}(\mathcal{R}, \mathcal{R})$. As $\mathcal{R}$ is correct w.r.t. $\mathcal{R}$, $\mathit{unf}(\mathcal{R}, \mathcal{R})$ also is (Prop. 3), *i.e.*, (p, q) also is. So, $\mathsf{while}(\mathsf{s}^{n+2}(x), \mathsf{s}^{n+1}(\mathsf{0})) \to^+_{\mathcal{R}} \mathsf{while}(\mathsf{s}^{2n+3}(x), \mathsf{s}^{n+2}(\mathsf{0}))$ holds for all $n \in \mathbb{N}$.

The next result allows one to infer correct pattern rules from a TRS. It considers pairs of rules that have the same form as (r_4, r_2), (r_4, r_3) and (r_6, r_5) in Ex. 4.

▶ **Proposition 9.** *Suppose that a TRS $\mathcal{R}$ contains two rules $r = (u, v)$, $r' = (u', v')$ s.t.*

- $u = c(c_1(x_1), \dots, c_m(x_m))$, $v = c'(c(x_1, \dots, x_m))$, $u' = c(t_1, \dots, t_m)$,
- $c_1, \dots, c_m, c'$ *are 1-contexts and c is an m-context with $\mathit{Var}(c_1, \dots, c_m, c', c) = \emptyset$,*
- $x_1, \dots, x_m \in X$ *are distinct and* $t_1, \dots, t_m \in T(\Sigma, X)$.

Let $\sigma = \{x_k \mapsto c_k(x_k) \mid 1 \le k \le m\}$ and $\mu = \{x_k \mapsto t_k \mid 1 \le k \le m\}$. Then, the pattern rule $\big(c(x_1, \dots, x_m) \star \sigma \star \mu, x_1 \star \{x_1 \mapsto c'(x_1)\} \star \{x_1 \mapsto v'\}\big)$ is correct w.r.t. $\mathcal{R}$

▶ **Example 10.** In Ex. 4, we have $r_4 = \big(c(c_1(x), c_2(y)), c'(c(x, y))\big)$ and $r_2 = (c(t_1, t_2), \mathsf{true})$ for $c = \mathsf{gt}(\square_1, \square_2)$, $c_1 = c_2 = \mathsf{s}(\square_1)$, $c' = \square_1$, $t_1 = \mathsf{s}(x)$ and $t_2 = \mathsf{0}$. So, by Prop. 9, (p_1, q_1) is correct w.r.t. $\mathcal{R}$ where $p_1 = \mathsf{gt}(x, y) \star \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{s}(y)\} \star \{x \mapsto \mathsf{s}(x), y \mapsto \mathsf{0}\}$ and $q_1 = x \star \emptyset \star \{x \mapsto \mathsf{true}\}$. Identically, from r_6 and r_5 one gets: (p_2, q_2) is correct w.r.t. $\mathcal{R}$ where $p_2 = \mathsf{add}(x, y) \star \{y \mapsto \mathsf{s}(y)\} \star \{y \mapsto \mathsf{0}\}$ and $q_2 = x \star \{x \mapsto \mathsf{s}(x)\} \star \emptyset$.

Unification for pattern terms is not considered in [2]. As we need it in our development (see Def. 13 below), we define it here.

▶ **Definition 11.** *Let p and q be pattern terms and θ be a pattern substitution. Then, θ is a* unifier *of p and q if $p(n)\theta(n) = q(n)\theta(n)$ for all $n \in \mathbb{N}$. Moreover, θ is a* most general unifier *(mgu) of p and q if $\theta(n) \in \mathit{mgu}(p(n), q(n))$ for all $n \in \mathbb{N}$. We let $\mathit{mgu}(p, q)$ be the set of all mgu's of p and q. All this is naturally extended to finite sequences of pattern terms.*

In Sect. 2.2, we have defined the equivalence class of a rule modulo renaming. We also need to adapt this concept to pattern rules.

▶ **Definition 12.** *For all pattern rules r, we let $[r]$ be the set of all pattern rules r' such that $\mathit{rules}(r') \subseteq [\mathit{rules}(r)]$. Moreover, for all sets of pattern rules R, we let $[R] = \bigcup_{r \in R}[r]$.*

Now, using the above concepts, we provide a counterpart of Def. 2 for pattern rules.

▶ **Definition 13.** *For all TRSs $\mathcal{R}$ and all sets of pattern rules R, we let*

$$U^{\pi}_{\mathcal{R}}(R) = \left[(p,q) \;\middle|\; \begin{array}{l} r = (u, c(s_1, \dots, s_n)) \in \mathcal{R},\ c \in \chi^{(n)} \\ ((p_1, v_1 \star \sigma_1 \star \mu_1), \dots, (p_n, v_n \star \sigma_n \star \mu_n)) \ll_r [R] \\ \sigma \star \mu \in \mathit{mgu}\left((s_1{}^{\star}, \dots, s_n{}^{\star}), (p_1, \dots, p_n)\right) \\ \sigma \text{ commutes with the } \sigma_i\text{s and } \mu_i\text{s} \\ p = u \star \sigma \star \mu \text{ and } q = c(v_1, \dots, v_n) \star \sigma_1 \dots \sigma_n \sigma \star \mu_1 \dots \mu_n \mu \end{array} \right]$$

The pattern unfolding of $\mathcal{R}$ from R *is the set $\mathit{patunf}(\mathcal{R}, R) = \big(U^{\pi}_{\mathcal{R}}\big)^*(R)$.*

▶ **Example 14** (Ex. 4 continued)**.** Let $R = \{(p_1, q_1), (p_2, q_2)\}$ (see Ex. 10) and

$$p'_1 = \mathsf{gt}(x_1, y_1) \star \{x_1 \mapsto \mathsf{s}(x_1), y_1 \mapsto \mathsf{s}(y_1)\} \star \{x_1 \mapsto \mathsf{s}(x_1), y_1 \mapsto \mathsf{0}\}$$
$$q'_1 = x_1 \star \emptyset \star \{x_1 \mapsto \mathsf{true}\}$$
$$p'_2 = \mathsf{add}(x_2, y_2) \star \{y_2 \mapsto \mathsf{s}(y_2)\} \star \{y_2 \mapsto \mathsf{0}\} \qquad q'_2 = x_2 \star \{x_2 \mapsto \mathsf{s}(x_2)\} \star \emptyset$$

Then, we have $((p'_1, q'_1), (p'_2, q'_2)) \ll_{r_1} [R]$ where $r_1 = (\mathsf{while}(\mathsf{true}, x, y), c(\mathsf{gt}(x, y), \mathsf{add}(x, y)))$ for $c = \mathsf{while}(\square_1, \square_2, \mathsf{s}(y)) \in \chi^{(2)}$. Moreover, $\rho \star \nu \in \mathit{mgu}\left(\left(\mathsf{gt}(x, y)^{\star}, \mathsf{add}(x, y)^{\star}\right), (p'_1, p'_2)\right)$

where $\rho = \{x \mapsto \mathsf{s}(x),\ y \mapsto \mathsf{s}(y),\ x_2 \mapsto \mathsf{s}(x_2)\}$ and $\nu = \{x \mapsto \mathsf{s}(x_1),\ y \mapsto 0,\ x_2 \mapsto \mathsf{s}(x_1)\}$. We note that ρ commutes with the substitutions of q'_1 and q'_2. So, $r^{\mathsf{while}} = (\mathsf{while}(\mathsf{true}, x, y) \star \rho \star \nu, c(x_1, x_2) \star \rho' \star \nu') \in U^{\pi}_{\mathcal{R}}(R)$ where $\rho' = \emptyset\{x_2 \mapsto \mathsf{s}(x_2)\}\rho = \{x \mapsto \mathsf{s}(x),\ y \mapsto \mathsf{s}(y),\ x_2 \mapsto \mathsf{s}^2(x_2)\}$ and $\nu' = \{x_1 \mapsto \mathsf{true}\}\emptyset\nu = \nu \cup \{x_1 \mapsto \mathsf{true}\}$.

The following result corresponds to the Soundness Thm. 7 of [2].

▶ **Theorem 15.** *Let $\mathcal{R}$ be a TRS and R be a set of pattern rules. If R is correct w.r.t. $\mathcal{R}$ then $patunf(\mathcal{R}, R)$ also is.*

Non-termination can be detected from a pattern rule using the following criterion.

▶ **Theorem 16** (Thm. 8 of [2]). *Let $(s \star \sigma_s \star \mu_s, t \star \sigma_t \star \mu_t)$ be correct w.r.t. a TRS $\mathcal{R}$ and let there be an $m \in \mathbb{N}$ such that $\sigma_t = \sigma_s^m \sigma'$ and $\mu_t = \mu_s \mu'$ for some $\sigma', \mu' \in S(\Sigma, X)$, where σ' commutes with σ_s and μ_s. If there is a $\pi \in Pos(t)$ and some $b \in \mathbb{N}$ such that $s\sigma_s^b = t|_\pi$, then $s\sigma_s^n \mu_s$ starts an infinite (possibly non-looping) $\to_{\mathcal{R}}$-chain for all $n \in \mathbb{N}$.*

We note that the infinite chain of Thm. 16 may contain a loop (*e.g.*, if $m = 1$ and $b = 0$). But, as the following example illustrates, this is not always the case.

▶ **Example 17** (Ex. 7 and Ex. 14 continued). Let us regard the pattern rule (p, q) of Ex. 7. As $rules((p, q)) \subseteq \{r_n^{\mathsf{while}} \mid n \in \mathbb{N}\} \subseteq [r_n^{\mathsf{while}} \mid n \in \mathbb{N}] = [rules(r^{\mathsf{while}})]$ (see Ex. 14), we have $(p, q) \in [r^{\mathsf{while}}]$ by Def. 12. So, as $[r^{\mathsf{while}}] \subseteq patunf(\mathcal{R}, R)$, we have $(p, q) \in patunf(\mathcal{R}, R)$. Moreover, as R is correct w.r.t. $\mathcal{R}$ (by Prop. 9), $patunf(\mathcal{R}, R)$ is correct w.r.t. $\mathcal{R}$ (by Thm. 15). So, (p, q) is correct w.r.t. $\mathcal{R}$. On the other hand, $(p, q) = (u\sigma \star \sigma \star \mu, u\sigma^2 \star \sigma\sigma' \star \mu)$ (see Ex. 7) and σ' commutes with σ and μ. Hence, by Thm. 16, for all $n \in \mathbb{N}$ the term $p(n) = u\sigma\sigma^n\mu = u\sigma^{n+1}\mu = \mathsf{while}(\mathsf{true}, \mathsf{s}^{n+2}(x), \mathsf{s}^{n+1}(0))$ starts an infinite $\to_{\mathcal{R}}$-chain. This implies that for all $n > m > 0$, $\mathsf{while}(\mathsf{true}, \mathsf{s}^n(0), \mathsf{s}^m(0))$ starts an infinite $\to_{\mathcal{R}}$-chain (Ex. 4).

4 Conclusion

We have presented a work in progress on the detection of infinite non-looping chains in TRSs. There are still many tasks to be completed. We have to implement our approach and to compare it with that of [2]: for the moment, we simply observed that it is a reformulation of [2], but we have to investigate further. Note that we already implemented a similar approach in logic programming [6] and that we tested it on logic programs obtained by translating TRSs introduced by the authors of [2] to evaluate their work. Our experiments suggest that our approach and that of [2] are not orthogonal and do not completely overlap. We also have to compare our work with other techniques for detecting non-looping chains [3, 4, 5].

References

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

2 F. Emmes, T. Enger, and J. Giesl. Proving non-looping non-termination automatically. In *Proc. IJCAR'12*, volume 7364 of *LNCS*, pages 225–240. Springer, 2012.

3 J. Endrullis and H. Zantema. Proving non-termination by finite automata. In *Proc. RTA'15*, volume 36 of *LIPIcs*, pages 160–176. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

4 M. Oppelt. Automatische Erkennung von Ableitungsmustern in nichtterminierenden Wortersetzungssystemen. Diploma Thesis, HTWK Leipzig, Germany, 2008.

5 É. Payet. Non-termination in term rewriting and logic programming. *Journal of Automated Reasoning*, 68(4):24 pages, 2024.

6 É. Payet. Non-termination of logic programs using patterns. To appear in Theory and Practice of Logic Programming, 2025.

A Dependency Pair Framework for Relative Termination of Term Rewriting*

Jan-Christoph Kassing
RWTH Aachen University, Aachen, Germany

Grigory Vartanyan
RWTH Aachen University, Aachen, Germany

Jürgen Giesl
RWTH Aachen University, Aachen, Germany

Dependency pairs (DPs) [1,4,6] are one of the most powerful techniques for proving termination of term rewrite systems (TRSs), and they are used in almost all tools for termination analysis of TRSs.

A combination of two TRSs (a *main* TRS $\mathcal{R}$ and a *base* TRS $\mathcal{B}$) is *relatively terminating* if there is no rewrite sequence that uses infinitely many steps with rules from $\mathcal{R}$ (whereas rules from $\mathcal{B}$ may be used infinitely often). Relative termination of TRSs has been studied since decades [3], and approaches based on relative rewriting are used for many applications.

However, while techniques and tools for analyzing ordinary termination of TRSs are very powerful due to the use of DPs, a direct application of standard DPs to analyze relative termination is not possible. Therefore, most existing approaches for automated analysis of relative termination are quite restricted in power. Hence, one of the largest open problems regarding DPs is Problem #106 of the RTA List of Open Problems [2]: *Can we use the dependency pair method to prove relative termination?* A first major step towards an answer to this question was presented in [7] by giving criteria for $\mathcal{R}$ and $\mathcal{B}$ that allow the use of ordinary DPs for relative termination.

Recently, we adapted DPs to prove almost-sure termination of probabilistic TRSs, by using *annotated dependency pairs (ADPs)* [8,9]. In this adaption, one considers all *defined* function symbols in the right-hand side of a rule at once, whereas ordinary DPs consider them separately.

We show that considering the defined symbols on right-hand sides separately (as for classical DPs) does not suffice for relative termination. On the other hand, we do not need to consider all of them at once either (i.e., we do not have to use the notion of ADPs from [8,9]). Instead, we introduce a new definition of ADPs that is suitable for relative termination and develop a corresponding ADP framework for automated relative termination proofs of TRSs. So while [7] presented conditions under which the *ordinary classical* DP framework can be used to prove relative termination, we develop the first *specific* DP framework for relative termination. We implemented our new ADP framework in our tool AProVE [5] and evaluate it in comparison to state-of-the-art tools for relative termination of TRSs.

Related Version Extended abstract of our IJCAR '24 paper [10]

References

1 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

2 Nachum Dershowitz. The RTA list of open problems. URL: https://www.cs.tau.ac.il/~nachum/rtaloop/.

* funded by the DFG Research Training Group 2236 UnRAVeL

3 Alfons Geser. *Relative Termination.* PhD thesis, University of Passau, Germany, 1990. URL: https://www.uni-ulm.de/fileadmin/website_uni_ulm/iui/Ulmer_Informatik_Berichte/1991/UIB-1991-03.pdf.

4 Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006. doi:10.1007/s10817-006-9057-7.

5 Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing program termination and complexity automatically with AProVE. *Journal of Automated Reasoning*, 58(1):3–31, 2017. doi:10.1007/s10817-016-9388-y.

6 Nao Hirokawa and Aart Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1-2):172–199, 2005. doi:10.1016/j.ic.2004.10.004.

7 José Iborra, Naoki Nishida, Germán Vidal, and Akihisa Yamada. Relative termination via dependency pairs. *Journal of Automated Reasoning*, 58(3):391–411, 2017. doi:10.1007/S10817-016-9373-5.

8 Jan-Christoph Kassing, Stefan Dollase, and Jürgen Giesl. A complete dependency pair framework for almost-sure innermost termination of probabilistic term rewriting. In *Proc. FLOPS '24*, LNCS 14659, pages 62–80, 2024. doi:10.1007/978-981-97-2300-3_4.

9 Jan-Christoph Kassing and Jürgen Giesl. Annotated dependency pairs for full almost-sure termination of probabilistic term rewriting. In *Principles of Verification: Cycling the Probabilistic Landscape*, LNCS 15260, pages 339–366, 2024. doi:10.1007/978-3-031-75783-9_14.

10 Jan-Christoph Kassing, Grigory Vartanyan, and Jürgen Giesl. A dependency pair framework for relative termination of term rewriting. In *Proc. IJCAR '24*, LNCS 14740, pages 360–380, 2024. doi:10.1007/978-3-031-63501-4_19.

Modularity of Termination in Probabilistic Term Rewriting

Jan-Christoph Kassing
RWTH Aachen University, Germany

Jürgen Giesl
RWTH Aachen University, Germany

Abstract

We investigate the modularity of probabilistic notions of termination in term rewriting. In the probabilistic setting, there are several interesting termination properties: Almost-sure termination (termination with probability 1), positive almost-sure termination (finite expected runtime of each rewrite sequence), and strong almost-sure termination (expected runtime is bounded by a constant for each start term). A property is called *modular* if it is preserved for certain unions of probabilistic term rewrite systems. We show that these three termination properties have different modularity behavior for innermost rewriting. Utilizing known relations between innermost and full probabilistic rewriting allows us to obtain modularity results for full probabilistic rewriting as well.

2012 ACM Subject Classification Theory of computation → Rewrite systems; Theory of computation → Equational logic and rewriting; Theory of computation → Logic and verification

Keywords and phrases Probabilistic Programming, Term Rewriting, Termination, Modularity

Related Version *See [8]. Full version, including all proofs*: `https://arxiv.org/abs/2409.17714`

Funding Funded by the DFG Research Training Group 2236 UnRAVeL.

1 Probabilistic Term Rewriting

We assume familiarity with non-probabilistic term rewriting [2] and briefly recapitulate probabilistic term rewriting, see, e.g., [1, 3, 6]. We write $\mathcal{T}(\Sigma, \mathcal{V})$ for the set of all *terms* over a (possibly infinite) countable set of *function symbols* $\Sigma = \biguplus_{k \in \mathbb{N}} \Sigma_k$ and a (possibly infinite) countable set of *variables* $\mathcal{V}$. In contrast to ordinary term rewrite systems (TRSs), a probabilistic term rewrite system (PTRS) has finite multi-distributions on the right-hand sides of its rewrite rules. A finite *multi-distribution* μ on $\mathcal{T}(\Sigma, \mathcal{V})$ is a finite multiset of pairs $(p : t)$, where $0 < p \leq 1$ is a probability and $t \in \mathcal{T}(\Sigma, \mathcal{V})$, such that $\sum_{(p:t) \in \mu} p = 1$. $\operatorname{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ is the set of all finite multi-distributions on $\mathcal{T}(\Sigma, \mathcal{V})$. For $\mu \in \operatorname{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$, its *support* is the multiset $\operatorname{Supp}(\mu) = \{t \mid (p : t) \in \mu \text{ for some } p\}$. A *probabilistic rewrite rule* is a pair $(\ell \to \mu) \in \mathcal{T}(\Sigma, \mathcal{V}) \times \operatorname{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ such that $\ell \notin \mathcal{V}$ and $\mathcal{V}(r) \subseteq \mathcal{V}(\ell)$ for every $r \in \operatorname{Supp}(\mu)$. A *probabilistic term rewrite system* is a (possibly infinite) countable set $\mathcal{P}$ of probabilistic rewrite rules. Similar to TRSs, the PTRS $\mathcal{P}$ induces a *rewrite relation* $\xrightarrow{\mathbf{f}}_{\mathcal{P}} \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \operatorname{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ where $s \xrightarrow{\mathbf{f}}_{\mathcal{P}} \{p_1 : t_1, \ldots, p_k : t_k\}$ if there is a position π, a rule $\ell \to \{p_1 : r_1, \ldots, p_k : r_k\} \in \mathcal{P}$, and a substitution σ such that $s|_\pi = \ell\sigma$ and $t_j = s[r_j\sigma]_\pi$ for all $1 \leq j \leq k$. Here, $\mathbf{f}$ stands for "full rewriting".[1] We call $s \xrightarrow{\mathbf{f}}_{\mathcal{P}} \mu$ an *innermost* rewrite step (denoted $s \xrightarrow{\mathbf{i}}_{\mathcal{P}} \mu$) if all proper subterms of the used redex $\ell\sigma$ are in normal form w.r.t. $\mathcal{P}$. For example, the PTRS $\mathcal{P}_{\mathsf{rw}}$ with the only rule $\mathsf{g} \to \{1/2 : \mathsf{c}(\mathsf{g}, \mathsf{g}),\ 1/2 : \mathsf{0}\}$ corresponds to a symmetric random walk on the number of g-symbols in a term.

[1] In the literature, one usually simply writes $\to_{\mathcal{P}}$ instead. Moreover, a "full" strategy sometimes refers to a specific strategy different from $\to_{\mathcal{P}}$, such as *full substitution rewriting* in Def. 4.9.5 (v) of [9]. We use $\xrightarrow{\mathbf{f}}_{\mathcal{P}} = \to_{\mathcal{P}}$ here to clearly distinguish between $\xrightarrow{\mathbf{f}}_{\mathcal{P}}$ and $\xrightarrow{\mathbf{i}}_{\mathcal{P}}$ in the following.

Next, we recapitulate the different notions of termination for PTRSs, see [1, 3, 6]. Let $\rightarrowtriangle \subseteq \mathcal{T}(\Sigma, \mathcal{V}) \times \mathrm{FDist}(\mathcal{T}(\Sigma, \mathcal{V}))$ be an arbitrary *probabilistic relation*, e.g., $\rightarrowtriangle \; = \; \xrightarrow{\mathtt{f}}_{\mathcal{P}}$ or $\rightarrowtriangle \; = \; \xrightarrow{\mathtt{i}}_{\mathcal{P}}$ for a PTRS $\mathcal{P}$, and let $\mathtt{NF}_{\rightarrowtriangle}$ be the set of all normal forms for $\rightarrowtriangle$. We track all possible rewrite sequences with their corresponding probabilities by lifting $\rightarrowtriangle$ to *rewrite sequence trees (RSTs)* [6]. The nodes v of an $\rightarrowtriangle$-RST are labeled by pairs $(p_v : t_v)$ of a probability p_v and a term t_v, where the root is always labeled with the probability 1. For each node v with the successors $w_1, \dots, w_k$, the edge relation represents a step with the relation $\rightarrowtriangle$, i.e., $t_v \rightarrowtriangle \{\frac{p_{w_1}}{p_v} : t_{w_1}, \dots, \frac{p_{w_k}}{p_v} : t_{w_k}\}$. For an $\rightarrowtriangle$-RST $\mathfrak{T}$, let $N^{\mathfrak{T}}$ denote the set of its nodes and $\mathrm{Leaf}^{\mathfrak{T}}$ denote the set of its leaves. For example, Fig. 1 depicts an $\xrightarrow{\mathtt{f}}_{\mathcal{P}_{\mathsf{rw}}}$-RST, which is also an $\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{rw}}}$-RST as it only uses innermost rewrite steps. As usual in term rewriting, a term might contain several redexes and several rules might be applicable to each redex. Thus, there can be several $\rightarrowtriangle$-RSTs with the same root.

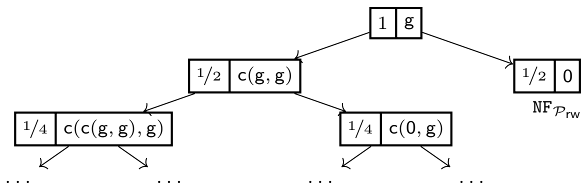

Figure 1 $\xrightarrow{\mathtt{f}}_{\mathcal{P}_{\mathsf{rw}}}$-RST which only uses innermost steps.

To express the concept of almost-sure termination, we have to determine the probabilities of the leaves of RSTs. While we define our notions of termination via RSTs, they are equivalent to the ones in [1, 3] where termination is defined via a lifting of $\rightarrowtriangle$ to multisets or via stochastic processes.

▶ **Definition 1** (Almost-Sure Termination). *For any $\rightarrowtriangle$-RST $\mathfrak{T}$ we define its* termination probability $|\mathfrak{T}| = \sum_{v \in \mathrm{Leaf}^{\mathfrak{T}}} p_v$. *Then* $\mathtt{AST}_{\rightarrowtriangle}$ *holds if for all $\rightarrowtriangle$-RSTs $\mathfrak{T}$ we have $|\mathfrak{T}| = 1$.*

▶ **Example 2.** For every extension $\mathfrak{T}$ of the $\xrightarrow{\mathtt{f}}_{\mathcal{P}_{\mathsf{rw}}}$-RST in Fig. 1, we have $|\mathfrak{T}| = 1$. Indeed, we have $\mathtt{AST}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}_{\mathsf{rw}}}}$ and thus also $\mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{rw}}}}$.

Next, we define *positive* almost-sure termination, which considers the *expected derivation length* $\mathrm{edl}(\mathfrak{T})$ of an RST $\mathfrak{T}$, i.e., the expected number of steps until one reaches a normal form. For positive almost-sure termination, we require that the expected derivation length of every possible rewrite sequence is finite.

▶ **Definition 3** (Positive Almost-Sure Termination, edl). *We define the* expected derivation length *of an $\rightarrowtriangle$-RST $\mathfrak{T}$ to be* $\mathrm{edl}(\mathfrak{T}) = \sum_{v \in N^{\mathfrak{T}} \setminus \mathrm{Leaf}^{\mathfrak{T}}} p_v$. *Then, we have* $\mathtt{PAST}_{\rightarrowtriangle}$ *if* $\mathrm{edl}(\mathfrak{T})$ *is finite for every $\rightarrowtriangle$-RST $\mathfrak{T}$.*

▶ **Example 4.** The expected derivation length $\mathrm{edl}(\mathfrak{T})$ is infinite for every infinite innermost extension $\mathfrak{T}$ of the $\xrightarrow{\mathtt{f}}_{\mathcal{P}_{\mathsf{rw}}}$-RST in Fig. 1 such that for every leaf $v \in \mathrm{Leaf}^{\mathfrak{T}}$ the corresponding term t_v is a normal form. Hence, $\mathtt{PAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_{\mathsf{rw}}}}$ does not hold, and $\mathtt{PAST}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}_{\mathsf{rw}}}}$ does not hold either.

Finally, we define *strong almost-sure termination* [1, 4], which is even stricter than $\mathtt{PAST}$ in case of non-determinism. It requires a finite bound on the expected derivation lengths of all rewrite sequences with the same start term. For a term $t \in \mathcal{T}(\Sigma, \mathcal{V})$, the *expected derivation height* $\mathrm{edh}_{\rightarrowtriangle}(t)$ considers all RSTs that start with t.

▶ **Definition 5** (Strong Almost-Sure Termination, edh). *We have* $\mathtt{SAST}_{\rightarrowtriangle}$ *if* $\mathrm{edh}_{\rightarrowtriangle}(t) = \sup\{\mathrm{edl}(\mathfrak{T}) \mid \mathfrak{T}$ *is an $\rightarrowtriangle$-RST whose root is labeled with* $(1 : t)\}$ *is finite for all* $t \in \mathcal{T}(\Sigma, \mathcal{V})$.

2 Modularity

For a PTRS $\mathcal{P}$, we decompose its signature $\Sigma = \Sigma_C \uplus \Sigma_D$ such that $f \in \Sigma_D$ if $f = \mathrm{root}(\ell)$ for some rule $\ell \to \mu \in \mathcal{P}$. The symbols in Σ_C and Σ_D are called *constructors* and *defined* *symbols*, respectively. To distinguish the functions symbols of different PTRSs $\mathcal{P}$, in the following we write $\Sigma_D^{\mathcal{P}}$, $\Sigma_C^{\mathcal{P}}$, and $\Sigma^{\mathcal{P}}$ for the defined symbols, constructor symbols, and all function symbols occurring in the rules of $\mathcal{P}$, respectively.

We study two different forms of unions, namely *disjoint unions* (Sect. 2.1), where both PTRSs do not share any function symbols, and *shared constructor unions* of PTRSs (Sect. 2.2) which may have common constructor symbols, but whose defined symbols are disjoint. In both cases, we restrict ourselves to innermost rewriting, as ordinary termination of TRSs for full rewriting is already not modular. One can lift our results on innermost rewriting to full rewriting for those classes of PTRSs $\mathcal{P}$ where, e.g., $\mathtt{AST}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}}} = \mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$, see [7, 8]. Finally in Sect. 2.3, we investigate the preservation of both full and innermost probabilistic rewriting under signature extensions.

2.1 Disjoint Unions

We first consider unions of systems that do not share any function symbols, i.e., we consider two PTRSs $\mathcal{P}^{(1)}$ and $\mathcal{P}^{(2)}$ such that $\Sigma^{\mathcal{P}^{(1)}} \cap \Sigma^{\mathcal{P}^{(2)}} = \varnothing$. In the non-probabilistic setting, [5] showed that innermost termination is modular for disjoint unions. This result can be lifted to $\mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$ and $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$, but it does not hold for $\mathtt{PAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$. We first investigate $\mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$, as illustrated by the following example.

▶ **Example 6.** Consider the PTRS $\mathcal{P}_1 = \mathcal{P}_1^{(1)} \cup \mathcal{P}_1^{(2)}$ given by

$$\mathcal{P}_1^{(1)} : \mathsf{f}(x) \to \{1/2 : \mathsf{f}(x), 1/2 : \mathsf{a}\} \qquad\qquad \mathcal{P}_1^{(2)} : \mathsf{g}(x) \to \{1/2 : \mathsf{g}(x), 1/2 : \mathsf{b}\}$$

$\mathcal{P}_1^{(1)}$ and $\mathcal{P}_1^{(2)}$ both correspond to a fair coin flip, where one terminates when obtaining heads. Hence, for both systems we have $\mathtt{AST}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}_1^{(1)}}}$ and $\mathtt{AST}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}_1^{(2)}}}$, and thus also $\mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_1^{(1)}}}$ and $\mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_1^{(2)}}}$. Furthermore, $\Sigma^{\mathcal{P}_1^{(1)}} \cap \Sigma^{\mathcal{P}_1^{(2)}} = \varnothing$, i.e., $\mathcal{P}_1$ is a disjoint union. When reducing a term like $\mathsf{f}(\mathsf{g}(x))$ which contains symbols from both systems, then we first reduce the innermost redex $\mathsf{g}(x)$ until we reach a normal form. Due to the innermost strategy, we cannot rewrite at the position of f beforehand. This reduction only uses one of the two systems, namely $\mathcal{P}_1^{(2)}$, hence it terminates with probability 1. Then, we use the next innermost redex, which will be $\mathsf{f}(\mathsf{b})$, using only rules of $\mathcal{P}_1^{(1)}$ until we reach a normal form, where symbols from $\Sigma^{\mathcal{P}_1^{(2)}}$ do not influence the reduction. The reason is that all subterms below or at positions of symbols from $\Sigma^{\mathcal{P}_1^{(2)}}$ are in normal form, and there is no symbol from $\Sigma^{\mathcal{P}_1^{(2)}}$ above the f at the root position. Again, this reduction terminates with probability 1. Thus, in the end, our reduction starting with $\mathsf{f}(\mathsf{g}(x))$ also terminates with probability 1, and the same holds for arbitrary start terms, which implies $\mathtt{AST}_{\xrightarrow{\mathtt{f}}_{\mathcal{P}_1}}$.

In Ex. 6, we considered the term $\mathsf{f}(\mathsf{g}(x))$ where we swap once between a symbol f from $\Sigma^{\mathcal{P}_1^{(1)}}$ and a symbol g from $\Sigma^{\mathcal{P}_1^{(2)}}$ on the path from the root to the "leaf" of the term. In the proof of Thm. 7 we lift the argumentation of Ex. 6 to arbitrary terms via induction. This proof idea was also used by [5] in the non-probabilistic setting to show the modularity of innermost termination for disjoint unions of TRSs.

▶ **Theorem 7** (Modularity of $\mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$ for Disjoint Unions). *Let $\mathcal{P}^{(1)}$ and $\mathcal{P}^{(2)}$ be PTRSs with $\Sigma^{\mathcal{P}^{(1)}} \cap \Sigma^{\mathcal{P}^{(2)}} = \varnothing$. Then we have: $\mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}^{(1)} \cup \mathcal{P}^{(2)}}} \iff \mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}^{(1)}}}$ and $\mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}^{(2)}}}$.*

In contrast to $\mathtt{AST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$, $\mathtt{PAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$ cannot be modular due to the potential extension of the signature (see Thm. 13, which shows that $\mathtt{PAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$ is not closed under signature extensions).

Finally, we consider $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$. To prove that $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$ is modular for disjoint unions, we have to show that the expected derivation height of any term t is finite. However, after rewriting t's proper subterms to normal forms, as we did in Ex. 6 and in the induction proof of Thm. 7, we may end up with infinitely many different terms. All their expected derivation heights have to be considered in order to compute the expected derivation height of t.

▶ **Example 8.** Consider the PTRSs $\mathcal{P}_2^{(1)}$ and $\mathcal{P}_2^{(2)}$ with

$$\begin{aligned} \mathcal{P}_2^{(1)} : \mathsf{f}(\mathsf{s}(x), y) &\to \{1 : \mathsf{f}(x, y)\} & \qquad \mathcal{P}_2^{(2)} : \mathsf{g}(x) \to \{1 : x\} \\ \mathsf{f}(x, \mathsf{s}(y)) &\to \{1 : \mathsf{f}(x, y)\} \\ \mathsf{a} &\to \{1/2 : \mathsf{0}, 1/2 : \mathsf{s}(\mathsf{a})\} \end{aligned}$$

Clearly, we have both $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_2^{(1)}}}$ and $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_2^{(2)}}}$. Now consider the term $t = \mathsf{f}(\mathsf{g}(\mathsf{a}), \mathsf{g}(\mathsf{a}))$. Due to the innermost strategy, we have to rewrite its proper subterms first. When proceeding in a similar way as in the induction proof of Thm. 7, then one would first construct bounds on the expected derivation heights of the proper subterms, and then use them to obtain a bound on the expected derivation height of the whole term t. However, reducing t's proper subterms can create infinitely many different terms, i.e., all terms of the form $\mathsf{f}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))$ for any $n, m \in \mathbb{N}$ can be reached with a certain probability. Since there is no finite supremum on the derivation height of $\mathsf{f}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))$ for all $n, m \in \mathbb{N}$, one would have to take the individual probabilities for reaching the terms $\mathsf{f}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))$ into account in order to prove that the expected derivation height of t is indeed finite.

As shown in Ex. 8, there may be infinitely many terms t' after an inductive step, e.g., in Ex. 8, t' can be any term of the form $\mathsf{f}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))$. However, this infinite set of terms can be over-approximated by a finite number of representatives (with finite expected derivation height), leading to the following result.

▶ **Theorem 9** (Modularity of $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$ for Disjoint Unions). *Let $\mathcal{P}^{(1)}$ and $\mathcal{P}^{(2)}$ be PTRSs with $\Sigma^{\mathcal{P}^{(1)}} \cap \Sigma^{\mathcal{P}^{(2)}} = \varnothing$. Then we have:* $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}^{(1)} \cup \mathcal{P}^{(2)}}} \iff \mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}^{(1)}}}$ *and* $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}^{(2)}}}$.

2.2 Shared Constructor Unions

Now we consider unions of PTRSs that may share constructor symbols, i.e., we consider two PTRSs $\mathcal{P}^{(1)}$ and $\mathcal{P}^{(2)}$ such that $\Sigma_D^{\mathcal{P}^{(1)}} \cap \Sigma_D^{\mathcal{P}^{(2)}} = \varnothing$, called *shared constructor unions.*

In the non-probabilistic setting, innermost termination is also modular for shared constructor unions [5]. However, $\mathtt{PAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$ was already not modular w.r.t. disjoint unions, so this also holds for shared constructor unions. Moreover, $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}}}$ also turns out to be not modular anymore for shared constructor unions.

Counterexample 10. Consider the PTRS $\mathcal{P}_3 = \mathcal{P}_3^{(1)} \cup \mathcal{P}_3^{(2)}$ with the rules

$$\begin{aligned} \mathcal{P}_3^{(1)} : \mathsf{f}(\mathsf{c}(x, y)) &\to \{1 : \mathsf{c}(\mathsf{f}(x), \mathsf{f}(y))\} \\ \mathsf{f}(\mathsf{0}) &\to \{1 : \mathsf{0}\} \\ \mathcal{P}_3^{(2)} : \mathsf{g}(x) &\to \{1/2 : \mathsf{g}(\mathsf{d}(x)), 1/2 : x\} \\ \mathsf{d}(x) &\to \{1 : \mathsf{c}(x, x)\} \end{aligned}$$

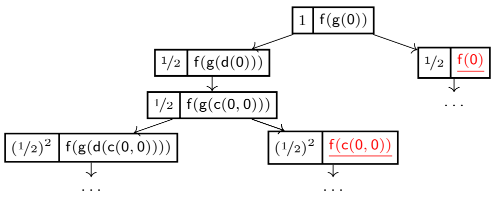

Figure 2 Infinite $\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}$-RST.

While $\mathcal{P}_3^{(1)}$ and $\mathcal{P}_3^{(2)}$ do not have any common defined symbols, they share the constructor c. We do not have $\mathtt{PAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}}$ (and thus, not $\mathtt{SAST}_{\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}}$ either), as the infinite $\xrightarrow{\mathtt{i}}_{\mathcal{P}_3}$-RST depicted in Fig. 2 has an infinite expected derivation length. For any $n \in \mathbb{N}$, each red underlined term $\mathsf{f}(\mathsf{c}^n(\mathsf{0}, \mathsf{0}))$ in the tree above can start a reduction of at least length 2^n, where $\mathsf{c}^n(\mathsf{0}, \mathsf{0})$ corresponds to the full binary tree of height n with c in inner nodes and 0 in the leaves. Hence, the term $\mathsf{f}(\mathsf{g}(0))$ has an expected derivation height of at least $\sum_{n=0}^{\infty} \frac{1}{2^{n+1}} \cdot 2^n = \sum_{n=0}^{\infty} \frac{1}{2}$, which diverges to infinity.

On the other hand, we have $\mathtt{SAST}_{\xrightarrow{\mathbf{i}}_{\mathcal{P}_3^{(1)}}}$, as $\mathcal{P}_3^{(1)}$ is a PTRS with only trivial probabilities that corresponds to a terminating TRS. Moreover, $\mathtt{SAST}_{\xrightarrow{\mathbf{i}}_{\mathcal{P}_3^{(2)}}}$ holds as well, as the d-rule can increase the number of c-symbols in a term exponentially, but those c-symbols will never be used. Thus, $\mathtt{SAST}_{\xrightarrow{\mathbf{i}}_{\mathcal{P}}}$ is not modular for shared constructor unions.

For $\mathtt{AST}_{\xrightarrow{\mathbf{i}}_{\mathcal{P}}}$ we obtain a similar result for shared constructor unions as for disjoint unions.

▶ **Theorem 11** (Modularity of $\mathtt{AST}_{\xrightarrow{\mathbf{i}}_{\mathcal{P}}}$ for Shared Constructor Unions). *Let $\mathcal{P}^{(1)}$ and $\mathcal{P}^{(2)}$ be PTRSs with $\Sigma_D^{\mathcal{P}^{(1)}} \cap \Sigma_D^{\mathcal{P}^{(2)}} = \varnothing$. Then we have: $\mathtt{AST}_{\xrightarrow{\mathbf{i}}_{\mathcal{P}^{(1)} \cup \mathcal{P}^{(2)}}} \iff \mathtt{AST}_{\xrightarrow{\mathbf{i}}_{\mathcal{P}^{(1)}}}$ and $\mathtt{AST}_{\xrightarrow{\mathbf{i}}_{\mathcal{P}^{(2)}}}$.*

2.3 Signature Extensions

Finally, we consider signature extensions. The following theorem shows that both $\mathtt{AST}$ and $\mathtt{SAST}$ are closed under extensions of the signature, for both innermost and full rewriting.

▶ **Theorem 12** (Signature Extensions for $\mathtt{AST}_{\xrightarrow{s}_{\mathcal{P}}}$ and $\mathtt{SAST}_{\xrightarrow{s}_{\mathcal{P}}}$). *Let $\mathcal{P}$ be a PTRS, $s \in \{\mathbf{f}, \mathbf{i}\}$, and let Σ' be some signature. Then we have: $\mathtt{AST}_{\xrightarrow{s}_{\mathcal{P}}}$ over $\Sigma^{\mathcal{P}} \iff \mathtt{AST}_{\xrightarrow{s}_{\mathcal{P}}}$ over $\Sigma^{\mathcal{P}} \cup \Sigma'$ and $\mathtt{SAST}_{\xrightarrow{s}_{\mathcal{P}}}$ over $\Sigma^{\mathcal{P}} \iff \mathtt{SAST}_{\xrightarrow{s}_{\mathcal{P}}}$ over $\Sigma^{\mathcal{P}} \cup \Sigma'$.*

However, $\mathtt{PAST}$ is not closed under signature extensions.

▶ **Theorem 13** (Signature Extensions for $\mathtt{PAST}_{\xrightarrow{s}_{\mathcal{P}}}$). *Let $s \in \{\mathbf{f}, \mathbf{i}\}$. There exists a PTRS $\mathcal{P}$ and signatures Σ, Σ' with $\Sigma \subset \Sigma'$ such that $\mathtt{PAST}_{\xrightarrow{s}_{\mathcal{P}}}$ holds over Σ, but not over Σ'.*

References

1 Martin Avanzini, Ugo Dal Lago, and Akihisa Yamada. On probabilistic term rewriting. *Sci. Comput. Program.*, 185, 2020. doi:10.1016/j.scico.2019.102338.

2 Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998. doi:10.1017/CBO9781139172752.

3 Olivier Bournez and Florent Garnier. Proving positive almost-sure termination. In *Proc. RTA '05*, LNCS 3467, pages 323–337, 2005. doi:10.1007/978-3-540-32033-3_24.

4 Hongfei Fu and Krishnendu Chatterjee. Termination of nondeterministic probabilistic programs. In *Proc. VMCAI '19*, LNCS 11388, pages 468–490, 2019. doi:10.1007/978-3-030-11245-5_22.

5 Bernhard Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundam. Informaticae*, 24:2–23, 1995. doi:10.3233/FI-1995-24121.

6 Jan-Christoph Kassing, Stefan Dollase, and Jürgen Giesl. A complete dependency pair framework for almost-sure innermost termination of probabilistic term rewriting. In *Proc. FLOPS '24*, LNCS 14659, 2024. doi:10.1007/978-981-97-2300-3_4.

7 Jan-Christoph Kassing, Florian Frohn, and Jürgen Giesl. From innermost to full almost-sure termination of probabilistic term rewriting. In *Proc. FoSSaCS '24*, LNCS 14575, pages 206–228, 2024. doi:10.1007/978-3-031-57231-9_10.

8 Jan-Christoph Kassing and Jürgen Giesl. From innermost to full probabilistic term rewriting: Almost-sure termination, complexity, and modularity, 2025. URL: https://arxiv.org/abs/2409.17714, arXiv:2409.17714.

9 Terese. *Term Rewriting Systems.* Cambridge University Press, 2001.

Deciding Termination of Simple Randomized Loops

Éléanore Meyer
RWTH Aachen University, Aachen, Germany

Jürgen Giesl
RWTH Aachen University, Aachen, Germany

Abstract

We show that universal positive almost sure termination (UPAST) is decidable for a class of simple randomized programs, i.e., it is decidable whether the expected runtime of such a program is finite for all inputs. Our class contains all programs that consist of a single loop, with a linear loop guard and a loop body composed of two linear commuting and diagonalizable updates. In each iteration of the loop, the update to be carried out is picked at random, according to a fixed probability. We show the decidability of UPAST for this class of programs, where the program's variables and inputs may range over various sub-semirings of the real numbers. In this way, we extend a line of research initiated by Tiwari in 2004 into the realm of randomized programs.

2012 ACM Subject Classification Theory of computation → Probabilistic computation; Theory of computation → Program analysis

Keywords and phrases decision procedures, randomized programs, linear loops, positive almost sure termination

Related Version The full version of this paper is published in [20]. Moreover, we provide an extended report [21] including all proofs, as well as proof sketches for all main results to help understanding the essential proof ideas.

Funding funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - DFG Research Training Group 2236 UnRAVeL

1 Introduction

We consider the problem of universal positive almost sure termination (UPAST), i.e., deciding whether a given randomized program has finite expected runtime on all inputs [5, 26]. This is a stronger property than universal almost sure termination (UAST) which requires that the probability of termination is 1. In [16], the authors showed that deciding UPAST is harder than deciding universal termination for non-randomized programs in terms of the arithmetic hierarchy. While the non-randomized problem is Π^0_2-complete, UPAST is Π^0_3-complete.

Our programs are simple randomized loops of the form

$$\texttt{while } \mathbf{C}\vec{x} > \vec{0} : \vec{x} \leftarrow \mathbf{A}\vec{x} \oplus_p \mathbf{B}\vec{x} \tag{1}$$

Here, $\vec{x} = (x_1, \ldots, x_n)$ denotes the vector of program variables that range over a semiring $\mathcal{S} \subseteq \mathbb{R}$, and $\mathbf{C} \in \mathbb{R}^{m\times n}$ is a matrix representing the loop guard with m linear constraints over the program variables. In each execution of the loop body, a matrix is chosen among $\mathbf{A}, \mathbf{B} \in \mathcal{S}^{n\times n}$ according to the probability $p \in [0, 1]$ and the value $\vec{x}$ is updated accordingly.

We show that UPAST is decidable for all $\mathcal{S} \in \{\mathbb{Z}, \mathbb{Q}, \mathbb{A}\}$ when limited to loops with diagonalizable commuting matrices $\mathbf{A}$ and $\mathbf{B}$, where $\mathbb{A}$ is the set of algebraic real numbers.[1]

[1] Our approach only considers *algebraic* p, $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, as it is not possible to represent arbitrary real numbers on computers. However, such a loop terminates for all algebraic inputs iff it terminates for all real inputs, see [20, Remark 38].

Thus, we extend previous results on the termination of linear and affine[2] non-randomized loops to the randomized setting. In addition to deciding universal termination, our approach can compute a non-termination witness $\vec{x} \in \mathcal{S}^n$, i.e., if the loop is non-terminating, then $\vec{x}$ is an input leading to an infinite expected runtime. So our programs go beyond single path loops as we might have $\mathbf{A} \neq \mathbf{B}$. Thus, for every $k \in \mathbb{N}$, there is not just a single execution of length k but instead one has a "range" of possible executions of length k where each execution occurs according to a known probability. To ensure tractability of the resulting problem we require commutativity of both updates, so that we can focus on how often each update has been selected in an execution, but we do not have to take the 2^k different orders into account in which the two updates might have been chosen. Moreover, we require diagonalizability in order to obtain closed forms of a certain shape, which allows us to analyze the behavior of a "range" of different executions at once.

To demonstrate the practical applicability of our decision procedure and the computation of non-termination witnesses, we provide a prototype implementation for the case $\mathcal{S} = \mathbb{A}$ in our tool SiRop (for "Simple Randomized Loops"). The tool and a corresponding collection of exemplary programs can be obtained from

`https://github.com/aprove-developers/SiRop`

In particular, our tool managed to find a non-terminating algebraic input for one[3] of the only two problems from the category `C Integer` which were not solved by any tool at the 2023 Termination Competition [12],[4] the other one being the Collatz problem. While our tool only considers $\mathcal{S} = \mathbb{A}$ (whereas the problem is formulated over the integers), the constraints generated by SiRop are unsatisfiable over $\mathbb{Z}$, which implies universal termination of the program over the integers.

Related Work: We continue a line of research started in 2004 by Tiwari [27] who showed decidability of universal termination for loops with an affine guard and an affine update as its body, where the guard, updates, and inputs range over the real numbers. In his proof, Tiwari reduced the affine to the linear case. In 2006, Braverman [6] proved that the problem remains decidable for loops and inputs ranging over the rational numbers $\mathbb{Q}$, and if the guard and update are linear, then he also showed decidability over the integers $\mathbb{Z}$. Similar to Tiwari, Braverman also reduced the affine case for $\mathbb{Q}$ to the linear case. In 2015, Ouaknine et al. proved [25] that the affine case is decidable over the integers $\mathbb{Z}$ whenever the update is of the form $\vec{x} \leftarrow \mathbf{A}\vec{x} + \vec{a}$, provided that $\mathbf{A} \in \mathbb{Z}^{n\times n}$ is diagonalizable. This restriction was removed by Hosseini et al. in 2019 [15]. In a related line of work, we proved decidability of universal termination over the integers $\mathbb{Z}$ for triangular affine loops, i.e., where the matrix $\mathbf{A} \in \mathbb{Z}^{n\times n}$ is triangular [10]. Later, we extended these results to triangular weakly non-linear loops which extend triangular loops by allowing certain non-linear updates [13, 14].

The only decidability results for termination of randomized programs that we are aware of consider probabilistic vector addition systems [7] or constant probability programs [11]. The programs in [7, 11] are orthogonal to the ones considered in our approach as they can only modify the program variables by adding constants, but do not allow for multiplication. Another related area of research [3, 18, 23] deals with prob-solvable loops and moment

[2] In an affine (or non-homogeneous) loop, the guard may have the form $\mathbf{C}\vec{x} > \vec{c}$ and the update may have the form $\vec{x} \leftarrow \mathbf{A}\vec{x} + \vec{a}$ for arbitrary vectors $\vec{c}$ and $\vec{a}$.

[3] `https://github.com/TermCOMP/TPDB/blob/11.3/C_Integer/Stroeder_15/ChenFlurMukhopadhyay-SAS2012-Ex2.06_false-termination.c`

[4] This category was not part of the 2024 competition.

invariants. However, in contrast to our method, these techniques require that all variables in the loop guard may only take finitely many values. Moreover, there are many automated approaches for tackling UPAST using ranking supermartingales (RSM), e.g., [1, 2, 4, 8, 9, 19, 22, 24, 28]. To generate a suitable RSM, one often uses techniques based on affine or polynomial templates, which renders the approach incomplete.

2 Outline of our Approach

For a loop as in (1), we consider (finite) executions f corresponding to words over the alphabet $\{A, B\}$, where the i-th symbol in f indicates which update matrix was used in the i-th application of the assignment $\vec{x} \leftarrow \mathbf{A}\vec{x} \oplus_p \mathbf{B}\vec{x}$. For such executions f, $|f|_A$ and $|f|_B$ denote the number of A- and B-symbols in f, respectively. Moreover, we introduce the function $\mathcal{V}al_{\vec{x}}$ that maps finite executions f to the values $\mathcal{V}al_{\vec{x}}(f) \in \mathbb{R}^m$ of the constraints in the loop guard after executing f on a concrete input $\vec{x} \in \mathbb{R}^n$, i.e., $\mathcal{V}al_{\vec{x}}(f) = \mathbf{C} \cdot \mathbf{A}^{|f|_A} \cdot \mathbf{B}^{|f|_B} \cdot \vec{x}$, since $\mathbf{A}$ and $\mathbf{B}$ commute. Our decision procedure does not search for a non-terminating input directly, but for an *eventually* non-terminating input $\vec{x}$. An input $\vec{x}$ is eventually non-terminating if by repeated execution of the loop body on $\vec{x}$ (while ignoring the guard), a non-terminating input can eventually be reached. We show that such a loop has an eventually non-terminating input iff it also has a non-terminating input. One can then show how such an eventually non-terminating input can be lifted to an actual non-terminating input.

We introduce a mapping $\mathcal{U}$ that maps executions f to the difference between the relative number $\frac{|f|_A}{|f|}$ of times that the update matrix $\mathbf{A}$ has been chosen in f and the probability p of choosing $\mathbf{A}$, i.e., $\mathcal{U}(f) = \frac{|f|_A}{|f|} - p$. Moreover, we essentially partition the set of indices $\{1, \ldots, n\}$ of all program variables into suitable sets $\mathfrak{D}_{(\mathfrak{i},\mathfrak{o})}$ with $(\mathfrak{i}, \mathfrak{o}) \in \mathcal{I}$ for some finite set $\mathcal{I} \subsetneq \mathbb{R}^2_{>0}$. We then show that for all $c \in \{1, \ldots, m\}$, and all executions f with $|f|_A, |f|_B \geq 1$, the value $(\mathcal{V}al_{\vec{x}}(f))_c$ of the c-th constraint after executing f on $\vec{x}$ is

$$(\mathcal{V}al_{\vec{x}}(f))_c = \sum_{(\mathfrak{i},\mathfrak{o})\in\mathcal{I}} \left(\mathfrak{i} \cdot \mathfrak{o}^{\mathcal{U}(f)}\right)^{|f|} \sum_{i\in\mathfrak{D}_{(\mathfrak{i},\mathfrak{o})}} \zeta_{i,\mathbf{A}}^{|f|_A}\, \zeta_{i,\mathbf{B}}^{|f|_B}\, \gamma_{c,i}(\vec{x}). \tag{2}$$

Here, all $\zeta_{i,\mathbf{A}}, \zeta_{i,\mathbf{B}}$ are complex numbers of modulus 1, i.e., $|\zeta_{i,\mathbf{A}}| = |\zeta_{i,\mathbf{B}}| = 1$ for all $i \in \{1, \ldots, n\}$, and the functions $\gamma_{c,i}$ are linear maps $\mathbb{R}^n \to \mathbb{C}$. The maps $\gamma_{c,i}$ and the values $\zeta_{i,\mathbf{A}}, \zeta_{i,\mathbf{B}} \in \mathbb{C}$ only depend on the matrices $\mathbf{C}$, $\mathbf{A}$, and $\mathbf{B}$, but not on the specific input $\vec{x}$. While weaker requirements would suffice to ensure that $(\mathcal{V}al_{\vec{x}}(f))_c$ has some closed form, diagonalizability of $\mathbf{A}$ and $\mathbf{B}$ guarantees that it has the form (2), which is crucial for our procedure. By a lexicographic comparison of those $(\mathfrak{i}, \mathfrak{o}) \in \mathcal{I}$ for which the inner sum in (2) is not 0 for all executions f, one can compute which of the pairs $(\mathfrak{i}, \mathfrak{o})$ is the "dominant" one. Here, a pair $(\mathfrak{i}, \mathfrak{o}) \in \mathcal{I}$ is considered dominant whenever the value of the first factor $\left(\mathfrak{i} \cdot \mathfrak{o}^{\mathcal{U}(f)}\right)^{|f|}$ of (2) grows the fastest if the execution of f is continued (i.e., if $|f| \to \infty$) and the corresponding inner sum is not 0 for all executions f. The dominant pair depends on the specific input $\vec{x}$ and on whether $\mathcal{U}(f)$ is $\mathfrak{p}$ositive or $\mathfrak{n}$egative, and correspondingly, one has to use different lexicographic comparisons to determine the dominant pair. For $d \in \{\mathfrak{n}, \mathfrak{p}\}$, let $\mathfrak{D}_{d,c,\vec{x}}$ denote the set $\mathfrak{D}_{(\mathfrak{i},\mathfrak{o})}$ where the pair $(\mathfrak{i}, \mathfrak{o})$ is dominant for input $\vec{x}$ and $c \in \{1, \ldots, m\}$ (and positive $\mathcal{U}(f)$ if $d = \mathfrak{p}$ or negative $\mathcal{U}(f)$ for $d = \mathfrak{n}$). Then, the sign of the "coefficient" $v(f) = \sum_{i\in\mathfrak{D}_{d,c,\vec{x}}} \zeta_{i,\mathbf{A}}^{|f|_A} \zeta_{i,\mathbf{B}}^{|f|_B} \gamma_{c,i}(\vec{x})$ of the dominant pair eventually determines the sign of $(\mathcal{V}al_{\vec{x}}(f))_c$, provided that $|v(f)|$ is large enough. This also indicates why an extension of our approach to programs with three instead of two update matrices would be problematic. Then instead of $\mathcal{U}(f)$ we would need a vector to express how much an execution deviates from the probabilities in the program. This would break the underlying concepts, since instead of $d \in \{\mathfrak{n}, \mathfrak{p}\}$, we would have to consider the "direction" of this deviation.

We then consider the rearrangement

$$v(f) \;=\; \underbrace{\textstyle\sum_{i\in\mathfrak{R}_{d,c,\vec{x}}} \gamma_{c,i}(\vec{x})}_{=R} + \textstyle\sum_{i\in\mathfrak{C}_{d,c,\vec{x}}} \zeta_{i,\mathbf{A}}^{|f|_A}\,\zeta_{i,\mathbf{B}}^{|f|_B}\,\gamma_{c,i}(\vec{x}) \tag{3}$$

where $\mathfrak{R}_{d,c,\vec{x}} = \{i \in \mathfrak{D}_{d,c,\vec{x}} \mid \zeta_{i,\mathbf{A}} = \zeta_{i,\mathbf{B}} = 1\}$ and $\mathfrak{C}_{d,c,\vec{x}} = \{i \in \mathfrak{D}_{d,c,\vec{x}} \mid \{\zeta_{i,\mathbf{A}}, \zeta_{i,\mathbf{B}}\} \neq \{1\}\}$. This leads us to a necessary condition for non-termination (and hence a sufficient condition for universal termination): If $\vec{x}$ is an eventually non-terminating input, then there must be a $d \in \{\mathfrak{n}, \mathfrak{p}\}$ such that we have $R > 0$ for all constraints c, with R as in (3).

This necessary condition for non-termination is then turned into a sufficient condition. To that end, we define the set W of witnesses for eventual non-termination containing all inputs $\vec{x} \in \mathcal{S}^n$ for which there is some $d \in \{\mathfrak{n}, \mathfrak{p}\}$ such that $R > \sum_{i\in\mathfrak{C}_{d,c,\vec{x}}} |\gamma_{c,i}(\vec{x})|$ holds for all constraints c. All witnesses $\vec{x} \in W$ are eventually non-terminating. While this condition is only sufficient for non-termination, one shows that if the program admits a non-terminating input, then there is also an input in W. So the considered program is non-terminating iff $W \neq \varnothing$, i.e., the program is universally terminating iff $W = \varnothing$.

Finally, we obtain a decision procedure for the problem UPAST. Here, we use that W is semi-algebraic and thus the emptiness problem is decidable over the real algebraic numbers, i.e., if $\mathcal{S} = \mathbb{A}$. Moreover, W can be represented as a finite union of convex semi-algebraic sets. Hence, emptiness of W can also be decided over the rationals and integers [17]. Moreover, witnesses for non-termination can be obtained from eventually non-terminating inputs $\vec{x} \in W$.

References

1 Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: An efficient approach to termination of probabilistic programs. *Proc. ACM Program. Lang.*, 2(POPL):34:1–34:32, 2018. doi:10.1145/3158122.

2 Martin Avanzini, Georg Moser, and Michael Schaper. A modular cost analysis for probabilistic programs. *Proc. ACM Program. Lang.*, 4(OOPSLA), 2020. doi:10.1145/3428240.

3 Ezio Bartocci, Laura Kovács, and Miroslav Stankovic. Automatic generation of moment-based invariants for prob-solvable loops. In *Proc. ATVA '19*, LNCS 11781, pages 255–276, 2019. doi:10.1007/978-3-030-31784-3_15.

4 Kevin Batz, Mingshuai Chen, Sebastian Junges, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Probabilistic program verification via inductive synthesis of inductive invariants. In *Proc. TACAS '23*, LNCS 13994, pages 410–429, 2023. doi:10.1007/978-3-031-30820-8_25.

5 Olivier Bournez and Florent Garnier. Proving positive almost-sure termination. In *Proc. RTA '05*, LNCS 3467, pages 323–337, 2005. doi:10.1007/978-3-540-32033-3_24.

6 Mark Braverman. Termination of integer linear programs. In *Proc. CAV '06*, LNCS 4144, pages 372–385, 2006. doi:10.1007/11817963_34.

7 Tomás Brázdil, Krishnendu Chatterjee, Antonín Kucera, Petr Novotný, and Dominik Velan. Deciding fast termination for probabilistic VASS with nondeterminism. In *Proc. ATVA '19*, LNCS 11781, pages 462–478, 2019. doi:10.1007/978-3-030-31784-3_27.

8 Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic program analysis with martingales. In *Proc. CAV '13*, LNCS 8044, pages 511–526, 2013. doi:10.1007/978-3-642-39799-8_34.

9 Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination analysis of probabilistic programs through positivstellensatz's. In *Proc. CAV '16*, LNCS 9779, pages 3–22, 2016. doi:10.1007/978-3-319-41528-4_1.

10 Florian Frohn and Jürgen Giesl. Termination of triangular integer loops is decidable. In *Proc. CAV '19*, LNCS 11562, pages 426–444, 2019. doi:10.1007/978-3-030-25543-5_24.

11 Jürgen Giesl, Peter Giesl, and Marcel Hark. Computing expected runtimes for constant probability programs. In *Proc. CADE '19*, LNCS 11716, pages 269–286, 2019. doi:10.1007/978-3-030-29436-6_16.

12 Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In *Proc. TACAS '19*, LNCS 11429, pages 156–166, 2019. Website of the Annual Termination Competition: https://termination-portal.org/wiki/Termination_Competition. doi:10.1007/978-3-030-17502-3_10.

13 Marcel Hark, Florian Frohn, and Jürgen Giesl. Polynomial loops: Beyond termination. In *Proc. LPAR '20*, EPiC 73, pages 279–297, 2020. doi:10.29007/nxv1.

14 Marcel Hark, Florian Frohn, and Jürgen Giesl. Termination of triangular polynomial loops. *Formal Methods in Syst. Des.*, 65(1):70–132, 2025. doi:10.1007/s10703-023-00440-z.

15 Mehran Hosseini, Joël Ouaknine, and James Worrell. Termination of linear loops over the integers. In *Proc. ICALP '19*, LIPIcs 132, 2019. doi:10.4230/LIPIcs.ICALP.2019.118.

16 Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. On the hardness of analyzing probabilistic programs. *Acta Informatica*, 56(3):255–285, 2019. URL: https://doi.org/10.1007/s00236-018-0321-1, doi:10.1007/S00236-018-0321-1.

17 Leonid Khachiyan and Lorant Porkolab. Computing integral points in convex semi-algebraic sets. In *Proc. FOCS '97*, pages 162–171, 1997. doi:10.1109/SFCS.1997.646105.

18 Andrey Kofnov, Marcel Moosbrugger, Miroslav Stankovic, Ezio Bartocci, and Efstathia Bura. Exact and approximate moment derivation for probabilistic loops with non-polynomial assignments. *ACM Trans. Model. Comput. Simul.*, 34(3):18:1–18:25, 2024. doi:10.1145/3641545.

19 Fabian Meyer, Marcel Hark, and Jürgen Giesl. Inferring expected runtimes of probabilistic integer programs using expected sizes. In *Proc. TACAS '21*, LNCS 12651, pages 250–269, 2021. doi:10.1007/978-3-030-72016-2_14.

20 Éléanore Meyer and Jürgen Giesl. Deciding termination of simple randomized loops. In *Proc. MFCS '25*, LIPIcs, 2025. To appear.

21 Éléanore Meyer and Jürgen Giesl. Deciding termination of simple randomized loops, 2025. Extended version of [20] with all proofs. URL: https://arxiv.org/abs/2506.18541, arXiv:2506.18541.

22 Marcel Moosbrugger, Ezio Bartocci, Joost-Pieter Katoen, and Laura Kovács. The probabilistic termination tool Amber. *Formal Methods Syst. Des.*, 61(1):90–109, 2022. URL: https://doi.org/10.1007/s10703-023-00424-z, doi:10.1007/S10703-023-00424-Z.

23 Marcel Moosbrugger, Miroslav Stankovic, Ezio Bartocci, and Laura Kovács. This is the Moment for Probabilistic Loops. *Proc. ACM Program. Lang.*, 6(OOPSLA2):1497–1525, 2022. doi:10.1145/3563341.

24 Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: Resource analysis for probabilistic programs. In *Proc. PLDI '18*, pages 496–512, 2018. doi:10.1145/3192366.3192394.

25 Joël Ouaknine, João Sousa Pinto, and James Worrell. On termination of integer linear loops. In *Proc. SODA '15*, pages 957–969, 2015. doi:10.1137/1.9781611973730.65.

26 Nasser Saheb-Djahromi. Probabilistic LCF. In *Proc. MFCS '78*, LNCS 64, pages 442–451, 1978. doi:10.1007/3-540-08921-7_92.

27 Ashish Tiwari. Termination of linear programs. In *Proc. CAV '04*, LNCS 3114, pages 70–82, 2004. doi:10.1007/978-3-540-27813-9_6.

28 Di Wang, David M. Kahn, and Jan Hoffmann. Raising expectations: Automating expected cost analysis with types. *Proc. ACM Program. Lang.*, 4(ICFP), 2020. doi:10.1145/3408992.

Control-Flow Refinement for Complexity Analysis of Probabilistic Programs

Nils Lommen
RWTH Aachen University, Germany

Éléanore Meyer
RWTH Aachen University, Germany

Jürgen Giesl
RWTH Aachen University, Germany

Abstract

Recently, we showed how to use control-flow refinement (CFR) to improve automatic complexity analysis of integer programs. While up to now CFR was limited to classical programs, we extend CFR to *probabilistic* programs and show its soundness for complexity analysis. To demonstrate its benefits, we implemented our new CFR technique in our complexity analysis tool KoAT.

2012 ACM Subject Classification Software and its engineering → Software verification and validation; Theory of computation → Automated reasoning; Theory of computation → Logic and verification; Theory of computation → Probabilistic computation

Keywords and phrases (Positive) Almost-Sure Termination, Control-Flow Refinement, Complexity Analysis

Related Version Extended Abstract of our Conference Article [26]

1 A Birds-Eye-View on Control-Flow Refinement

There exist numerous tools for complexity analysis of (non-probabilistic) programs, e.g., [2–6, 10, 11, 15, 16, 18, 19, 24, 25, 28, 30, 32]. Our tool KoAT infers upper runtime and size bounds for (non-probabilistic) integer programs in a modular way by analyzing subprograms separately and lifting the obtained results to global bounds on the whole program [10]. Recently, we developed several improvements of KoAT [18, 24, 25] and showed that incorporating control-flow refinement (CFR) [13, 14] increases the power of automated complexity analysis significantly [18].

There are also several approaches for complexity analysis of *probabilistic* programs, e.g., [1, 7, 9, 21–23, 27, 29, 31, 34]. In particular, we also adapted KoAT's approach for runtime and size bounds, and introduced a modular framework for automated complexity analysis of probabilistic integer programs in [27]. However, the improvements of KoAT from [18, 24, 25] had not yet been adapted to the probabilistic setting. In particular, we are not aware of any existing technique to combine CFR with complexity analysis of probabilistic programs.

Thus, we develop a novel CFR technique for probabilistic programs which could be used as a black box by every complexity analysis tool. Moreover, to reduce the overhead by CFR, we integrate CFR natively into KoAT by calling it on-demand in a modular way. Our experiments show that CFR increases the power of KoAT for complexity analysis of probabilistic programs substantially.

The idea of CFR is to gain information on the values of program variables and to sort out infeasible program paths. For example, consider the probabilistic **while**-loop (1). Here, we flip a (fair) coin and either set x to 0 or do nothing.

$$\textbf{while}\ \ x > 0\ \ \textbf{do}\ \ x \leftarrow 0 \oplus_{1/2} \texttt{noop}\ \ \textbf{end} \tag{1}$$

The update $x \leftarrow 0$ is in a loop. However, after setting x to 0, the loop cannot be executed again. To simplify its analysis, CFR "unrolls" the loop resulting in (2).

```
while $x > 0$ do break $\oplus_{1/2}$ noop end
if $x > 0$ then $x \leftarrow 0$ end                                        (2)
```

Here, x is updated in a separate, *non-probabilistic* **if**-statement and the loop does not change variables. Thus, we sorted out (infeasible) paths where $x \leftarrow 0$ was executed repeatedly. Now, techniques for probabilistic programs can be used for the **while**-loop. The rest of the program can be analyzed by techniques for non-probabilistic programs. In particular, this is important if (1) is part of a larger program.

Our novel CFR algorithm for *probabilistic* integer programs is based on the partial evaluation technique for non-probabilistic programs from [13, 14, 18]. In particular, our algorithm coincides with the classical CFR technique when the program is non-probabilistic. The goal of CFR is to transform a program $\mathcal{P}$ into a program $\mathcal{P}'$ which is "easier" to analyze. In the full version of this paper, we prove that both $\mathcal{P}$ and $\mathcal{P}'$ have the same *expected* runtime (see [26, Thm. 4]). Thus, our approach is not only sound but it also does not increase the expected runtime. We apply CFR only *on-demand* on a subprogram (thus, CFR can be performed in a *modular* way for different subprograms). In practice, we choose the subprogram heuristically and use CFR only on parts of the program where our currently inferred runtime bounds are "not yet good enough".

2 Implementation and Evaluation

Up to now, our complexity analyzer KoAT used the tool iRankFinder [13] for CFR of non-probabilistic programs [18]. To demonstrate the benefits of CFR for complexity analysis of probabilistic programs, we now replaced the call to iRankFinder in KoAT by a native implementation of our new CFR algorithm. KoAT is written in OCaml and it uses Z3 [12] for SMT solving, Apron [20] to generate invariants, and the Parma Polyhedra Library [8] for computations with polyhedra.

We used all 75 probabilistic benchmarks from [27, 29] and added 15 new benchmarks including our leading example and problems adapted from the *Termination Problem Data Base* [33], e.g., a probabilistic version of McCarthy's 91 function. Our benchmarks also contain examples where CFR is useful even if it cannot separate probabilistic from non-probabilistic program parts as in our leading example.

Table 1 shows the results of our experiments. We compared the configuration of KoAT with CFR ("KoAT+CFR") against KoAT without CFR. Moreover, as in [27], we also compared with the main other recent tools for inferring upper bounds on the expected runtimes of probabilistic integer programs (Absynth [29] and eco-imp [7]). As in the *Termination Competition* [17], we used a timeout of 5 minutes per example. The first entry in every cell is the number of benchmarks for which the tool inferred the respective bound. In brackets, we give the corresponding number when only regarding our new examples. The runtime bounds computed by the tools are compared asymptotically as functions which depend on the largest initial absolute value n of all program variables. So for example, KoAT+CFR finds a finite expected runtime bound for 84 of the 90 examples. A linear expected bound (i.e., in $\mathcal{O}(n)$) is found for 56 of these 84 examples, where 12 of these benchmarks are from our new set. AVG(s) is the average runtime in seconds on all benchmarks and AVG^{+}(s) is the average runtime on all successful runs.

	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^{>2})$	$\mathcal{O}(EXP)$	$< \omega$	AVG$^+$(s)	AVG(s)
KoAT+CFR	11 (2)	56 (12)	14	2	1	84 (14)	11.68	11.34
KoAT	9	41 (1)	16 (1)	2	1	69 (2)	2.71	2.41
Absynth	7	35	9	0	0	51	2.86	37.48
eco-imp	8	35	6	0	0	49	0.34	68.02

Table 1 Evaluation of CFR on Probabilistic Programs

The experiments show that similar to its benefits for non-probabilistic programs [18], CFR also increases the power of automated complexity analysis for probabilistic programs substantially, while the runtime of the analyzer may become longer since CFR increases the size of the program. The experiments also indicate that a related CFR technique is not available in the other complexity analyzers. Thus, we conjecture that other tools for complexity or termination analysis of PIPs would also benefit from the integration of our CFR technique.
KoAT's source code, a binary, and a Docker image are available at:

https://koat.verify.rwth-aachen.de/prob_cfr

The website also explains how to use our CFR implementation separately (without the rest of KoAT), in order to access it as a black box by other tools. Moreover, the website provides a *web interface* to directly run KoAT online, and details on our experiments, including our benchmark collection.

References

1 Sheshansh Agrawal, Krishnendu Chatterjee, and Petr Novotný. Lexicographic ranking supermartingales: An efficient approach to termination of probabilistic programs. *Proc. ACM Program. Lang.*, 2(POPL), 2017. doi:10.1145/3158122.

2 Elvira Albert, Puri Arenas, Samir Genaim, and Germán Puebla. Automatic inference of upper bounds for recurrence relations in cost analysis. In *Proc. SAS*, LNCS 5079, pages 221–237, 2008. doi:10.1007/978-3-540-69166-2_15.

3 Elvira Albert, Puri Arenas, Samir Genaim, Germán Puebla, and Damiano Zanardini. Cost analysis of object-oriented bytecode programs. *Theor. Comput. Sci.*, 413(1):142–159, 2012. doi:10.1016/j.tcs.2011.07.009.

4 Elvira Albert, Miquel Bofill, Cristina Borralleras, Enrique Martín-Martín, and Albert Rubio. Resource analysis driven by (conditional) termination proofs. *Theory Pract. Log. Program.*, 19(5-6):722–739, 2019. doi:10.1017/S1471068419000152.

5 Christophe Alias, Alain Darte, Paul Feautrier, and Laure Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *Proc. SAS*, LNCS 6337, pages 117–133, 2010. doi:10.1007/978-3-642-15769-1_8.

6 Martin Avanzini and Georg Moser. A Combination Framework for Complexity. In *Proc. RTA*, LIPIcs 21, pages 55–70, 2013. doi:10.4230/LIPIcs.RTA.2013.55.

7 Martin Avanzini, Georg Moser, and Michael Schaper. A modular cost analysis for probabilistic programs. *Proc. ACM Program. Lang.*, 4(OOPSLA), 2020. URL: https://doi.org/10.1145/3428240.

8 Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. Comput. Program.*, 72:3–21, 2008. doi:10.1016/j.scico.2007.08.001.

9 Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Lena Verscht. A calculus for amortized expected runtimes. *Proc. ACM Program. Lang.*, 7(POPL), 2023. doi:10.1145/3571260.

10 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl. Analyzing Runtime and Size Complexity of Integer Programs. *ACM Trans. Program. Lang. Syst.*, 38:1–50, 2016. doi:10.1145/2866575.

11 Quentin Carbonneaux, Jan Hoffmann, and Zhong Shao. Compositional certified resource bounds. In *Proc. PLDI*, pages 467–478, 2015. doi:10.1145/2737924.2737955.

12 Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *Proc. TACAS*, LNCS 4963, pages 337–340, 2008. doi:10.1007/978-3-540-78800-3_24.

13 Jesús J. Doménech and Samir Genaim. iRankFinder. In *Proc. WST*, page 83, 2018. http://wst2018.webs.upv.es/wst2018proceedings.pdf.

14 Jesús J. Doménech, John P. Gallagher, and Samir Genaim. Control-flow refinement by partial evaluation, and its application to termination and cost analysis. *Theory Pract. Log. Program.*, 19(5-6):990–1005, 2019. doi:10.1017/S1471068419000310.

15 Antonio Flores-Montoya. Upper and lower amortized cost bounds of programs expressed as cost relations. In *Proc. FM*, LNCS 9995, pages 254–273, 2016. doi:10.1007/978-3-319-48989-6_16.

16 Florian Frohn and Jürgen Giesl. Complexity analysis for Java with AProVE. In *Proc. iFM*, LNCS 10510, pages 85–101, 2017. doi:10.1007/978-3-319-66845-1_6.

17 Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In *Proc. TACAS*, LNCS 11429, pages 156–166, 2019. doi:10.1007/978-3-030-17502-3_10.

18 Jürgen Giesl, Nils Lommen, Marcel Hark, and Fabian Meyer. Improving Automatic Complexity Analysis of Integer Programs. In *The Logic of Software. A Tasting Menu of Formal Methods*, LNCS 13360, pages 193–228, 2022. doi:10.1007/978-3-031-08166-8_10.

19 Jan Hoffmann, Ankush Das, and Shu-Chun Weng. Towards automatic resource bound analysis for OCaml. In *Proc. POPL*, pages 359–373, 2017. doi:10.1145/3009837.3009842.

20 Bertrand Jeannet and Antoine Miné. Apron: A library of numerical abstract domains for static analysis. In *Proc. CAV*, LNCS 5643, pages 661–667, 2009. doi:10.1007/978-3-642-02658-4_52.

21 Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest precondition reasoning for expected runtimes of randomized algorithms. *J. ACM*, 65:1–68, 2018. doi:10.1145/3208102.

22 Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. Expected runtime analyis by program verification. In Gilles Barthe, Joost-Pieter Katoen, and Alexandra Silva, editors, *Foundations of Probabilistic Programming*, page 185–220. Cambridge University Press, 2020. doi:10.1017/9781108770750.007.

23 Lorenz Leutgeb, Georg Moser, and Florian Zuleger. Automated expected amortised cost analysis of probabilistic data structures. In *Proc. CAV*, LNCS 13372, pages 70–91, 2022. doi:10.1007/978-3-031-13188-2_4.

24 Nils Lommen, Fabian Meyer, and Jürgen Giesl. Automatic Complexity Analysis of Integer Programs via Triangular Weakly Non-Linear Loops. In *Proc. IJCAR*, LNCS 13385, pages 734–754, 2022. doi:10.1007/978-3-031-10769-6_43.

25 Nils Lommen and Jürgen Giesl. Targeting Completeness: Using Closed Forms for Size Bounds of Integer Programs. In *Proc. FroCoS*, LNCS 14279, pages 3–22, 2023. doi:10.1007/978-3-031-43369-6_1.

26 Nils Lommen, Éléanore Meyer, and Jürgen Giesl. Control-Flow Refinement for Complexity Analysis of Probabilistic Programs in KoAT (Short Paper). In *Proc. IJCAR*, LNCS 14739, pages 233–243, 2024. doi:10.1007/978-3-031-63498-7_14.

27 Fabian Meyer, Marcel Hark, and Jürgen Giesl. Inferring Expected Runtimes of Probabilistic Integer Programs Using Expected Sizes. In *Proc. TACAS*, LNCS 12651, pages 250–269, 2021. doi:10.1007/978-3-030-72016-2_14.

28 Georg Moser and Michael Schaper. From Jinja bytecode to term rewriting: A complexity reflecting transformation. *Inf. Comput.*, 261:116–143, 2018. doi:10.1016/j.ic.2018.05.007.

29 Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: Resource analysis for probabilistic programs. In *Proc. PLDI*, pages 496–512, 2018. URL: https://doi.org/10.1145/3192366.3192394.

30 Lars Noschinski, Fabian Emmes, and Jürgen Giesl. Analyzing Innermost Runtime Complexity of Term Rewriting by Dependency Pairs. *J. Autom. Reason.*, 51:27–56, 2013. doi:10.1007/s10817-013-9277-6.

31 Philipp Schröer, Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, and Christoph Matheja. A Deductive Verification Infrastructure for Probabilistic Programs. *Proc. ACM Program. Lang.*, 7(OOPSLA):2052–2082, 2023. doi:10.1145/3622870.

32 Moritz Sinn, Florian Zuleger, and Helmut Veith. Complexity and resource bound analysis of imperative programs using difference constraints. *J. Autom. Reason.*, 59(1):3–45, 2017. doi:10.1007/s10817-016-9402-4.

33 TPDB (Termination Problem Data Base). URL: https://github.com/TermCOMP/TPDB.

34 Di Wang, David M. Kahn, and Jan Hoffmann. Raising expectations: Automating expected cost analysis with types. *Proc. ACM Program. Lang.*, 4(ICFP), 2020. URL: https://doi.org/10.1145/3408992.

On Closures in String Rewriting

Alfons Geser
HTWK Leipzig

Abstract

The set of overlap closures can be characterized as the semantics of composition trees. Rewrite systems for composition trees can be used to prove that the existence of loops implies the existence of looping forward closures, or to prove the correctness of a characterization of the set of right-hand sides of closures. We show that, for a quasi-terminating string rewrite system, the existence of cycles is equivalent to the existence of looping forward closures. This improves upon a result of Guttag et al. 1983.

2012 ACM Subject Classification Theory of computation → Rewrite systems

Keywords and phrases Termination, String rewriting, Forward Closure, Overlap Closure, Composition Tree, Looping Derivation, Cyclic Derivation

1 Overlap Closures and Composition Trees

We generally assume that a string rewrite system R over an alphabet Σ is given.

A set of closures is usually defined inductively. Guttag et al. [5] introduced overlap closures for term rewriting. The following is a version for string rewriting.

▶ **Definition 1.** *The set OC of* overlap closures *is defined as the least set of Σ^*-pairs that*
(1) *includes R and satisfies*
(2) *if $(s, tx) \in OC$ and $(xu, v) \in OC$ for some $t, x, u \neq \epsilon$ then $(su, tv) \in OC$;*
(2') *if $(s, xt) \in OC$ and $(ux, v) \in OC$ for some $t, x, u \neq \epsilon$ then $(us, vt) \in OC$;*
(3) *if $(s, xuy) \in OC$ and $(u, v) \in OC$ then $(s, xvy) \in OC$;*
(3') *if $(u, v) \in OC$ and $(xvy, t) \in OC$ then $(xuy, t) \in OC$.*

When dealing with closures, it is useful to represent a closure by a composition tree. The composition tree describes the way the closure is formed.

▶ **Definition 2** ([4]). *Define the signature $\Omega = \{1, 2, 2', 3, 3', 4\}$, where 1 is nullary, 4 is ternary, and the other symbols are binary. The set CT of* composition trees *is defined as the set of ground terms over Ω.*

▶ **Definition 3.** *A composition tree c represents a set $\langle c \rangle$ of Σ^*-pairs, as follows:*

$$\begin{aligned}
\langle 1 \rangle &= R, \\
\langle 2(c_1, c_2) \rangle &= \{(su, tv) \mid (s, tx) \in \langle c_1 \rangle, (xu, v) \in \langle c_2 \rangle, t, x, u \neq \epsilon\}, \\
\langle 2'(c_1, c_2) \rangle &= \{(us, vt) \mid (s, xt) \in \langle c_1 \rangle, (ux, v) \in \langle c_2 \rangle, t, x, u \neq \epsilon\}, \\
\langle 3(c_1, c_2) \rangle &= \{(s, xvy) \mid (s, xuy) \in \langle c_1 \rangle, (u, v) \in \langle c_2 \rangle\}, \\
\langle 3'(c_1, c_2) \rangle &= \{(xuy, t) \mid (u, v) \in \langle c_1 \rangle, (xvy, t) \in \langle c_2 \rangle, \}, \\
\langle 4(c_1, c_2, c_3) \rangle &= \{(swu, tzv), \mid (s, tx) \in \langle c_1 \rangle, (u, yv) \in \langle c_2 \rangle, (xwy, z) \in \langle c_3 \rangle, t, x, y, v \neq \epsilon\} \ .
\end{aligned}$$

This is conveniently extended to sets S of composition trees:

$$\langle S \rangle = \bigcup_{c \in S} \langle c \rangle.$$

For each case in the inductive definition, we have a symbol in Ω that names this case. For instance symbol 1 means Case (1): the overlap closure is a rule in R. If an overlap closure (s, tx) is represented by the composition tree c_1 and an overlap closure (xu, v) is represented by the composition tree c_2, then the overlap closure (su, tv) obtained by Case (2), is represented by the composition tree $2(c_1, c_2)$. We may so write Case (2) more succinctly as:

If $\langle c_1 \rangle \subseteq \mathsf{OC}$ and $\langle c_2 \rangle \subseteq \mathsf{OC}$ then $\langle 2(c_1, c_2) \rangle \subseteq \mathsf{OC}$.

Let CT_0 denote the composition trees that do not contain the symbol 4. Then the set of overlap closures is characterized like this:

▶ **Lemma 4.** *$OC = \langle CT_0 \rangle$.*

Adding symbol 4 does not increase expressiveness, since we have $\langle 4(c_1, c_2, c_3) \rangle \subseteq \langle 2(c_1, 2'(c_2, c_3)) \rangle$.

▶ **Lemma 5.** *$OC = \langle CT \rangle$.*

Symbol 4 will be needed later in Section 3.

▶ **Example 6.** Suppose $R = \{ad \to abb, bd \to db, bdbdd \to c\}$. Then (bd, db) is a closure by Case (1). Using this closure twice, we get a closure (bdd, ddb), by Case (2) with $t = d = u$, $x = b$. From the two closures (bd, db) and $(bdbdd, c)$, we get a closure $(bbddd, c)$ by Case (3′) with $v = db$, $x = b$, $y = dd$. From the closures (ad, abb), (bdd, ddb), and $(bbddd, c)$, we get a symbol-4 closure $(addbdd, acb)$ by $s = ad$, $t = a$, $x = bb$, $u = bdd$, $y = dd$, $v = b$, $w = d$, $z = c$. With composition trees, this is expressed as $(bd, db) \in \langle 1 \rangle$, $(bdd, ddb) \in \langle 2(1, 1) \rangle$, $(bbddd, c) \in \langle 3'(1, 1) \rangle$, and $(addbdd, acb) \in \langle 4(1, 2(1, 1), 3'(1, 1)) \rangle$. The same closure, $(addbdd, acb)$, may be assigned a different composition tree: $(add, abdb) \in \langle 2(1, 1) \rangle$, $(bdd, ddb) \in \langle 2(1, 1) \rangle$, and $(bdbdd \to c) \in \langle 1 \rangle$ yield the symbol-4 closure $(addbdd, acb) \in \langle 4(2(1, 1), 2(1, 1), 1) \rangle$, by $s = add$, $t = a$, $x = bdb$, $u = bdd$, $y = dd$, $v = b$, $w = \epsilon$, $z = c$. ◀

2 Looping Closures and Tree Rearrangement

In order to demonstrate what one can do with composition trees, we recall a characterization result for the existence of loops, i.e. of derivations of the form $t \to^+ utv$. Let us call an overlap closure of the form (t, utv) a *looping overlap closure.* Likewise, let us call a forward closure of the form (t, utv) a *looping forward closure.*

The starting point is the following characterization of overlap closures.

▶ **Lemma 7.** *[4] OC is equal to the set of pairs (t, u) such that there is a derivation from t to u in which every inner interposition of t is touched.*

Here "inner interposition" means position between letters.

▶ **Example 8.** In Example 6, to the overlap closure $(addbdd, acb)$ there are the derivations

$$\underline{a|d}|d|b|d|d \to abb|d|\underline{b|d}|d \to abb|d|d\underline{b|d} \to ab\underline{b|d}|ddb \to a\underline{bdb|dd}b \to acb,$$

$$a|d|d|\underline{b|d}|d \to a|d|d|d\underline{b|d} \to \underline{a|d}|d|ddb \to ab\underline{b|d}|ddb \to a\underline{bdb|dd}b \to acb,$$

corresponding to the composition trees $4(1, 2(1, 1), 3'(1, 1))$ and $4(2(1, 1), 2(1, 1), 1)$, respectively. Matching left hand sides are underlined. The inner interpositions that are not yet touched are indicated by a stroke. ◀

▶ **Lemma 9.** *[4] Every SRS that has a loop, has a looping overlap closure.*

Proof. If $(\epsilon, r) \in R$ then R has a loop $\epsilon \to r$, and $(\epsilon, r) \in \mathsf{OC}$. So assume that $\epsilon \notin \mathsf{lhs}(R)$. The proof is by induction on the length of t in the given derivation $t = t_0 \to t_1 \to \cdots \to t_n = utv$. If every inner interposition of t is touched during this derivation, then we have an overlap closure (t, utv) by Lemma 7. Otherwise, there is an inner inperposition in t that is not touched, and which has therefore a residual, unique by $\epsilon \notin \mathsf{lhs}(R)$, in every string t_i. At these residuals, the derivation can be split into two derivations $t'_0 \to^= t'_1 \to^= \cdots \to^= t'_n$ and $t''_0 \to^= t''_1 \to^= \cdots \to^= t''_n$ such that $t_i = t'_i t''_i$ for all $i \in \{0, \ldots, n\}$. We have $utv = ut'_0 t''_0 v = t'_n t''_n$. By a case analysis on whether $|ut'_0| < |t'_n|$ or $|ut'_0| > |t'_n|$ or $|ut'_0| = |t'_n|$, one of the two derivations forms a loop, and by the inductive hypothesis the claim follows. ◀

Lemma 9 can be strengthened towards forward closures. The set of forward closures, introduced by Lankford and Musser [7] for term rewriting, can be defined in the string rewriting case simply as follows:

▶ **Definition 10.** *For a string rewrite system R the set* FC *of forward closures is defined as* $\mathsf{FC} = \langle \mathsf{Term}(\{1, 2, 3\}) \rangle$.

Here, $\mathsf{Term}(\{1, 2, 3\})$ denotes the set of ground terms over the signature $\{1, 2, 3\}$, a subsignature of Ω.

▶ **Theorem 11.** *[4] Every SRS that has a loop, has a looping forward closure.*

For the proof, we first introduce an interpretation ϱ on composition trees by $\varrho(1) = 2$,

$$\varrho(2(c_1, c_2)) = \varrho(3(c_1, c_2)) = \varrho(c_1)\varrho(c_2), \qquad \varrho(2'(c_1, c_2)) = \varrho(3'(c_1, c_2)) = \varrho(c_1)\varrho(c_2) + 1 \ .$$

For composition trees c_1, c_2 we define $c_1 <_\varrho c_2$ by $\varrho(c_1) < \varrho(c_2)$, and we say that c_1 is smaller than c_2. The order $<_\varrho$ is a reduction order.

▶ **Lemma 12.** *If there is a looping overlap closure with the composition tree $3'(c_1, c_2)$ or $2'(c_1, c_2)$, then there is also a looping overlap closure with a smaller composition tree.*

Proof. Let $(u, v) \in \langle c_1 \rangle$ and let $(xvy, t) \in \langle c_2 \rangle$ such that $(xuy, t) \in \langle 3'(c_1, c_2) \rangle$, by the definition of $\langle 3'(c_1, c_2) \rangle$. Moreover let this overlap closure be looping, i.e. let $t = rxuys$ for some $r, s \in \Sigma^*$. Then (xvy, t) and (u, v) form the looping overlap closure $(xvy, rxvys) \in \langle 3(c_2, c_1) \rangle$, and $\varrho(3(c_2, c_1)) < \varrho(3'(c_1, c_2))$ holds.

Similarly, one can show, by a case analysis, that to a looping overlap closure with composition tree $2'(c_1, c_2)$ there is a looping overlap closure with one of the smaller composition trees $c_1, 2(c_1, c_2), 3(c_2, c_1)$. ◀

▶ **Definition 13.** *The sets $[T]$ and $[F]$ of composition trees are given by the regular tree grammar with variables T, F, start variable T and rules*

$$T \to F \mid 2'(T, T) \mid 3'(T, T), \qquad F \to 1 \mid 2(F, F) \mid 3(F, F).$$

Note that by definition, $[F] = \mathsf{Term}(\{1, 2, 3\})$ and so $\langle [F] \rangle = \mathsf{FC}$ holds.

We now construct a term rewrite system P on Ω that satisfies three conditions:

1. P has a subset of $[T]$ as its normal forms. It does so by removing all $2'$ and $3'$ symbols from the left and right arguments of 2 and of 3. In other words, its left-hand sides are $f(g(c_1, c_2), c_3)$ and $f(c_1, g(c_2, c_3))$ where $f \in \{2, 3\}$ and $g \in \{2', 3'\}$.
2. P is terminating by $<_\varrho$.
3. For every left-hand side u in P, and the set V of all corresponding right-hand sides, we get $\langle u \rangle \subseteq \langle V \rangle$. Let us call this condition *semantic coverage.*

Semantic coverage can be rephrased as follows:

▶ **Lemma 14.** *For every $c \in \mathsf{CT}$ that admits a P rewrite step, and for every $(s,t) \in \langle c\rangle$ there is a $c' \in \mathsf{CT}$ such that both $c \to_P c'$ and $(s,t) \in \langle c'\rangle$.*

Note that c' may depend on s and t. Suitable such rules from P each with left-hand side $3(2'(c_1,c_2),c_3)$ are $3(2'(c_1,c_2),c_3) \to 2'(c_1,2(c_2,c_3))$, $3(2'(c_1,c_2),c_3) \to 2'(3(c_1,c_3),c_2)$, $3(2'(c_1,c_2),c_3) \to 2'(c_1,3(c_2,c_3))$. The term rewrite system P has 14 rules.

Proof of Theorem 11. Suppose R has a loop. By Lemma 9, it has a looping overlap closure (t,utv). We use induction on the size of the composition tree c of (t,utv). If c contains no $2'$ nor $3'$ symbols then it is the composition tree of a forward closure, and we are done. If $c = 3'(c_1,c_2)$ or $c = 2'(c_1,c_2)$ for some c_1,c_2, then by Lemma 12, there is an overlap closure with a smaller composition tree, and the claim follows by inductive hypothesis. Otherwise, the composition tree c admits a P rewrite step, $c \to_P c'$, and (t,utv) admits the smaller composition tree c'. Then again the claim follows by inductive hypothesis. ◀

Guttag et al. [5] show that a quasi-terminating string rewrite system has a cycle if and only if, it has a cyclic overlap closure, i.e. an overlap closure of the form (t,t). This result can be strengthened towards forward closure, as is shown next.

▶ **Theorem 15.** *If R is quasi-terminating then every loop is a cycle, i.e. whenever $t \to^+ utv$ then $u = \epsilon = v$.*

Proof. Suppose that $t \to^+ utv$ is a proper loop, i.e. $|uv| > 0$ holds. Then the infinite derivation $t \to^+ utv \to^+ u^2tv^2 \to^+ \dots$ shows that t has infinitely many descendants: $|t| < |utv| < |u^2tv^2| < \dots$. Hence R is not quasi-terminating. ◀

So for quasi-terminating R, if R has a cycle, then it has a loop, then it has a looping forward closure by Theorem 11. Conversely, if R has a looping forward closure, it has a loop, which is a cycle by Theorem 15. This proves:

▶ **Corollary 16.** *Let R be quasi-terminating. Then R has a cycle if, and only if, R has a looping* forward closure.

3 Right-hand Sides of Closures

In order to arrive at an inductive definition of the set of right-hand sides of forward closures, one needs an inductive definition of the set of forward closures that descends only at the left, i.e. the composition tree has only 1 in all its right branches. Hermann [6, Corollaire 2.16] introduced such a characterization for term rewriting:

▶ **Definition 17.** *For a string rewrite system R, let the set FC' be defined as the least set of derivations that includes R and satisfies*
1. *if $(s,tx) \in \mathsf{FC}'$ and $(xu,v) \in R$ for $t,x,u \neq \epsilon$ then $(su,tv) \in \mathsf{FC}'$.*
2. *if $(s,xuy) \in \mathsf{FC}'$ and $(u,v) \in R$ then $(s,xvy) \in \mathsf{FC}'$,*

▶ **Theorem 18.** *For every string rewrite system R, we have $\mathsf{FC}' = \mathsf{FC}$.*

The point of Hermann's characterization is that an impoverished variant inductively characterizes the set $\mathsf{rhs}(\mathsf{FC})$ of right-hand sides of forward closures.

▶ **Corollary 19.** *[2] For any string rewrite system R, the set $\mathsf{rhs}(\mathsf{FC})$ is equal to the least set S of strings that includes $\mathsf{rhs}(R)$ and satisfies*

1. *if $tx \in S$ and $(xu, v) \in R$ for $t, x, u \neq \epsilon$ then $tv \in S$.*
2. *if $xuy \in S$ and $(u, v) \in R$ then $xvy \in S$,*

This characterization of $\mathsf{rhs}(\mathsf{FC})$ is the starting point of several termination proof methods based on Dershowitz's [1] characterization of termination by termination on $\mathsf{rhs}(\mathsf{FC})$.

One can use composition trees to render Hermann's definition more compactly:

▶ **Definition 20.** *The set $[H]$ is given by the regular tree grammar with the start variable H and rules $H \to 1 \mid 2(H, 1) \mid 3(H, 1)$.*

▶ **Lemma 21.** *For every string rewrite system R, we have $\mathsf{FC}' = \langle [H] \rangle$.*

Theorem 18 then amounts to $\langle \mathsf{Term}[\{1, 2, 3\}) \rangle = \langle [H] \rangle$. A proof can be done like in Section 2: First an interpretation δ on $\mathsf{Term}(\{1, 2, 3\})$ is defined by $\delta(1) = 2$ and $\delta(f(c_1, c_2)) = \delta(c_1) + 2 \cdot \delta(c_2)$ for all $f \in \{2, 3\}$. The interpretation δ induces a reduction order $<_\delta$ on $\mathsf{Term}(\{1, 2, 3\})$. Then a term rewrite system Q_F on $\{1, 2, 3\}$ is defined which (1) has a subset of $[H]$ as its set of normal forms, (2) terminates by $<_\delta$, and (3) satisfies semantic coverage. For this purpose, it must remove all 2 and 3 symbols from the right arguments of 2 and of 3. In other words, its left-hand sides are $f(c_1, g(c_2, c_3))$ where $f, g \in \{2, 3\}$. It turns out that Q_F must comprise the rules

$$\begin{array}{ll}
2(c_1, 2(c_2, c_3)) \to 2(2(c_1, c_2), c_3), & 2(c_1, 2(c_2, c_3)) \to 2(3(c_1, c_2), c_3), \\
2(c_1, 3(c_2, c_3)) \to 3(2(c_1, c_2), c_3), & 3(c_1, 2(c_2, c_3)) \to 3(3(c_1, c_2), c_3), \\
3(c_1, 3(c_2, c_3)) \to 3(3(c_1, c_2), c_3). &
\end{array}$$

This finishes the proof sketch of Theorem 18.

Now we aim at a similar characterization of the set of *overlap closures*. The following material is an excerpt from Geser et al. [3].

▶ **Definition 22.** *The set CT_N is given by the regular tree grammar with variables T, D (top, deep), start variable T, and rules*

$$T \to 3'(1, T) \mid D, \qquad D \to 1 \mid 2(D, 1) \mid 2'(D, 1) \mid 3(D, 1) \mid 4(D, D, 1).$$

▶ **Definition 23.** *The sets OC_N and OC' are defined as $\mathsf{OC}_N = \langle \mathsf{CT}_N \rangle$ and $\mathsf{OC}' = \langle [D] \rangle$.*

The rules for T represent an initial derivation before a closure in OC':

▶ **Lemma 24.** $\mathsf{OC}_N = \{(s, t) \mid s \to_R^* s' \wedge (s', t) \in \mathsf{OC}'\}$.

The definition of OC' could also be spelled out as an inductive definition similar to that of OC, where closures are overlapped with rules, Case ($3'$) is dropped and Case (4) is added.

▶ **Theorem 25.** *For every string rewrite system R, we have $\mathsf{OC} = \mathsf{OC}_N$.*

Then by Lemma 24, we have $\mathsf{rhs}(\mathsf{OC}) = \mathsf{rhs}(\mathsf{OC}_N) = \mathsf{rhs}(\mathsf{OC}')$, whence we get:

▶ **Corollary 26.** *$\mathsf{rhs}(\mathsf{OC})$ is equal to the least set S that includes $\mathsf{rhs}(R)$ and satisfies*

1. *if $tx \in S$ and $(xu, v) \in R$ for some $t, x, u \neq \epsilon$ then $tv \in S$;*
2. *if $xt \in S$ and $(ux, v) \in R$ for some $t, x, u \neq \epsilon$ then $vt \in S$;*
3. *if $xuy \in S$ and $(u, v) \in R$ then $xvy \in S$;*
4. *if $tx \in S$ and $yv \in S$ and $(xwy, z) \in R$ for some $t, x, y, v \neq \epsilon$ then $tzv \in S$.*

This characterization of $\mathsf{rhs}(\mathsf{OC})$ turns out useful in proofs of relative termination, where the right-hand sides of forward closures cannot be applied.

In the remainder of this section, a proof of Theorem 25 will be sketched again in a similar way to the proof in Section 2. We are going to construct a term rewrite system Q on Ω that has a subset of CT_N as its set of normal forms. It must remove all non-1 symbols from the left argument of $3'$, and remove all non-1 symbols from the rightmost argument of $2, 2', 3$, and 4. Also, it must remove all $3'$ that are below some non-$3'$. These conditions already determine the set of left-hand sides of Q. For each left-hand side ℓ, the set of corresponding right-hand sides must cover ℓ semantically. These right-hand sides are obtained by a case analysis. The term rewrite system Q has 55 rules. For lack of space, only a few rules are exemplified here.

We bubble-up $3'$ symbols, e. g., $2(3'(c_1, c_2), c_3) \to 3'(c_1, 2(c_2, c_3))$, and we rotate to move non-1 symbols, e. g., $2(c_1, 2(c_2, c_3)) \to 2(2(c_1, c_2), c_3)$. Rotation below $3'$ goes from left to right, all other rotations go from right to left. The rules $2(c_1, 2'(c_2, c_3)) \to 4(c_1, c_2, c_3)$ and $2'(c_1, 2(c_2, c_3)) \to 4(c_2, c_1, c_3)$ show that symbol 4 cannot be avoided. Of course, Q_F is a subset of Q.

Termination of Q follows from the reduction order $>$ given by a lexicographic combination of an interpretation ρ that decreases under rotation, and an interpretation σ that decreases under bubbling. A lemma like Lemma 14 takes care of the semantic coverage property.

Proof of Theorem 25. For "$\supseteq$", observe that $\mathsf{CT} \supseteq \mathsf{CT}_N$ whence $\mathsf{OC} = \langle \mathsf{CT} \rangle \supseteq \langle \mathsf{CT}_N \rangle = \mathsf{OC}_N$ by Lemma 5 and the definition of OC_N. For "$\subseteq$", we prove that $c \in \mathsf{CT}$ and $(s, t) \in \langle c \rangle$ implies $(s, t) \in \langle \mathsf{CT}_N \rangle$. We do so by induction on c, ordered by the reduction order $>$. If c is in Q-normal form then $c \in \mathsf{CT}_N$, and so $\langle c \rangle \subseteq \langle \mathsf{CT}_N \rangle$. Now suppose that c admits a Q rewrite step. Then by semantic coverage there is $c' \in \mathsf{CT}$ such that both $c \to_Q c'$ and $(s, t) \in \langle c' \rangle$. From $c > c'$, the claim follows by inductive hypothesis for c'. ◀

Acknowledgements: I am indebted to Dieter Hofbauer and Johannes Waldmann for their plentiful feedback. Hans Zantema† encouraged me to study looping derivations. An anonymous referee provided valuable advice.

References

1 Nachum Dershowitz. Termination of linear rewriting systems. In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981, Proceedings*, volume 115 of *LNCS*, pages 448–458. Springer, 1981.

2 Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting systems. *Appl. Algebra Eng. Commun. Comput.*, 15(3-4):149–171, 2004.

3 Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Sparse Tiling Through Overlap Closures for Termination of String Rewriting. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019)*, volume 131 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 21:1–21:21, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

4 Alfons Geser and Hans Zantema. Non-looping string rewriting. *Theoret. Informatics Appl.*, 33(3):279–301, 1999.

5 John V. Guttag, Deepak Kapur, and David R. Musser. On proving uniform termination and restricted termination of rewriting systems. *SIAM J. Comput.*, 12(1):189–214, 1983.

6 Miki Hermann. *Divergence des systèmes de réécriture et schématisation des ensembles infinis de termes.* Habilitation, Université de Nancy, France, March 1994.

7 Dallas S. Lankford and D. R. Musser. A finite termination criterion. Technical report, Information Sciences Institute, Univ. of Southern California, Marina-del-Rey, CA, 1978.

A New Proof for Soundness of Right-Forward Closures for Termination Analysis

René Thiemann
University of Innsbruck, Austria

Abstract

We consider the termination problem of term rewrite systems (TRSs). Dershowitz proved that termination starting from arbitrary terms is equivalent to termination starting from all terms in the right-forward closure (RFC), provided that the TRS is right-linear or orthogonal. In this paper we provide a new proof of this result, and also include a later result that one can weaken orthogonality to locally confluent overlay TRSs. All proofs have been verified in Isabelle/HOL.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Logic and verification

Keywords and phrases Term rewriting, Narrowing, Right-Forward Closures, Isabelle/HOL

Acknowledgements I want to thank the WST reviewers for their helpful comments.

1 Introduction

A relation $\to$ is strongly normalizing w.r.t. a set of starting objects T, written $SN(\to, T)$, iff there is no infinite sequence

$$t_0 \to t_1 \to t_2 \to t_3 \to \dots \qquad (\star)$$

with $t_0 \in T$. If there is no restriction on the set of starting objects we just write $SN(\to)$.

In this paper we consider termination of first-order unsorted term rewrite systems $\mathcal{R}$ where $\to$ is the rewrite relation $\to_{\mathcal{R}}$ of $\mathcal{R}$ and T is either the set of all terms, or $RFC(\mathcal{R})$, the right-hand sides of forward closure of $\mathcal{R}$, also known as the right-forward closure of $\mathcal{R}$.

This paper will illustrate a novel proof of the following result of Dershowitz, and this theorem will also be generalized to a larger class of TRSs.

▶ **Theorem 1** ([3]). *Let $\mathcal{R}$ be a right-linear or orthogonal TRS. Then*

$$SN(\to_{\mathcal{R}}) \longleftrightarrow SN(\to_{\mathcal{R}}, RFC(\mathcal{R})).$$

Note that Theorem 1 has applications in termination proving, e.g., in combination with the match-bounds technique [5, 6, 10]. Here, the change from all starting terms to just $RFC(\mathcal{R})$ can become crucial for a successful termination proof.

There is a challenge in verifying the proof of Theorem 1, e.g., in a formal setting.

One of the problems is that the definition of $RFC(\mathcal{R})$ as it is understood nowadays (defined via narrowing), differs from Dershowitz' original definition [3] (defined via infinite chains, where in infinite derivations—such as ($\star$)—active positions are marked and have to satisfy certain conditions). Moreover, the original proof stays at an intuitive level, e.g., by claiming that certain steps can be reordered in an infinite derivation, without providing further details.

Dershowitz and Hoot provide an alternative definition of forward closures [4, Definition 6] that is based on narrowing, and there is a clear correspondence to $RFC(\mathcal{R})$. Unfortunately, the connection between infinite chains [3] and *infinite* forward closures as they occur in [4, Theorem 6] remains at an intuitive level in this paper. Note that this paper also includes an improved result: Theorem 1 is still true if one replaces *orthogonal TRS* by *locally confluent overlay TRS* [4, Theorem 6].

The definition of *infinite* forward closures is made formal by Geupel. He also provides a detailed proof of Theorem 1 for non-overlapping TRSs [8, Theorem 2], but does not consider right-linear TRSs.

A detailed proof of Theorem 1 for the case of right-linear TRSs is partly provided by Zantema [13, Section 6]: he restricts the theorem to string rewrite systems.

Our main contributions are:

- The development of a novel proof of Theorem 1 for a definition of $\mathit{RFC}(\mathcal{R})$ that is based on narrowing, with the inclusion of locally confluent overlay TRSs.
- The proof does not require any reordering of steps as in the original proof.
- The full proof is formally verified in Isabelle/HOL, cf. IsaFoR version 3.5 [12].

2 Preliminaries

We assume familiarity with term rewriting [2] and recall important notions and notations.

We consider first order terms $s, t, \ell, r, \ldots \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ that consist of variables $x, y, z, \ldots \in \mathcal{V}$ and function applications $f(t_1, \ldots, t_n)$ for n-ary symbols $f \in \mathcal{F}$. Substitutions $\sigma, \delta, \mu, \ldots$ are mappings from $\mathcal{V}$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$, and we write $t\sigma$ to substitute each variable x in a term t by $\sigma(x)$. Given a substitution σ and some set of variables X we write $\sigma \restriction X$ for the substitution that restricts σ to X, i.e., $(\sigma \restriction X)(x) = \sigma(x)$ if $x \in X$, and $(\sigma \restriction X)(x) = x$ if $x \notin X$. We denote the set of positions of t by $\mathit{Pos}(t)$, $t|_p$ is the subterm of t at position p and $t[s]_p$ replaces $t|_p$ by s in t at position p. The set $\mathit{FPos}(t)$ is the set of function positions, i.e., $\mathit{FPos}(t) = \{p \in \mathit{Pos}(t) \mid t|_p \notin \mathcal{V}\}$. The strict subterm relation is denoted by $\rhd$, i.e., $s \rhd t$ iff $s \neq t$ and $s|_p = t$ for some $p \in \mathit{Pos}(s)$. Term t is linear, if no variable occurs more than once in t; $\mathit{Vars}(t)$ is the set of variables of term t. We write $\mathit{mgu}(s, t) = \sigma$ if σ is a most general unifier of s and t.

A TRS $\mathcal{R}$ is a set of rules $\ell \to r$ such that $\ell \notin \mathcal{V}$ and $\mathit{Vars}(\ell) \supseteq \mathit{Vars}(r)$. The rewrite relation of $\mathcal{R}$ is defined as $s \to_{\mathcal{R},p} t$ iff $p \in \mathit{FPos}(t)$, $\ell \to r \in \mathcal{R}$, $s|_p = \ell\mu$, and $t = s[r\mu]_p$ for some p, ℓ, r, μ. Narrowing of $\mathcal{R}$ is defined as $s \rightsquigarrow_{\mathcal{R},\mu \restriction \mathit{Vars}(s),p} t$ iff $p \in \mathit{FPos}(t)$, $\ell \to r \in \mathcal{R}$, $\mathit{mgu}(s|_p, \ell) = \mu$, and $t = s[r]_p\mu$, for some p, ℓ, r, μ where—in contrast to the rewrite relation—unification is used instead of matching. For narrowing it is always assumed that the variables of rule $\ell \to r$ are renamed apart so that they are disjoint to $\mathit{Vars}(s)$. We often omit the subscripts in the rewrite and narrowing relation if these are not relevant or if they are clear from the context.

The set of right-hand sides of a TRS $\mathcal{R}$ is denoted by $\mathit{rhs}(\mathcal{R})$, and $\mathcal{R}$ is right-linear if all terms in $\mathit{rhs}(\mathcal{R})$ are linear. We refer to the definition of critical pairs to the textbook [2]. An *orthogonal* TRS does not have any critical pairs. In an *overlay* TRS all critical pairs stem from overlaps at the root. A TRS is *locally confluent* if all its critical pairs are joinable.

We are now able to define the right-forward closure of a TRS and recast Dershowitz' and our result using the notations that have been introduced.

▶ **Definition 2** ($\mathit{RFC}(\mathcal{R})$). *$\mathit{RFC}(\mathcal{R})$ is the least set that contains $\mathit{rhs}(\mathcal{R})$ and is closed under narrowing.*

$$\mathit{RFC}(\mathcal{R}) = \{t \mid s \in \mathit{rhs}(R) \wedge s \rightsquigarrow^*_{\mathcal{R}} t\} = \rightsquigarrow^*_{\mathcal{R}}(\mathit{rhs}(\mathcal{R}))$$

▶ **Theorem 3** ([3]). *Let $\mathcal{R}$ be a right-linear or orthogonal TRS. Then $\mathit{SN}(\to_{\mathcal{R}})$ is satisfied, if $\mathit{SN}(\to_{\mathcal{R}}, \mathit{RFC}(\mathcal{R}))$ or $\mathit{SN}(\rightsquigarrow_{\mathcal{R}}, \mathit{rhs}(\mathcal{R}))$.*

▶ **Theorem 4** ([4, 12]). *Theorem 3 is still valid, if one weakens "orthogonal TRS" to "locally confluent overlay TRS".*

▶ **Example 5.** A subset of Toyama's TRS is easily proved to be terminating by Theorems 3 and 4. For the TRS $\mathcal{R} = \{f(a,b,x) \to f(x,x,x), g(x,y) \to x\}$ we obtain $\mathit{rhs}(\mathcal{R}) = \{f(x,x,x), x\}$. Since both terms in $\mathit{rhs}(\mathcal{R})$ are normal forms w.r.t. narrowing, in particular $\mathit{SN}(\rightsquigarrow_{\mathcal{R}}, \mathit{rhs}(\mathcal{R}))$ is satisfied. Thus, by Theorem 3 (or by the stronger Theorem 4) we may conclude termination of $\mathcal{R}$.

▶ **Example 6.** The full version of Toyama's TRS $\{f(a,b,x) \to f(x,x,x), g(x,y) \to x, g(x,y) \to y\}$ is non-terminating. This TRS shows that one cannot just drop the pre-conditions on the TRS in Theorems 3 and 4. The reason is that—similarly to Example 5—narrowing is terminating if one starts from an arbitrary right-hand side of this TRS.

3 A Novel Proof of Theorem 3 and of Theorem 4

Before we start with the main proof of Theorem 4, let us first state some easy connections between rewriting and narrowing.

▶ **Lemma 7.**
- *If $s \to_p t$ then $s \rightsquigarrow_{\mu,p} t\mu$ for some variable renaming μ.*
- *If $s \rightsquigarrow_{\mu,p} t$ then $s\mu \to_p t$.*
- *If $s \rightsquigarrow_{\mu,p} t$ for some variable renaming μ then $s \to_p t\mu^{-1}$.*

The first property of Lemma 7 shows that narrowing can simulate rewriting; the second property is for the reverse direction, provided that one instantiates the starting term s by the substitution μ from the narrowing step; and finally, if narrowing just uses variable renamings μ, then it is basically rewriting (modulo variable renamings).

Note that $\mathit{SN}(\to_{\mathcal{R}})$ is equivalent to $\mathit{SN}(\to_{\mathcal{R}}, \sigma(\mathit{rhs}(\mathcal{R})))$, i.e., for termination analysis it suffices to consider all instances $r\sigma$ of right-hand sides r of $\mathcal{R}$: this can be seen by using a minimal non-terminating term argument as it is used for dependency pairs [1].

The idea of the RFC termination argument is now that narrowing can recover σ, so one just needs to start from some right-hand side and will then piecewise reconstruct σ to simulate the non-terminating sequence, and at some point all narrowing steps will become rewrite steps as in the third property of Lemma 7. To this end, we need the following result which permits us to connect a rewrite step with source $s\sigma$ with a narrowing step with source s, provided that the position of the rewrite step is within s: one can decompose σ into $\mu\sigma'$ and the rewrite step is already possible with $s\mu$.

▶ **Lemma 8** (One-step simulation). *If $s\sigma \to_p t$ and $p \in \mathit{FPos}(s)$ then there are s' and μ and σ' such that*

$$s \rightsquigarrow_{\mu,p} s' \text{ and } \sigma = \mu\sigma' \text{ and } t = s'\sigma'$$

or illustrated a bit differently where now the rewrite step is underlined:

$$s\sigma = \underline{s\mu}\sigma' \to \underline{s'}\sigma' = t \quad \text{and} \quad s \rightsquigarrow_{\mu} s'$$

Now assume there is some non-terminating derivation

$$s_0\sigma_0 = t_0 \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \dots \tag{1}$$

The aim is to find some invariant on s and σ such that it is satisfied for s_0 and σ_0. Moreover, the invariant must be preserved by Lemma 8, i.e., the terms s' and substitution σ' of this

$$\begin{tikzcd}
t_0 \arrow[r] \arrow[d, equal] & t_1 \arrow[r] \arrow[d, equal] & t_2 \arrow[r] \arrow[d, equal] & \cdots \\
s_0\sigma_0 \arrow[d, equal] & s_1\sigma_1 \arrow[d, equal] & s_2\sigma_2 \arrow[d, equal] & \cdots \\
s_0\mu_1\sigma_1 \arrow[ur] & s_1\mu_2\sigma_2 \arrow[ur] & s_2\mu_3\sigma_3 \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
s_0 \arrow[r, squiggly, "\mu_1"] & s_1 \arrow[r, squiggly, "\mu_2"] & s_2 \arrow[r, squiggly, "\mu_3"] & \cdots
\end{tikzcd}$$

Figure 1 Infinite Application of One-Step Simulation

lemma should also satisfy it. The reason is that with such an invariant an infinite narrowing sequence can be constructed as in Figure 1. Moreover, one can further prove that eventually all narrowing steps must be rewrite steps. Thus, $\neg SN(\rightsquigarrow_{\mathcal{R}}, \{s_0\})$ and $\neg SN(\to_{\mathcal{R}}, \rightsquigarrow_{\mathcal{R}} (\{s_0\}))$ can be derived, and hence both $SN(\rightsquigarrow_{\mathcal{R}}, \{s_0\})$ and $SN(\to_{\mathcal{R}}, \rightsquigarrow^*_{\mathcal{R}} (\{s_0\}))$ are sufficient criteria for proving $SN(\to_{\mathcal{R}}, \{s_0\sigma_0\})$.

▶ **Lemma 9.** *Let I be some invariant such that all of the following is satisfied:*
- *there is an infinite derivation as in (1),*
- *$I(s_0, \sigma_0)$,*
- *if $I(s, \sigma)$ and $s\sigma \to_p t$ then $p \in FPos(s)$, and*
- *if $I(s, \sigma)$ and $s \rightsquigarrow_\mu s'$ and $\sigma = \mu\sigma'$ then $I(s', \sigma')$.*

Then $\neg SN(\rightsquigarrow_{\mathcal{R}}, \{s_0\})$ and $\neg SN(\to_{\mathcal{R}}, \rightsquigarrow^*_{\mathcal{R}} (\{s_0\}))$.

Proof. The proof works as follows. First, by using the invariant one can indeed construct the infinite narrowing sequence as indicated in Figure 1. This already proves $\neg SN(\rightsquigarrow_{\mathcal{R}}, \{s_0\})$. Afterwards, it only needs to be shown that eventually all μ_i are variable renamings, so that the narrowing steps in Figure 1 can eventually be turned into rewrite steps by Lemma 7.

To this end, observe that

$$t_0 = s_0\sigma_0 = s_0\mu_1\sigma_1 = s_0\mu_1\mu_2\sigma_2 = s_0\mu_1\mu_2\mu_3\sigma_3 = \dots.$$

Thus, $s_0\mu_1\mu_2 \dots \mu_n$ is an instance of t_0 for all n. Therefore, eventually all μ_i are variable substitutions, i.e., $\mu_i : \mathcal{V} \to \mathcal{V}$ for almost every i. For these variable substitutions we infer

$$Vars(s_{i+1}\mu_{i+1}) = \mu_{i+1}(Vars(s_{i+1})) \subseteq \mu_{i+1}(Vars(s_i\mu_i))$$

because $s_i\mu_i \to s_{i+1}$ implies $Vars(s_i\mu_i) \supseteq Vars(s_{i+1})$, and hence

$$|Vars(s_{i+1}\mu_{i+1})| \leq |\mu_{i+1}(Vars(s_i\mu_i))| \leq |Vars(s_i\mu_i)|$$

shows that $|Vars(s_i\mu_i)|$ weakly decreases with increasing i. Thus, at some point $|Vars(s_i\mu_i)|$ becomes constant, and hence for all future i, μ_{i+1} is a variable renaming. ◀

We now only need to find suitable invariants I to obtain Theorem 4.

Proof of Theorem 4: locally confluent overlay TRSs. Proving this second part of Theorem 4 is quite simple with the help of Lemma 9: we define I such that $I(s, \sigma)$ is satisfied if σ is a normal form substitution, i.e., $\sigma(x)$ is in normal form w.r.t. $\to_{\mathcal{R}}$ for all x. Hence, by

Lemma 9 we conclude that both $SN(\rightsquigarrow_{\mathcal{R}}, \{s\})$ and $SN(\rightarrow_{\mathcal{R}}, \rightsquigarrow^*_{\mathcal{R}}(\{s\}))$ imply $SN(\rightarrow_{\mathcal{R}}, \{s\sigma\})$ for all normal form substitutions σ, or replacing $\{s\}$ by $rhs(\mathcal{R})$ we arrive at:

$$SN(\rightsquigarrow_{\mathcal{R}}, rhs(\mathcal{R})) \vee SN(\rightarrow_{\mathcal{R}}, RFC(\mathcal{R})) \quad \text{implies} \quad SN(\rightarrow_{\mathcal{R}}, \sigma(rhs(\mathcal{R})))$$

The switch from $SN(\rightarrow_{\mathcal{R}}, \sigma(rhs(\mathcal{R})))$ for all normal form substitutions σ to full $SN(\rightarrow_{\mathcal{R}})$ is now performed using a result of Gramlich [9]: for locally confluent overlay TRSs, innermost termination and termination coincide, and thus the restriction to normal form substitution can be assumed without loss of generality. ◀

Proof of Theorem 4: right-linear TRSs. For this part of the proof we define I differently, namely $I(s, \sigma)$ is satisfied if s is linear and σ is a strongly normalizing substitution, i.e., $SN(\rightarrow_{\mathcal{R}}, \{\sigma(x) \mid x \in \mathcal{V}\})$.

Given some infinite derivation (1), we cannot immediately apply Lemma 9, but first have to do some preprocessing as follows.

For an arbitrary rewrite step $s\sigma \rightarrow_p t$, it is either the case that $p \in FPos(s)$ (and we use Lemma 8), or the rewrite step is completely inside σ, e.g., below variable x. If s is linear, then in the latter case the step $s\sigma \rightarrow_p t$ can be simulated by narrowing with zero steps as follows: $s \rightsquigarrow^0 s$ and $\sigma \rightarrow^+ \sigma'$ where σ' is defined as the substitution that is obtained by rewriting $x\sigma$ to the corresponding subterm of t. In this way one obtains $s\sigma \rightarrow t = s\sigma'$. So in both cases ($p \in FPos(s)$ is satisfied or not), one can find a term s' and substitution σ' such that $t = s'\sigma'$ and $I(s', \sigma')$ is satisfied. Moreover, $s \rightsquigarrow^* s'$. Hence, we can again obtain an infinite simulation of the rewrite sequence by narrowing steps as indicated in Figure 1, except that now some of the narrowing steps need to be replaced by equalities.

Note that we further obtain $\sigma_i(Vars(s_i))$ $(\rightarrow_{\mathcal{R}} \cup \rhd)^*_{mul}$ $\sigma_{i+1}(Vars(s_{i+1}))$ where here $Vars(s_i)$ is interpreted as the multiset of variables of a term, and $(\rightarrow_{\mathcal{R}} \cup \rhd)_{mul}$ is the multiset extension of relation $\rightarrow_{\mathcal{R}} \cup \rhd$. Since I enforces that the substitutions are strongly normalizing, there must be some point k such indeed a strict decrease is not possible anymore, i.e., for all $i \geq k$, the relation $\sigma_i(Vars(s_i))$ $(\rightarrow_{\mathcal{R}} \cup \rhd)^+_{mul}$ $\sigma_{i+1}(Vars(s_{i+1}))$ is not satisfied. Since such a strict decrease is always obtained if the rewrite step is completely inside the substitution, we know that for each $i \geq k$, indeed the position of the rewrite step must be in s_i. Hence, we can apply Lemma 9 on the infinite derivation $t_k = s_k\sigma_k \rightarrow_{\mathcal{R}} s_{k+1}\sigma_{k+1} \rightarrow_{\mathcal{R}} \dots$ and obtain $\neg SN(\rightsquigarrow_{\mathcal{R}}, \{s_k\})$ and $\neg SN(\rightarrow_{\mathcal{R}}, \rightsquigarrow^*_{\mathcal{R}}(\{s_k\}))$. In combination with $s_0 \rightsquigarrow^* s_k$ this nearly completes the proof as in the case for locally confluent overlay TRSs. There are two additional steps that still need to proven.

First, we argue that invariant I is initially satisfied. To this end, we again refer to the minimal non-terminating term argument: if $\mathcal{R}$ is not terminating, then there must be a minimal non-terminating term u w.r.t. the subterm relation. Any infinite derivation must make a root step at some point, so $u \rightarrow^*_{\mathcal{R}} \ell\sigma \rightarrow_{\mathcal{R}} r\sigma$ for some rule $\ell \rightarrow r \in \mathcal{R}$, and $r\sigma$ is non-terminating. By minimality of u it is easy to see that $I(r, \sigma)$ is satisfied, and we choose $s_0 = r$ and $\sigma_0 = \sigma$ as starting point of derivation (1).

The final issue is the preservation of the invariant, i.e., in particular we must show for every narrowing step $s_i \rightsquigarrow_{\mathcal{R}} s_{i+1}$ that linearity of s_i is carried over to linearity of s_{i+1}. This fact is stated in the upcoming Lemma 11. ◀

Before we show that narrowing preserves linearity, we first need the result that unification preservers linearity. Note that not all participating terms have to be linear, which is crucial to obtain linearity preservation of narrowing for TRSs that are right-linear, but not necessarily left-linear.

▶ **Lemma 10** (Unification Involving Linear Terms, [11]). *Let $\mathit{Vars}(s) \cap \mathit{Vars}(t) = \mathit{Vars}(u) \cap \mathit{Vars}(t) = \emptyset$, let t and u be linear. If s and t are unifiable with $mgu(s,t) = \sigma$ then $u\sigma$ is linear.*

Lemma 10 allows non-linearity of the left term s of the unification problem (s,t), whereas the right term t must be linear. This lemma is proven by generalizing (s,t) to multisets of such unification problems, and then following the computation of a swap-free unification algorithm. In such an algorithm there is no transition of the form that replaces (s,t) by (t,s), with the overhead that there must be two rules for treating substitutions, one for pairs (x,t) and one for (t,x). Being swap-free is important to maintain the invariant regarding linearity of the right-hand sides of a unification problem.

▶ **Lemma 11** (Linearity Preservation of Narrowing, [11]). *If $v \rightsquigarrow_{\mathcal{R},p} w$ and $\mathcal{R}$ is right-linear and v is linear, then also w is linear.*

Proof. We mainly need to choose suitable parameters in Lemma 10 in order to prove that w is linear: we instantiate s by ℓ, t by $v|_p$, u by $v[r]_p$, and σ by μ. Here $\ell \to r$ and μ are the applied rule and the unifier from the narrowing step, respectively. ◀

It remains as future work to verify computable approximations of $\mathit{RFC}(\mathcal{R})$ (one might start with string rewrite systems [13, Section 6], extend this to linear TRSs [7, Section 8] and even further to right-linear TRSs [10, Section 8]) and then combine these approximations with CeTA's checker for match-bounds in order to support RFC match-bounds in CeTA.

References

1 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theor. Comput. Sci.*, 236(1-2):133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

2 Franz Baader and Tobias Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

3 Nachum Dershowitz. Termination of linear rewriting systems (preliminary version). In Shimon Even and Oded Kariv, editors, *Automata, Languages and Programming, 8th Colloquium, Acre (Akko), Israel, July 13-17, 1981, Proceedings*, volume 115 of *Lecture Notes in Computer Science*, pages 448–458. Springer, 1981. doi:10.1007/3-540-10843-2_36.

4 Nachum Dershowitz and Charles Hoot. Natural termination. *Theor. Comput. Sci.*, 142(2):179–207, 1995. doi:10.1016/0304-3975(94)00275-4.

5 Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Match-bounded string rewriting systems. *Appl. Algebra Eng. Commun. Comput.*, 15(3-4):149–171, 2004. URL: https://doi.org/10.1007/s00200-004-0162-8, doi:10.1007/S00200-004-0162-8.

6 Alfons Geser, Dieter Hofbauer, Johannes Waldmann, and Hans Zantema. Finding finite automata that certify termination of string rewriting. In Michael Domaratzki, Alexander Okhotin, Kai Salomaa, and Sheng Yu, editors, *Implementation and Application of Automata, 9th International Conference, CIAA 2004, Kingston, Canada, July 22-24, 2004, Revised Selected Papers*, volume 3317 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 2004. doi:10.1007/978-3-540-30500-2_13.

7 Alfons Geser, Dieter Hofbauer, Johannes Waldmann, and Hans Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Inf. Comput.*, 205(4):512–534, 2007. URL: https://doi.org/10.1016/j.ic.2006.08.007, doi:10.1016/J.IC.2006.08.007.

8 Oliver Geupel. Overlap closures and termination of term rewriting systems. Technical Report MIP-8922, Universität Passau, Passau, 1989.

9 Bernhard Gramlich. Relating innermost, weak, uniform and modular termination of term rewriting systems. In Andrei Voronkov, editor, *Logic Programming and Automated Reasoning,International Conference LPAR'92, St. Petersburg, Russia, July 15-20, 1992, Proceedings*,

volume 624 of *Lecture Notes in Computer Science*, pages 285–296. Springer, 1992. URL: `https://doi.org/10.1007/BFb0013069`, `doi:10.1007/BFB0013069`.

10 Martin Korp and Aart Middeldorp. Match-bounds revisited. *Inf. Comput.*, 207(11):1259–1283, 2009. URL: `https://doi.org/10.1016/j.ic.2009.02.010`, `doi:10.1016/J.IC.2009.02.010`.

11 Theory `Linear_Narrowing`. IsaFoR/CeTA version 3.5. `http://cl2-informatik.uibk.ac.at/rewriting/mercurial.cgi/IsaFoR/file/v3.5/thys/Termination_and_Complexity/Linear_Narrowing.thy`.

12 Theory `Right_Forward_Closure`. IsaFoR/CeTA version 3.5. `http://cl2-informatik.uibk.ac.at/rewriting/mercurial.cgi/IsaFoR/file/v3.5/thys/Termination_and_Complexity/Right_Forward_Closure.thy`.

13 Hans Zantema. Termination of string rewriting proved automatically. *J. Autom. Reason.*, 34(2):105–139, 2005. URL: `https://doi.org/10.1007/s10817-005-6545-0`, `doi:10.1007/S10817-005-6545-0`.

Cetera: Certified Termination with Agda

Dieter Hofbauer
ASW Saarland

Johannes Waldmann
HTWK Leipzig

Abstract

We report on Cetera, a program and library for checking termination certificates for string rewriting. The library is certified in the dependently typed programming language Agda. For now, Cetera contains well-founded monotone monoids (instantiated to weights and standard matrices) and allows modular removal of rules (by relative termination). We are currently working on the RFC (right-hand sides of forward closures) theorem. Proofs in Agda are constructive, so they sometimes differ from standard presentation.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Rewrite systems

Keywords and phrases termination, string rewriting

1 Introduction

The termination method of sparse tiling [8] for string rewriting systems, implemented in Matchbox and MultumNonMulta, turned out to be quite powerful (for present benchmarks, at least). It uses the RFC (right-hand sides of forward closures) method, which was absent from libraries for verified termination [6, 4, 11]. The goal of our library Cetera (`https://git.imn.htwk-leipzig.de/waldmann/cetera/`) is to verify sparse tiling.

Matchbox is implemented in the pure functional programming language Haskell. Haskell's type system is moving towards dependent types (for example, it has generalized algebraic data types), and this suggests verification with Agda [5], a dependently typed programming language based on Martin-Löf type theory.

2 Termination and Non-Termination in Agda

Termination is formalized constructively by an accessibility predicate [10]. The following Agda type `SN R x` specifies that for relation R on a set a, the element $x \in a$ is terminating (strongly normalizing), if each R-successor y of x is terminating.

```
data SN {a : Set} (R : Rel a) (x : a) : Set  where
  sn : (forall (y : a) -> R x y -> SN R y) -> SN R x
```

A termination proof of R is an Agda expression of type `forall (x : a) -> SN R x`, that is, an Agda-computable function that builds (more precisely, bounds the depth of) the R-derivation tree of input x. Ultimately, this method reduces termination of R to termination of Agda programs in our library, checked by Agda's type checker [1, 2]. There are terminating R that have no such proof, since Agda is, by design, not Turing-complete.

The element x is non-terminating if there is a function $f : \mathbb{N} \to a$ that represents an infinite R-derivation $x = f(0) \to_R f(1) \to_R \dots$, formally,

```
data INF {a : Set} (R : Rel a) (x : a) : Set  where
  inf : (f : Nat -> a) -> (f zero == x)
      -> (forall (y : Nat) -> R (f y) (f (succ y))) -> INF R x
```

Again, since the function f is Agda-computable, we miss some forms of non-termination.

We can easily prove in Agda that `SN R x` and `INF R x` cannot hold at the same time.

3 The Present: Well-founded Monotone Monoids

We consider pairs $(>, \geq)$ where $>$ is terminating, $\geq$ is reflexive and transitive, and $> \circ \geq\ \subseteq\ >$ as well as $\geq \circ > \ \subseteq\ >$. Then $R \subseteq\ >$ and $S \subseteq\ \geq$ implies termination of R relative to S, allowing to remove rules (of R) in an incremental proof of termination (of $R \cup S$). We realize such pairs by orders on matrices [9] of shape $E_{1,d} = \{m \mid m \in \mathbb{N}^{d \times d}, m_{1,1} \geq 1, m_{d,d} \geq 1\}$ where $L > R$ if $(L - R) \in P_{1,n} = \{m \mid m \in \mathbb{N}^{d \times d}, m_{1,d} \geq 1\}$ and $L \geq R$ if $(L - R) \in \mathbb{N}^{d \times d}$.

In the Agda program, we represent matrices as functions `Fin d -> Fin d -> Nat`, where `Fin d` denotes $\{0, \ldots, d-1\}$. This allows for easier proofs than the concrete representation `Vec d (Vec d Nat)`, where `Vec d` denotes lists of length d. If we want to use Agda's built-in equality for functional matrices, then we need the extensionality axiom.

The extracted code for checking certificates does indeed multiply matrices. The functional matrix representation does not share computations for intermediate products, so a chain of matrix multiplications has exponential cost. This can be reduced to linear cost in the length of the chain by conversion from function to data, which allows sharing.

4 Implementation and Experiments

For now, Cetera uses Haskell syntax for systems and certificates. Identifiers are renamed to numbers. Printers and parsers are obtained from `deriving (Show, Read)`. A certificate for $\{abaabb \to aababba\}$ (Zantema_04/z049) is

```
Certificate
  { system = [Rule   { lhs = [0, 1, 0, 0, 1, 1], rhs = [0, 0, 1, 0, 1, 1, 0]}]
  , reason = MatrixInterpretation
    { minterpretation = MatrixInterpretation
      { dim = 4
      , int = [ (1, [[1, 0, 0, 0], [0, 0, 1, 0], [0, 1, 1, 1], [0, 0, 0, 1]])
              , (0, [[1, 1, 0, 0], [0, 1, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1]]) ]}
    , remove = [Rule   { lhs = [0, 1, 0, 0, 1, 1], rhs = [0, 0, 1, 0, 1, 1, 0]}]
    , sub = Certificate   { system = [], reason = Empty}}}
```

Using only weights and matrix interpretations, with a time-out of 30 seconds, Matchbox proves termination for 488 (of 1658) benchmarks in SRS_Standard. The largest certificate occurs for ICFP_2010/26871. It has 60k non-whitespace characters, and takes 0.07 seconds to verify. Total verification time over all certificates is < 5 seconds.

Larger certificates will appear with semantic labelling via full or sparse tiling [8].

5 The Future: Right-Hand Sides of Forward Closures

Termination of R follows from termination of R on the set of right-sides of forward closures $\mathsf{RFC}(R)$ [7]. We formalize this claim in Agda as

```
sn-RFC : {C : Set l} (R : Rel (List C) l)
  -> (forall (x : List C) -> RFC R x -> SN-from (Rewrites R) x) -> SN (Rewrites R)
```

The classical proof is non-constructive, as it derives a contradiction from the existence of an infinite derivation. For a constructive proof, we need to embed an arbitrary R-derivation into a well-founded order. The idea is to simulate the original derivation by a derivation on sequences of *blocks* where a block is a single letter, or the right-hand side of a forward closure. The key idea is that a step on a sequence of blocks is length-lexicographically (from the right) decreasing w.r.t. the order $(\to_R \cup \sqsupset_s)^+$ on blocks. Here

$\sqsubset_s$ is the strict suffix relation, and it comes into play since an extension (narrowing) step of a closure eats a prefix of the neighbouring block. We use a commutation property [3] $(T \circ S) \subseteq (S \circ T) \wedge \mathsf{SN}(T) \wedge \mathsf{SN}(S) \Rightarrow \mathsf{SN}(S \cup T)$ for $T = \sqsupset_s$, $S = \to_R$. Note that $\to_R^+$ is only used when starting from $\mathsf{RFC}(R)$, as in the premise of Theorem `sn-RFC`.

We are currently working on proving the theorem. To use it for certificate checking, Cetera is still missing a finite description of (an over-approximation of) $\mathsf{RFC}(R)$. This could be a matchbound automaton, or a shift automaton (of tiles) [8].

References

1 Andreas Abel. foetus - termination checker for simple functional programs. 1998. `doi:10.48550/ARXIV.2407.06924`.

2 Andreas Abel. Termination checking with types. *RAIRO Theor. Informatics Appl.*, 38(4):277–319, 2004. `doi:10.1051/ITA:2004015`.

3 Leo Bachmair and Nachum Dershowitz. Commutation, transformation, and termination. In *Proc. Conference on Automated Deduction, CADE*, volume 230 of *LNCS*, pages 5–20. Springer, 1986. `doi:10.1007/3-540-16780-3_76`.

4 Frédéric Blanqui and Adam Koprowski. CoLoR: a Coq library on well-founded rewrite relations and its application to the automated verification of termination certificates. *Math. Struct. Comput. Sci.*, 21(4):827–859, 2011. `doi:10.1017/S0960129511000120`.

5 Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda - A functional language with dependent types. In *Proc. Theorem Proving in Higher Order Logics, TPHOLs*, volume 5674 of *LNCS*, pages 73–78. Springer, 2009. `doi:10.1007/978-3-642-03359-9_6`.

6 Evelyne Contejean, Pierre Courtieu, Julien Forest, Olivier Pons, and Xavier Urbain. Certification of automated termination proofs. In *Proc. Frontiers of Combining Systems, FroCoS*, volume 4720 of *LNCS*, pages 148–162. Springer, 2007. `doi:10.1007/978-3-540-74621-8_10`.

7 Nachum Dershowitz. Termination of linear rewriting systems. In *Proc. International Colloquium on Automata, Languages and Programming, ICALP*, pages 448–458, 1981. `doi:10.1007/3-540-10843-2_36`.

8 Alfons Geser, Dieter Hofbauer, and Johannes Waldmann. Sparse tiling through overlap closures for termination of string rewriting. In *Proc. Formal Structures for Computation and Deduction, FSCD*, volume 131 of *LIPIcs*, pages 21:1–21:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.FSCD.2019.21`.

9 Dieter Hofbauer and Johannes Waldmann. Termination of string rewriting with matrix interpretations. In *Proc. Term Rewriting and Applications, RTA*, volume 4098 of *LNCS*, pages 328–342. Springer, 2006. `doi:10.1007/11805618_25`.

10 Lawrence C. Paulson. Constructing recursion operators in intuitionistic type theory. *J. Symb. Comput.*, 2(4):325–355, 1986. `doi:10.1016/S0747-7171(86)80002-5`.

11 René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In *Proc. Theorem Proving in Higher Order Logics, TPHOLs*, volume 5674 of *LNCS*, pages 452–468. Springer, 2009. `doi:10.1007/978-3-642-03359-9_31`.

MU-TERM GTRS: A Tool for Proving Termination of Generalized Term Rewriting Systems

Raúl Gutiérrez
DSIC & VRAIN, Universitat Politècnica de València, Spain

Salvador Lucas
DSIC & VRAIN, Universitat Politècnica de València, Spain

Abstract

We present MU-TERM GTRS, a tool for proving termination of Generalized Term Rewriting Systems (GTRSs), where rewriting computations are defined by rules that may include not only conditions involving reachability, joinability, and conversion, but also atoms defined by Horn clauses—particularly conditions expressing relations among terms (e.g., sort information, subterm relations, etc.). Termination control is enhanced through the use of a *replacement map*, which explicitly specifies the argument positions of function symbols where rewriting is allowed. GTRSs offer a powerful framework for modeling computations in advanced reduction-based languages such as Maude.

2012 ACM Subject Classification Theory of computation Equational logic and rewriting, Logic and verification, Automated reasoning

Keywords and phrases Generalized Term Rewriting Systems, Termination

Acknowledgements Supported by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe" (PID2021-122830OB-C42 and PID2021-122830OB-C44) and by the grant CIPROM/2022/6 funded by Generalitat Valenciana

1 Introduction

A *Generalized Term Rewriting System (GTRS [10])* is a tuple $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$, where $\mathcal{F}$ is a signature of *function symbols*; Π is a signature of *predicate symbols*, including $\to$ and $\to^*$; μ is a *replacement map* i.e., for all k-ary function symbols $f \in \mathcal{F}$, $\mu(f) \subseteq \{1, \ldots, k\}$ is the set of *active* arguments on which rewriting steps are allowed [9]; H is a set of definite Horn clauses $A \Leftarrow c$, where the predicate symbol of A is not $\to$ or $\to^*$; and R is a set of rewrite rules $\ell \to r \Leftarrow c$ [10, Definition 51]. In both cases, c is a sequence of atoms. Programs of reduction-based systems like Maude [2] can often be encoded as GTRSs [10, 11, Section 8].

▶ **Example 1.** The Maude program `OvConsOS` in Figure 1 (left) exemplifies the use of functions on both finite lists (predicate `NatList`) and infinite lists (predicate `NatIList`) of natural numbers 0, s(0),..., built using the list constructor cons, which is made "lazy" for the evaluation of the second argument by means of a `strat` annotation. Symbol zeros represents an infinite list of 0. Although take can be used to obtain a finite prefix of an infinite list, length should be applied to *finite* lists only. Thus, explicit *sorts* are given to inputs and output of functions (e.g., `length : NatList -> Nat`). The GTRS $\mathcal{R}$ in Figure 1 (right), given in the extended COPS format for GTRSs of [6], permits a proof of termination of `OvConsOS`, which depends on the appropriate sorting of function symbols (`length` can only be applied to finite lists) and the use of $\mu(\texttt{cons}) = \{1\}$.

Termination of GTRSs $\mathcal{R}$ in the usual sense, i.e., as the absence of infinite rewrite sequences $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$, has been studied in [11] using appropriate notions of *dependency pairs*. This paper presents MU-TERM GTRS[1], a tool for automatically (dis)proving termination of

[1] `http://zenon.dsic.upv.es/muterm-gtrs/`

```
fmod OvConsOS is
 sorts Nat NatList NatIList .
 subsort NatList < NatIList .
 op 0 : -> Nat .
 op s : Nat -> Nat .
 op zeros : -> NatIList .
 op nil : -> NatList .
 op cons : Nat NatIList -> NatIList [strat (1 0)] .
 op cons : Nat NatList -> NatList [strat (1 0)] .
 op take : Nat NatIList -> NatList .
 op length : NatList -> Nat .
 vars M N : Nat .
 var IL : NatIList .
 var L : NatList .
 eq zeros = cons(0,zeros) .
 eq take(0,IL) = nil .
 eq take(s(M),cons(N,IL)) = cons(N,take(M,IL)) .
 eq length(nil) = 0 .
 eq length(cons(N,L)) = s(length(L)) .
endfm
```

```
(VAR M N L IL)
(REPLACEMENT-MAP
 (cons 1)
)
(HORN-CLAUSES
 NatIList(L) | NatList(L)
 Nat(0)
 Nat(s(N)) | Nat(N)
 NatList(nil)
 NatList(cons(N,L)) | Nat(N), NatList(L)
 NatIList(zeros)
 NatIList(cons(N,IL)) | Nat(N), NatIList(IL)
 NatList(take(N,IL)) | Nat(N), NatIList(IL)
 Nat(length(L)) | NatList(L)
)
(RULES
 zeros -> cons(0,zeros)
 take(0,IL) -> nil | NatIList(IL)
 take(s(M),cons(N,IL)) -> cons(N,take(M,IL))
   | Nat(M), Nat(N), NatIList(IL)
 length(nil) -> 0
 length(cons(N,L)) -> s(length(L))
   | Nat(N), NatList(L)
)
```

Figure 1 The Maude module OvConsOS in [3, Figure 1] and MU-TERM GTRS module

Table 1 Generic sentences of the first-order theory of rewriting

Label	Sentence
(Rf)	$(\forall x)\ x \to^* x$
(Co)	$(\forall x, y, z)\ x \to y \wedge y \to^* z \Rightarrow x \to^* z$
$(\mathrm{Pr})_{f,i}$	$(\forall x_1, \ldots, x_k, y_i)\ x_i \to y_i \Rightarrow f(x_1, \ldots, x_i, \ldots, x_k) \to f(x_1, \ldots, y_i, \ldots, x_k)$
$(\mathrm{HC})_{A \Leftarrow A_1,\ldots,A_n}$	$(\forall x_1, \ldots, x_p)\ A_1 \wedge \cdots \wedge A_n \Rightarrow A$
	where $x_1, \ldots, x_p$ are the variables occurring in $A_1, \ldots, A_n$ and A

GTRSs based on the Dependency Pair Framework introduced in [11].

2 Termination of GTRSs using Dependency Pairs

A GTRS $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ determines a first-order theory $\overline{\mathcal{R}} = \{(\mathrm{Rf}), (\mathrm{Co})\} \cup \{(\mathrm{Pr})_{f,i} \mid f \in \mathcal{F}, i \in \mu(f)\} \cup \{(\mathrm{HC})_\alpha \mid \alpha \in H \cup R\}$ (see Table 1). Note that rules in R are Horn clauses which are often given a label α.

▶ **Example 2.** The GTRS in Figure 1 (right) is $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ in Figure 2, where $\mathcal{F} = \{\mathsf{0}, \mathsf{s}, \mathsf{nil}, \mathsf{cons}, \mathsf{zeros}, \mathsf{take}, \mathsf{length}\}$, $\Pi = \{\to, \to^*, \mathsf{Nat}, \mathsf{NatList}, \mathsf{NatIList}\}$, $\mu(\mathsf{cons}) = \{1\}$ and $\mu(f) = \{1, \ldots, k\}$ for all k-ary $f \in \mathcal{F} - \{\mathsf{cons}\}$, $H = \{(1), \ldots, (9)\}$, and $R = \{(10), \ldots, (14)\}$.

For all terms s and t, we write $s \to_{\mathcal{R}} t$ (resp. $s \to^*_{\mathcal{R}} t$) iff $\overline{\mathcal{R}} \vdash s \to t$ (resp. $\overline{\mathcal{R}} \vdash s \to^* t$); $\mathcal{R}$ is terminating iff there is no infinite sequence $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$.

Symbols $f \in \mathcal{F}$ are called *defined* if $root(\ell) = f$ for some $\ell \to r \Leftarrow c \in R$. A subterm t of s is *active* (regarding μ, written $s \trianglerighteq_\mu t$) if $s = t$ or $s = f(s_1, \ldots, s_i, \ldots, s_k)$ and $s_i \trianglerighteq_\mu t$ for some $i \in \mu(f)$. We say that t is *frozen* in s (written $s \rhd_{\not\mu} t$) if $s \trianglerighteq t$ and $s \ntrianglerighteq_\mu t$. As for the DP approach for TRSs [1, 8], 'classical' dependency pairs for GTRSs $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ are:

$$H_{DP}(\mathcal{R}) = \{\ell^\sharp \xrightarrow{\mathsf{t}} v^\sharp \Leftarrow c \mid \ell \to r \Leftarrow c \in R, r \trianglerighteq_\mu v,\ \ root(v) \text{ is defined, and } \ell \ntrianglerighteq_\mu v\}$$

with $\xrightarrow{\mathsf{t}}$ a new predicate symbol. For $t = f(t_1, \ldots, t_k)$, let $t^\sharp = f^\sharp(t_1, \ldots, t_k)$ for a *new* symbol $f^\sharp$, often *capitalized* as F. For $\mathcal{R}$ in Ex. 2, $H_{DP}(\mathcal{R}) = \{(15)\}$ with

$$\mathsf{LENGTH}(\mathsf{cons}(N, L)) \xrightarrow{\mathsf{t}} \mathsf{LENGTH}(L) \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L) \quad (15)$$

$$\begin{array}{llcll}
\mathsf{NatIList}(L) \Leftarrow \mathsf{NatList}(L) & (1) & & \mathsf{NatIList}(\mathsf{cons}(N,IL)) \Leftarrow \mathsf{Nat}(N), \mathsf{NatIList}(IL) & (6)\\
\mathsf{Nat}(0) & (2) & & \mathsf{NatList}(\mathsf{cons}(N,L)) \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L) & (7)\\
\mathsf{Nat}(\mathsf{s}(N)) \Leftarrow \mathsf{Nat}(N) & (3) & & \mathsf{NatList}(\mathsf{take}(N,IL)) \Leftarrow \mathsf{Nat}(N), \mathsf{NatIList}(IL) & (8)\\
\mathsf{NatList}(\mathsf{nil}) & (4) & & \mathsf{Nat}(\mathsf{length}(L)) \Leftarrow \mathsf{NatList}(L) & (9)\\
\mathsf{NatIList}(\mathsf{zeros}) & (5) & & &
\end{array}$$

$$\begin{array}{lr}
\mathsf{zeros} \to \mathsf{cons}(0, \mathsf{zeros}) & (10)\\
\mathsf{take}(0, IL) \to \mathsf{nil} \Leftarrow \mathsf{NatIList}(IL) & (11)\\
\mathsf{take}(\mathsf{s}(M), \mathsf{cons}(N, IL)) \to \mathsf{cons}(N, \mathsf{take}(M, IL)) \Leftarrow \mathsf{Nat}(M), \mathsf{Nat}(N), \mathsf{NatIList}(IL) & (12)\\
\mathsf{length}(\mathsf{nil}) \to 0 & (13)\\
\mathsf{length}(\mathsf{cons}(N, L)) \to \mathsf{s}(\mathsf{length}(L)) \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L) & (14)
\end{array}$$

Figure 2 GTRS for the Maude module OvConsOS in [3, Figure 1]

Dependency pairs for TRSs are useful to encode (possibly) infinite rewrite sequences (see [8]). However, in order to properly capture termination of GTRSs we also need *collapsing* dependency pairs [11, Sect. 5] to handle computations involving conditions and context-sensitive replacement restrictions. By expressing special features of conditional and context-sensitive dependency pairs as Horn clauses, we obtain a new valid GTRS as a result. While [11, Def. 16] defines them for GTRSs, we refine this using the richer framework from [4], based on hidden terms and hiding contexts.

A term t such that $root(t)$ is a *defined symbol* is *hidden* if there is a rule $\alpha : \ell \to r \Leftarrow c$ such that t is a frozen subterm of r: $r \rhd_{\not\mu} t$; $\mathcal{HT}(\mathcal{R})$ is the set of hidden terms in $\mathcal{R}$.

▶ **Example 3.** For $\mathcal{R}$ in Example 2, $\mathcal{HT}(\mathcal{R}) = \{\mathsf{zeros}, \mathsf{take}(M, IL)\}$ (in red on the right-hand sides of (10) and (12)).

Instances of hidden terms are the only ones which may *become active* when matched by a left-hand side. More precisely, a function symbol f *hides the active argument position* $i \in \mu(f)$ *in the right hand side* r *of* α if $r \rhd_{\not\mu} f(r_1, \ldots, r_k)$ for some terms $r_1, \ldots, r_k$, and r_i contains (a) an *active defined symbol* or (b) an active variable which is (b.1) frozen both in ℓ and r, and (b.2) not active in ℓ or r.

When dealing with *conditional* rules $\ell \to r \Leftarrow c$, collapsing dependency pairs are also necessary to capture the continuation of infinite rewrite sequences on subterms s of instances $\sigma(x)$ of variables x occurring in r but *not occurring* in ℓ and possibly occurring in the conditional part c, see [15, Sect. 4.3]. Let $\varpi_{\trianglerighteq_{\mathsf{unh}}}$, $\varpi_{\trianglerighteq_\mu}$ and Mk be new (binary) predicate symbols (denoting the *hiding* and *active subterm* relation on terms, and the *marking* of terms), respectively defined by sets of clauses $Unh(\mathcal{F}) = \{x \varpi_{\trianglerighteq_{\mathsf{unh}}} x\} \cup \{f(x_1, \ldots, x_i, \ldots, x_k) \varpi_{\trianglerighteq_{\mathsf{unh}}} x'_i \Leftarrow x_i \varpi_{\trianglerighteq_\mu} x'_i \mid f \in \mathcal{F}, f \text{ hides } i\}$, $\mathcal{S}ubt(\mathcal{F}, \mu) = \{x \varpi_{\trianglerighteq_\mu} x\} \cup \{f(x_1, \ldots, x_i, \ldots, x_k) \varpi_{\trianglerighteq_\mu} x'_i \Leftarrow x_i \varpi_{\trianglerighteq_\mu} x'_i \mid f \in \mathcal{F}, k = ar(f), i \in \mu(f)\}$. and $\mathcal{M}ark(\mathcal{F}) = \{\mathsf{Mk}(f(x_1, \ldots, x_k), f^\sharp(x_1, \ldots, x_k)) \mid f \in \mathcal{F}, k = ar(f)\}$, respectively. Then, we let

$$\begin{array}{rcl}
H_{DPC}(\mathcal{R}) & = & \{\ell^\sharp \xrightarrow{\mathsf{t}} t^\sharp \Leftarrow c, x \;\varpi_{\trianglerighteq_{\mathsf{unh}}}\; t \mid \ell \to r \Leftarrow c \in R, t \in \mathcal{HT}(\mathcal{R}), \text{ and}\\
& & \quad x \in \mathcal{V}ar(\ell) \cap [\mathcal{V}ar^\mu(r) - \mathcal{V}ar^\mu(\ell)]\}\\
& \cup & \{\ell^\sharp \xrightarrow{\mathsf{t}} x'' \Leftarrow c, x \;\varpi_{\trianglerighteq_\mu}\; x', \mathsf{Mk}(x', x'') \mid \ell \to r \Leftarrow c \in R, \text{ and}\\
& & \quad x \in \mathcal{V}ar^\mu(r) - \mathcal{V}ar(\ell) \text{ and } x' \text{ and } x'' \text{ are fresh variables}\} \qquad (16)
\end{array}$$

where, given a term t, $\mathcal{V}ar^\mu(t)$ is the set of variables which occur active in t.

▶ Remark 4. In contrast to (16), in [11, Def. 16] $H_{DPC}(\mathcal{R})$ only considers the second group of *collapsing* pairs *with* $x \in \mathcal{V}ar^{\mu}(r) - \mathcal{V}ar^{\mu}(\ell)$. Then, for $\mathcal{R}$ in Example 2, $H_{DPC}(\mathcal{R})$ would consist of a single (collapsing) pair

$$\mathsf{LENGTH}(\mathsf{cons}(N, L)) \overset{\mathsf{t}}{\to} L'' \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L), L \mathrel{\varpi_{\trianglerighteq_{\mu}}} L', \mathsf{Mk}(L', L'') \tag{17}$$

▶ **Example 5.** For $\mathcal{R}$ in Example 2, using (16), $H_{DPC}(\mathcal{R}) = \{(18), (19)\}$ with

$$\mathsf{LENGTH}(\mathsf{cons}(N, L)) \overset{\mathsf{t}}{\to} \mathsf{ZEROS} \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L), L \mathrel{\varpi_{\trianglerighteq_{\mathsf{unh}}}} \mathsf{zeros} \tag{18}$$

$$\mathsf{LENGTH}(\mathsf{cons}(N, L)) \overset{\mathsf{t}}{\to} \mathsf{TAKE}(M, IL) \Leftarrow \mathsf{Nat}(N), \mathsf{NatList}(L), L \mathrel{\varpi_{\trianglerighteq_{\mathsf{unh}}}} \mathsf{take}(M, IL) \tag{19}$$

▶ **Definition 6.** *Let* $\mathcal{R} = (\mathcal{F}, \Pi, \mu, H, R)$ *be a GTRS whose set of defined symbols is* $\mathcal{D}$. *The GTRS* $\mathsf{DP}_{HC}(\mathcal{R}) = (\mathcal{F}_{HC}, \Pi_{HC}, \mu^{\sharp}, H_{HC}, R)$*, where:* $\mathcal{F}_{HC} = \mathcal{F} \cup \mathcal{D}^{\sharp}$*;* $\Pi_{HC} = \Pi \cup \{\varpi_{\trianglerighteq_{\mathsf{unh}}}, \varpi_{\trianglerighteq_{\mu}}, \mathsf{Mk}\}$*; for all* $f \in \mathcal{F}_{HC}$*,* $\mu^{\sharp}(f) = \mu(f)$ *if* $f \in \mathcal{F}$ *and* $\mu(g)$ *if* $f = g^{\sharp}$ *for some* $g \in \mathcal{D}$*; and* $H_{HC} = H \cup Unh(\mathcal{F}) \cup \mathcal{S}ubt(\mathcal{F} \cup \mathcal{D}^{\sharp}, \mu^{\sharp}) \cup \mathcal{M}ark(\mathcal{D}) \cup H_{DP}(\mathcal{R}) \cup H_{DPC}(\mathcal{R})$.

▶ **Theorem 7.** *A GTRS* $\mathcal{R}$ *is* terminating *if there is no infinite minimal* $\mathsf{DP}_{HC}(\mathcal{R})$*-chain*[2]. *If there is an infinite* $\mathsf{DP}_{HC}(\mathcal{R})$*-chain, then* $\mathcal{R}$ *is not terminating.*

In the following, given a GTRS $\mathcal{R}$, we write $P_{\mathcal{R}}$ to denote the subset of Horn clauses in $\mathcal{R}$ of the form $u \overset{\mathsf{t}}{\to} v \Leftarrow c$. Note that $P_{\mathsf{DP}_{HC}(\mathcal{R})} = H_{DP}(\mathcal{R}) \cup H_{DPC}(\mathcal{R})$.

3 Implementation of the DP Framework for GTRSs.

MU-TERM GTRS is implemented in 27 Haskell modules including around 4000 *loc*. The tool has been developed independently from MU-TERM [5] only using semantic processors, so they are independent tools. A specific parser has been created to accept the TPDB notation[3] enriched with a new block `HORN-CLAUSES` to specify them (see Figure 1)[4]. To implement the framework, the followign data type is used:

```haskell
data GTRS c
   = GTRS { gName :: Map Int String, gArity :: Map Int Int, gMu :: Map Int [Int]
          , gSymbols :: Set Int, gVariables :: Set Int , gPredicates :: Set Int
          , gEquations :: Set (CEquation c), gHornClauses :: Set (HornClause c)
          , gRules :: Set (CRule c), gLabel :: String} deriving (Eq,Show)
```

We use the `muterm-framework` library to define instances of problems, processors, and also strategy combinators to easily define our strategy (see below).

```haskell
data Problem typ p = Problem typ p
-- | GRewriting problem
data GRewriting = GRewriting PScheme deriving (Eq, Ord, Show)
-- | Problems contains a rewrite system
class IsProblem typ problem | typ -> problem where
    getProblemType :: Problem typ problem -> typ
    getR :: Problem typ problem -> problem
-- | GRewriting problem is a Problem
instance IsProblem GRewriting where
  data Problem GRewriting a = GRew PScheme a deriving (Eq, Ord, Show)
  getProblemType (GRew m _) = GRewriting m t
  getR (GRew _ r) = r
```

[2] We use the natural extension of chain, see [11, Definition 14]
[3] `https://www.lri.fr/~marche/tpdb/format.html`
[4] As part of future work, we aim to extend the ARI format as well.

where `PScheme` is a pair that records whether the current problem can be proved terminating or non-terminating. Initially, `PScheme` has the value (m, a); this means that to test termination, we start with minimal sequences, but to test non-termination, we start with arbitrary sequences. A processor modifies the pair values depending on whether the processor is sound or complete. These flags can only have the following values: m for minimal, a for arbitrary, or $\bullet$ to indicate that the processor returns an incompatible type. The computation of the initial GTRS problem is treated as a special *dependency pair processor*:

```haskell
data DPProcessor = DPProcessor
-- | The information obtained is the new GTRS
data DPProcInfo problem
    = DPProcInfo { outProblem :: problem }
-- DPProcessor
instance (... functional dependencies ... )
  => Processor info DPProcessor (Problem GRewriting trs)
                                (Problem GRewriting trs) where
    apply DPProcessor inP = singleP (DPProcInfo outP) inP outP where ...
```

Each processor has its own name. The strategy combines the different processors in order to find a proof tree. We use the following strategy:

```haskell
grewStrat
 = idProc .&. (mace4irProc .|. return) .&. dpProc .&. (infProc .|. simpProc .|. return)
    .&. sccProc .&. fixSolver ((subProc .|. mace4rpProc .|. agesrpProc) .&. sccProc)
```

where `mace4irProc` removes infeasible rules of the input system, and the other names refer to the processors described in [7, Section 5.1] and in [11, Section 7]. `simpProc` applies simplification processors, `infProc` applies non-termination techniques and we use Mace4 and AGES for the semantic RP processor. Finally, `.&.` and `.|.` are the 'and' and 'or' strategy combinators, and `fixSolver` is a fix-point application strategy.

4 Conclusions and Future Work

We have presented MU-TERM GTRS, a new tool for proving termination of GTRSs. The tool implements the Dependency Pair Framework for proving termination of GTRSs [11], although we have introduced some relevant improvements in the treatment of collapsing dependency pairs induced by the replacement restrictions (see (16) and Remark 4), advantageously adapting the developments in [4]. All processors described in [11] have been implemented. Besides, some processors developed for the Confluence Framework for proving confluence of GTRSs [7] have been adapted and implemented as well. Since GTRSs can be used to model Maude programs (see Example 1), the tool can be used to prove termination of Maude programs. The stronger property of *operational termination* [13], which implies termination (but not vice versa) has been defined for GTRSs in [12]. Operational termination of GTRSs is also a subject for future work as it is important for a practical use of GTRSs, as discussed in [14] in the context of rewrite theories.

References

1 Thomas Arts and Jürgen Giesl. Termination of term rewriting using dependency pairs. *Theor. Comput. Sci.*, 236(1-2):133–178, 2000. doi:10.1016/S0304-3975(99)00207-8.

2 Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn L. Talcott. *All About Maude - A High-Performance Logical Framework,*

How to Specify, Program and Verify Systems in Rewriting Logic, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007. doi:10.1007/978-3-540-71999-1.

3 Francisco Durán, Salvador Lucas, José Meseguer, Claude Marché, and Xavier Urbain. Proving termination of membership equational programs. In Nevin Heintze and Peter Sestoft, editors, *Proceedings of the 2004 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, 2004*, pages 147–158. ACM, 2004. doi:10.1145/1014007.1014022.

4 Raúl Gutiérrez and Salvador Lucas. Proving termination in the context-sensitive dependency pair framework. In Peter Csaba Ölveczky, editor, *Rewriting Logic and Its Applications - 8th International Workshop, WRLA 2010, Held as a Satellite Event of ETAPS 2010, Revised Selected Papers*, volume 6381 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2010. doi:10.1007/978-3-642-16310-4_3.

5 Raúl Gutiérrez and Salvador Lucas. MU-TERM: Verify Termination Properties Automatically (System Description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 436–447. Springer, 2020. doi:10.1007/978-3-030-51054-1_28.

6 Raúl Gutiérrez and Salvador Lucas. Proving and disproving feasibility with infChecker. In Raúl Gutérrez and Naoki Nishida, editors, *14th International Workshop on Confluence, IWC 2025*, page to appear, 2025.

7 Raúl Gutiérrez, Salvador Lucas, and Miguel Vítores. Proving Confluence in the Confluence Framework with CONFident. *Fundamenta Informaticae*, 192(2):167–217, 2024. doi:10.3233/FI-242192.

8 Nao Hirokawa and Aart Middeldorp. Dependency pairs revisited. In Vincent van Oostrom, editor, *Rewriting Techniques and Applications, 15th International Conference, RTA 2004, Aachen, Germany, June 3-5, 2004, Proceedings*, volume 3091 of *Lecture Notes in Computer Science*, pages 249–268. Springer, 2004. doi:10.1007/978-3-540-25979-4_18.

9 Salvador Lucas. Context-sensitive Rewriting. *ACM Comput. Surv.*, 53(4):78:1–78:36, 2020. doi:10.1145/3397677.

10 Salvador Lucas. Local confluence of conditional and generalized term rewriting systems. *Journal of Logical and Algebraic Methods in Programming*, 136:paper 100926, pages 1–23, 2024. doi:10.1016/j.jlamp.2023.100926.

11 Salvador Lucas. Termination of Generalized Term Rewriting Systems. In Jakob Rehof, editor, *9th International Conference on Formal Structures for Computation and Deduction (FSCD 2024)*, volume 299 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 32:1–32:18, Dagstuhl, Germany, 2024. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSCD.2024.32.

12 Salvador Lucas. Semantic Properties of Computations Defined by Elementary Inference Systems. In Emmanuel D'Angelis and Florian Frohn, editors, *12th International Workshop on Horn Clauses in Verification and Synthesis, HCVS 2025*, 2025. to appear.

13 Salvador Lucas, Claude Marché, and José Meseguer. Operational termination of conditional term rewriting systems. *Inf. Process. Lett.*, 95(4):446–453, 2005. doi:10.1016/j.ipl.2005.05.002.

14 Salvador Lucas and José Meseguer. Normal forms and normal theories in conditional rewriting. *J. Log. Algebr. Meth. Program.*, 85(1):67–97, 2016. doi:10.1016/j.jlamp.2015.06.001.

15 Salvador Lucas and José Meseguer. Dependency pairs for proving termination properties of conditional term rewriting systems. *J. Log. Algebr. Meth. Program.*, 86(1):236–268, 2017. doi:10.1016/j.jlamp.2016.03.003.

AProVE: Becoming Open Source and Recent Improvements

Florian Frohn
RWTH Aachen University, Germany

Carsten Fuhs
Birkbeck, University of London, UK

Jürgen Giesl
RWTH Aachen University, Germany

Jan-Christoph Kassing
RWTH Aachen University, Germany

Nils Lommen
RWTH Aachen University, Germany

Abstract

AProVE (Automated Program Verification Environment) is a tool for fully automatic program verification. More precisely, AProVE is able to analyze termination, complexity, and safety of different programming languages, e.g., Java, C, Haskell, and Prolog. To ensure the correctness of its analysis, AProVE can generate certificates that can be checked by external certification tools. For further details on AProVE's general approach, see [1].

Since the last workshop on termination, several improvements were developed within AProVE. First of all, we want to announce that AProVE will be released as an open source tool by the end of this year. Opening the tool to the community will allow researchers and developers to explore its techniques and experiment with new strategies (e.g., for termination proofs). We hope this will foster the development of novel methods and lead to improved strategies for all supported techniques. In celebration of this milestone, we present a new logo, see Fig. 1.

Furthermore, AProVE will participate in this year's confluence competition (CoCo) [9] for the first time. Several of AProVE's termination techniques rely on confluence properties, so participating in CoCo provides an opportunity to evaluate how well AProVE's techniques to prove confluence compare with specialized tools in that area.

Finally, AProVE will keep on participating in the annual termination competition [2]. Compared to previous years, we have integrated several improvements in multiple categories. In particular, we improved the termination and complexity analysis of probabilistic term rewriting [3, 4, 5], the termination analysis of relative term rewriting [6], the derivational complexity analysis of standard term rewriting [7], and more. Moreover, the analysis of C-programs has been improved by integrating new versions of the tools KoAT and LoAT [8].

Figure 1 New AProVE Logo

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Termination, Complexity, Safety, Automation

References

1 Jürgen Giesl, Cornelius Aschermann, Marc Brockschmidt, Fabian Emmes, Florian Frohn, Carsten Fuhs, Jera Hensel, Carsten Otto, Martin Plücker, Peter Schneider-Kamp, Thomas Ströder, Stephanie Swiderski, and René Thiemann. Analyzing program termination and complexity automatically with AProVE. *J. Autom. Reason.*, 58(1):3–31, 2017. `doi:10.1007/s10817-016-9388-y`.

2 Jürgen Giesl, Albert Rubio, Christian Sternagel, Johannes Waldmann, and Akihisa Yamada. The termination and complexity competition. In *Proc. TACAS '19*, LNCS 11429, pages 156–166, 2019. Website of *TermComp*: `https://termination-portal.org/wiki/Termination_Competition`. `doi:10.1007/978-3-030-17502-3_10`.

3 Jan-Christoph Kassing, Stefan Dollase, and Jürgen Giesl. A complete dependency pair framework for almost-sure innermost termination of probabilistic term rewriting. In *Proc. FLOPS '24*, LNCS 14659, pages 62–80, 2024. `doi:10.1007/978-981-97-2300-3_4`.

4 Jan-Christoph Kassing, Florian Frohn, and Jürgen Giesl. From innermost to full almost-sure termination of probabilistic term rewriting. In *Proc. FoSSaCS '24*, LNCS 14575, pages 206–228, 2024. `doi:10.1007/978-3-031-57231-9_10`.

5 Jan-Christoph Kassing and Jürgen Giesl. Annotated dependency pairs for full almost-sure termination of probabilistic term rewriting. In *Principles of Verification: Cycling the Probabilistic Landscape*, LNCS 15260, pages 339–366, 2024. `doi:10.1007/978-3-031-75783-9_14`.

6 Jan-Christoph Kassing, Grigory Vartanyan, and Jürgen Giesl. A dependency pair framework for relative termination of term rewriting. In *Proc. IJCAR '24*, LNCS 14740, pages 360–380, 2024. `doi:10.1007/978-3-031-63501-4_19`.

7 Jan-Christoph Kassing and Jürgen Giesl. From innermost to full probabilistic term rewriting: Almost-sure termination, complexity, and modularity, 2025. URL: `https://arxiv.org/abs/2409.17714`, `arXiv:2409.17714`.

8 Nils Lommen and Jürgen Giesl. AProVE (KoAT+ LoAT). In *Proc. TACAS '25*, LNCS 15698, pages 205–211, 2025. `doi:10.1007/978-3-031-90660-2_13`.

9 Aart Middeldorp, Julian Nagele, and Kiraku Shintani. Confluence competition 2019. In *Proc. TACAS '19*, LNCS 11429, pages 25–40, 2019. `doi:10.1007/978-3-030-17502-3_2`.

KoAT: An Automatic Complexity Analysis Tool for (Probabilistic) Integer Programs

Nils Lommen
RWTH Aachen University, Germany

Éléanore Meyer
RWTH Aachen University, Germany

Jürgen Giesl
RWTH Aachen University, Germany

Abstract

KoAT is an automated tool for inferring runtime and size bounds – and proving termination – of (probabilistic) integer programs by analyzing program fragments in a modular way and combining their bounds. It uses multiphase linear ranking functions to capture execution "phases" of loops and a technique for triangular weakly non-linear loops (*twn*-loops), enabling the analysis of programs with non-linear arithmetic.

2012 ACM Subject Classification Software and its engineering → Software verification and validation; Theory of computation → Automated reasoning; Theory of computation → Logic and verification

Keywords and phrases Complexity Analysis, Termination Analysis

KoAT is a tool to automatically infer complexity bounds for (probabilistic) integer programs, based on an alternating modular inference of upper runtime and size bounds for program parts [2]. By inferring runtime bounds for individual subprograms repeatedly, in the end we obtain a bound on the runtime complexity of the whole program. Furthermore, KoAT can be used to automatically prove termination of programs.

To infer runtime bounds for subprograms, we use multiphase linear ranking functions (MΦRFs) [1,3]. In contrast to classical ranking functions, MΦRFs can also represent bounds on multiple "phases" of program executions. Moreover, we embedded a complete technique for inferring runtime bounds of triangular weakly non-linear loops (*twn*-loops) in our complexity analysis tool [4,7]. The update of a *twn*-loop is triangular, i.e., we can order the program variables such that the update of any x_i does not depend on the variables $x_1, \dots, x_{i-1}$ with smaller indices. So the restriction to triangular updates prohibits "cyclic dependencies" of variables. Due to this, one can compute closed forms for the repeated updates of *twn*-loops, which makes them especially suitable for automatic program analysis. In particular, this approach also allows us to analyze the complexity of programs with non-linear arithmetic. In addition, we developed a novel procedure to infer size bounds via closed forms as well, which is also suitable for programs with non-linear arithmetic [5,7]. Furthermore, recently we extended our framework such that (possibly recursive) function calls can be represented naturally and analyzed by our tool KoAT. KoAT is also used in the framework AProVE (KoAT + LoAT) [8] to automatically prove termination of C programs.

We adapted KoAT's framework for automated complexity analysis to *probabilistic* integer programs in [9]. To improve the power of automatic complexity analysis further, we integrated control-flow refinement via partial evaluation into KoAT's approach [3] and recently adapted this refinement for probabilistic programs as well [6].
KoAT's source code, a web interface, a binary, and a Docker image are available at

https://koat.verify.rwth-aachen.de.

References

1 Amir M. Ben-Amram and Samir Genaim. On Multiphase-Linear Ranking Functions. In *Proc. CAV '17*, LNCS 10427, pages 601–620, 2017. doi:10.1007/978-3-319-63390-9_32.

2 Marc Brockschmidt, Fabian Emmes, Stephan Falke, Carsten Fuhs, and Jürgen Giesl. Analyzing Runtime and Size Complexity of Integer Programs. *ACM Trans. Program. Lang. Syst.*, 38:1–50, 2016. doi:10.1145/2866575.

3 Jürgen Giesl, Nils Lommen, Marcel Hark, and Fabian Meyer. Improving Automatic Complexity Analysis of Integer Programs. In *The Logic of Software. A Tasting Menu of Formal Methods*, LNCS 13360, pages 193–228, 2022. doi:10.1007/978-3-031-08166-8_10.

4 Nils Lommen, Fabian Meyer, and Jürgen Giesl. Automatic Complexity Analysis of Integer Programs via Triangular Weakly Non-Linear Loops. In *Proc. IJCAR '22*, LNCS 13385, pages 734–754, 2022. doi:10.1007/978-3-031-10769-6_43.

5 Nils Lommen and Jürgen Giesl. Targeting Completeness: Using Closed Forms for Size Bounds of Integer Programs. In *Proc. FroCoS '23*, LNCS 14279, pages 3–22, 2023. doi:10.1007/978-3-031-43369-6_1.

6 Nils Lommen, Éléanore Meyer, and Jürgen Giesl. Control-Flow Refinement for Complexity Analysis of Probabilistic Programs in KoAT (Short Paper). In *Proc. IJCAR '24*, LNCS 14739, pages 233–243, 2024. doi:10.1007/978-3-031-63498-7_14.

7 Nils Lommen, Éléanore Meyer, and Jürgen Giesl. Targeting Completeness: Automated Complexity Analysis of Integer Programs. *Corr*, abs/2412.01832, 2024. doi:10.48550/arXiv.2412.01832.

8 Nils Lommen and Jürgen Giesl. AProVE (KoAT + LoAT). In *Proc. TACAS '25*, LNCS 15698, pages 205–211, 2025. doi:10.1007/978-3-031-90660-2_13.

9 Fabian Meyer, Marcel Hark, and Jürgen Giesl. Inferring Expected Runtimes of Probabilistic Integer Programs Using Expected Sizes. In *Proc. TACAS '21*, LNCS 12651, pages 250–269, 2021. doi:10.1007/978-3-030-72016-2_14.