Ann Math Artif Intell manuscript No.
(will be inserted by the editor)

Increasing Interpretations

Harald Zankl · Aart Middeldorp

Received: date / Accepted: date

Abstract The article 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. One 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 interpretations. Furthermore, this generalization perfectly fits the dependency pair framework which is implemented in virtually every termination prover dealing with term rewrite systems. We present two dependency pair processors for increasing interpretations. The novelty of the second one is that it can be used to eliminate single edges from the dependency graph.

Keywords Term rewriting · Termination · Polynomial interpretations

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 [20] were the first to encode a termination method in propositional logic. In 2006 for the first time

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

H. Zankl · A. Middeldorp
Institute of Computer Science, University of Innsbruck, Austria
E-mail: Harald.Zankl@uibk.ac.at, Aart.Middeldorp@uibk.ac.at

[1] `http://termination-portal.org/wiki/Termination_Competition`

termination analyzers incorporated translations to SAT (Jambox [4] and Matchbox [25]) 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 article satisfies both demands, (a) it is modular and local in the sense that it can be formulated as a DP processor and (b) it allows an efficient implementation using SAT solving.

The article is organized as follows. In Section 2 the necessary definitions for graph reasoning and polynomial interpretations are given. Afterwards, the dependency pair framework is introduced in Section 3. Section 4 motivates our approach by means of an example and already suggests that special care is needed to formulate a sound processor. Afterwards in Section 5 two dependency pair processors are introduced. Implementation details and optimizations are presented in Section 6. An assessment of our contribution can be found in Section 7 before ideas for future work are addressed in Section 8.

Parts of this article appeared in an earlier conference paper [26]. Here we integrate our results directly in the dependency pair framework. Furthermore, the second DP processor (Section 5) and Sections 6.2 and 7.1 are new.

2 Preliminaries

We assume familiarity with term rewriting [2] in general and termination [27] in particular. In this section we fix definitions and notions for graphs and polynomial interpretations.

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 $\ell\colon N \to \mathbb{L}$ that assigns to every node a label.[2] 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 $1 \leqslant i < m$. In the sequel we only consider non-empty paths, i.e., $m > 1$. A path is called *elementary* if all its nodes are distinct. The *cost* of a path $[n_1, \ldots, n_{m-1}, n_m]$ is $\ell(n_1) + \cdots + \ell(n_{m-1})$ and its *length* is $m - 1$. The *distance* between two nodes a and b is the maximal cost of an elementary path from a to b and denoted by $\mathsf{d}(a, b)$. A path $[n_1, \ldots, n_m]$ is called *cyclic* if $n_1 = n_m$. A *cycle* $[n_1, \ldots, n_m]$ is a cyclic path where $(n_i, n_{i+1}) \neq (n_j, n_{j+1})$ for all $0 < i < j < m$. 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 cost carries over naturally from paths to cycles. A cycle is called *decreasing* (*increasing*) if its cost is less (greater) than zero. Furthermore we define the *distance* $\mathsf{d}(n)$ for a single node n as the maximal cost 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 (not necessarily distinct) other node. Maximality means that the property of being an SCC is lost if a further node is added. For aesthetic reasons, labels of nodes are associated to edges in graphical representations of graphs throughout the article, where edges (n, m) are labeled with $\ell(n)$.

[2] In our setting we deal not only with concrete labels (e.g. $\mathbb{N}$, $\mathbb{Z}$, or $\mathbb{Q}$) but also with abstract labels (e.g. propositional formulas representing numbers in binary) that are closed under certain operations (addition, subtraction, maximum) and allow comparison. Further details are discussed in Section 6.

$$\begin{tikzcd}
1 \arrow[r, "+1"] \arrow[d, bend right, "+1"'] & 2 \arrow[d, bend right, "0"'] \\
4 \arrow[u, bend right, "0"'] & 3 \arrow[l, "-1"] \arrow[u, bend right, "-1"']
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
1 \arrow[r, bend left, "+1"] & 2 \arrow[l, bend left, "-2"] \arrow[r, bend left, "-2"] & 3 \arrow[l, bend left, "+1"]
\end{tikzcd}$$

(b)

Fig. 1 Labeled graphs.

Example 1 In the labeled graph of Figure 1(a), $p_1 = [1,2,3,4,1]$ 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 $\text{cost}(p_1) = 0$ and $\text{cost}(p_2) = 2$. The distance of node 1 is 1 since it is the maximal cost of the elementary cycles $[1,4,1]$ and $[1,2,3,4,1]$. Note that in Figure 1(b) $[1,2,3,2,1]$ is a cycle but not an elementary one.

2.2 Polynomial Interpretations

For a signature $\mathcal{F}$ a polynomial interpretation $\mathcal{I}$ [21] maps each n-ary function symbol $f \in \mathcal{F}$ to a polynomial $f_{\mathcal{I}}$ over a carrier A in n indeterminates. (Here A will be $\mathbb{N}$.) 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 from the carrier. The comparison $s \geqslant_{\mathcal{I}} t$ is similarly defined. For *non-linear* 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 [17]. 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 exist generalizations of polynomial interpretations, e.g., to rational [22, 8] and real coefficients [22] or to negative constants as well as coefficients [14]. Other approaches allow less restrictive kinds of polynomials including max [7]. Furthermore matrix [5], quasi-periodic [29], and arctic [18] 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 1) refinement is employed.

3 Dependency Pair Framework

In this section we sketch a simplified version of the dependency pair framework [1, 11, 13, 16, 24] which we specialize to fit our requirements. Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. As usual TRSs are assumed to be finite. The signature $\mathcal{F}$ is extended with *dependency pair symbols* $f^\sharp$ for every symbol $f \in \{\text{root}(l) \mid l \to r \in \mathcal{R}\}$, where $f^\sharp$ has the same arity as f, resulting in the signature $\mathcal{F}^\sharp$.[3] In examples one usually 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 a defined root symbol that is not a proper subterm of l then the rule $l^\sharp \to t^\sharp$ is a *dependency pair* of $\mathcal{R}$. Here $l^\sharp$ and $t^\sharp$ are the result of replacing the root symbols in l and t by the corresponding dependency pair symbols. The set of dependency pairs of $\mathcal{R}$ is denoted by $\mathsf{DP}(\mathcal{R})$. A *DP problem* is a pair of TRSs $(\mathcal{P},\mathcal{R})$ such that the root symbols of the rules in $\mathcal{P}$ do neither occur in $\mathcal{R}$ nor in proper subterms of the left- and right-hand sides of rules in $\mathcal{P}$. The problem is said to be *finite* if there is no infinite sequence $s_1 \to_{\mathcal{P}} t_1 \to^*_{\mathcal{R}} s_2 \to_{\mathcal{P}} t_2 \to^*_{\mathcal{R}} \cdots$ such that all terms $t_1, t_2, \ldots$ are terminating with respect to $\mathcal{R}$. Such an infinite sequence is said to be *minimal*. The main result underlying the dependency pair approach states that termination of a TRS $\mathcal{R}$ is equivalent to finiteness of the DP problem $(\mathsf{DP}(\mathcal{R}),\mathcal{R})$.

In order to prove a DP problem finite, a number of *DP processors* have been developed. DP processors are functions that take a DP problem as input and return a set of DP problems as output. In order to be employed to prove termination they need to be *sound*, that is, if all DP problems in a set returned by a DP processor are finite then the initial DP problem is finite. In addition, to ensure that a DP processor can be used to prove non-termination it must be *complete* which means that if one of the DP problems returned by the DP processor is not finite then the original DP problem is not finite.

One important DP processor is the dependency graph. In general it is not computable but sound approximations exist [1,12,15,23]. 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.

Definition 1 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 $t\sigma \to^*_{\mathcal{R}} u\tau$.

To make use of this definition a corresponding DP processor is formulated.

Theorem 1 *The following processor is sound and complete. For a DP problem $(\mathcal{P},\mathcal{R})$ the processor returns $\{(\mathcal{P}_1,\mathcal{R}),\ldots,(\mathcal{P}_n,\mathcal{R})\}$ where $\mathcal{P}_1,\ldots,\mathcal{P}_n$ are the SCCs of $\mathsf{DG}(\mathcal{R})$.* □

Next, the notion of a reduction pair is defined.

Definition 2 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.

Reduction pairs give rise to DP processors.

Theorem 2 *Let $(\gtrsim,>)$ be a reduction pair. The processor that maps a DP problem $(\mathcal{P},\mathcal{R})$ to*

- $\{(\mathcal{P}\setminus>,\mathcal{R})\}$ *if* $\mathcal{P} \subseteq \gtrsim \cup >$ *and* $\mathcal{R} \subseteq \gtrsim$
- $\{(\mathcal{P},\mathcal{R})\}$ *otherwise*

is sound and complete. □

Due to the following theorem natural polynomials can be used to get valid reduction pairs.

[3] Function symbols that appear as a root of a left-hand side are called *defined*.

Theorem 3 *Let $\mathcal{I}$ be a weakly monotone (polynomial) interpretation over a well-founded carrier. Then $(\geqslant_{\mathcal{I}}, >_{\mathcal{I}})$ is a reduction pair.* □

Corollary 1 *Let $\mathcal{I}$ be a weakly monotone (polynomial) interpretation. The processor that maps a DP problem $(\mathcal{P},\mathcal{R})$ to*

- $\{(\mathcal{P} \setminus >_{\mathcal{I}}, \mathcal{R})\}$ *if* $\mathcal{P} \subseteq \geqslant_{\mathcal{I}} \cup >_{\mathcal{I}}$ *and* $\mathcal{R} \subseteq \geqslant_{\mathcal{I}}$
- $\{(\mathcal{P},\mathcal{R})\}$ *otherwise*

is sound and complete. □

4 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 2 Consider the TRS $\mathcal{R}$ 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:

$$\begin{tikzcd} 7 & 4 \arrow[l] & 6 \arrow[l] \arrow[r, bend left] & 5 \arrow[l, bend left] \end{tikzcd}$$

According to the processor of Theorem 1 it suffices to consider the DP problem $(\mathcal{P},\mathcal{R})$ with $\mathcal{P} = \{5,6\}$. To make further progress we have to find a reduction pair $(\gtrsim, >)$ such that all rules in $\mathcal{P} \cup \mathcal{R}$ decrease weakly and at least one rule in $\mathcal{P}$ decreases strictly. In the sequel we will show that the current DP problem $(\mathcal{P},\mathcal{R})$ 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 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 DP problem mentioned above results in the constraints

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

Table 1 Rules with increasing interpretations.

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

where at least one inequality is strict. By simple mathematics the inequalities 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 inequality 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

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

which orients almost all rules of interest correctly as can be seen in Table 1. Here, rule 5 is not correctly oriented. 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

$$\begin{tikzcd} 7 & 4 \arrow[l, "0"'] & 6 \arrow[l, "-2"'] \arrow[r, bend left, "-2"] & 5 \arrow[l, bend left, "+1"] \end{tikzcd}$$

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

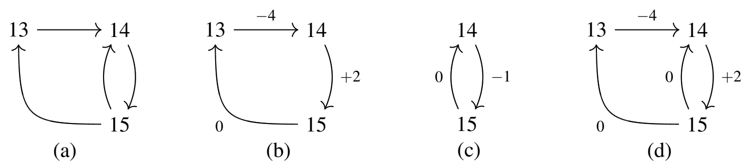


Fig. 2 Different parts of (labeled) dependency graphs.

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 (because the interpretation of every term is assumed to be non-negative).

Before moving to more challenging dependency graphs we stress that refinements such as usable rules or real coefficients do not allow linear polynomials to prove this system terminating.

4.1 From Cycles to SCCs

The above idea naturally extends from plain cycles to SCCs as described below. Consequently this allows to formulate a DP processor where DP problems of arbitrary shape can be handled. 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

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

and the only SCC

$$\mathsf{F}(\mathsf{0},\mathsf{0},x,\mathsf{g}(\mathsf{g}(\mathsf{g}(\mathsf{g}(y))))) \to \mathsf{F}(\mathsf{0},\mathsf{1},\mathsf{g}(\mathsf{g}(x)),y) \tag{13}$$

$$\mathsf{F}(\mathsf{0},\mathsf{1},\mathsf{g}(x),y) \to \mathsf{F}(\mathsf{1},\mathsf{1},x,\mathsf{g}(\mathsf{g}(y))) \tag{14}$$

$$\mathsf{F}(\mathsf{1},\mathsf{1},x,y) \to \mathsf{F}(\mathsf{0},x,x,y) \tag{15}$$

The corresponding SCC of the dependency graph depicted in Figure 2(a) contains the two cycles $[13,14,15,13]$ and $[14,15,14]$. The first one is handled by the increasing interpretation

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

For the second one we take the interpretation as above but with $\mathsf{F}_{\mathbb{N}}(x,y,z,w) = z$. Hence for the elementary cycle $[13,14,15,13]$ the interpreted value decreases by 2 in every loop. Similarly there is a decrease of 1 for the elementary cycle $[14,15,14]$. 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

$$\begin{tikzcd}
13 \arrow[r, "-1"] & 14 \arrow[d, bend left, "+1"] \\
 & 15 \arrow[u, bend left, "-1"] \arrow[ul, bend left, "-1"]
\end{tikzcd}$$

(a) The labeled DG.

$$\begin{tikzcd}
14 \arrow[d, bend left] \\
15 \arrow[u, bend left]
\end{tikzcd}$$

(b) The remaining DG.

Fig. 3 A hypothetically labeled DG.

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

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 (non-increasing) setting it suffices to consider the two cycles separately. This is the case because a strict decrease in every single cycle ensures a strict decrease in larger cyclic paths 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 path $[13,14,15,14,15,13]$ where the interpreted value is increased in the elementary cycle $[14,15,14]$ and consequently the cost of $[13,14,15,14,15,13]$ is zero and there is no decrease. Considering (infinitely many!) possibly non-elementary cyclic paths is undoable. But if a graph contains no increasing cycles then the above scenario is not possible. To recognize dangerous cyclic paths, it suffices to compute the distance for every node. For the graph in Figure 2(d) we have $\mathsf{d}(13) = -2$, $\mathsf{d}(14) = 2$, and $\mathsf{d}(15) = 2$. Only if for every node the distance is smaller than or equal to zero we know that problematic paths as demonstrated above cannot occur. Furthermore we know that in such a case we can delete nodes with negative distance because on every possible cyclic path containing such a node 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 terminating) then deleting node 13 would have been possible since $\mathsf{d}(13) = -1$, $\mathsf{d}(14) = 0$, and $\mathsf{d}(15) = 0$. In such a situation one could proceed with the simpler graph depicted in Figure 3(b) with a possibly totally different interpretation.

5 Two DP Processors

The example in the preceding section suggests that DP problems that consist of more than just one cycle need special attention. 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 $s \to t$, the change of the interpreted value is $[t]_{\mathcal{I}} - [s]_{\mathcal{I}}$. The idea is to label every node by the constant part of that difference.

Definition 3 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 write $\mathsf{ncp}_{\mathcal{I}}(t)$ as $\mathsf{ncp}([t]_{\mathcal{I}})$. This notation naturally extends to rules and TRSs, e.g., $\mathsf{ncp}_{\mathcal{I}}(s \to t) = \mathsf{ncp}_{\mathcal{I}}(s) \to \mathsf{ncp}_{\mathcal{I}}(t)$ and $\mathsf{ncp}_{\mathcal{I}}(\mathcal{R}) = \{\mathsf{ncp}_{\mathcal{I}}(s \to t) \mid s \to t \in \mathcal{R}\}$. The same notation is freely used for $\mathsf{cp}_{\mathcal{I}}$.

Definition 4 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 $\ell(s \to t) = \mathsf{cp}([t]_\mathcal{I} - [s]_\mathcal{I})$ for every node $s \to t$ in DG. By $\mathsf{d}_\mathcal{I}(n)$ we denote the distance of a node n in $\mathsf{DG}_\mathcal{I}$.

The above definition does not explicitly mention the set of labels $\mathbb{L}$. This is not necessary since closing A under subtraction yields $\mathbb{L}$. The next definition presents the first DP processor in the setting of increasing interpretations.

Definition 5 Let $(\mathcal{P}, \mathcal{R})$ be a DP problem, $\mathcal{G}$ the corresponding (part of the) dependency graph with nodes $\mathcal{P}$, and $\mathcal{I}$ an interpretation. We define the DP processor $\mathit{Proc}_\mathcal{I}$ as follows: $\mathit{Proc}_\mathcal{I}(\mathcal{P}, \mathcal{R})$ returns

- $\{(\mathcal{P} \setminus \{p \in \mathcal{P} \mid \mathsf{d}_\mathcal{I}(p) < 0\}, \mathcal{R})\}$
  if $(\geqslant_\mathcal{I}, >_\mathcal{I})$ is a reduction pair, $\mathcal{R} \subseteq \geqslant_\mathcal{I}$, $\mathsf{ncp}_\mathcal{I}(\mathcal{P}) \subseteq \geqslant$, and $\mathsf{d}_\mathcal{I}(\mathcal{P}) \subseteq \mathbb{L}^{\leqslant 0}$
- $\{(\mathcal{P}, \mathcal{R})\}$ otherwise.

Here $\mathbb{L}^{\leqslant 0} = \{n \in \mathbb{L} \mid n \leqslant 0\}$.

The above processor formulates that under the condition that no increasing cycles exist ($\mathsf{d}_\mathcal{I}(\mathcal{P}) \subseteq \mathbb{L}^{\leqslant 0}$), nodes that are only part of decreasing cycles ($\mathsf{d}_\mathcal{I}(p) < 0$) can be removed. Before we present the (first) main theorem we show that increasing interpretations subsume the conventional non-increasing case. Together with the discussion in Section 4 this shows that increasing interpretations are really more powerful.

Lemma 1 *The processor of Definition 5 subsumes the processor of Corollary 1.*

Proof Assume that the processor of Corollary 1 applies. Hence there exists a reduction pair $(\geqslant_\mathcal{I}, >_\mathcal{I})$ such that

$$\mathcal{R} \subseteq \geqslant_\mathcal{I} \tag{16}$$

and

$$\mathcal{P} \subseteq \geqslant_\mathcal{I} \cup >_\mathcal{I} \tag{17}$$

We have to show that under the assumption of (16) and (17) also

$$\mathcal{R} \subseteq \geqslant_\mathcal{I} \tag{18}$$
$$\mathsf{ncp}_\mathcal{I}(\mathcal{P}) \subseteq \geqslant \tag{19}$$

and

$$\mathsf{d}_\mathcal{I}(\mathcal{P}) \subseteq \mathbb{L}^{\leqslant 0} \tag{20}$$

Constraint (16) is identical to (18). That (17) implies (19) is trivial. For the other implication note that from (17) the property $\ell(\mathcal{P}) \subseteq \mathbb{L}^{\leqslant 0}$ follows which proves (20) (cf. Definition 4). Furthermore $s >_\mathcal{I} t$ implies $\mathsf{cp}_\mathcal{I}(t) > \mathsf{cp}_\mathcal{I}(s)$ which gives $\mathsf{d}_\mathcal{I}(s \to t) < 0$. This ensures that the processor of Definition 5 deletes the same nodes as the processor of Theorem 2. ◻

Theorem 4 *The DP processor $\mathit{Proc}_\mathcal{I}$ is sound and complete.*

Proof To shorten notation we abbreviate $\mathcal{P} \setminus \{p \in \mathcal{P} \mid \mathsf{d}_{\mathcal{I}}(\mathcal{P}) < 0\}$ by $\mathcal{P}^0$ and $\mathcal{P} \setminus \mathcal{P}^0$ by $\mathcal{P}^{<0}$. First we show soundness. Suppose the DP problem $(\mathcal{P}^0, \mathcal{R})$ is finite. We have to show that $(\mathcal{P}, \mathcal{R})$ is finite. Suppose to the contrary that $(\mathcal{P}, \mathcal{R})$ is not finite. So there exists a minimal rewrite sequence

$$s_0 \to_{\mathcal{P}} t_0 \to_{\mathcal{R}}^* s_1 \to_{\mathcal{P}} t_1 \to_{\mathcal{R}}^* \cdots \tag{21}$$

We consider two cases:

– A rule $s \to t \in \mathcal{P}^{<0}$ is applied infinitely often in the sequence (21). Then one can extract a sequence
$$s_0' \to_{s\to t} t_0' \to_{\mathcal{P}\cup\mathcal{R}}^* s_1' \to_{s\to t} t_1' \to_{\mathcal{P}\cup\mathcal{R}}^* \cdots$$
Since $\geqslant_{\mathcal{I}}$ is closed under contexts and substitutions, for all terms u, v, and all rules $l \to r \in \mathcal{R} \cup \mathcal{P}$ 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 $\mathsf{d}_{\mathcal{I}}(\mathcal{P}) \subseteq \mathbb{L}^{\leqslant 0}$ ensures that every cyclic path 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
$$\mathsf{cp}_{\mathcal{I}}(s_0') > \mathsf{cp}_{\mathcal{I}}(s_1') > \mathsf{cp}_{\mathcal{I}}(s_2') > \mathsf{cp}_{\mathcal{I}}(s_3') > \cdots$$
which contradicts the well-foundedness of $>$. Consequently there cannot be an infinite sequence (21) contradicting the assumption that $(\mathcal{P}, \mathcal{R})$ is not finite.
– No rule $s \to t \in \mathcal{P}^{<0}$ is applied infinitely often in (21). Then there exists a tail of (21) such that all $\mathcal{P}$-steps are from $\mathcal{P}^0$. By assumption the sequence (21) is minimal. Hence the tail is minimal as well and thus the DP problem $(\mathcal{P}^0, \mathcal{R})$ is not finite.

Completeness says that if $(\mathcal{P}^0, \mathcal{R})$ is not finite then $(\mathcal{P}, \mathcal{R})$ is not finite. This is trivial since any minimal
$$s_0 \to_{\mathcal{P}^0} t_0 \to_{\mathcal{R}}^* s_1 \to_{\mathcal{P}^0} t_1 \to_{\mathcal{R}}^* \cdots$$
sequence is at the same time a minimal
$$s_0 \to_{\mathcal{P}} t_0 \to_{\mathcal{R}}^* s_1 \to_{\mathcal{P}} t_1 \to_{\mathcal{R}}^* \cdots$$
sequence (because clearly $\mathcal{P}^0 \subseteq \mathcal{P}$). □

Next we show that increasing interpretations can be used to delete single edges in the dependency graph, resulting in a finer DP processor than the one of Definition 5. Similar as in [24] the dependency graph is added to DP problems—resulting in the notion of *extended* DP problems. An extended DP problem is a triple $(\mathcal{P}, \mathcal{R}, \mathcal{G})$. So in addition to ordinary DP problems now also some processing on the third component (the dependency graph) is possible. Ordinary DP processors remain sound if this third component is added. Furthermore the initial DP problem is now set to $(\mathsf{DP}(\mathcal{R}), \mathcal{R}, \mathsf{DG}(\mathcal{R}))$ and for an infinite minimal sequence additionally it is demanded that consecutive $\mathcal{P}$ steps correspond to an edge in $\mathcal{G}$. It makes sense to redefine the processor of Theorem 1 such that for the (extended) DP problem $(\mathcal{P}, \mathcal{R}, \mathcal{G})$ it returns the set of extended DP problems $\{(\mathcal{P}_i, \mathcal{R}, \mathcal{G}_i)\}$ such that $\mathcal{P}_i$ are the SCCs of $\mathcal{G}$ and $\mathcal{G}_i$ is the restriction of $\mathcal{G}$ to $\mathcal{P}_i$.

The following example motivates why deleting edges can be advantageous.

Example 3 In the labeled graph

$$\begin{tikzcd}
\cdot \arrow[loop left, "-1"] \arrow[r, bend left, "-1"] & \cdot \arrow[l, bend left, "+1"]
\end{tikzcd}$$

all preconditions for applying the processor $Proc_{\mathcal{I}}$ are satisfied. But it cannot make progress since for all nodes $n \in \mathcal{P}$ the property $\mathsf{d}(n) = 0$ holds. Nevertheless it is obvious that there cannot be an infinite run that uses the leftmost edge infinitely often.

Next the improved processor is presented that allows removing edges from the dependency graph. For this processor (and the discussion in Section 6.2) labels are associated to edges instead of nodes. A node-labeling ℓ_n is transformed into an edge-labeling ℓ_e as follows: $\ell_e(a,b) = \ell_n(a)$ if $(a,b) \in E$. Concepts as cost, distance, etc. carry over naturally from node-labeled to edge-labeled graphs.

Definition 6 Let $(\mathcal{P},\mathcal{R},\mathcal{G})$ be an extended DP problem such that $\mathcal{G} = (\mathcal{P},\mathcal{E})$,[4] $\mathcal{I}$ is an interpretation, and $\mathcal{G}_{\mathcal{I}} = (\mathcal{G},\ell)$ the labeled variant of $\mathcal{G}$. We define the DP processor $Proc'_{\mathcal{I}}$ as follows:

– $\{(\mathcal{P},\mathcal{R},(\mathcal{P},\mathcal{E} \setminus \{(a,b) \in \mathcal{E} \mid \ell_{\mathcal{I}}(a,b) + \mathsf{d}_{\mathcal{I}}(b,a) < 0\}))\}$
  if $(\geqslant_{\mathcal{I}}, >_{\mathcal{I}})$ is a reduction pair, $\mathcal{R} \subseteq \geqslant_{\mathcal{I}}$, $\mathsf{ncp}_{\mathcal{I}}(\mathcal{P}) \subseteq \geqslant$, and $\mathsf{d}_{\mathcal{I}}(\mathcal{P}) \subseteq \mathbb{L}^{\leqslant 0}$
– $\{(\mathcal{P},\mathcal{R},\mathcal{G})\}$ otherwise.

The idea of the processor above is that edges that appear on decreasing cycles only ($\ell(a,b) + \mathsf{d}(b,a) < 0$) do not give rise to infinite reductions.

Theorem 5 *The processor $Proc'_{\mathcal{I}}$ is sound and complete.*

Proof The proof is quite similar to the one of Theorem 4. For soundness we assume a minimal infinite sequence that applies an edge $(s \to t, u \to v)$ infinitely often. This means that the minimal sequence applies rule $u \to v$ after $s \to t$ infinitely often (with possible $\mathcal{R}$-steps in between but without any $\mathcal{P}$-steps), i.e.,

$$s_0 \to_{s\to t} t_0 \to^*_{\mathcal{R}} s_1 \to_{u\to v} t_1 \to^*_{\mathcal{P}\cup\mathcal{R}} s_2 \to_{s\to t} t_2 \to^*_{\mathcal{R}} s_3 \to_{u\to v} t_3 \to^*_{\mathcal{P}\cup\mathcal{R}} s_4 \to_{s\to t} \cdots$$

But then the sequence

$$\mathsf{cp}_{\mathcal{I}}(s_0) > \mathsf{cp}_{\mathcal{I}}(s_2) > \mathsf{cp}_{\mathcal{I}}(s_4) > \cdots$$

gives the desired contradiction. This sequence exists because of the condition $\mathsf{d}_{\mathcal{I}}(\mathcal{P}) \subseteq \mathbb{L}^{\leqslant 0}$ and the assumption that the edge $(s \to t, u \to v)$ appears on decreasing cycles only (since $\ell_{\mathcal{I}}(s \to t, u \to v) + \mathsf{d}_{\mathcal{I}}(u \to v, s \to t) < 0$). Completeness is trivial. $\square$

[4] Demanding that the nodes of $\mathcal{G}$ equal $\mathcal{P}$ is no restriction since one can always apply the (extended) dependency graph processor beforehand.

6 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 recent 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 numbers at hand but some propositional formulas which abstractly encode the range of all possible values. Thus existing algorithms from graph theory cannot be employed because they require concrete numbers. Since encoding polynomials in SAT has already been described in detail [6], in this article we refrain from giving all implementation issues. The only part of the encoding which is discussed here deals with graphs that are labeled by abstract numbers. This includes computing the distance $\mathsf{d}(n)$ for a node n and expressing if an edge (a,b) occurs on decreasing cycles only.

6.1 Computing the Distance of a Node

The idea is to compute the distance of a node by means of a transitivity closure. The numerical 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 reached kind of a fixed point, i.e., $R_{abk'} = -\infty$ if and only if $R_{abk} = -\infty$ 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 $-\infty$, the operations addition and maximum are extended naturally, i.e., $n + -\infty = -\infty + n = -\infty$ and $\max(n, -\infty) = \max(-\infty, n) = n$ for all $n \in \mathbb{L} \cup \{-\infty\}$. For $0 \leqslant i < k$ we define

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

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[5] 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, i.e., a path of maximal cost. 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 some cyclic path is considered). Nevertheless the idea remains sound because of the condition that for all nodes $R_{nnk} \leqslant 0$ there are no increasing cycles. In such a case the problem sketched here does not occur. For a demonstration consider the following example.

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

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

$\mathsf{d}(1) = 1$	$\mathsf{d}(2) = 0$	$\mathsf{d}(3) = 0$	$\mathsf{d}(4) = 1$
$R_{112} = 2$	$R_{222} = 0$	$R_{332} = 0$	$R_{442} = 2$

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}.

But nevertheless one can formulate the following two lemmata which allow to implement the DP processor of Definition 5.

Lemma 2 *Let $((N,E),\ell)$ be a labeled graph and $k = \lceil \log_2(|N|) \rceil$. If for all nodes $n \in N$ the property $R_{nnk} \leqslant 0$ holds then for all nodes $n \in N$ the property $\mathsf{d}(n) \leqslant 0$ holds.*

Proof Assume for a contradiction that there is a node $n \in N$ with $\mathsf{d}(n) > 0$. Hence there exists an elementary cycle that is increasing and contains n. But then clearly this cycle gives $R_{nnk} > 0$. □

Lemma 3 *Let $((N,E),\ell)$ be a labeled graph, $k = \lceil \log_2(|N|) \rceil$, and for all $n \in N$ $\mathsf{d}(n) \leqslant 0$. Then $\mathsf{d}(a,b) = R_{abk}$ for all distinct nodes $a,b \in N$ and $\mathsf{d}(a) = R_{aak}$ for all $a \in N$.*

Proof We show the first equality, the second one is proved analogously. The distance from a to b is the maximal cost of an elementary path. Clearly by construction $R_{abk} \geqslant \mathsf{d}(a,b)$ since every elementary path from a to b is covered by an R_{abk} variable. By assumption there are no increasing cycles and hence non-elementary paths from a to b cannot have a larger cost than an elementary one. Hence R_{abk} represents the maximal cost of an elementary path from a to b. □

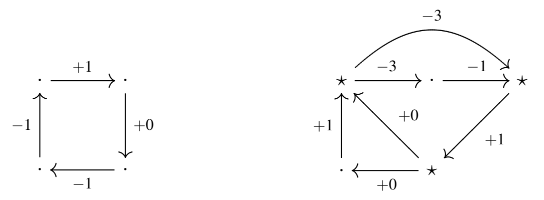
Both DP processors can be implemented with the help of the R_{abi} variables. For the processor from Definition 5 the formula (in the sequel referred to as $\mathsf{direct\_n}$)

$$\bigwedge_{n \in N} R_{nnk} \leqslant 0 \ \wedge \ \bigvee_{n \in N} R_{nnk} < 0$$

is employed whereas Definition 6 amounts to the formula ($\mathsf{direct\_e}$)

$$\bigwedge_{n \in N} R_{nnk} \leqslant 0 \ \wedge \ \bigvee_{(a,b) \in E} \ell(a,b) + R_{bak} < 0$$

Then from a satisfying assignment the nodes with negative distance or edges that are just part of decreasing cycles can easily be determined and removed. Clearly satisfiability of the first formula implies satisfiability of the second formula. The reason why both are presented is that the first one produces a smaller encoding. That this can be advantageous is demonstrated by experimental data in Section 7.



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

Fig. 4 Two special shapes of SCCs.

6.1.1 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 graph. To get a faster implementation we can optimize the encoding for SCCs that have a special shape:

(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 decreases when cycling. The encoding is linear in the size of the nodes.

(b) Almost simple SCCs: An SCC is called *almost simple* if 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 elementary 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 $-(\ell(n_1) + \cdots + \ell(n_p)) > \ell(m_1) + \cdots + \ell(m_q)$ holds where $n_1, \ldots, n_p$ are the selected nodes and $m_1, \ldots, m_q$ are the (non-selected) nodes in the SCC that have a positive label. The underlying idea is that the selected nodes n_i decrease the interpretation more than all other nodes together might increase it. In Figure 4(b) the inequality amounts to $-(-3+0+1) > 1$ which is satisfiable since $2 > 1$. The encoding again is linear in the size of the nodes. Furthermore it specializes exactly to the one in (a) for simple SCCs.

Note that the specialization for case (a) is exact whereas (b) is an approximation, i.e., there are labeled graphs where a node with negative distance exists but is not found. E.g., the graph in Figure 1(b) produces the constraint $-(-2) > 1+1$ which is not satisfiable although node 2 has negative distance.

6.2 Compressing Graphs

In this section a solution is presented to implement (a strong heuristic for) the processor of Definition 6. Here the idea is to process the graph. In an iterative manner nodes are removed from the graph one by one. While removing nodes, information about the cycles in the

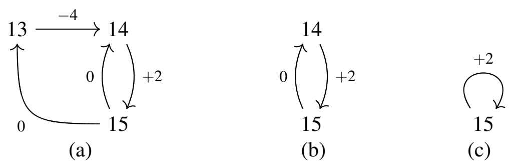


Fig. 5 Compressing labeled graphs.

graph is obtained. This information allows to formulate a propositional formula such that satisfiability of this formula implies the existence of a decreasing cycle. At the end of this section we discuss how this approach allows to implement the DP processors of the previous section.

For a simpler presentation of the algorithm, labels of non-existing edges are assumed to be $-\infty$. The intuition behind Definition 7 is as follows. In step 1 cycles of length one are removed from the graph. Then in steps 2 and 3 a graph (N',E') is built that contains all edges of the graph from step 1, except that the chosen node n is bypassed. The bypass edges are exactly the ones in $i(n) \times o(n)$ where $i(n) = \{m \in N \mid (m,n) \in E\}$ and $o(n) = \{p \in N \mid (n,p) \in E\}$. Bypassing nodes might produce a multi-graph. Reasoning about multi-graphs is prevented by altering the labeling function ℓ which chooses the maximal edge between two nodes in step 4. Note that if $a \notin i(n)$ or $b \notin o(n)$ then $\ell'(a,b) = \ell''(a,b)$.

Definition 7 (Compression Algorithm) Given a labeled graph $((N,E),\ell)$ the procedure works as follows: Set $C = \varnothing$.

1. Set $C = C \cup \{\ell(a,a) \mid \ell(a,a) \neq -\infty\}$, $E'' = E \setminus \{(a,a) \mid a \in N\}$, and
$$\ell''(a,b) = \begin{cases} \ell(a,b) & \text{if } a \neq b, \\ -\infty & \text{otherwise.} \end{cases}$$
2. If $|N| = 1$ then return C, otherwise choose a node $n \in N$.
3. Set $N' = N \setminus \{n\}$ and $E' = (E'' \cap N' \times N') \cup i(n) \times o(n)$.
4. Set $\ell'(a,b) = \max\{\ell''(a,b), \ell''(a,n) + \ell''(n,b)\}$.
5. Repeat step 1 with $((N',E'),\ell')$.

The following example performs the compression algorithm on a concrete graph.

Example 5 Consider the graph depicted in Figure 5(a). Step 1 is easy since there are no cycles of length one. In step 2 we select node 13. Clearly $i(13) = \{15\}$ and $o(13) = \{14\}$. Hence in step 3 we obtain $N' = \{14,15\}$ and $E' = \{(14,15),(15,14)\}$. In step 4 the labeling ℓ'' looks like $\ell''(14,15) = +2$ and $\ell''(15,14) = \max\{0, 0-4\} = 0$ and all other labels are $-\infty$. The result of the algorithm after one iteration is shown in Figure 5(b). In the second iteration there are no changes in step 1. Now choose $n = 14$. This results in a graph with only one remaining node 15 and $\ell''(15,15) = +2$, shown in Figure 5(c). In the next iteration the algorithm sets $C = \{+2\}$, $\ell''(15,15) = -\infty$, and terminates in step 2.

The property that all cycles are non-increasing is fulfilled if for all values $c \in C$ we have $c \leqslant 0$ which is not the case in the example above. Here Definition 7 was applied to a labeled graph with concrete labels. In the setting of increasing interpretations the algorithm is applied to a labeled dependency graph (with abstract labels) and thus returns a set of abstract numbers. To demand that there is no increase in any elementary cycle the expression $\bigwedge_{c\in C} c \leqslant 0$ (correctness) is employed. Furthermore progress (that there is a decreasing cycle) is achieved by $\bigvee_{c\in C} c < 0$. In contrast to the approach with the variables R_{abi} one cannot directly determine from a satisfying assignment which nodes (if any) have negative distance and thus can be removed. But from a satisfying assignment the concrete labels can be inferred. Afterwards maximal distances can easily be computed. Additionally edges (a,b) which are just contained in decreasing cycles ($\ell(a,b)+\mathsf{d}(b,a) < 0$) can be searched and removed instantaneously. This allows to implement both processors presented earlier (Definitions 5 and 6) at the same time. But first we prove the correctness of the approach.

Lemma 4 *One iteration of the algorithm in Definition 7 fulfills the property: There exists a node $x \in N$ with $\mathsf{d}(x) > 0$ if and only if there exists a node $x' \in N'$ with $\mathsf{d}(x') > 0$ or there exists a $c \in C$ with $c > 0$.*

Proof Assume there is a node $x \in N$ such that $\mathsf{d}(x) > 0$. We consider two cases:

– There is a node y with $\ell(y,y) > 0$. Then in step 1 $\ell(y,y) \in C$ and thus the claim holds.
– There is no y with $\ell(y,y) > 0$. We again consider two cases.
  – The chosen node n is on a cycle involving x with positive distance. By the assumptions such a cycle must contain at least two different nodes and since x is existentially quantified we can choose it such that $x \neq n$. Hence we may denote the cycle by $[x,\dots,a,n,b,\dots,x]$. But then also $\mathsf{d}(x) > 0$ for the cycle $[x,\dots,a,b,\dots,x]$ in N' because by construction $\ell'(a,b) \geqslant \ell''(a,n)+\ell''(n,b)$. Hence the result holds by taking $x' = x$.
  – If the chosen node n is not on a cycle involving x then we take $x' = x$ and the result trivially holds.

For the other direction again two cases are considered:

– First assume there is a node $x' \in N'$ such that $\mathsf{d}(x') > 0$. Without loss of generality we can assume that a bypass edge (a,b) is contained in the cycle $[x',\dots,a,b,\dots,x']$. By construction $\ell'(a,b) = \ell''(a,b)$ or $\ell'(a,b) = \ell''(a,n)+\ell''(n,b)$. Choosing the appropriate edge(s) gives a cycle in the original graph with $\mathsf{d}(x) > 0$ where $x = x'$.
– The other case is if there is a $c \in C$ with $c > 0$. Then in step 1 there must be a node $x \in N$ with $\ell(x,x) > 0$ ensuring $\mathsf{d}(x) > 0$. □

Corollary 2 *Let $((N,E),\ell)$ be a labeled graph. The following properties are equivalent:*

– *The distance $\mathsf{d}(x) \leqslant 0$ for all $x \in N$.*
– *The conjunction $c \leqslant 0$ for $c \in C$ is satisfiable where C is the output of the algorithm in Definition 7.* □

The above corollary states that the algorithm of Definition 7 is sound and thus can be used to implement the DP processors from the previous section. This means that whenever the conjunction of $\bigwedge_{c\in C} c \leqslant 0$ is satisfiable then $\mathsf{d}(n) \leqslant 0$ for all $n \in N$ which is demanded by both DP processors. However in order to ensure progress of the processors (removal of one node or edge) there must exist a $c \in C$ with $c < 0$. In the sequel we abbreviate the conjunction of $\bigwedge_{c\in C} c \leqslant 0$ and $\bigvee_{c\in C} c < 0$ by $\mathrm{C}^{\wedge}_{\vee}$. Note that one cannot conclude from the satisfiability of $\mathrm{C}^{\wedge}_{\vee}$ the existence of a node n with $\mathsf{d}(n) < 0$. Example 3 shows such a case. But satisfiability of $\mathrm{C}^{\wedge}_{\vee}$ ensures a decreasing cycle. Thus the removal of edges as in the processor of Definition 6 applies.

To further demonstrate the compression algorithm the labeled graph from Figure 6(a) is considered. Clearly the leftmost edge labeled -1 cannot be taken indefinitely and could

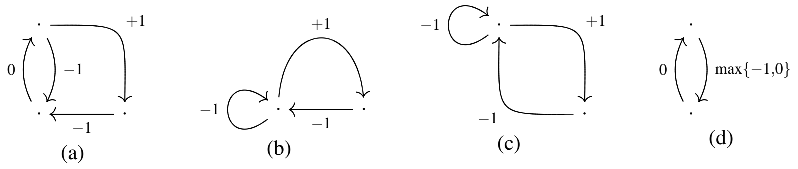


Fig. 6 Problems of the compression algorithm.

be removed since it is part of decreasing cycles only. But depending on the selection of the nodes in the compression algorithm, the formula $C_{\vee}^{\wedge}$ may be satisfiable or not. The reason is that multi-edges are removed by maximizing over the labels and consequently decreasing cycles can be overlooked. Figures 6(b), 6(c), and 6(d) show the result of the algorithm after one step if the left-top, left-bottom, and right-bottom node is chosen to be eliminated. Hence our implementation provides just a heuristic for the processor $\mathit{Proc}'_{\mathcal{I}}$. However if there exists a node with negative distance, then maximizing does not destroy satisfiability of $C_{\vee}^{\wedge}$ (since all elementary cycles containing this node are decreasing).

7 Assessment

In this article we showed that increasing interpretations are strictly more powerful than standard (non-increasing) interpretations. Clearly for DP problems $(\mathcal{P},\mathcal{R})$ where $\mathcal{P}$ is a singleton set they are of equal power.

The reason why the TRS of Example 2 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 and 2008 editions 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 [10] and T$_\mathsf{T}$T$_2$[6] use dependency pair analysis whereas Jambox [4] performs a reduction of right-hand sides [28].

It is an easy exercise to construct (larger) TRSs than Example 2 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 T$_\mathsf{T}$T$_2$ can still prove these examples by bounds [9, 19] we ensure the TRS to be not left-linear which makes increasing interpretations the only successful method.

Example 6 The TRS below resembles a binary counter. The rules behave in a way such that the counter increments by one (adding a function symbol g in f's fourth argument) seven times in a row before the value is decremented by eight.

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

[6] `http://cl-informatik.uibk.ac.at/software/ttt2/`

Table 2 Experimental results.

(a) 1381 TRSs.

method	yes	timeout	total time
direct_n	493	238	17873
direct_e	490	253	18577
a	203	12	1262
b	224	17	2333
compress	536	144	12191

(b) 724 SRSs.

method	yes	timeout	total time
direct_n	46	59	5465
direct_e	44	69	6473
a	10	0	24
b	15	0	115
compress	56	13	1909

None of the current 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. The general approaches described in Section 6.1, the specialization (b) from Section 6.1.1, and the compression approach from Section 6.2 yield the increasing interpretation

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

which ensures that all nodes have a negative distance and hence the whole problematic SCC can be removed. The only difference between the four approaches is that it takes the direct methods more than a second whereas the other approaches succeed within a fraction of a second.

In Table 2 the different approaches presented in this article are compared against each other.[7] As a test set the 1381 TRSs and 724 SRSs of the standard rewriting category from the Termination Problems Data Base[8] have been employed. All tests have been performed on a single core of a server equipped with eight dual-core AMD Opteron® processors 885 running at a clock rate of 2.6GHz and 64GB of main memory. In Table 2 direct refers to the direct approach in Section 6.1, where the suffix n (e) indicates if the formula to remove nodes (edges) is employed. Rows a and b indicate the usage of the optimizations (a) and (b) from Section 6.1.1 and compress the compression algorithm presented in Definition 7. The column labeled yes indicates the number of TRSs that could be proved terminating. If the algorithm could not determine a result within 60 seconds, the computation was stopped (column timeout). The last column presents the total time in seconds used by the specific method. In Table 2 linear polynomials with coefficients from $\{0,\ldots,7\}$ have been employed. Intermediate results are restricted to numbers less than 32. Summing up, the compression algorithm performs better than the direct approach in both measurements: time and termination proving power (due to the time limit). Especially for SRSs the compression algorithm is much faster as can be inferred from Table 2(b). The tables also show that the optimizations from Section 6.1.1 allow a fast implementation for special cases that appear surprisingly frequent.

7.1 Extension to Other Semantic Based Methods

The theory of increasing interpretations as described above directly applies to other reduction pairs based on interpretations. A careful inspection of the proofs in Section 5 reveals

[7] Detailed experiments are available from `http://colo6-c703.uibk.ac.at/ttt2/increasing/`.

[8] `http://www.lri.fr/~marche/tpdb/`

that our results extend to polynomial interpretations with rational [22, 8], and real coefficients [22] as well as negative constants [14]. Furthermore it generalizes to the matrix method [5] since there the interpretation of a dependency pair amounts to a natural number. Consequently the dependency graph is labeled in exactly the same fashion. The recent technique from [7] to allow the maximum operation can also be used in combination with the method proposed here. The reason is that the (more complex) max-polynomials only occur in intermediate steps. When the constraints are given to a SAT solver, all occurrences of max have already been removed.

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

$$\begin{tikzcd} \cdot \arrow[r, bend left, "-x-1"] & \cdot \arrow[l, bend left, "+x"] \end{tikzcd}$$

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 minimal 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 minimal 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 by instantiation and narrowing [1, 13].

To conclude, we state 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 minimal non-terminating sequence 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.

Acknowledgements 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 $\mathsf{T_TT_2}$.

References

1. Arts, T., Giesl, J.: Termination of term rewriting using dependency pairs. Theor. Comput. Sci. **236**(1-2), 133–178 (2000)
2. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)

3. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Proc. 6th International Conference on Theory and Applications of Satisfiability Testing, *LNCS*, vol. 2919, pp. 502–518 (2003)
4. Endrullis, J.: Jambox (2007). Available from `http://joerg.endrullis.de`
5. Endrullis, J., Waldmann, J., Zantema, H.: Matrix interpretations for proving termination of term rewriting. J. Autom. Reason. **40**(2-3), 195–220 (2008)
6. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: SAT solving for termination analysis with polynomial interpretations. In: Proc. 10th International Conference on Theory and Applications of Satisfiability Testing, *LNCS*, vol. 4501, pp. 340–354 (2007)
7. Fuhs, C., Giesl, J., Middeldorp, A., Schneider-Kamp, P., Thiemann, R., Zankl, H.: Maximal termination. In: Proc. 19th International Conference on Rewriting Techniques and Applications, *LNCS*, vol. 5117, pp. 110–125 (2008)
8. Fuhs, C., Navarro-Marset, R., Otto, C., Giesl, J., Lucas, S., Schneider-Kamp, P.: Search techniques for rational polynomial orders. In: Proc. 9th International Conference on Artificial Intelligence and Symbolic Computation, *LNCS (LNAI)*, vol. 5144, pp. 109–124 (2008)
9. Geser, A., Hofbauer, D., Waldmann, J., Zantema, H.: On tree automata that certify termination of left-linear term rewriting systems. Inf. Comput. **205**(4), 512–534 (2007)
10. Giesl, J., Schneider-Kamp, P., Thiemann, R.: AProVE 1.2: Automatic termination proofs in the dependency pair framework. In: Proc. 3rd International Joint Conference on Automated Reasoning, *LNCS (LNAI)*, vol. 4130, pp. 281–286 (2006)
11. Giesl, J., Thiemann, R., Schneider-Kamp, P.: The dependency pair framework: Combining techniques for automated termination proofs. In: Proc. 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning, *LNCS (LNAI)*, vol. 3425, pp. 301–331 (2004)
12. Giesl, J., Thiemann, R., Schneider-Kamp, P.: Proving and disproving termination of higher-order functions. In: Proc. 5th International Workshop on Frontiers of Combining Systems, *LNCS (LNAI)*, vol. 3717, pp. 216–231 (2005)
13. Giesl, J., Thiemann, R., Schneider-Kamp, P., Falke, S.: Mechanizing and improving dependency pairs. J. Autom. Reason. **37**(3), 155–203 (2006)
14. Hirokawa, N., Middeldorp, A.: Polynomial interpretations with negative coefficients. In: Proc. 7th International Conference on Artificial Intelligence and Symbolic Computation, *LNCS (LNAI)*, vol. 3249, pp. 185–198 (2004)
15. Hirokawa, N., Middeldorp, A.: Automating the dependency pair method. Inf. Comput. **199**(1-2), 172–199 (2005)
16. Hirokawa, N., Middeldorp, A.: Tyrolean termination tool: Techniques and features. Inf. Comput. **205**(4), 474–511 (2007)
17. Hong, H., Jakuš, D.: Testing positiveness of polynomials. J. Autom. Reason. **21**(1), 23–38 (1998)
18. Koprowski, A., Waldmann, J.: Arctic termination ... below zero. In: Proc. 19th International Conference on Rewriting Techniques and Applications, *LNCS*, vol. 5117, pp. 202–216 (2008)
19. Korp, M., Middeldorp, A.: Proving termination of rewrite systems using bounds. In: Proc. 18th International Conference on Rewriting Techniques and Applications, *LNCS*, vol. 4533, pp. 273–287 (2007)
20. Kurihara, M., Kondo, H.: 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, *LNCS (LNAI)*, vol. 3029, pp. 827–837 (2004)
21. Lankford, D.: On proving term rewrite systems are noetherian. Tech. Rep. MTP-3, Louisiana Technical University, Ruston, LA, USA (1979)
22. Lucas, S.: Practical use of polynomials over the reals in proofs of termination. In: Proc. 9th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming, pp. 39–50 (2007)
23. Middeldorp, A.: Approximations for strategies and termination. Electr. Notes Theor. Comput. Sci. **70**(6), 1–20 (2002)
24. Thiemann, R.: The DP framework for proving termination of term rewriting. Ph.D. thesis, RWTH Aachen (2007). Available as technical report AIB-2007-17
25. Waldmann, J.: Matchbox: A tool for match-bounded string rewriting. In: Proc. 15th International Conference on Rewriting Techniques and Applications, *LNCS*, vol. 3091, pp. 85–94 (2004)
26. Zankl, H., Middeldorp, A.: Increasing interpretations. In: Proc. 9th International Conference on Artificial Intelligence and Symbolic Computation, *LNCS (LNAI)*, vol. 5144, pp. 191–204 (2008)
27. Zantema, H.: Termination. In: Terese (ed.) Term Rewriting Systems, *Cambridge Tracts in Theoretical Computer Science*, vol. 55, pp. 181–259. Cambridge University Press (2003)
28. Zantema, H.: 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, *LNCS*, vol. 3838, pp. 173–197 (2005)
29. Zantema, H., Waldmann, J.: Termination by quasi-periodic interpretations. In: Proc. 18th International Conference on Rewriting Techniques and Applications, *LNCS*, vol. 4533, pp. 404–418 (2007)