Increasing Interpretations*

Harald Zankl and Aart Middeldorp

Institute of Computer Science
University of Innsbruck
6020 Innsbruck, Austria
{harald.zankl,aart.middeldorp}@uibk.ac.at

Abstract. The paper at hand introduces a refinement of interpretation based termination criteria for term rewrite systems in the dependency pair setting. Traditional methods share the property that—in order to be successful—all rewrite rules must (weakly) decrease with respect to some measure. The novelty of our approach is that we allow some rules to increase the interpreted value. These rules are found by simultaneously searching for adequate polynomial interpretations while considering the information of the dependency graph. We prove that our method extends the termination proving power of linear natural interpretations. Furthermore, this generalization perfectly fits the recursive SCC decomposition algorithm which is implemented in virtually every termination prover dealing with term rewrite systems.

Keywords: term rewriting, termination, polynomial interpretations

Related Topics: implementations of symbolic computation systems, logic and symbolic computing

1 Introduction

Termination of term rewriting systems (TRSs) has been a very active area of research for the last decades. In the early days many different (mostly non-modular) techniques have been developed based on syntactic and/or semantic aspects. In the recent past the demand for suitable ways for automating the methods grew. The international competition of termination tools[1] gave a strong stimulus in that direction. In this competition every tool can only spend a fixed amount of time on checking a rewrite system for (non-)termination. Since a vast number of termination criteria are known (and implemented), tool authors have to cleverly select a strategy which determines the order in which to apply the different methods and/or come up with fast implementations of termination criteria. In 2004 Kurihara and Kondo [17] were the first to encode a termination method in propositional logic. In 2006 for the first time termination analyzers

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

[1] http://www.lri.fr/~marche/termination-competition/

incorporated translations to SAT (Jambox [4] and Matchbox [20]) in the competition and astonished the termination community by the gains in power and speed. Another important issue of a termination method is locality which means that the method should fit the dependency pair method [1]. The technique we propose in this paper satisfies both demands, (a) it is modular and local in the sense that it perfectly fits the recursive SCC decomposition algorithm [12] and (b) it allows an efficient implementation using SAT solving.

The paper is organized as follows. In Section 2 the necessary definitions for graph reasoning, polynomial interpretations, and dependency pairs are given. Section 3 motivates our approach by means of an example and already suggests that special care is needed for generalizing the approach to the recursive SCC algorithm. Afterwards in Section 4 the main theorem is formally stated. Implementation details are presented in Section 5. An assessment of our contribution can be found in Section 6 before ideas for future work are addressed in Section 7.

2 Preliminaries

The termination method we present relies on (dependency) graph reasoning. The next subsection defines graphs and related concepts.

2.1 Graphs

Let N be a finite set. A *graph* $\mathcal{G} = (N, E)$ is a pair such that $E \subseteq N \times N$. Elements of N (E) are called *nodes* (*edges*). A *labeled graph* is a pair $(\mathcal{G}, \ell)$ consisting of a graph $\mathcal{G} = (N, E)$ and a labeling function $\ell\colon N \to \mathbb{Z}$ that assigns to every node an integer. A *path* from n_1 to n_m in a graph $\mathcal{G} = (N, E)$ is a finite sequence $[n_1, \ldots, n_m]$ of nodes such that $(n_i, n_{i+1}) \in E$ for all $1 \leqslant i < m$. A path is called *elementary* if all its nodes are distinct. The *length* (or *cost*) of a path $[n_1, \ldots, n_{m-1}, n_m]$ is $\ell(n_1)+\cdots+\ell(n_{m-1})$. The *distance* between two nodes a and b is the maximal length of an elementary path from a to b. A *cycle* $[n_1, \ldots, n_m]$ is a path with $m > 1$, $n_1 = n_m$, and $i \neq j$ implies $(n_i, n_{i+1}) \neq (n_j, n_{j+1})$ for all $0 \leqslant i, j < m$. A cycle $[n_1, \ldots, n_{m-1}, n_m]$ is called *elementary* if $n_1, \ldots, n_{m-1}$ are pairwise distinct. The definition of length carries over naturally from paths to cycles. Furthermore we define the *distance* $\mathsf{d}(n)$ for a single node n as the maximal length of an elementary cycle starting in n if such a cycle exists. A *strongly connected component* (SCC) is a maximal set of nodes such that there is a path from every node to every other node. Maximality means that the property of being an SCC is lost if a further node is added. For esthetic reasons, labels of nodes are associated to edges in graphical representations of graphs throughout the paper, where edges (n, m) are labeled with $\ell(n)$.

Example 1. In the labeled graph of Figure 1, $p_1 = [1, 2, 3, 4, 1]$ is an example of a (non-elementary) path and an elementary cycle. The (non-elementary) path $p_2 = [1, 4, 1, 4, 1]$ is no cycle since the edge $(1, 4)$ appears twice. We have $\text{length}(p_1) = 0$ and $\text{length}(p_2) = 2$. The distance of node 1 is 1 since it is the maximum length of the elementary cycles $[1, 4, 1]$ and $[1, 2, 3, 4, 1]$.

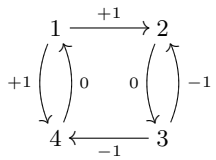


Fig. 1. A labeled graph.

2.2 Polynomial Interpretations

For a signature $\mathcal{F}$ a polynomial interpretation $\mathcal{I}$ [18] maps each n-ary function symbol $f \in \mathcal{F}$ to a polynomial $f_{\mathcal{I}}$ over the natural numbers in n indeterminates. The induced mapping from terms to polynomials is denoted by $[\cdot]_{\mathcal{I}}$. For two terms s and t we have $s >_{\mathcal{I}} t$ if $[s]_{\mathcal{I}} > [t]_{\mathcal{I}}$ holds for all possible instantiations of variables by natural numbers. The comparison $s \geqslant_{\mathcal{I}} t$ is similarly defined. For polynomials with coefficients ranging over the natural numbers these problems are known to be undecidable (Hilbert's 10^{th} problem). By fixing an upper bound for the coefficients the search space becomes finite. In typical implementations polynomials are ordered by absolute positiveness criteria [14]. Thus, in order to test whether $p > q$ holds for *linear* polynomials $p = c_0x_0 + \cdots + c_nx_n + c_{n+1}$ and $q = d_0x_0 + \cdots + d_nx_n + d_{n+1}$, a sufficient condition is $c_i \geqslant d_i$ for all $0 \leqslant i \leqslant n$ and $c_{n+1} > d_{n+1}$. The test $p \geqslant q$ is similar except for the constant case, i.e., $c_{n+1} \geqslant d_{n+1}$.

There already exist generalizations of polynomial interpretations, e.g., to rational and real coefficients [19] or to negative constants as well as coefficients [11]. Furthermore matrix [5], quasi-periodic [22], and arctic [15] interpretations do also extend the termination proving power significantly. All these extensions share the property that the rewrite rules under consideration must weakly decrease and at least one rule has to decrease strictly. Our approach differs from these ones in the sense that we allow a possible increase for some rules (under the side condition that some other rules eliminate that increase). In order to detect possible candidates where the interpreted value might increase when applying a rule, the dependency pair method in combination with the dependency graph (Definition 3) refinement is employed.

2.3 Dependency Pairs

We assume basic familiarity with term rewriting [2]. In the recent past there has been much research related to the dependency pair method [1] and its refinements. In this subsection we just recall the very basic definitions.

Definition 2. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. The defined symbols are the root symbols of the left-hand sides of the rewrite rules in $\mathcal{R}$. The original signature $\mathcal{F}$ is extended to a signature $\mathcal{F}^\sharp$ by adding for every defined symbol f a fresh symbol $f^\sharp$ with the same arity as f. For a term $t = f(t_1, \ldots, t_n)$*

with defined symbol f we denote $f^\sharp(t_1, \ldots, t_n)$ by $t^\sharp$. In examples one often uses capitalization, i.e., one writes F for $f^\sharp$. If $l \to r \in \mathcal{R}$ and t is a subterm of r with defined root symbol, then the rule $l^\sharp \to t^\sharp$ is a dependency pair of $\mathcal{R}$. We write $\mathsf{DP}(\mathcal{R})$ for the set of all dependency pairs of $\mathcal{R}$.

Dependency pairs correspond to recursive function calls. They are the basic ingredient for the dependency graph [1], which is kind of a call-graph that visualizes the order in which these recursive calls can be performed.

Definition 3. *Let $\mathcal{R}$ be a TRS. The nodes of the* dependency graph $\mathsf{DG}(\mathcal{R})$ *are the dependency pairs of $\mathcal{R}$ and there is an edge from node $s \to t$ to node $u \to v$ if there exist substitutions σ and τ such that $t\sigma \to_{\mathcal{R}}^* u\tau$.*

The dependency graph is not computable in general but sound approximations exist. Here soundness means that every edge in the original graph is also an edge in the estimated graph and hence it forms an over-approximation of the actual dependency graph.

Next, the notion of a reduction pair [1] is defined. We simplify the original definition by omitting argument filterings since they are automatically built in when dealing with polynomial interpretations (as zero coefficients correspond to deleting positions of an argument filtering).

Definition 4. *A reduction pair $(\gtrsim, >)$ consists of a rewrite pre-order $\gtrsim$ (a pre-order on terms that is closed under contexts and substitutions) and a well-founded order $>$ that is closed under substitutions such that the inclusion $\gtrsim \cdot > \cdot \gtrsim \subseteq >$ (compatibility) holds.*

The main theorem dealing with dependency pairs and including a dependency graph formulation is not given here but in Section 4 since then it is easier to see the differences between the usual theorem and our formulation.

3 A Simple Example

This section demonstrates the limitations of polynomial interpretations and suggests an improvement by additionally considering the order of recursive calls encoded in the dependency graph.

Example 5. Consider the TRS consisting of the following three rules:

$$\mathsf{f}(\mathsf{0}, x) \to \mathsf{f}(\mathsf{1}, \mathsf{g}(x)) \tag{1}$$
$$\mathsf{f}(\mathsf{1}, \mathsf{g}(\mathsf{g}(x))) \to \mathsf{f}(\mathsf{0}, x) \tag{2}$$
$$\mathsf{g}(\mathsf{1}) \to \mathsf{g}(\mathsf{0}) \tag{3}$$

The dependency pairs

$$\mathsf{F}(\mathsf{0}, x) \to \mathsf{G}(x) \tag{4}$$
$$\mathsf{F}(\mathsf{0}, x) \to \mathsf{F}(\mathsf{1}, \mathsf{g}(x)) \tag{5}$$
$$\mathsf{F}(\mathsf{1}, \mathsf{g}(\mathsf{g}(x))) \to \mathsf{F}(\mathsf{0}, x) \tag{6}$$
$$\mathsf{G}(\mathsf{1}) \to \mathsf{G}(\mathsf{0}) \tag{7}$$

admit the following dependency graph:

$$(7) \longleftarrow (4) \longleftarrow (6) \rightleftarrows (5)$$

The idea in [12] is to find a reduction pair $(\gtrsim, >)$ for every SCC $\mathcal{S}$ such that all rules in $\mathcal{S} \cup \mathcal{R}$ decrease weakly and at least one rule in $\mathcal{S}$ decreases strictly. In the sequel we will show that the (only) SCC consisting of the nodes (5) and (6) cannot be handled by reduction pairs based on traditional implementations of linear polynomial interpretations. To be able to address all possible polynomial interpretations, we consider our problem as an abstract constraint satisfaction problem. Consequently the coefficients for the polynomials are variables whose values are natural numbers. Similarly to [6] a term $\mathsf{F}(x, y)$ is transformed into an abstract linear polynomial $\mathsf{F}_0 x + \mathsf{F}_1 y + \mathsf{F}_2$. Doing so for the SCC mentioned above results in the constraints

$$\begin{aligned} \mathsf{F}_0\mathsf{0}_0 + \mathsf{F}_1 x + \mathsf{F}_2 &\geqslant \mathsf{F}_0\mathsf{1}_0 + \mathsf{F}_1(\mathsf{g}_0 x + \mathsf{g}_1) + \mathsf{F}_2 \\ \mathsf{F}_0\mathsf{1}_0 + \mathsf{F}_1(\mathsf{g}_0(\mathsf{g}_0 x + \mathsf{g}_1) + \mathsf{g}_1) + \mathsf{F}_2 &\geqslant \mathsf{F}_0\mathsf{0}_0 + \mathsf{F}_1 x + \mathsf{F}_2 \end{aligned}$$

where at least one inequality is strict. By simple mathematics the inequations simplify to

$$\mathsf{F}_0\mathsf{0}_0 + \mathsf{F}_1 x \geqslant \mathsf{F}_0\mathsf{1}_0 + \mathsf{F}_1\mathsf{g}_0 x + \mathsf{F}_1\mathsf{g}_1 \tag{8}$$

$$\mathsf{F}_0\mathsf{1}_0 + \mathsf{F}_1\mathsf{g}_0\mathsf{g}_0 x + \mathsf{F}_1\mathsf{g}_0\mathsf{g}_1 + \mathsf{F}_1\mathsf{g}_1 \geqslant \mathsf{F}_0\mathsf{0}_0 + \mathsf{F}_1 x \tag{9}$$

From the fact that one of the above inequalities has to be strict it is obvious that $\mathsf{F}_1 > 0$. The constraints for x in (8) demand $\mathsf{g}_0 \leqslant 1$ and similarly (9) gives $\mathsf{g}_0 \geqslant 1$. Hence the constraint problem is equivalent to

$$\mathsf{F}_0\mathsf{0}_0 \geqslant \mathsf{F}_0\mathsf{1}_0 + \mathsf{F}_1\mathsf{g}_1 \tag{10}$$

$$\mathsf{F}_0\mathsf{1}_0 + \mathsf{F}_1\mathsf{g}_1 + \mathsf{F}_1\mathsf{g}_1 \geqslant \mathsf{F}_0\mathsf{0}_0 \tag{11}$$

which demands $\mathsf{g}_1 > 0$ to make one inequation strict. The (simplified) constraint for rule (3) amounts to

$$\mathsf{1}_0 \geqslant \mathsf{0}_0 \tag{12}$$

The proof is concluded by the contradictory sequence

$$\mathsf{F}_0\mathsf{0}_0 \geqslant \mathsf{F}_0\mathsf{1}_0 + \mathsf{F}_1\mathsf{g}_1 \geqslant \mathsf{F}_0\mathsf{0}_0 + \mathsf{F}_1\mathsf{g}_1$$

where the first inequality derives from (10), the second one from (12), and the contradiction from the fact that $\mathsf{F}_1, \mathsf{g}_1 > 0$ which we learned earlier.

Although we just proved that there is no termination proof for the system above with linear polynomials, we will present a termination proof right now. Assume the weakly monotone interpretation

$$\mathsf{F}_{\mathbb{N}}(x, y) = x + y \quad \mathsf{f}_{\mathbb{N}}(x, y) = 0 \quad \mathsf{g}_{\mathbb{N}}(x) = x + 1 \quad \mathsf{0}_{\mathbb{N}} = 0 \quad \mathsf{1}_{\mathbb{N}} = 0$$

$$\begin{aligned}
\mathsf{f}(0,x) &\to \mathsf{f}(1,\mathsf{g}(x)) && 0 \geqslant 0 && (1)\\
\mathsf{f}(1,\mathsf{g}(\mathsf{g}(x))) &\to \mathsf{f}(0,x) && 0 \geqslant 0 && (2)\\
\mathsf{g}(1) &\to \mathsf{g}(0) && 1 \geqslant 1 && (3)\\
\mathsf{F}(0,x) &\to \mathsf{F}(1,\mathsf{g}(x)) && x \geqslant x+1 && (5)\\
\mathsf{F}(1,\mathsf{g}(\mathsf{g}(x))) &\to \mathsf{F}(0,x) && x+2 \geqslant x && (6)
\end{aligned}$$

Table 1. Rules with increasing interpretations.

which orients almost all rules of interest correctly as can be seen in Table 1.

The idea to turn this interpretation into a valid termination proof is to combine the information of the dependency graph with the interpretation. From the (labeled) dependency graph

$$(7) \xleftarrow{\;0\;} (4) \xleftarrow{\;-2\;} (6) \underset{+1}{\overset{-2}{\rightleftarrows}} (5)$$

one infers that the two dependency pairs (5) and (6) are used alternately. The labels of the graph are computed as follows: From Table 1 one infers that an application of rule (6) *decreases* the interpreted value by the constant 2 (hence label -2) whereas rule (5) *increases* the value by the constant 1 (hence label $+1$). Consequently, after performing the cycle once the total value decreases by at least one. Therefore, the cycle cannot give rise to an infinite rewrite sequence.

3.1 From Cycles to SCCs

The above idea naturally extends from plain cycles to SCCs as described below. Nevertheless some care is needed when the dependency graph contains more complicated SCCs as the following example demonstrates. Consider the TRS $\mathcal{R}$ consisting of the five rules

$$\begin{aligned}
\mathsf{f}(0,0,x,\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(y))))) &\to \mathsf{f}(0,1,\mathsf{g}(\mathsf{g}(x)),y)\\
\mathsf{f}(0,1,\mathsf{g}(x),y) &\to \mathsf{f}(1,1,x,\mathsf{g}(\mathsf{g}(y)))\\
\mathsf{f}(1,1,x,y) &\to \mathsf{f}(0,x,x,y)\\
\mathsf{g}(0) &\to \mathsf{g}(1)\\
\mathsf{g}(x) &\to x
\end{aligned}$$

and the only SCC

$$\begin{aligned}
\mathsf{F}(0,0,x,\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(y))))) &\to \mathsf{F}(0,1,\mathsf{g}(\mathsf{g}(x)),y) && (1)\\
\mathsf{F}(0,1,\mathsf{g}(x),y) &\to \mathsf{F}(1,1,x,\mathsf{g}(\mathsf{g}(y))) && (2)\\
\mathsf{F}(1,1,x,y) &\to \mathsf{F}(0,x,x,y) && (3)
\end{aligned}$$

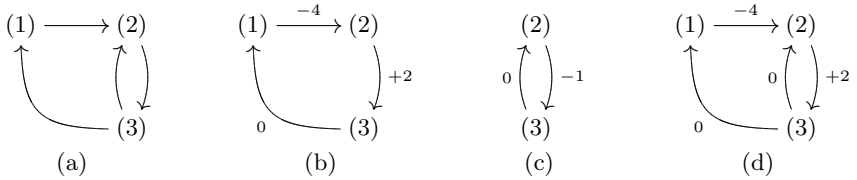


Fig. 2. Different parts of (labeled) dependency graphs.

The corresponding SCC of the dependency graph depicted in Figure 2(a) contains the two cycles $[1, 2, 3, 1]$ and $[2, 3, 2]$. The first one is handled by the increasing interpretation

$$\mathsf{F}_{\mathbb{N}}(x, y, z, w) = w \quad \mathsf{f}_{\mathbb{N}}(x, y, z, w) = 0 \quad \mathsf{g}_{\mathbb{N}}(x) = x + 1 \quad \mathsf{0}_{\mathbb{N}} = 0 \quad \mathsf{1}_{\mathbb{N}} = 0$$

For the second we take the interpretation as above but with $\mathsf{F}_{\mathbb{N}}(x, y, z, w) = z$. Hence for the elementary cycle $[1, 2, 3, 1]$ the interpreted value decreases by 2 in every loop. Similarly there is a decrease of 1 for the elementary cycle $[2, 3, 2]$. The two labeled graphs in Figures 2(b) and 2(c) describe the symbiosis of the interpretations and the elementary cycles. The only problem is, that

$$\begin{aligned}
&\mathsf{f}(\mathsf{0}, \mathsf{0}, \mathsf{0}, \mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(y)))))) \to \mathsf{f}(\mathsf{0}, \mathsf{1}, \mathsf{g}(\mathsf{g}(\mathsf{0})), y) \to \mathsf{f}(\mathsf{1}, \mathsf{1}, \mathsf{g}(\mathsf{0}), \mathsf{g}(\mathsf{g}(y)))\\
&\quad \to \mathsf{f}(\mathsf{0}, \mathsf{g}(\mathsf{0}), \mathsf{g}(\mathsf{0}), \mathsf{g}(\mathsf{g}(y))) \to \mathsf{f}(\mathsf{0}, \mathsf{g}(\mathsf{1}), \mathsf{g}(\mathsf{0}), \mathsf{g}(\mathsf{g}(y)))\\
&\quad \to \mathsf{f}(\mathsf{0}, \mathsf{1}, \mathsf{g}(\mathsf{0}), \mathsf{g}(\mathsf{g}(y))) \to \mathsf{f}(\mathsf{1}, \mathsf{1}, \mathsf{0}, \mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(y))))))\\
&\quad \to \mathsf{f}(\mathsf{0}, \mathsf{0}, \mathsf{0}, \mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(y)))))) \to \dots
\end{aligned}$$

constitutes a non-terminating sequence in this TRS. What exactly went wrong can be seen when considering the whole SCC of the labeled dependency graph (using the first interpretation, cf. Figure 2(d)). In the conventional setting it suffices to consider only the two cycles. This is the case because a strict decrease in every single cycle ensures a strict decrease in larger cycles by combining the partial proofs lexicographically. The example above shows that this is no longer true for increasing interpretations. The problematic non-terminating sequence corresponds to a run $[1, 2, 3, 2, 3, 1]$ where the interpreted value is increased in the elementary cycle $[2, 3, 2]$ and consequently the length of $[1, 2, 3, 2, 3, 1]$ is zero and there is no decrease. Considering (infinitely many!) possibly non-elementary cyclic paths is undoable. Hence the smart thing is to work with SCCs instead. To recognize dangerous runs, it suffices to compute the distance for every node. For the graph in Figure 2(d) we have $\mathsf{d}(1) = -2$, $\mathsf{d}(2) = 2$, and $\mathsf{d}(3) = 2$. Only if for every node the distance is smaller than or equal to zero we know that problematic runs as demonstrated above cannot occur. Furthermore we know that in such a case we can delete nodes with negative distance because on every possible run the interpreted value decreases. If for the SCC under consideration one had managed to find a weakly monotone interpretation with labeled dependency graph like the one in Figure 3(a) (which is of course impossible since the system at hand is not

(a) The labeled DG. (b) The remaining DG.

Fig. 3. A hypothetically labeled DG.

terminating) then deleting node (1) would have been possible since $\mathsf{d}(1) = -1$, $\mathsf{d}(2) = 0$, and $\mathsf{d}(3) = 0$. In such a situation node (1) could safely be removed and one could proceed with the simpler graph in Figure 3(b) with a possibly totally different interpretation.

4 Correctness of the Approach

The example in the preceding section shows that SCCs that consist of more than just one cycle need special attention. For usual reduction pairs it is sufficient to consider single cycles and hence in the literature theorems are usually dealing with cycles; theoretically there is no difference in power when considering cycles or SCCs but all fast implementations follow the recursive SCC approach [12]. The reason is that normally the formulation for cycles is a bit easier but in our setting it is essential to switch to an SCC treatment in order to avoid reasoning about an infinite number of possibly non-elementary cyclic paths as the example of the previous section demonstrates.

It is well known that (linear) weakly monotone polynomial interpretations over the naturals form a valid reduction pair. Note that there are strictly stronger formulations of the theorem since both restrictions—to polynomials and natural numbers—are severe.

Theorem 6. *Let $\mathcal{I}$ be a weakly monotone polynomial interpretation over the naturals. Then $(\gtrsim_{\mathcal{I}}, >_{\mathcal{I}})$ is a reduction pair.*

Definition 7 ([12]). *Let $\mathcal{R}$ be a TRS, $\mathcal{S}$ a subset of the dependency pairs in $\mathsf{DG}(\mathcal{R})$, and $(\gtrsim, >)$ a reduction pair. The notation $(\gtrsim, >) \models_\exists \mathcal{R}, \mathcal{S}$ means that*

$$\mathcal{R} \subseteq \gtrsim \qquad \mathcal{S} \subseteq \gtrsim \cup > \qquad \mathcal{S} \cap > \neq \varnothing$$

In words the above definition says that all considered rules ($\mathcal{R}$ and $\mathcal{S}$) are weakly decreasing and at least one rule in $\mathcal{S}$ is strictly decreasing. The most basic theorem concerning dependency pairs (using the notation of [12]) and including the usage of the dependency graph is then formulated as follows.

Theorem 8 ([1]). *A TRS $\mathcal{R}$ is terminating if and only if for every cycle $\mathcal{C}$ in $\mathsf{DG}(\mathcal{R})$ there exists a reduction pair $(\gtrsim, >)$ such that $(\gtrsim, >) \models_\exists \mathcal{R}, \mathcal{C}$.*

There are many generalizations of the theorem above—usable rules [1,9,10], argument filterings [1], and reduction triples [13]—to name a few. To keep the presentation and discussion simple we present our work without these refinements (although our results directly generalize).

Definition 9 ([12]). *Let $\mathcal{R}$ be a TRS and $\mathcal{S}$ a subset of the dependency pairs in $\mathsf{DG}(\mathcal{R})$. We write $\models \mathcal{R}, \mathcal{S}$ if there exists a reduction pair $(\gtrsim, >)$ such that $(\gtrsim, >) \models_\exists \mathcal{R}, \mathcal{S}$ and $\models \mathcal{R}, \mathcal{S}'$ for all SCCs $\mathcal{S}'$ of the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by the pairs $l \to r \in \mathcal{S}$ such that $l \not> r$.*

The theorem below states that concerning termination proving power it makes no difference if one considers cycles or performs a recursive SCC computation. The latter has the advantage that the number of SCCs is linear in the number of nodes in the dependency graph whereas the former might be exponential.

Theorem 10 ([12]). *Let $\mathcal{R}$ be a TRS. The following conditions are equivalent:*

- *$\models \mathcal{R}, \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$*
- *$\models_\exists \mathcal{R}, \mathcal{C}$ for every cycle $\mathcal{C}$ in $\mathsf{DG}(\mathcal{R})$*

We now show how to label the dependency graph by a given interpretation $\mathcal{I}$. When considering a root rewrite step which applies a rule $l \to r$, the change of the interpreted value is $[r]_\mathcal{I} - [l]_\mathcal{I}$. The idea is to label every edge by the constant part of that difference.

Definition 11. *For a polynomial p we denote the constant (non-constant) part of p by $\mathsf{cp}(p)$ ($\mathsf{ncp}(p)$). For a term t and a polynomial interpretation $\mathcal{I}$ we abbreviate $\mathsf{ncp}([t]_\mathcal{I})$ by $\mathsf{ncp}_\mathcal{I}(t)$. This notation naturally extends to rules and TRSs, e.g., $\mathsf{ncp}_\mathcal{I}(l \to r) = \mathsf{ncp}_\mathcal{I}(l) \to \mathsf{ncp}_\mathcal{I}(r)$ and $\mathsf{ncp}_\mathcal{I}(\mathcal{R}) = \{\mathsf{ncp}_\mathcal{I}(l \to r) \mid l \to r \in \mathcal{R}\}$. The same notation is freely used for $\mathsf{cp}_\mathcal{I}$.*

Definition 12. *Let $\mathcal{I}$ be an interpretation and DG a dependency graph. The labeled dependency graph $\mathsf{DG}_\mathcal{I}$ is defined as (DG, ℓ) with $\ell(l \to r) = \mathsf{cp}([r]_\mathcal{I} - [l]_\mathcal{I})$ for every node $l \to r$ in DG. By $\mathsf{d}_\mathcal{I}(n)$ we denote the distance of a node $n \in \mathsf{DG}_\mathcal{I}$.*

The next definition presents analogous versions of Definitions 7 and 9 in the setting of increasing interpretations.

Definition 13. *Let $\mathcal{R}$ be a TRS and $\mathcal{S}$ a subset of the dependency pairs in $\mathsf{DG}(\mathcal{R})$. We write $\models^\mathcal{I}_\exists \mathcal{R}, \mathcal{S}$ if $\mathcal{I}$ is an interpretation over the naturals and*

$$\mathcal{R} \cup \mathsf{ncp}_\mathcal{I}(\mathcal{S}) \subseteq \geqslant_\mathcal{I} \quad \mathsf{d}_\mathcal{I}(\mathcal{S}) \subseteq \mathbb{Z}^{\leqslant 0} \quad \mathsf{d}_\mathcal{I}(\mathcal{S}) \cap \mathbb{Z}^{<0} \neq \varnothing$$

Consequently $\models^\mathcal{I} \mathcal{R}, \mathcal{S}$ if $\models^\mathcal{I}_\exists \mathcal{R}, \mathcal{S}$ and for all SCCs $\mathcal{S}'$ of the subgraph of $\mathsf{DG}_\mathcal{I}(\mathcal{R})$ induced by the pairs $l \to r \in \mathcal{S}$ such that $\mathsf{d}_\mathcal{I}(l \to r) \not< 0$ there exists an interpretation $\mathcal{I}'$ such that $\models^{\mathcal{I}'} \mathcal{R}, \mathcal{S}'$.

Now we are ready to present the main theorem. In Section 3 we already showed that this extends the termination proving power of natural linear interpretations.

Theorem 14. *A TRS $\mathcal{R}$ is terminating if for every SCC $\mathcal{S}$ in the dependency graph $\mathsf{DG}(\mathcal{R})$ there exists a weakly monotone polynomial interpretation $\mathcal{I}$ over the naturals such that $\models^{\mathcal{I}} \mathcal{R}, \mathcal{S}$ holds.*

Proof. We show that under the assumption $\models^{\mathcal{I}}_{\exists} \mathcal{R}, \mathcal{S}$ with $s \to t \in \mathcal{S}$ satisfying $\mathsf{d}_{\mathcal{I}}(s \to t) < 0$ there cannot be a non-terminating rewrite sequence that applies $s \to t$ indefinitely. The theorem follows immediately from that property. For a proof by contradiction assume the existence of such a sequence:

$$s_0 \to_{s \to t} t_0 \to^*_{\mathcal{S} \cup \mathcal{R}} s_1 \to_{s \to t} t_1 \to^*_{\mathcal{S} \cup \mathcal{R}} s_2 \to_{s \to t} t_2 \to^*_{\mathcal{S} \cup \mathcal{R}} s_3 \to \dots$$

Since $\geqslant_{\mathcal{I}}$ is closed under contexts and substitutions, for all terms u, v, and all rules $l \to r \in \mathcal{R} \cup \mathcal{S}$ with $u \to_{l \to r} v$ we get $\mathsf{ncp}_{\mathcal{I}}(u) \geqslant \mathsf{ncp}_{\mathcal{I}}(v)$. Because the infinite sequence was chosen such that the rule $s \to t$ is used infinitely often it is obvious that when starting from term s_0 one must cycle in the dependency graph in order to reach s_1. The fact that $\mathsf{d}_{\mathcal{I}}(s \to t) < 0$ together with $\mathsf{d}_{\mathcal{I}}(\mathcal{S}) \subseteq \mathbb{Z}^{\leqslant 0}$ ensures that every cycle containing the node $s \to t$ decreases the constant part of the interpretation strictly (note that $\mathsf{cp}_{\mathcal{I}}(\mathcal{R}) \subseteq \geqslant$ by definition). Hence, $\mathsf{cp}_{\mathcal{I}}(s_0) > \mathsf{cp}_{\mathcal{I}}(s_1)$. Repeating this argument gives rise to the sequence

$$\mathsf{cp}_{\mathcal{I}}(s_0) > \mathsf{cp}_{\mathcal{I}}(s_1) > \mathsf{cp}_{\mathcal{I}}(s_2) > \mathsf{cp}_{\mathcal{I}}(s_3) > \dots$$

which contradicts the well-foundedness of $>$ over the natural numbers. □

5 Implementation

Almost all fast implementations of polynomial interpretations are based on a transformation to a SAT problem. Also many other termination criteria are very suitable for a SAT encoding as can be seen by the vast amount of literature. The major drawback is that one has to work with abstract encodings all the time. Hence when labeling the dependency graph one does not have concrete integers at hand but some propositional formulas which abstractly encode the range of all possible values. Since encoding polynomials in SAT has already been described in detail [6], in this paper we refrain from giving all implementation issues. The only encoding which is discussed here is how to compute the distance between two (not necessarily distinct) nodes within a labeled graph.

5.1 General Algorithm

The idea is to compute the distance of a node by means of a transitivity closure. The integer variable R_{abi} is $-\infty$ if b is not reachable in at most 2^i steps from a and otherwise this variable keeps the (currently known) distance from a to b.

It is obvious that in a graph (N, E) an elementary cycle contains at most $|N|$ edges and hence for $k' \geqslant k := \lceil log_2(|N|) \rceil$ one has surely reached a fixed point, i.e., $R_{abk'} = R_{abk}$ for all $a, b \in N$.

More precisely, the variables R_{ab0} reflect the edges of the graph and hence b is reachable from a with a cost of $\ell(a)$ if $(a, b) \in E$ and it is unreachable if $(a, b) \notin E$. Thus we initialize these variables as follows:

$$R_{ab0} = \begin{cases} \ell(a) & \text{if } (a, b) \in E \\ -\infty & \text{otherwise} \end{cases}$$

Since R_{abi} might be $-\infty$, addition and maximum operation are extended naturally, i.e., $n + -\infty = -\infty + n = -\infty$ and $\mathsf{max}(n, -\infty) = \mathsf{max}(-\infty, n) = n$ for all $n \in \mathbb{Z} \cup \{-\infty\}$. For $0 \leqslant i < k$ we define

$$R_{ab(i+1)} = \mathsf{max}(R_{abi}, \mathsf{max}_{m \in N}\{R_{ami} + R_{mbi}\})$$

If one first forgets about the max then the above formula expresses that b is reachable from a in at most 2^{i+1} steps with a cost of $R_{ab(i+1)}$ if it is already reachable within 2^i steps with that cost or there is a mid-point[2] m and the cost from a to m and the one from m to b just sum up. Taking the maximum of all possible costs ensures that we consider a worst case scenario. In the end we want to test if $R_{nnk} \leqslant 0$ for all $n \in N$. Note that it might happen that the value R_{nnk} does not emerge from an elementary cycle (because it might happen that one cycles more than once). Nevertheless the idea remains sound because if the length of a maximal elementary cycle is smaller than zero, then the length remains smaller than zero if we go along that cycle more often. Dually this property holds for distances greater than zero. For a demonstration consider the following example.

Example 15. In the labeled graph from Example 1 we have $k = \lceil log_2(4) \rceil = 2$ and

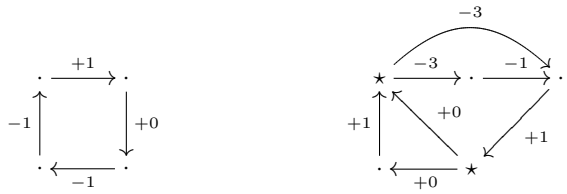
$$\begin{array}{cccc} \mathsf{d}(1) = 1 & \mathsf{d}(2) = 0 & \mathsf{d}(3) = 0 & \mathsf{d}(4) = 1 \\ R_{112} = 2 & R_{222} = 0 & R_{332} = 0 & R_{442} = 2 \end{array}$$

The reason for the different values is that R_{112} does not correspond to an elementary cycle; we have $\mathsf{d}(1) = 1$ (see Example 1) but $R_{112} = 2$ since it derives from the cyclic path $[1, 4, 1, 4, 1]$. A similar argument explains the discrepancy of $\mathsf{d}(4)$ and R_{442}.

5.2 Special Algorithms

The encoding for computing maximal paths in SAT from the previous subsection has complexity $\mathcal{O}(n^2 log(n))$ where n is the number of nodes in the underlying SCC of the labeled DG. To get a faster implementation we specialize the algorithm for SCCs that have a special shape:

[2] Fortunately Zeno of Elea was wrong and this approach constitutes a valid method for computing reachability.



(a) A simple SCC. (b) An almost simple SCC.

Fig. 4. Two special shapes of SCCs.

(a) Simple SCCs: An SCC is called *simple* if it contains exactly one cycle, i.e., omitting any edge would destroy the property of being an SCC. An example of this shape is depicted in Figure 4(a). Linear time suffices to decide if a given SCC $\mathcal{S}$ is simple (the number of edges equals the number of nodes). In such a case the encoding specializes to

$$\sum_{n \in \mathcal{S}} \ell(n) < 0$$

which expresses that the constant part of the interpretation $\mathcal{I}$ decreases when cycling. The encoding is linear in the size of the nodes.

(b) Almost simple SCCs: An SCC is called *almost simple* if it is not simple and there exists a node n (called *selected node*) such that after deleting *all* outgoing edges of n there is no non-empty sub-SCC left. Here we will exploit the fact that in every cycle within this SCC we pass the node n. The nodes indicated with $\star$ in Figure 4(b) satisfy this property. In the encoding we demand that $-\ell(n) > \ell(m_1) + \cdots + \ell(m_p)$ holds where n is the selected node and $m_1, \ldots, m_p$ are the nodes in the SCC that have a positive label. The underlying idea is that node n decreases the interpretation more than all other rules together might increase it and since that node n must be passed in every cyclic run there cannot be infinite reductions. For every selected node n the encoding is of linear size.

Note that the specialization for case (a) is exact whereas (b) is an approximation.

6 Assessment

In this paper we showed that increasing interpretations are strictly more powerful than standard linear interpretations over the naturals. Clearly for SCCs consisting of just a single rule they are of equal power.

The reason why the TRS of Example 5 cannot be proved terminating by means of linear polynomials is that we cannot differentiate constant `0` from `1` by the interpretation. Hence it is not so astonishing that the problematic SCC can be handled by matrix interpretations [5] of dimension two. Actually all of the tools (dedicated to proving termination) participating in the TRS category

of the 2007 edition of the international termination competition can handle this system. All the proofs rely on matrix interpretations with dimension two. As a pre-processing step AProVE [8] and TTT2[3] use dependency pair analysis whereas Jambox [4] performs a reduction of right-hand sides [21].

It is an easy exercise to construct (larger) TRSs than Example 5 such that all tools of the termination competition fail. To disallow Jambox the rewriting of right-hand sides we introduce overlaps. To knock-out the matrix method just increasing the size of the system suffices. Since TTT2 can still prove these examples by bounds [7,16] we ensure the TRS to be not left-linear which makes increasing interpretations the only successful method.

Example 16. For the TRS where $\mathsf{g}^8(x)$ is a shortcut for $\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(x))))))))$

$$\begin{aligned}
\mathsf{f}(0,0,0,x) &\to \mathsf{f}(0,0,1,\mathsf{g}(x)) & \mathsf{f}(0,0,1,x) &\to \mathsf{f}(0,1,0,\mathsf{g}(x))\\
\mathsf{f}(0,1,0,x) &\to \mathsf{f}(0,1,1,\mathsf{g}(x)) & \mathsf{f}(0,1,1,x) &\to \mathsf{f}(1,0,0,\mathsf{g}(x))\\
\mathsf{f}(1,0,0,x) &\to \mathsf{f}(1,0,1,\mathsf{g}(x)) & \mathsf{f}(1,0,1,x) &\to \mathsf{f}(1,1,0,\mathsf{g}(x))\\
\mathsf{f}(1,1,0,x) &\to \mathsf{f}(1,1,1,\mathsf{g}(x)) & \mathsf{f}(y,y,y,\mathsf{g}^8(x)) &\to \mathsf{f}(0,0,0,x)\\
\mathsf{g}(\mathsf{g}(0)) &\to 1 & \mathsf{g}(\mathsf{g}(1)) &\to \mathsf{g}(\mathsf{g}(0))
\end{aligned}$$

none of the existing termination tools succeeds in proving termination within a 60 seconds time limit. Increasing interpretations produce a successful—and very intuitive—proof for the challenging SCC. It considers the changes of F's fourth argument. Both the general approach described in Section 5.1 and the specialization (b) from Section 5.2 yield the increasing interpretation $\mathsf{F}_{\mathbb{N}}(x,y,z,w) = w$, $\mathsf{g}_{\mathbb{N}}(x) = x+1$, $\mathsf{f}_{\mathbb{N}}(x,y,z,w) = 0_{\mathbb{N}} = 1_{\mathbb{N}} = 0$ which ensures that all nodes have a negative distance and hence the whole problematic SCC can be removed. The only difference between the two is that it takes the first method almost half a minute whereas the optimized encoding succeeds within a fraction of a second.

The theory of increasing interpretations as described above directly applies to the matrix method [5] as well. Note that when interpreting dependency pairs the constant part amounts to a natural number and hence the dependency graph is labeled in exactly the same fashion.

7 Future Work

Generalizing the approach in such a way that not only the constant part of the interpretation is used as additional information in the dependency graph but also the non-constant part, is highly desirable. We anticipate that this would make the approach significantly more powerful. The only drawback is that probably this generalization applies to a very restricted class of TRSs only. To get a feeling for the problems that arise consider the non-terminating system

$$\mathsf{f}(\mathsf{s}(x)) \to \mathsf{g}(\mathsf{s}(x)) \qquad \mathsf{g}(x) \to \mathsf{f}(x)$$

[3] `http://colo6-c703.uibk.ac.at/ttt2`

which admits the dependency pairs (1) $\mathsf{F}(\mathsf{s}(x)) \to \mathsf{G}(\mathsf{s}(x))$ and (2) $\mathsf{G}(x) \to \mathsf{F}(x)$. The increasing interpretation $\mathsf{F}_{\mathbb{N}}(x) = 2x$, $\mathsf{G}_{\mathbb{N}}(x) = x$, $\mathsf{f}_{\mathbb{N}}(x) = 0$, $\mathsf{s}_{\mathbb{N}}(x) = x + 1$ would remove both dependency pairs since there is a strict decrease for every cycle in the labeled dependency graph, which looks like

$$(1) \overset{-x-1}{\underset{+x}{\rightleftarrows}} (2)$$

The problem in this example is that in the two dependency pairs the variable x does not correspond to the same term. For this example it is obvious that in any minimally non-terminating sequence, $\mathsf{s}(x)$ is substituted for the variable x in the second rule. Hence, one should not consider the original system but immediately change the variable x in the second rule on both sides to $\mathsf{s}(x)$. Then increasing interpretations are no longer successful. However such a transformation is not always possible. In the example above for every minimally non-terminating sequence there are no $\mathcal{R}$-steps and hence one can compute the substitution for x in the second rule by unification. Similar cases can be dealt with narrowing [1].

To conclude, we summarize that increasing interpretations can be extended to allow an increase also in the variable part if the TRS under consideration satisfies two properties: (a) all dependency pairs are variable disjoint (this can always be achieved by renaming) and (b) for every minimally non-terminating sequence

$$s_0 \to_{\mathsf{DP}(\mathcal{R})} t_0 \to^*_{\mathcal{R}} s_1 \to_{\mathsf{DP}(\mathcal{R})} t_1 \to^*_{\mathcal{R}} s_2 \to_{\mathsf{DP}(\mathcal{R})} t_2 \to^*_{\mathcal{R}} \cdots$$

the $\mathcal{R}$-sequences are empty (and hence the values for variables can possibly be computed by unification). Note that one sufficient condition for (b) is that the set of usable rules is empty.

Acknowledgments We thank Niklas Eén and Niklas Sörensson for developing and providing MiniSat [3]. We thank Sarah Winkler for writing a suitable OCaml interface for MiniSat that paved the way for an integration of increasing interpretations into the termination prover TTT2.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
3. N. Eén and N Sörensson. An extensible SAT-solver. In *Proc. 6th International Conference on Theory and Applications of Satisfiability Testing*, volume 2919 of *LNCS*, pages 502–518, 2003.
4. J. Endrullis. Jambox, 2007. Available from `http://joerg.endrullis.de`.
5. J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. *Journal of Automated Reasoning*, 40(2-3):195–220, 2008.

6. C. Fuhs, J. Giesl, A. Middeldorp, P. Schneider-Kamp, R. Thiemann, and H. Zankl. SAT solving for termination analysis with polynomial interpretations. In *Proc. 10th International Conference on Theory and Applications of Satisfiability Testing*, volume 4501 of *LNCS*, pages 340–354, 2007.
7. Alfons Geser, Dieter Hofbauer, Johannes Waldmann, and Hans Zantema. On tree automata that certify termination of left-linear term rewriting systems. *Information and Computation*, 205(4):512–534, 2007.
8. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proc. 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *LNAI*, pages 281–286, 2006.
9. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and improving dependency pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
10. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *Proc. 15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *LNCS*, pages 249–268, 2004.
11. N. Hirokawa and A. Middeldorp. Polynomial interpretations with negative coefficients. In *Proc. 7th International Conference on Artificial Intelligence and Symbolic Mathematical Computation*, volume 3249 of *LNAI*, pages 185–198, 2004.
12. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.
13. N. Hirokawa and A. Middeldorp. Tyrolean termination tool: Techniques and features. *Information and Computation*, 205(4):474–511, 2007.
14. H. Hong and D. Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21(1):23–38, 1998.
15. A. Koprowski and J. Waldmann. Arctic termination ... below zero. In *Proc. 19th International Conference on Rewriting Techniques and Applications*, LNCS, 2008. To appear.
16. M. Korp and A. Middeldorp. Proving termination of rewrite systems using bounds. In *Proc. 18th International Conference on Rewriting Techniques and Applications*, volume 4533 of *LNCS*, pages 273–287, 2007.
17. M. Kurihara and H. Kondo. Efficient BDD encodings for partial order constraints with application to expert systems in software verification. In *Proc. 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, volume 3029 of *LNAI*, pages 827–837, 2004.
18. D. Lankford. On proving term rewrite systems are noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
19. S. Lucas. On the relative power of polynomials with real, rational, and integer coefficients in proofs of termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 17(1):49–73, 2006.
20. J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proc. 15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *LNCS*, pages 85–94, 2004.
21. H. Zantema. Reducing right-hand sides for termination. In *Processes, Terms and Cycles: Steps on the Road to Infinity, Essays Dedicated to Jan Willem Klop, on the Occasion of His 60th Birthday*, volume 3838 of *LNCS*, pages 173–197, 2005.
22. H. Zantema and J. Waldmann. Termination by quasi-periodic interpretations. In *Proc. 18th International Conference on Rewriting Techniques and Applications*, volume 4533 of *LNCS*, pages 404–418, 2007.