Tyrolean Termination Tool: Techniques and Features

Nao Hirokawa Aart Middeldorp

Institute of Computer Science
University of Innsbruck
6020 Innsbruck, Austria

Abstract

The *Tyrolean Termination Tool* (TTT for short) is a powerful tool for automatically proving termination of rewrite systems. It incorporates several new refinements of the dependency pair method that are easy to implement, increase the power of the method, result in simpler termination proofs, and make the method more efficient. TTT employs polynomial interpretations with negative coefficients, like $x - 1$ for a unary function symbol or $x - y$ for a binary function symbol, which are useful for extending the class of rewrite systems that can be proved terminating automatically. Besides a detailed account of these techniques, we describe the convenient web interface of TTT and provide some implementation details.

Key words: Term rewriting, Termination, Automation, Dependency pair method, Polynomial interpretations

1 Introduction

The dependency pair method (Arts and Giesl [5]) and the monotonic semantic path order (Borralleras, Ferreira, and Rubio [9]) are two powerful methods which facilitate termination proofs that can be obtained automatically. After the introduction of these methods, there is a renewed interest in the study of termination for term rewrite systems. Three important issues which receive a lot of attention in current research on termination are to make these methods faster, to improve the methods such that more and more (challenging) rewrite

Email addresses: nao.hirokawa@uibk.ac.at (Nao Hirokawa), aart.middeldorp@uibk.ac.at (Aart Middeldorp).

systems can be handled, and to extend the methods beyond the realm of ordinary first-order term rewriting. Especially in connection with the dependency pair method many improvements, extensions, and refinements have been published. The dependency pair method forms an important ingredient in several software tools for proving termination. This paper describes the Tyrolean Termination Tool (T$_T$T in the sequel), the successor of the Tsukuba Termination Tool [23]. We explain its web interface, provide implementation details, and give a detailed account of the new techniques it supports.

In this paper we go back to the foundations of the dependency pair method. Starting from scratch, we give a systematic account of the method in the next two sections. Along the way we derive two new refinements—the *subterm criterion* in Section 2 and the *usable rule criterion* in Section 3—that are very easy to implement, increase the termination proving power, make the method much faster, and give rise to shorter termination proofs involving simpler techniques. In Section 4 we explain how to use polynomial interpretations with *negative coefficients* for proving termination in connection with the dependency pair method.

The web interface as well as some implementation details of T$_T$T are described in Section 5. In Section 6 we report on the many experiments that we performed to test the usefulness of the new techniques. The final section contains a short comparison with other systems for automatically proving termination of term rewrite systems.

Parts of this paper appeared in earlier conference proceedings [24,25,27]. New results are Theorem 23 in Section 3 and Theorem 40 in Section 4.

2 Dependency Pairs and Subterm Criterion

We assume familiarity with the basics of term rewriting [7,37]. We just recall some basic notation and terminology. The set of terms $\mathcal{T}(\mathcal{F},\mathcal{V})$ constructed from a signature $\mathcal{F}$ and a disjoint set $\mathcal{V}$ of variables is abbreviated to $\mathcal{T}$ when no confusion can arise. The set of variables appearing in a term t is denoted by $\mathcal{V}\mathsf{ar}(t)$. The root symbol of a term t is denoted by $\mathrm{root}(t)$. Defined function symbols are root symbols of left-hand sides of rewrite rules. A substitution is a mapping σ from variables to terms such that its domain $\mathcal{D}\mathsf{om}(\sigma) = \{x \mid x \neq \sigma(x)\}$ is finite. We write $t\sigma$ to denote the result of applying the substitution σ to the term t. A relation R on terms is closed under substitutions if $(s\sigma, t\sigma) \in R$ whenever $(s,t) \in R$, for all substitutions σ. We say that R is closed under contexts if $(u[s]_p, u[t]_p) \in R$ whenever $(s,t) \in R$, for all terms u and positions p in u. We use $\xrightarrow{\epsilon}$ to denote root rewrite steps and $\xrightarrow{>\epsilon}$ to denote rewrite steps in which the selected redex occurs below the root. The superterm relation is

denoted by $\trianglerighteq$ and $\triangleright$ denotes its strict part.

We use the following TRS from Dershowitz [12] extended with an associativity rule for $\vee$ to illustrate the developments in this and the next section:

$$\begin{array}{lll} 1\colon & \neg\neg x \to x & 4\colon\ x \wedge (y \vee z) \to (x \wedge y) \vee (x \wedge z) \\ 2\colon & \neg(x \vee y) \to \neg x \wedge \neg y & 5\colon\ (y \vee z) \wedge x \to (x \wedge y) \vee (x \wedge z) \\ 3\colon & \neg(x \wedge y) \to \neg x \vee \neg y & 6\colon\ (x \vee y) \vee z \to x \vee (y \vee z) \end{array}$$

Although this TRS is simply terminating, traditional simplification orders like the recursive path order, the Knuth-Bendix order, and polynomial interpretations do not apply. Modern termination tools incorporate many more powerful techniques, but automatically proving termination of the above TRS remains a challenge for many tools.

Let us start with some easy observations. If a TRS $\mathcal{R}$ is not terminating then there must be a *minimal* non-terminating term, minimal in the sense that all its proper subterms are terminating. Let us denote the set of all minimal non-terminating terms by $\mathcal{T}_\infty$. The following lemma is implicit in the proof of Theorem 6 in [5].

Lemma 1 *For every term $t \in \mathcal{T}_\infty$ there exist a rewrite rule $l \to r$, a substitution σ, and a non-variable subterm u of r such that $t \xrightarrow{>\epsilon}{}^* l\sigma \xrightarrow{\epsilon} r\sigma \trianglerighteq u\sigma$ and $u\sigma \in \mathcal{T}_\infty$.*

PROOF. Let A be an infinite rewrite sequence starting at t. Since all proper subterms of t are terminating, A must contain a root rewrite step. By considering the first root rewrite step in A it follows that there exist a rewrite rule $l \to r$ and a substitution σ such that A starts with $t \xrightarrow{>\epsilon}{}^* l\sigma \xrightarrow{\epsilon} r\sigma$. Write $l = f(l_1, \ldots, l_n)$. Since the rewrite steps in $t \to^* l\sigma$ take place below the root, $t = f(t_1, \ldots, t_n)$ and $t_i \to^* l_i\sigma$ for all $1 \leqslant i \leqslant n$. By assumption the arguments $t_1, \ldots, t_n$ of t are terminating. Hence so are the terms $l_1\sigma, \ldots, l_n\sigma$. It follows that $\sigma(x)$ is terminating for every $x \in \mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l)$. As $r\sigma$ is non-terminating it has a subterm $t' \in \mathcal{T}_\infty$. Because non-terminating terms cannot occur in the substitution part, there must be a non-variable subterm u of r such that $t' = u\sigma$. $\square$

Observe that the term $l\sigma$ in Lemma 1 belongs to $\mathcal{T}_\infty$ as well. Since all arguments of $l\sigma$ are terminating, $u\sigma$ cannot be a proper subterm of $l\sigma$. Moreover, since the root symbols of t and l are identical, every term in $\mathcal{T}_\infty$ has a defined root symbol.

If we were to define a new TRS $\mathcal{R}'$ consisting of all rewrite rules $l \to u$ for which there exist a rewrite rule $l \to r \in \mathcal{R}$ and a subterm u of r with defined

function symbol, then the sequence in the conclusion of Lemma 1 should be of the form $\xrightarrow{>\epsilon}{}^*_{\mathcal{R}} \cdot \xrightarrow{\epsilon}_{\mathcal{R}'}$. The idea is now to get rid of the position constraints by marking the root symbols of the terms in the rewrite rules of $\mathcal{R}'$.

Definition 2 *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. Let $\mathcal{F}^\sharp$ denote the union of $\mathcal{F}$ and $\{f^\sharp \mid f$ is a defined symbol of $\mathcal{R}\}$ where $f^\sharp$ is a fresh function symbol with the same arity as f. We call these new symbols* dependency pair symbols. *Given a term $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with f a defined symbol, we write $t^\sharp$ for the term $f^\sharp(t_1, \ldots, t_n)$. For any subset $T \subseteq \mathcal{T}$ consisting of terms with a defined root symbol, we denote the set $\{t^\sharp \mid t \in T\}$ by $T^\sharp$. If $l \to r \in \mathcal{R}$ and u is a subterm of r with defined root symbol such that u is not a proper subterm of l then the rewrite rule $l^\sharp \to u^\sharp$ is called a* dependency pair *of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted by $\mathsf{DP}(\mathcal{R})$.*

The idea of excluding dependency pairs $l^\sharp \to u^\sharp$ where u is a proper subterm of l is due to Dershowitz [13]. Although dependency pair symbols are defined symbols of $\mathsf{DP}(\mathcal{R})$, they are not defined symbols of the original TRS $\mathcal{R}$. In the following, defined symbols always refer to the original TRS $\mathcal{R}$.

Example 3 *The example at the beginning of this section admits the following 14 dependency pairs:*

7:	$\neg^\sharp(x \vee y) \to \neg x \wedge^\sharp \neg y$	14:	$x \wedge^\sharp (y \vee z) \to x \wedge^\sharp y$
8:	$\neg^\sharp(x \vee y) \to \neg^\sharp x$	15:	$x \wedge^\sharp (y \vee z) \to x \wedge^\sharp z$
9:	$\neg^\sharp(x \vee y) \to \neg^\sharp y$	16:	$(y \vee z) \wedge^\sharp x \to (x \wedge y) \vee^\sharp (x \wedge z)$
10:	$\neg^\sharp(x \wedge y) \to \neg x \vee^\sharp \neg y$	17:	$(y \vee z) \wedge^\sharp x \to x \wedge^\sharp y$
11:	$\neg^\sharp(x \wedge y) \to \neg^\sharp x$	18:	$(y \vee z) \wedge^\sharp x \to x \wedge^\sharp z$
12:	$\neg^\sharp(x \wedge y) \to \neg^\sharp y$	19:	$(x \vee y) \vee^\sharp z \to x \vee^\sharp (y \vee z)$
13:	$x \wedge^\sharp (y \vee z) \to (x \wedge y) \vee^\sharp (x \wedge z)$	20:	$(x \vee y) \vee^\sharp z \to y \vee^\sharp z$

An immediate consequence of Lemma 1 and the previous definition is that for every term $s \in \mathcal{T}_\infty$ there exist terms $t, u \in \mathcal{T}_\infty$ such that $s^\sharp \to^*_{\mathcal{R}} t^\sharp \to_{\mathsf{DP}(\mathcal{R})} u^\sharp$. Hence, every non-terminating TRS $\mathcal{R}$ admits an infinite rewrite sequence of the form

$$t_1 \to^*_{\mathcal{R}} t_2 \to_{\mathsf{DP}(\mathcal{R})} t_3 \to^*_{\mathcal{R}} t_4 \to_{\mathsf{DP}(\mathcal{R})} \cdots$$

with $t_i \in \mathcal{T}^\sharp_\infty$ for all $i \geqslant 1$. Consequently, to prove termination of a TRS $\mathcal{R}$ it is sufficient to show that $\mathcal{R} \cup \mathsf{DP}(\mathcal{R})$ admits no such infinite sequences. For finite $\mathcal{R}$, every such sequence contains a tail in which all applied dependency pairs are used infinitely many times. The set of those dependency pairs forms a cycle in the dependency graph, which is defined below.[1] *From now on, we assume that all TRSs are finite.*

[1] But not every cycle in the dependency graph can be obtained in this way.

Definition 4 *The nodes of the* dependency graph $\mathsf{DG}(\mathcal{R})$ *are the dependency pairs of* $\mathcal{R}$ *and there is an arrow from* $s \to t$ *to* $u \to v$ *if and only if there exist substitutions* σ *and* τ *such that* $t\sigma \to_{\mathcal{R}}^* u\tau$*. A* cycle *is a nonempty subset* $\mathcal{C}$ *of dependency pairs of* $\mathsf{DP}(\mathcal{R})$ *if for every two (not necessarily distinct) pairs* $s \to t$ *and* $u \to v$ *in* $\mathcal{C}$ *there exists a nonempty path in* $\mathcal{C}$ *from* $s \to t$ *to* $u \to v$*.*

The following theorem corresponds to Theorem 3.3 in [18].

Theorem 5 *For every non-terminating TRS* $\mathcal{R}$ *there exist a cycle* $\mathcal{C}$ *in* $\mathsf{DG}(\mathcal{R})$ *and an infinite rewrite sequence in* $\mathcal{R} \cup \mathcal{C}$ *of the form*

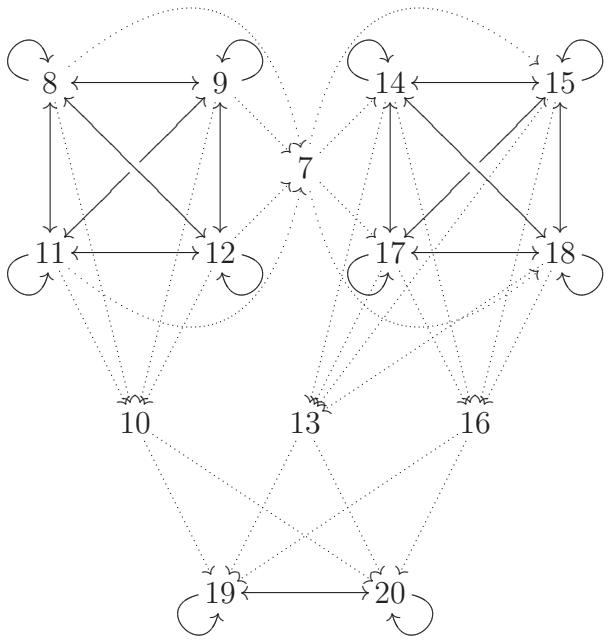
$$\mathcal{T}_\infty^\sharp \ni t_1 \to_{\mathcal{R}}^* t_2 \to_{\mathcal{C}} t_3 \to_{\mathcal{R}}^* t_4 \to_{\mathcal{C}} \cdots$$

in which all rules of $\mathcal{C}$ *are applied infinitely often.* □

We call such an infinite rewrite sequence $\mathcal{C}$*-minimal*. The difference with the related concept of *infinite* $\mathcal{R}$*-chain of dependency pairs* in [5,18] is that we impose minimality (i.e., the condition $t_1 \in \mathcal{T}_\infty^\sharp$). Minimality is essential for the new results (Theorems 8 and 20) that we present below.

So proving termination of a TRS $\mathcal{R}$ boils down to proving the absence of $\mathcal{C}$-minimal rewrite sequences, for any cycle $\mathcal{C}$ in the dependency graph $\mathsf{DG}(\mathcal{R})$.

Example 6 *Our leading example has the following dependency graph (dotted arrows do not contribute to cycles):*



It contains 33 cycles: all nonempty subsets of $\{8, 9, 11, 12\}$*,* $\{14, 15, 17, 18\}$*, and* $\{19, 20\}$*.*

Although the dependency graph is not computable in general, sound approximations exist that can be computed efficiently [5,26]. Soundness here means that every cycle in the real dependency graph is a cycle in the approximated graph. For our example all known approximations compute the real dependency graph.

We now present a new criterion which permits us to ignore certain cycles of the dependency graph.

Definition 7 *Let $\mathcal{R}$ be a TRS and $\mathcal{C} \subseteq \mathsf{DP}(\mathcal{R})$ such that every dependency pair symbol in $\mathcal{C}$ has positive arity. A* simple projection *for $\mathcal{C}$ is a mapping π that assigns to every n-ary dependency pair symbol $f^\sharp$ in $\mathcal{C}$ an argument position $i \in \{1, \ldots, n\}$. The mapping that assigns to every term $f^\sharp(t_1, \ldots, t_n) \in \mathcal{T}^\sharp$, with $f^\sharp$ a dependency pair symbol in $\mathcal{C}$, its argument at position $\pi(f^\sharp)$ is also denoted by π.*

If R is a set of rewrite rules and O is a relation on terms then the expression $\pi(R)$ denotes the set $\{\pi(l) \to \pi(r) \mid l \to r \in R\}$, the inclusion $R \subseteq O$ abbreviates "$(l, r) \in O$ for all $l \to r \in R$", and the inequality $R \cap O \neq \varnothing$ abbreviates "$(l, r) \in O$ for at least one $l \to r \in R$". So the conditions in the following theorem state that after applying the simple projection π, every rule in $\mathcal{C}$ is turned into an identity or a rule whose right-hand side is a proper subterm of the left-hand side. Moreover, the latter case applies at least once.

Theorem 8 *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$. If there exists a simple projection π for $\mathcal{C}$ such that $\pi(\mathcal{C}) \subseteq \trianglerighteq$ and $\pi(\mathcal{C}) \cap \triangleright \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.*

PROOF. Suppose to the contrary that there exists a $\mathcal{C}$-minimal rewrite sequence:

$$\mathcal{T}^\sharp_\infty \ni t_1 \to^*_\mathcal{R} u_1 \to_\mathcal{C} t_2 \to^*_\mathcal{R} u_2 \to_\mathcal{C} t_3 \to^*_\mathcal{R} \cdots \tag{1}$$

All terms in this sequence have a dependency pair symbol in $\mathcal{C}$ as root symbol. We apply the simple projection π to (1). Let $i \geqslant 1$.

- First consider the dependency pair step $u_i \to_\mathcal{C} t_{i+1}$. There exist a dependency pair $l \to r \in \mathcal{C}$ and a substitution σ such that $u_i = l\sigma$ and $t_{i+1} = r\sigma$. We have $\pi(u_i) = \pi(l)\sigma$ and $\pi(t_{i+1}) = \pi(r)\sigma$. We have $\pi(l) \trianglerighteq \pi(r)$ by assumption. So $\pi(l) = \pi(r)$ or $\pi(l) \triangleright \pi(r)$. In the former case we trivially have $\pi(u_i) = \pi(t_{i+1})$. In the latter case the closure under substitutions of $\triangleright$ yields $\pi(u_i) \triangleright \pi(t_{i+1})$. Because of the assumption $\pi(\mathcal{C}) \cap \triangleright \neq \varnothing$, the latter holds for infinitely many i.
- Next consider the rewrite sequence $t_i \to^*_\mathcal{R} u_i$. All steps in this sequence take place below the root and thus we obtain the (possibly shorter) sequence

$\pi(t_i) \to_{\mathcal{R}}^* \pi(u_i)$.

So by applying the simple projection π, sequence (1) is transformed into an infinite $\to_{\mathcal{R}} \cup \rhd$ sequence containing infinitely many $\rhd$ steps, starting from the term $\pi(t_1)$. Since the relation $\rhd$ is well-founded, the infinite sequence must also contain infinitely many $\to_{\mathcal{R}}$ steps. By making repeated use of the well-known relational inclusion $\rhd \cdot \to_{\mathcal{R}} \subseteq \to_{\mathcal{R}} \cdot \rhd$ ($\rhd$ commutes over $\to_{\mathcal{R}}$ in the terminology of [8]), we obtain an infinite $\to_{\mathcal{R}}$ sequence starting from $\pi(t_1)$. In other words, the term $\pi(t_1)$ is non-terminating with respect to $\mathcal{R}$. Let $t_1 = f^\sharp(s_1, \ldots, s_n)$. Because $t_1 \in \mathcal{T}^\sharp_\infty$, $f(s_1, \ldots, s_n)$ is a minimal non-terminating term. Consequently, its argument $\pi(t_1) = s_{\pi(f^\sharp)}$ is terminating with respect to $\mathcal{R}$, providing the desired contradiction. $\Box$

In contrast to the standard dependency pair approach (cf. Theorem 12 below), the above theorem permits us to discard cycles of the dependency graph without considering any rewrite rules. This is extremely useful. Moreover, the criterion is very simple to check.

Example 9 *First consider the cycle* $\mathcal{C}_1 = \{8, 9, 11, 12\}$. *The only dependency pair symbol in* $\mathcal{C}_1$ *is* $\neg^\sharp$. *Since* $\neg^\sharp$ *is a unary function symbol, there is just one simple projection for* $\mathcal{C}_1$*:* $\pi_1(\neg^\sharp) = 1$. *By applying* π_1 *to* $\mathcal{C}_1$, *we obtain*

$$\begin{array}{ll} 8\colon\ x \vee y \to x & \qquad 11\colon\ x \wedge y \to x \\ 9\colon\ x \vee y \to y & \qquad 12\colon\ x \wedge y \to y \end{array}$$

We clearly have $\pi_1(\mathcal{C}_1) \subseteq \rhd$. *Hence we can ignore* $\mathcal{C}_1$ *(and all its subcycles). Next consider the cycle* $\mathcal{C}_2 = \{19, 20\}$. *The only dependency pair symbol in* $\mathcal{C}_2$ *is* $\vee^\sharp$. *By applying the simple projection* $\pi_2(\vee^\sharp) = 1$ *to* $\mathcal{C}_2$, *we obtain*

$$19\colon\ x \vee y \to x \qquad\qquad 20\colon\ x \vee y \to y$$

We clearly have $\pi_2(\mathcal{C}_2) \subseteq \rhd$. *Hence we can ignore* $\mathcal{C}_2$ *(and its two subcycles). The only cycles that are not handled by the criterion of Theorem 8 are the ones that involve* 17 *or* 18*; applying the simple projection* $\pi(\wedge^\sharp) = 1$ *produces*

$$17, 18\colon\ y \vee z \to x$$

whereas $\pi(\wedge^\sharp) = 2$ *gives*

$$17\colon\ x \to y \qquad\qquad 18\colon\ x \to z$$

None of these rules are compatible with $\trianglerighteq$.

In implementations one should not compute all cycles of the dependency graph (since there can be exponentially many in the number of dependency pairs),

but use the technique of Hirokawa and Middeldorp [26] to solve strongly connected components[2] recursively (which gives rise to a linear algorithm): if all pairs in a strongly connected component (SCC for short) are compatible with $\trianglerighteq$ after applying a simple projection, the ones that are compatible with $\triangleright$ are removed and new SCCs among the remaining pairs are computed. This is illustrated in the following example. This example furthermore shows that the subterm criterion is capable of proving the termination of TRSs that were considered to be challenging in the termination literature (cf. remarks in [20, Example 9]).

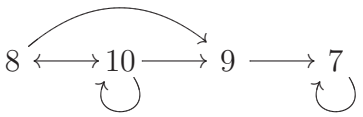
Example 10 *Consider the following TRS from [36]:*

$$\begin{aligned}
1\colon&\quad \mathsf{intlist}([\,]) \to [\,]\\
2\colon&\quad \mathsf{intlist}(x : y) \to \mathsf{s}(x) : \mathsf{intlist}(y)\\
3\colon&\quad \mathsf{int}(0,0) \to 0 : [\,]\\
4\colon&\quad \mathsf{int}(0,\mathsf{s}(y)) \to 0 : \mathsf{int}(\mathsf{s}(0),\mathsf{s}(y))\\
5\colon&\quad \mathsf{int}(\mathsf{s}(x),0) \to [\,]\\
6\colon&\quad \mathsf{int}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{intlist}(\mathsf{int}(x,y))
\end{aligned}$$

The term $\mathsf{int}(\mathsf{s}^m(0),\mathsf{s}^n(0))$ *evaluates to the list* $[\mathsf{s}^m(0),\mathsf{s}^{m+1}(0),\ldots,\mathsf{s}^n(0)]$*; lists are constructed using* $:$ *and* $[\,]$*. There are 4 dependency pairs:*

$$\begin{aligned}
7\colon&\quad \mathsf{intlist}^\sharp(x : y) \to \mathsf{intlist}^\sharp(y)\\
8\colon&\quad \mathsf{int}^\sharp(0,\mathsf{s}(y)) \to \mathsf{int}^\sharp(\mathsf{s}(0),\mathsf{s}(y))\\
9\colon&\quad \mathsf{int}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{intlist}^\sharp(\mathsf{int}(x,y))\\
10\colon&\quad \mathsf{int}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{int}^\sharp(x,y)
\end{aligned}$$

The dependency graph



contains 2 SCCs: $\{7\}$ *and* $\{8,10\}$*. The first one is handled by the simple projection* $\pi(\mathsf{intlist}^\sharp) = 1$*:*

$$7\colon\quad x : y \to y$$

For the second one we use the simple projection $\pi(\mathsf{int}^\sharp) = 2$*:*

$$8\colon\quad \mathsf{s}(y) \to \mathsf{s}(y) \qquad\qquad 10\colon\quad \mathsf{s}(y) \to y$$

After removing the strictly decreasing pair 10*, we are left with* 8*. Since the restriction of the dependency graph to the remaining pair* 8 *contains no SCCs, the TRS is terminating.*

[2] A strongly connected component is a maximal cycle in the dependency graph.

An empirical evaluation of the subterm criterion can be found in Section 6.

3 Usable Rules

What to do with cycles $\mathcal{C}$ of the dependency graph that cannot be handled by the criterion of the preceding section? In the dependency pair approach one uses orders $\gtrsim$, $\geqslant$, and $>$ that satisfy the properties stated below such that

(1) all rules in $\mathcal{R}$ are oriented by $\gtrsim$,
(2) all rules in $\mathcal{C}$ are oriented by $\geqslant$, and
(3) at least one rule in $\mathcal{C}$ is oriented by $>$.

Definition 11 *A* reduction triple $(\gtrsim, \geqslant, >)$ *consists of three relations that are closed under substitutions such that* $\gtrsim$ *and* $\geqslant$ *are preorders,* $\gtrsim$ *is closed under contexts,* $>$ *is a well-founded order, and the following compatibility condition holds: both* $\gtrsim \cdot > \subseteq >$ *and* $\geqslant \cdot > \subseteq >$ *or both* $> \cdot \gtrsim \subseteq >$ *and* $> \cdot \geqslant \subseteq >$*. We say that* $(\gtrsim, >)$ *is a reduction* pair *if* $(\gtrsim, \gtrsim \cup >, >)$ *is a reduction triple.*

Since we do not demand that $>$ is the strict part of the preorders $\gtrsim$ or $\geqslant$, the identities $\gtrsim \cdot > = >$ and $\geqslant \cdot > = >$ need not hold. Note that every reduction pair $(\gtrsim, >)$ gives rise to the reduction triple $(\gtrsim, \gtrsim \cup >, >)$. In earlier papers on dependency pairs, reduction pairs are used and the relation $\geqslant$ in condition (2) above is consequently replaced by $\gtrsim \cup >$. The formulation using reduction triples is slightly more general and anticipates the developments in Section 4.

A typical example of a reduction pair is $(\geq_{\text{lpo}}, >_{\text{lpo}})$, where $>_{\text{lpo}}$ is the lexicographic path order induced by the (strict) precedence $>$, and $\geq_{\text{lpo}}$ denotes its reflexive closure. In order to benefit from the fact that closure under contexts is not required, the conditions (1), (2), and (3) mentioned at the beginning of this section may be simplified by applying an argument filtering [5] to delete certain (arguments of) function symbols occurring in $\mathcal{R}$ and $\mathcal{C}$ before testing orientability.

A general semantic construction of reduction pairs, which covers polynomial interpretations, is based on the concept of algebra. If we equip the carrier A of an algebra $\mathcal{A} = (A, \{f_A\}_{f \in \mathcal{F}})$ with a well-founded order $>$ such that every interpretation function is weakly monotone in all arguments (i.e., $f_A(x_1, \ldots, x_n) \geqslant f_A(y_1, \ldots, y_n)$ whenever $x_i \geqslant y_i$ for all $1 \leqslant i \leqslant n$, for every n-ary function symbol $f \in \mathcal{F}$) then $(\geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ is a reduction pair. Here the relations $\geqslant_{\mathcal{A}}$ and $>_{\mathcal{A}}$ are defined as follows: $s \geqslant_{\mathcal{A}} t$ if $[\alpha]_{\mathcal{A}}(s) \geqslant [\alpha]_{\mathcal{A}}(t)$ and $s >_{\mathcal{A}} t$ if $[\alpha]_{\mathcal{A}}(s) > [\alpha]_{\mathcal{A}}(t)$, for all assignments α of elements of A to the variables in s and t ($[\alpha]_{\mathcal{A}}(\cdot)$ denotes the usual evaluation function associated with the algebra $\mathcal{A}$).

We now present the standard dependency pair approach [18, Theorem 3.5] to the treatment of cycles in the dependency graph using reduction triples.

Theorem 12 *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$. If there exists a reduction triple $(\gtrsim, \geqslant, >)$ such that $\mathcal{R} \subseteq \gtrsim$, $\mathcal{C} \subseteq \geqslant$, and $\mathcal{C} \cap > \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.* $\square$

The proof of Theorem 12 does not use the fact that $\mathcal{C}$-minimal rewrite sequences start from terms in $\mathcal{T}^\sharp_\infty$. In the remainder of this section we show that by restoring the use of minimality, we can get rid of some of the constraints originating from $\mathcal{R}$.

Arts and Giesl [5] proved that for establishing *innermost* termination, the constraint $\mathcal{R} \subseteq \gtrsim$ can be weakened to $\mathcal{U}(\mathcal{C}) \subseteq \gtrsim$. Here $\mathcal{U}(\mathcal{C})$ denotes the set of *usable* rules of $\mathcal{C}$, which is defined as follows.

Definition 13 *We write $f \rhd_\mathrm{d} g$ if there exists a rewrite rule $l \to r \in \mathcal{R}$ such that $f = \mathrm{root}(l)$ and g is a defined function symbol in $\mathcal{F}\mathsf{un}(r)$. For a set $\mathcal{G}$ of defined function symbols we denote by $\mathcal{R}{\restriction}\mathcal{G}$ the set of rewrite rules $l \to r \in \mathcal{R}$ with $\mathrm{root}(l) \in \mathcal{G}$. The set $\mathcal{U}(t)$ of usable rules of a term t is defined as $\mathcal{R}{\restriction}\{g \mid f \rhd_\mathrm{d}^* g$ for some defined function symbol f in $\mathcal{F}\mathsf{un}(t)\}$. Finally, if $\mathcal{C}$ is a set of dependency pairs then*

$$\mathcal{U}(\mathcal{C}) = \bigcup_{l \to r \in \mathcal{C}} \mathcal{U}(r)$$

Example 14 *None of the dependency pairs that appear in the SCCs $\{8, 9, 11, 12\}$ and $\{14, 15, 17, 18\}$ in our leading example have defined symbols in their right-hand sides, so for both SCCs the set of usable rules is empty. The right-hand side of dependency pair 19 contains the defined symbol $\vee$. We have $\{g \mid \vee \rhd_\mathrm{d}^* g\} = \{\vee\}$ and hence the usable rules of the SCC $\{19, 20\}$ are the rules that define $\vee$, which is just rule 6.*

Urbain [42] obtained a first significant result for termination. By combining a modularity result of Gramlich [21] with a weaker version of the dependency pair method, he essentially showed that the constraint $\mathcal{R} \subseteq \gtrsim$ can be replaced by $\mathcal{R}' \cup \mathcal{P} \subseteq \gtrsim$ for some subset $\mathcal{R}'$ of $\mathcal{R}$. Here $\mathcal{P}$ denotes the TRS consisting of the two projection rules

$$\mathsf{cons}(x, y) \to x$$
$$\mathsf{cons}(x, y) \to y$$

for a fresh function symbol cons. By adapting this result to the formulation of the dependency pair method in Theorem 12, Giesl *et al.* [20] obtained a strictly more powerful result. In both approaches $\mathcal{R}'$ is a superset of $\mathcal{U}(\mathcal{C})$.

Below we show that the constraint $\mathcal{U}(\mathcal{C}) \cup \mathcal{P} \subseteq \gtrsim$ is sufficient. To this end we transform a presupposed $\mathcal{C}$-minimal rewrite sequence

$$t_1 \to_{\mathcal{R}}^* u_1 \to_{\mathcal{C}} t_2 \to_{\mathcal{R}}^* u_2 \to_{\mathcal{C}} t_3 \to_{\mathcal{R}}^* \cdots$$

into an infinite rewrite sequence in $\mathcal{U}(\mathcal{C}) \cup \mathcal{C} \cup \mathcal{P}$ in which every rule of $\mathcal{C}$ is used infinitely often, providing a contradiction with the assumptions $\mathcal{U}(\mathcal{C}) \cup \mathcal{P} \subseteq \gtrsim$, $\mathcal{C} \subseteq \geqslant$, and $\mathcal{C} \cap > \neq \varnothing$. The transformation is achieved by applying the mapping $I_{\mathcal{G}}$ defined below, where $\mathcal{G}$ is the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$, to all terms in the above sequence.

Definition 15 *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and let $\mathcal{G} \subseteq \mathcal{F}$. The* interpretation *$I_{\mathcal{G}}$ is a mapping from terminating terms in $\mathcal{T}(\mathcal{F}^\sharp, \mathcal{V})$ to terms in $\mathcal{T}(\mathcal{F}^\sharp \cup \{\mathsf{nil}, \mathsf{cons}\}, \mathcal{V})$, where* nil *and* cons *are fresh function symbols, inductively defined as follows:*

$$I_{\mathcal{G}}(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ f(I_{\mathcal{G}}(t_1), \ldots, I_{\mathcal{G}}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } f \notin \mathcal{G} \\ \mathsf{cons}(f(I_{\mathcal{G}}(t_1), \ldots, I_{\mathcal{G}}(t_n)), t') & \text{if } t = f(t_1, \ldots, t_n) \text{ and } f \in \mathcal{G} \end{cases}$$

where in the last clause t' denotes the term $\mathrm{order}(\{I_{\mathcal{G}}(u) \mid t \to_{\mathcal{R}} u\})$ *with*

$$\mathrm{order}(T) = \begin{cases} \mathsf{nil} & \text{if } T = \varnothing \\ \mathsf{cons}(t, \mathrm{order}(T \setminus \{t\})) & \text{if } t \text{ is the minimum element of } T \end{cases}$$

Here we assume an arbitrary but fixed total order on $\mathcal{T}(\mathcal{F}^\sharp \cup \{\mathsf{nil}, \mathsf{cons}\}, \mathcal{V})$.

The idea behind the mapping $I_{\mathcal{G}}$ is to anticipate the applications of non-usable rules by collecting in a recursive manner the corresponding reducts into a list. The rules of $\mathcal{P}$ will be used to extract appropriate elements from the lists constructed by $I_{\mathcal{G}}$.

Because we deal with finite TRSs, the relation $\to_{\mathcal{R}}$ is finitely branching and hence the set $\{u \mid t \to_{\mathcal{R}} u\}$ of one-step reducts of t is finite. Moreover, every term in this set is terminating. The well-definedness of $I_{\mathcal{G}}$ now follows by a straightforward induction argument.

The above definition is a variation of a similar definition in Urbain [42], which in turn is based on a definition in Gramlich [21]. The difference with Urbain's definition is that we insert $f(I_{\mathcal{G}}(t_1), \ldots, I_{\mathcal{G}}(t_n))$ in the list t' when $f \in \mathcal{G}$. This modification is crucial for obtaining Theorem 20 below.

We start with some preliminary results. The first one addresses the behaviour of $I_{\mathcal{G}}$ on instantiated terms.

Definition 16 *If σ is a substitution that assigns to every variable a termi-*

nating term then we denote the substitution that assigns to every variable x the term $I_{\mathcal{G}}(x\sigma)$ by $\sigma_{I_{\mathcal{G}}}$.

Lemma 17 *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and let $\mathcal{G} \subseteq \mathcal{F}$. Let t be a term and σ a substitution. If $t\sigma$ is terminating then $I_{\mathcal{G}}(t\sigma) \to_{\mathcal{P}}^* t\sigma_{I_{\mathcal{G}}}$ and, if t does not contain $\mathcal{G}$-symbols, $I_{\mathcal{G}}(t\sigma) = t\sigma_{I_{\mathcal{G}}}$.*

PROOF. We use induction on t. If t is a variable then $I_{\mathcal{G}}(t\sigma) = t\sigma_{I_{\mathcal{G}}}$. Let $t = f(t_1, \ldots, t_n)$. We distinguish two cases.

- If $f \notin \mathcal{G}$ then $I_{\mathcal{G}}(t\sigma) = f(I_{\mathcal{G}}(t_1\sigma), \ldots, I_{\mathcal{G}}(t_n\sigma))$. The induction hypothesis yields $I_{\mathcal{G}}(t_i\sigma) \to_{\mathcal{P}}^* t_i\sigma_{I_{\mathcal{G}}}$ for $1 \leqslant i \leqslant n$ and thus

$$I_{\mathcal{G}}(t\sigma) \to_{\mathcal{P}}^* f(t_1\sigma_{I_{\mathcal{G}}}, \ldots, t_n\sigma_{I_{\mathcal{G}}}) = t\sigma_{I_{\mathcal{G}}}$$

  If there are no $\mathcal{G}$-symbols in $t_1, \ldots, t_n$ then we obtain $I_{\mathcal{G}}(t_i\sigma) = t_i\sigma_{I_{\mathcal{G}}}$ for all $1 \leqslant i \leqslant n$ from the induction hypothesis and thus $I_{\mathcal{G}}(t\sigma) = t\sigma_{I_{\mathcal{G}}}$.
- If $f \in \mathcal{G}$ then

$$I_{\mathcal{G}}(t\sigma) = \mathsf{cons}(f(I_{\mathcal{G}}(t_1\sigma), \ldots, I_{\mathcal{G}}(t_n\sigma)), t') \to_{\mathcal{P}} f(I_{\mathcal{G}}(t_1\sigma), \ldots, I_{\mathcal{G}}(t_n\sigma))$$

  for some term t'. We obtain $f(I_{\mathcal{G}}(t_1\sigma), \ldots, I_{\mathcal{G}}(t_n\sigma)) \to_{\mathcal{P}}^* t\sigma_{I_{\mathcal{G}}}$ as in the preceding case and thus $I_{\mathcal{G}}(t\sigma) \to_{\mathcal{P}}^* t\sigma_{I_{\mathcal{G}}}$ as desired. □

The second preliminary result states that $I_{\mathcal{G}}$ preserves any top part without $\mathcal{G}$-symbols. We omit the straightforward proof.

Lemma 18 *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and let $\mathcal{G} \subseteq \mathcal{F}$. If $t = C[t_1, \ldots, t_n]$ is terminating and the context C contains no $\mathcal{G}$-symbols then $I_{\mathcal{G}}(t) = C[I_{\mathcal{G}}(t_1), \ldots, I_{\mathcal{G}}(t_n)]$.* □

The following lemma is the key result. It states that rewrite steps in $\mathcal{R}$ are transformed by $I_{\mathcal{G}}$ into rewrite sequences in $\mathcal{U}(\mathcal{C}) \cup \mathcal{P}$, provided $\mathcal{G}$ is the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$.

Lemma 19 *Let $\mathcal{R}$ be a TRS and let $\mathcal{C} \subseteq \mathsf{DP}(\mathcal{R})$. Furthermore, let $\mathcal{G}$ be the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$. If terms s and t are terminating and $s \to_{\mathcal{R}} t$ then $I_{\mathcal{G}}(s) \to_{\mathcal{U}(\mathcal{C}) \cup \mathcal{P}}^+ I_{\mathcal{G}}(t)$.*

PROOF. Let p be the position of the rewrite step $s \to_{\mathcal{R}} t$. We distinguish two cases.

- First suppose that there is a function symbol from $\mathcal{G}$ at a position above p. In this case we may write $s = C[s_1, \ldots, s_i, \ldots, s_n]$ and $t = C[s_1, \ldots, t_i, \ldots, s_n]$

with $s_i \to_{\mathcal{R}} t_i$, where $\mathrm{root}(s_i) \in \mathcal{G}$ and the context C contains no $\mathcal{G}$-symbols. We have $I_{\mathcal{G}}(s_i) \to_{\mathcal{P}} \mathrm{order}(\{I_{\mathcal{G}}(u) \mid s_i \to_{\mathcal{R}} u\})$. Since $s_i \to_{\mathcal{R}} t_i$, we can extract $I_{\mathcal{G}}(t_i)$ from the term $\mathrm{order}(\{I_{\mathcal{G}}(u) \mid s_i \to_{\mathcal{R}} u\})$ by appropriate $\mathcal{P}$ steps, so $I_{\mathcal{G}}(s_i) \to_{\mathcal{P}}^{+} I_{\mathcal{G}}(t_i)$. We now obtain $I_{\mathcal{G}}(s) \to_{\mathcal{P}}^{+} I_{\mathcal{G}}(t)$ from Lemma 18.

- In the other case $s = C[s_1, \ldots, s_i, \ldots, s_n]$ and $t = C[s_1, \ldots, t_i, \ldots, s_n]$ with $s_i \xrightarrow{\epsilon}_{\mathcal{R}} t_i$, where $\mathrm{root}(s_i) \notin \mathcal{G}$ and the context C contains no $\mathcal{G}$-symbols. Since $\mathrm{root}(s_i) \notin \mathcal{G}$ and $\mathcal{R} = \mathcal{U}(\mathcal{C}) \cup (\mathcal{R}\restriction\mathcal{G})$, the applied rewrite rule $l \to r$ in the step $s_i \xrightarrow{\epsilon}_{\mathcal{R}} t_i$ must come from $\mathcal{U}(\mathcal{C})$. Let σ be the substitution with $\mathcal{D}\mathsf{om}(\sigma) \subseteq \mathcal{V}\mathsf{ar}(l)$ such that $s_i = l\sigma$ and $t_i = r\sigma$. We obtain $I_{\mathcal{G}}(s_i) \to_{\mathcal{P}}^{*} l\sigma_{I_{\mathcal{G}}}$ from Lemma 17. Because right-hand sides of rules in $\mathcal{U}(\mathcal{C})$ do not contain $\mathcal{G}$-symbols, the same lemma yields $I_{\mathcal{G}}(t_i) = r\sigma_{I_{\mathcal{G}}}$. Clearly $l\sigma_{I_{\mathcal{G}}} \to_{\mathcal{U}(\mathcal{C})} r\sigma_{I_{\mathcal{G}}}$ and thus $I_{\mathcal{G}}(s_i) \to_{\mathcal{U}(\mathcal{C})\cup\mathcal{P}}^{+} I_{\mathcal{G}}(t_i)$. Lemma 18 now yields the desired $I_{\mathcal{G}}(s) \to_{\mathcal{U}(\mathcal{C})\cup\mathcal{P}}^{+} I_{\mathcal{G}}(t)$. □

After these preparations, the main result [3] of this section is easily proved.

Theorem 20 *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$. If there exists a reduction triple $(\gtrsim, \geqslant, >)$ such that $\mathcal{U}(\mathcal{C}) \cup \mathcal{P} \subseteq \gtrsim$, $\mathcal{C} \subseteq \geqslant$, and $\mathcal{C} \cap > \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.*

PROOF. Suppose to the contrary that there exists a $\mathcal{C}$-minimal rewrite sequence:

$$t_1 \to_{\mathcal{R}}^{*} u_1 \to_{\mathcal{C}} t_2 \to_{\mathcal{R}}^{*} u_2 \to_{\mathcal{C}} t_3 \to_{\mathcal{R}}^{*} \cdots \tag{2}$$

Let $\mathcal{G}$ be the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$. We show that after applying the interpretation $I_{\mathcal{G}}$ we obtain an infinite rewrite sequence in $\mathcal{U}(\mathcal{C}) \cup \mathcal{P} \cup \mathcal{C}$ in which every rule of $\mathcal{C}$ is used infinitely often. Since all terms in (2) belong to $\mathcal{T}_{\infty}^{\sharp}$, they are terminating with respect to $\mathcal{R}$ and hence we can indeed apply the interpretation $I_{\mathcal{G}}$. Let $i \geqslant 1$.

- First consider the dependency pair step $u_i \to_{\mathcal{C}} t_{i+1}$. There exist a dependency pair $l \to r \in \mathcal{C}$ and a substitution σ such that $u_i = l\sigma$ and $t_{i+1} = r\sigma$. We may assume that $\mathcal{D}\mathsf{om}(\sigma) \subseteq \mathcal{V}\mathsf{ar}(l)$. Since $u_i \in \mathcal{T}_{\infty}^{\sharp}$, $\sigma(x)$ is terminating for every variable $x \in \mathcal{V}\mathsf{ar}(l)$. Hence the substitution $\sigma_{I_{\mathcal{G}}}$ is well-defined. Since right-hand sides of rules in $\mathcal{U}(\mathcal{C})$ lack $\mathcal{G}$-symbols, we have $I_{\mathcal{G}}(r\sigma) = r\sigma_{I_{\mathcal{G}}}$ by Lemma 17. The same lemma also yields $I_{\mathcal{G}}(l\sigma) \to_{\mathcal{P}}^{*} l\sigma_{I_{\mathcal{G}}}$. Hence
$$I_{\mathcal{G}}(u_i) \to_{\mathcal{P}}^{*} l\sigma_{I_{\mathcal{G}}} \to_{\mathcal{C}} r\sigma_{I_{\mathcal{G}}} = I_{\mathcal{G}}(t_{i+1})$$
- Next consider the rewrite sequence $t_i \to_{\mathcal{R}}^{*} u_i$. Because all terms in this sequence are terminating, we obtain $I_{\mathcal{G}}(t_i) \to_{\mathcal{U}(\mathcal{C})\cup\mathcal{P}}^{*} I_{\mathcal{G}}(u_i)$ by repeated applications of Lemma 19.

[3] This result has been independently obtained by Thiemann *et al.* [40].

So we obtain the infinite rewrite sequence

$$\begin{aligned} I_{\mathcal{G}}(t_1) &\to^*_{\mathcal{U}(\mathcal{C})\cup\mathcal{P}} I_{\mathcal{G}}(u_1) \to^*_{\mathcal{P}} \cdot \to_{\mathcal{C}} I_{\mathcal{G}}(t_2) \\ &\to^*_{\mathcal{U}(\mathcal{C})\cup\mathcal{P}} I_{\mathcal{G}}(u_2) \to^*_{\mathcal{P}} \cdot \to_{\mathcal{C}} I_{\mathcal{G}}(t_3) \to^*_{\mathcal{U}(\mathcal{C})\cup\mathcal{P}} \cdots \end{aligned}$$

in which all rules in $\mathcal{C}$ are infinitely often applied. Using the assumptions of the theorem, the latter sequence is transformed into an infinite sequence consisting of $\gtrsim$, $\geqslant$, and infinitely many $>$ steps. Using the compatibility condition, we obtain a contradiction with the well-foundedness of $>$. $\square$

Example 21 *Let us take a final look at the SCC* $\{14, 15, 17, 18\}$ *in our leading example. There are no usable rules. By taking the linear polynomial interpretation* $\wedge^\sharp_{\mathbb{N}}(x, y) = x + y$ *and* $\vee_{\mathbb{N}}(x, y) = x + y + 1$ *the involved dependency pairs result in the following inequalities over* $\mathbb{N}$*:*

$$\begin{aligned} 14, 17\colon &\quad x + y + z + 1 > x + y \\ 15, 18\colon &\quad x + y + z + 1 > x + z \end{aligned}$$

Hence there are no $\mathcal{C}$*-minimal rewrite sequences for any nonempty subset* $\mathcal{C} \subseteq \{14, 15, 17, 18\}$ *and we conclude that the TRS is terminating.*

Linear polynomial interpretations are insufficient to prove termination of the TRS in our leading example with the earlier modularity result in [20]. For the SCC $\{14, 15, 17, 18\}$ it identifies the rules in $\{4, 5, 6\}$ as usable. The interpretation $\vee_{\mathbb{N}}(x, y) = x + y + 1$ turns rule 6 into the identity $(x + y + 1) + z + 1 = x + (y + z + 1) + 1$, but there is no linear polynomial interpretation for $\wedge$ such that rules 4 and 5 are turned into valid weakly decreasing inequalities.

Since $\mathcal{U}(\mathcal{C})$ in general is a proper subset of $\mathcal{R}$, the condition $\mathcal{U}(\mathcal{C}) \subseteq \gtrsim$ is easier to satisfy than the condition $\mathcal{R} \subseteq \gtrsim$ of Theorem 12. What about the additional condition $\mathcal{P} \subseteq \gtrsim$, i.e., $\mathsf{cons}(x, y) \gtrsim x$ and $\mathsf{cons}(x, y) \gtrsim y$? Almost all reduction pairs that are used in termination tools can be extended to satisfy this condition. For reduction pairs that are based on simplification orders (in connection with argument filterings) this is obvious. For reduction pairs based on well-founded algebras with weakly monotone interpretations a sufficient condition for $\mathcal{P}$-compatibility is that each pair of elements of the carrier has an upper bound. For interpretations in the set $\mathbb{N}$ of natural numbers equipped with the standard order this is obviously satisfied. The necessity of the upper bound condition follows by considering the term algebra associated with the famous rule $\mathsf{f}(\mathsf{a}, \mathsf{b}, x) \to \mathsf{f}(x, x, x)$ of Toyama [41] equipped with the well-founded order $\to^+$: a and b do not have an upper bound and $\mathcal{P}$-compatibility does not hold because the union of $\mathsf{f}(\mathsf{a}, \mathsf{b}, x) \to \mathsf{f}(x, x, x)$ and $\mathcal{P}$ is not terminating. In Section 4 we introduce reduction triples based on polynomial interpretations with negative coefficients that are *not* compatible with $\mathcal{P}$.

As a matter of fact, due to the condition $\mathcal{P} \subseteq \gtrsim$, Theorem 20 provides only a sufficient condition for the absence of $\mathcal{C}$-minimal rewrite sequences. This is in contrast to Theorem 12, which provides a sufficient and necessary condition for termination. The reason is that termination of a TRS $\mathcal{R}$ is equivalent to the termination of $\mathcal{R} \cup \mathsf{DP}(\mathcal{R})$, a result due to [5] (see [34] for a simple proof based on type introduction). An example of a terminating TRS that cannot be proved terminating by the criterion of Theorem 20 is presented below.

Example 22 *Consider the terminating TRS $\mathcal{R}$ consisting of the following two rewrite rules:*

$$\begin{aligned} 1\colon&\quad \mathsf{f}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b}),x) \to \mathsf{f}(x,x,x)\\ 2\colon&\quad \mathsf{g}(\mathsf{f}(\mathsf{s}(x),\mathsf{s}(y),z)) \to \mathsf{g}(\mathsf{f}(x,y,z)) \end{aligned}$$

There are three dependency pairs:

$$\begin{aligned} 3\colon&\quad \mathsf{f}^\sharp(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b}),x) \to \mathsf{f}^\sharp(x,x,x)\\ 4\colon&\quad \mathsf{g}^\sharp(\mathsf{f}(\mathsf{s}(x),\mathsf{s}(y),z)) \to \mathsf{g}^\sharp(\mathsf{f}(x,y,z))\\ 5\colon&\quad \mathsf{g}^\sharp(\mathsf{f}(\mathsf{s}(x),\mathsf{s}(y),z)) \to \mathsf{f}^\sharp(x,y,z) \end{aligned}$$

The dependency graph contains 1 cycle: $\mathcal{C} = \{4\}$. We have $\mathcal{U}(\mathcal{C}) = \{1\}$. We claim that the conditions $\mathcal{U}(\mathcal{C}) \cup \mathcal{P} \subseteq \gtrsim$, $\mathcal{C} \subseteq \geqslant$, and $\mathcal{C} \cap > \neq \varnothing$ are not satisfied by any reduction triples $(\gtrsim, \geqslant, >)$. The reason is simply that the term $t = \mathsf{g}^\sharp(\mathsf{f}(u,u,u))$ with $u = \mathsf{s}(\mathsf{cons}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b})))$ admits the following cyclic reduction in $\mathcal{U}(\mathcal{C}) \cup \mathcal{P} \cup \mathcal{C}$:

$$\begin{aligned} t \to_{\mathcal{C}}\ & \mathsf{g}^\sharp(\mathsf{f}(\mathsf{cons}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b})),\mathsf{cons}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b})),u))\\ \to_{\mathcal{P}}\ & \mathsf{g}^\sharp(\mathsf{f}(\mathsf{s}(\mathsf{a}),\mathsf{cons}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b})),u))\\ \to_{\mathcal{P}}\ & \mathsf{g}^\sharp(\mathsf{f}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b}),u))\\ \to_{\mathcal{U}(\mathcal{C})}\ & t \end{aligned}$$

The final result [4] of this section gives two sufficient conditions that allow us to ignore $\mathcal{P}$ for cycles $\mathcal{C}$ in Theorem 20. The experimental results described in Section 6 make clear that this is very useful for the polynomial interpretations with negative coefficients introduced in the next section, which give rise to reduction triples that do not satisfy $\mathcal{P} \subseteq \gtrsim$.

Theorem 23 *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$ such that $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ is non-duplicating or $\mathcal{C}$ contains a right-ground rule. If there exists a reduction triple $(\gtrsim, \geqslant, >)$ such that $\mathcal{U}(\mathcal{C}) \subseteq \gtrsim$, $\mathcal{C} \subseteq \geqslant$, and $\mathcal{C} \cap > \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.* □

[4] This is a new result compared to the conference paper [25].

PROOF. If there are $\mathcal{C}$-minimal rewrite sequences then we obtain an infinite rewrite sequence $t_1 \to^*_{\mathcal{U}(\mathcal{C})\cup\mathcal{P}} t_2 \to_{\mathcal{C}} t_3 \to^*_{\mathcal{U}(\mathcal{C})\cup\mathcal{P}} t_4 \to_{\mathcal{C}} \cdots$ as in the proof of Theorem 20. If $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ is non-duplicating then, because the rules in $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ do not introduce cons symbols, there can be only finitely many $\mathcal{P}$-steps in this sequence. The same holds if $\mathcal{C}$ contains a right-ground rule because after applying this rule there are no cons symbols left. So there is an index n such that $t_n \to^*_{\mathcal{U}(\mathcal{C})} t_{n+1} \to_{\mathcal{C}} t_{n+2} \to^*_{\mathcal{U}(\mathcal{C})} t_{n+3} \to_{\mathcal{C}} \cdots$, contradicting the assumptions. □

4 Polynomial Interpretations with Negative Coefficients

In this section we develop polynomial interpretations with negative coefficients which can be used in connection with the dependency pair method. Polynomial interpretations that are used for direct termination proofs need to be strictly monotone in all arguments, which precludes interpretations like $x+1$ for binary function symbols. In connection with the dependency pair method, weak monotonicity is sufficient and hence $x+1$ or even 0 as interpretation of a binary function symbol causes no problems. Monotonicity is typically guaranteed by demanding that all coefficients are positive. We go a step further. We show that polynomial interpretations over the integers with negative coefficients like $x-1$ and $x-y+1$ can also be used for termination proofs.

To make the discussion more concrete, let us consider two examples.

Example 24 *Consider the TRS consisting of the following rewrite rules:*

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

The function $\mathsf{half}(x)$ *computes* $\lceil \frac{x}{2} \rceil$ *and* $\mathsf{bits}(x)$ *computes the number of bits that are needed to represent all numbers less than or equal to* x*. Termination of this TRS is proved in [6] by using the dependency pair method together with the narrowing refinement. There are three dependency pairs:*

$$\begin{array}{ll} 6\colon & \mathsf{half}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{half}^\sharp(x) \\ 7\colon & \mathsf{bits}^\sharp(\mathsf{s}(x)) \to \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x))) \\ 8\colon & \mathsf{bits}^\sharp(\mathsf{s}(x)) \to \mathsf{half}^\sharp(\mathsf{s}(x)) \end{array}$$

and the dependency graph contains two SCCs: $\{6\}$ *and* $\{7\}$*. The SCC* $\{6\}$ *is handled by the subterm criterion with the simple projection* $\pi(\mathsf{half}^\sharp) = 1$*. Consider the SCC* $\{7\}$*. The usable rules are* $\mathcal{U}(\{7\}) = \{1,2,3\}$*. By taking the polynomial interpretation* $\mathsf{0}_{\mathbb{Z}} = 0$*,* $\mathsf{half}_{\mathbb{Z}}(x) = x - 1$*,* $\mathsf{bits}_{\mathbb{Z}}(x) = \mathsf{bits}^\sharp_{\mathbb{Z}}(x) =$

$\mathsf{half}^\sharp_{\mathbb{Z}}(x) = x$, *and* $\mathsf{s}_{\mathbb{Z}}(x) = x + 1$ *over the integers, the rule and dependency pair constraints reduce to the following inequalities:*

$$1\colon\ -1 \geqslant 0 \qquad 2\colon\ 0 \geqslant 0 \qquad 3\colon\ x + 1 \geqslant x \qquad 7\colon\ x + 1 > x$$

These constraints are obviously not satisfied. Nevertheless, we will argue that the given polynomial interpretation can be used to conclude termination and, moreover, that the search for appropriate interpretations can be efficiently implemented.

The next example shows that negative coefficients in polynomial interpretations can be useful.

Example 25 *Consider the following variation of a TRS in [6]:*

$$\begin{array}{rrcl}
1\colon & 0 \leqslant y & \to & \mathsf{true} \\
2\colon & \mathsf{s}(x) \leqslant 0 & \to & \mathsf{false} \\
3\colon & \mathsf{s}(x) \leqslant \mathsf{s}(y) & \to & x \leqslant y \\
4\colon & \mathsf{mod}(0, \mathsf{s}(y)) & \to & 0 \\
5\colon & \mathsf{mod}(\mathsf{s}(x), 0) & \to & 0 \\
6\colon & \mathsf{mod}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{if}(y \leqslant x, \mathsf{mod}(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(y)), \mathsf{s}(x)) \\
7\colon & x - 0 & \to & x \\
8\colon & \mathsf{s}(x) - \mathsf{s}(y) & \to & x - y \\
9\colon & \mathsf{if}(\mathsf{true}, x, y) & \to & x \\
10\colon & \mathsf{if}(\mathsf{false}, x, y) & \to & y
\end{array}$$

There are six dependency pairs:

$$\begin{array}{rrcl}
11\colon & \mathsf{s}(x) \leqslant^\sharp \mathsf{s}(y) & \to & x \leqslant^\sharp y \\
12\colon & \mathsf{s}(x) -^\sharp \mathsf{s}(y) & \to & x -^\sharp y \\
13\colon & \mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{if}^\sharp(y \leqslant x, \mathsf{mod}(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(y)), \mathsf{s}(x)) \\
14\colon & \mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & y \leqslant^\sharp x \\
15\colon & \mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{mod}^\sharp(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(y)) \\
16\colon & \mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(x) -^\sharp \mathsf{s}(y)
\end{array}$$

The dependency graph contains three SCCs: $\{11\}$, $\{12\}$, *and* $\{15\}$. *The first two are handled by the subterm criterion (take* $\pi(\leqslant^\sharp) = 1$, *and* $\pi(-^\sharp) = 1$*). The SCC* $\{15\}$ *is problematic. The usable rules are* $\mathcal{U}(\{15\}) = \{7, 8\}$. *We need to find an* $\mathcal{P}$*-compatible reduction triple* $(\gtrsim, \geqslant, >)$ *such that rules* 7 *and* 8 *are oriented by* $\gtrsim$ *and dependency pair* 15 *is oriented by* $>$. *The only way to achieve the latter is by using the observation that* $\mathsf{s}(x)$ *is semantically greater than the syntactically larger term* $\mathsf{s}(x) - \mathsf{s}(y)$. *If we take the interpretation* $-_{\mathbb{Z}}(x, y) = x - y$, $\mathsf{s}_{\mathbb{Z}}(x) = x + 1$, *and* $0_{\mathbb{Z}} = 0$, *together with* $\mathsf{mod}^\sharp_{\mathbb{Z}}(x, y) = x$ *then we obtain the following induced ordering constraints:*

$$7\colon\ x \geqslant x \qquad 8\colon\ x - y \geqslant x - y \qquad 15\colon\ x + 1 > x - y$$

which are satisfied for all natural numbers x *and* y. *However, are we allowed to use an interpretation like* $-_{\mathbb{Z}}(x, y) = x - y$ *in termination proofs?*

In the next subsection we present the theoretical foundations for using polynomial interpretations with negative coefficients. Afterwards we discuss how to automate the proposed framework.

4.1 Theoretical Framework

When using polynomial interpretations with negative coefficients, the first challenge we face is that the standard order $>$ on $\mathbb{Z}$ is not well-founded. Restricting the domain to the set $\mathbb{N}$ of natural numbers makes an interpretation like $\mathsf{half}_{\mathcal{Z}}(x) = x - 1$ ill-defined. We propose to ensure well-definedness by modifying the interpretation of half to $\mathsf{half}_{\mathbb{N}}(x) = \max\{0, x - 1\}$.

Definition 26 *Let $\mathcal{F}$ be a signature and let $\mathcal{Z}$ be an $\mathcal{F}$-algebra over $\mathbb{Z}$. The interpretation functions of the* induced *algebra $\mathcal{N} = (\mathbb{N}, \{f_{\mathbb{N}}\}_{f \in \mathcal{F}})$ are defined as follows: $f_{\mathbb{N}}(x_1, \ldots, x_n) = \max\{0, f_{\mathbb{Z}}(x_1, \ldots, x_n)\}$ for all $x_1, \ldots, x_n \in \mathbb{N}$.*

With respect to the interpretations in Example 24, we obtain $\mathsf{s}_{\mathbb{N}}(\mathsf{half}_{\mathbb{N}}(x)) = \max\{0, \max\{0, x - 1\} + 1\} = \max\{0, x - 1\} + 1$, $\mathsf{half}_{\mathbb{N}}(\mathsf{0}_{\mathbb{N}}) = \max\{0, 0\} = 0$, and $\mathsf{half}_{\mathbb{N}}(\mathsf{s}_{\mathbb{N}}(x)) = \max\{0, \max\{0, x + 1\} - 1\} = x$.

Lemma 27 *If $\mathcal{Z}$ is an algebra such that every interpretation function is weakly monotone in all its arguments then $(\geqslant_{\mathcal{N}}, \geqslant_{\mathcal{N}}, >_{\mathcal{N}})$ is a reduction triple.*

PROOF. It is easy to show that the interpretation functions of the induced algebra are weakly monotone in all arguments. Routine arguments reveal that the relation $>_{\mathcal{N}}$ is a well-founded order which is closed under substitutions and that $\geqslant_{\mathcal{N}}$ is a preorder closed under contexts and substitutions. Moreover, the identity $>_{\mathcal{N}} \cdot \geqslant_{\mathcal{N}} = >_{\mathcal{N}}$ holds. Hence $(\geqslant_{\mathcal{N}}, \geqslant_{\mathcal{N}}, >_{\mathcal{N}})$ is a reduction triple. □

The reduction triples of Lemma 27 can be used in connection with Theorem 20 because they can be made $\mathcal{P}$-compatible by simply defining $\mathsf{cons}_{\mathbb{N}}(x, y) = \max\{x, y\}$.

Corollary 28 *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$. If there exist an algebra $\mathcal{Z}$ with weakly monotone interpretations such that $\mathcal{U}(\mathcal{C}) \cup \mathcal{C} \subseteq \geqslant_{\mathcal{N}}$ and $\mathcal{C} \cap >_{\mathcal{N}} \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.* □

Example 29 *Consider again the TRS of Example 24. For the SCC $\{7\}$ we obtain the following constraints over $\mathbb{N}$:*

$$1, 2\colon\ 0 \geqslant 0 \qquad 3\colon\ x + 1 \geqslant \max\{0, x - 1\} + 1 \qquad 7\colon\ x + 1 > x$$

Polynomial interpretations with negative constants are weakly monotone, but admitting negative coefficients like in Example 25 destroys weak monotonicity and without weak monotonicity of the interpretation functions, the order $\geqslant_{\mathcal{N}}$ is *not* closed under contexts: if $s \geqslant_{\mathcal{N}} t$ then it may happen that $C[s] \leqslant_{\mathcal{N}} C[t]$. Consequently, we do not obtain a reduction triple. However, if we have $s =_{\mathcal{N}} t$ rather than $s \geqslant_{\mathcal{N}} t$, closure under contexts is obtained for free. So for polynomial interpretations with negative coefficients we can take $(=_{\mathcal{N}}, \geqslant_{\mathcal{N}}, >_{\mathcal{N}})$ as reduction triple. This works fine in Example 25 because the induced algebra is a model of the set of usable rules $\{7, 8\}$ and $>_{\mathcal{N}}$ orients dependency pair 15.

We stress that using the reduction pair $(=_{\mathcal{N}}, >_{\mathcal{N}})$ instead of the reduction triple $(=_{\mathcal{N}}, \geqslant_{\mathcal{N}}, >_{\mathcal{N}})$ would result in the requirement that all dependency pairs in a cycle are compatible with $=_{\mathcal{N}} \cup >_{\mathcal{N}}$, which is rather restrictive because dependency pairs that are transformed into a polynomial constraint of the form $x \geqslant 0$ or $x + y \geqslant x$ cannot be handled.

Lemma 30 *Let $\mathcal{A}$ be an algebra equipped with a well-founded order $>$. The triple $(=_{\mathcal{A}}, \geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ is a reduction triple.*

PROOF. Straightforward. □

Corollary 31 *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in its dependency graph. If there exists an algebra $\mathcal{A}$ equipped with a well-founded order $>$ such that $\mathcal{R} \subseteq =_{\mathcal{A}}$, $\mathcal{C} \subseteq \geqslant_{\mathcal{A}}$, and $\mathcal{C} \cap >_{\mathcal{A}} \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.* □

The constraint $\mathcal{R} \subseteq =_{\mathcal{A}}$ in Corollary 31 means that $\mathcal{A}$ is a model of $\mathcal{R}$. Replacing it by $\mathcal{U}(\mathcal{C}) \cup \mathcal{P} \subseteq =_{\mathcal{A}}$ will not work. The reason is that $\mathcal{P}$ does not admit any nontrivial models—in any model $\mathcal{A}$ of $\mathcal{P}$ we have $x = \mathsf{cons}_A(x, y) = y$ for all x, y in the carrier A of $\mathcal{A}$—and trivial models are useless since they admit only the empty well-founded order $>$. This is a problem for the TRS in Example 25. There we have $\mathcal{U}(\{15\}) \subseteq =_{\mathcal{N}}$ but one easily checks that $\mathcal{R} \subseteq =_{\mathcal{N}}$ implies $\mathsf{mod}_{\mathbb{N}}(x, y) = x \bmod y$, which cannot be represented with polynomials of finite degree. The following example shows that replacing $\mathcal{R} \subseteq =_{\mathcal{A}}$ by $\mathcal{U}(\mathcal{C}) \subseteq =_{\mathcal{A}}$ is unsound.

Example 32 *Consider the following non-terminating TRS $\mathcal{R}$ (Toyama [41]):*

$$1\colon\ \mathsf{f}(\mathsf{a}, \mathsf{b}, x) \to \mathsf{f}(x, x, x) \qquad 2\colon\ \mathsf{g}(x, y) \to x \qquad 3\colon\ \mathsf{g}(x, y) \to y$$

The only dependency pair $\mathsf{f}^\sharp(\mathsf{a}, \mathsf{b}, x) \to \mathsf{f}^\sharp(x, x, x)$ forms a cycle in the dependency graph. There are no usable rules. If we take the polynomial interpretation $\mathsf{a}_{\mathbb{Z}} = 1$, $\mathsf{b}_{\mathbb{Z}} = 0$, and $\mathsf{f}^\sharp_{\mathbb{Z}}(x, y, z) = x - y$ then $\mathsf{f}^\sharp(\mathsf{a}, \mathsf{b}, x) >_{\mathcal{N}} \mathsf{f}^\sharp(x, x, x)$ as the dependency pair is transformed into $1 - 0 = 1 > 0 = x - x$.

The problem in the above example is that the non-right-linearity of $\mathcal{C}$, which is essential for non-termination, is eliminated by applying the interpretation. In the following we prove that we may replace $\mathcal{R} \subseteq =_{\mathcal{A}}$ by $\mathcal{U}(\mathcal{C}) \subseteq =_{\mathcal{A}}$, provided $>$ is a well-order (i.e., a total well-founded order) and, more importantly, $\mathcal{A}$ regards $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ as right-linear. [5] The latter concept is defined as follows.

Definition 33 *A linear term s is called a* linearization *of a term t if $s\sigma = t$ for some substitution σ that maps variables to variables. Let $\mathcal{A}$ be an algebra. A TRS $\mathcal{R}$ is* $\mathcal{A}$-right-linear *if for every rule $l \to r \in \mathcal{R}$ there exists a linearization r' of r such that $r =_{\mathcal{A}} r'$.*

Example 34 *The dependency pair $\mathsf{f}^\sharp(\mathsf{a}, \mathsf{b}, x) \to \mathsf{f}^\sharp(x, x, x)$ in Example 32 is not $\mathcal{N}$-right-linear with respect to the induced algebra $\mathcal{N}$ as*

$$\mathsf{f}^\sharp_{\mathbb{N}}(x, x, x) = \max\{0, x - x\} = 0$$

and $\mathsf{f}^\sharp_{\mathbb{N}}(x', y', z') \neq 0$ for every linearization $\mathsf{f}^\sharp(x', y', z')$ of $\mathsf{f}^\sharp(x, x, x)$. For instance, $\mathsf{f}^\sharp_{\mathbb{N}}(y, x, z) = \max\{0, y - x\}$. Consider the SCC $\mathcal{C} = \{15\}$ in Example 25. We claim that $\mathcal{N}$ regards $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ as right-linear. For the rules in $\mathcal{U}(\mathcal{C}) = \{7, 8\}$ this is clear as they are right-linear. Because of the interpretation $\mathsf{mod}^\sharp_{\mathbb{Z}}(x, y) = x$, we have

$$\mathsf{mod}^\sharp(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(y)) =_{\mathcal{N}} \mathsf{s}(x) - \mathsf{s}(y) =_{\mathcal{N}} \mathsf{mod}^\sharp(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(z))$$

and thus the single rule in $\mathcal{C}$ is regarded as right-linear.

The following definition introduces a new algebraic construction that is used to prove the desired result.

Definition 35 *Let $\mathcal{F}$ be a signature and let $\mathcal{A} = (A, \{f_A\}_{f \in \mathcal{F}})$ be an $\mathcal{F}$-algebra equipped with a well-order $>$. Let S be the set of all nonempty finite subsets of A. We define the* set extension *of $\mathcal{A}$ as the $(\mathcal{F} \cup \{\mathsf{cons}\})$-algebra $\mathcal{S}$ with carrier S and interpretations $\mathsf{cons}_{\mathcal{S}}(X, Y) = X \cup Y$ and*

$$f_S(X_1, \ldots, X_n) = \{f_A(x_1, \ldots, x_n) \mid x_1 \in X_1, \ldots, x_n \in X_n\}$$

for all $f \in \mathcal{F}$. The relations $\supseteq_{\mathcal{S}}$, $\geqslant_{\mathcal{S}}$, and $>_{\mathcal{S}}$ are defined on terms as follows:

$$\begin{array}{ll} s \supseteq_{\mathcal{S}} t & \text{if } [\alpha]_{\mathcal{S}}(s) \supseteq [\alpha]_{\mathcal{S}}(t) \\ s \geqslant_{\mathcal{S}} t & \text{if } \max([\alpha]_{\mathcal{S}}(s)) \geqslant \max([\alpha]_{\mathcal{S}}(t)) \\ s >_{\mathcal{S}} t & \text{if } \max([\alpha]_{\mathcal{S}}(s)) > \max([\alpha]_{\mathcal{S}}(t)) \end{array}$$

for all assignments $\alpha \colon \mathcal{V} \to S$.

Note that since $[\alpha]_{\mathcal{S}}(u)$ is a finite nonempty set for every term u and $>$ is a well-order, the relations $\geqslant_{\mathcal{S}}$ and $>_{\mathcal{S}}$ are well-defined.

[5] This is a new result compared to the conference papers [24,25].

Lemma 36 *The triple* $(\supseteq_{\mathcal{S}}, \geqslant_{\mathcal{S}}, >_{\mathcal{S}})$ *is a* $\mathcal{P}$*-compatible reduction triple.*

PROOF. The relations $\supseteq_{\mathcal{S}}$ and $\geqslant_{\mathcal{S}}$ are clearly preorders. Closure under contexts of $\supseteq_{\mathcal{S}}$ follows because all interpretations in $\mathcal{S}$ are weakly monotone with respect to set inclusion. We show that $\supseteq_{\mathcal{S}}$ is closed under substitutions. Suppose that $s \supseteq_{\mathcal{S}} t$ and let σ be a substitution. Let $\alpha\colon \mathcal{V} \to \mathcal{S}$ be an arbitrary assignment. We have to show that $[\alpha]_{\mathcal{S}}(s\sigma) \supseteq [\alpha]_{\mathcal{S}}(t\sigma)$. Define the assignment β as $\beta(x) = [\alpha]_{\mathcal{S}}(x\sigma)$ for all $x \in \mathcal{V}$. It is not difficult to show that $[\alpha]_{\mathcal{S}}(s\sigma) = [\beta]_{\mathcal{S}}(s)$ and $[\alpha]_{\mathcal{S}}(t\sigma) = [\beta]_{\mathcal{S}}(t)$. The assumption $s \supseteq_{\mathcal{S}} t$ yields $[\beta]_{\mathcal{S}}(s) \supseteq [\beta]_{\mathcal{S}}(t)$. Closure under substitutions of $\geqslant_{\mathcal{S}}$ and $>_{\mathcal{S}}$ follows in the same way. The relation $>_{\mathcal{S}}$ is a strict partial order. It inherits well-foundedness from $>$. Since $\supseteq_{\mathcal{S}} \subseteq \geqslant_{\mathcal{S}}$ and $\geqslant_{\mathcal{S}} \cdot >_{\mathcal{S}} = >_{\mathcal{S}}$, compatibility holds. We have $\mathcal{P} \subseteq \supseteq_{\mathcal{S}}$ by the definition of $\mathsf{cons}_{\mathcal{S}}$. $\square$

The next example illustrates the difference between $>_{\mathcal{N}}$ and $>_{\mathcal{S}}$.

Example 37 *Consider again the TRS and the interpretation of Example 32. If we take an assignment* α *with* $\alpha(x) = \{0, 1\}$ *then* $[\alpha]_{\mathcal{S}}(\mathsf{a}) = \mathsf{a}_S = \{\mathsf{a}_A\} = \{1\}$*,* $[\alpha]_{\mathcal{S}}(\mathsf{b}) = \mathsf{b}_S = \{\mathsf{b}_A\} = \{0\}$*,* $[\alpha]_{\mathcal{S}}(x) = \alpha(x) = \{0, 1\}$ *and thus*

$$\begin{aligned}[\alpha]_{\mathcal{S}}(\mathsf{f}^\sharp(\mathsf{a}, \mathsf{b}, x)) &= \mathsf{f}^\sharp_S([\alpha]_{\mathcal{S}}(\mathsf{a}), [\alpha]_{\mathcal{S}}(\mathsf{b}), [\alpha]_{\mathcal{S}}(x)) = \mathsf{f}^\sharp_S(\{1\}, \{0\}, \{0, 1\}) \\ &= \{\mathsf{f}^\sharp_A(1, 0, 0), \mathsf{f}^\sharp_A(1, 0, 1)\} = \{1\}\end{aligned}$$

and

$$\begin{aligned}[\alpha]_{\mathcal{S}}(\mathsf{f}^\sharp(x, x, x)) &= \mathsf{f}^\sharp_S(\{0, 1\}, \{0, 1\}, \{0, 1\}) \\ &= \{\mathsf{f}^\sharp_A(x_1, x_2, x_3) \mid x_1, x_2, x_3 \in \{0, 1\}\} = \{0, 1\}\end{aligned}$$

Hence $\mathsf{f}^\sharp(\mathsf{a}, \mathsf{b}, x) >_{\mathcal{S}} \mathsf{f}^\sharp(x, x, x)$ *does not hold.*

In the following lemma $\beta \in \alpha$ abbreviates "$\beta(x) \in \alpha(x)$ for all $x \in \mathcal{V}$".

Lemma 38 *Let* t *be a term. For every assignment* $\alpha\colon \mathcal{V} \to S$ *we have*

$$[\alpha]_{\mathcal{S}}(t) \supseteq \{[\beta]_{\mathcal{A}}(t) \mid \beta \in \alpha\}.$$

Moreover, if t *is linear then the reverse inclusion also holds.*

PROOF. We show the result by induction on t. If t is a variable then $[\beta]_{\mathcal{A}}(t) = \beta(t) \in \alpha(t) = [\alpha]_{\mathcal{S}}(t)$ and thus

$$[\alpha]_{\mathcal{S}}(t) = \alpha(t) = \{\beta(t) \mid \beta \in \alpha\} = \{[\beta]_{\mathcal{A}}(t) \mid \beta \in \alpha\}.$$

Suppose $t = f(t_1, \ldots, t_n)$ and let $\beta \in \alpha$. The induction hypothesis yields $[\beta]_{\mathcal{A}}(t_i) \in [\alpha]_{\mathcal{S}}(t_i)$ for all $i \in \{1, \ldots, n\}$. Hence, by the definition of f_S,

$$[\beta]_{\mathcal{A}}(t) = f_A([\beta]_{\mathcal{A}}(t_1), \ldots, [\beta]_{\mathcal{A}}(t_n)) \in f_S([\alpha]_{\mathcal{S}}(t_1), \ldots, [\alpha]_{\mathcal{S}}(t_n)) = [\alpha]_{\mathcal{S}}(t).$$

Now suppose that t is linear. We show the reverse inclusion

$$[\alpha]_{\mathcal{S}}(t) \subseteq \{[\beta]_{\mathcal{A}}(t) \mid \beta \in \alpha\}$$

The induction hypothesis yields

$$[\alpha]_{\mathcal{S}}(t_i) \subseteq \{[\beta]_{\mathcal{A}}(t_i) \mid \beta \in \alpha\}$$

for all $i \in \{1, \ldots, n\}$. Because t is linear, the variables in $t_1, \ldots, t_n$ are pairwise disjoint and hence $[\alpha]_{\mathcal{S}}(t_1) \times \cdots \times [\alpha]_{\mathcal{S}}(t_n) \subseteq \{([\beta]_{\mathcal{A}}(t_1), \ldots, [\beta]_{\mathcal{A}}(t_n)) \mid \beta \in \alpha\}$. Consequently,

$$\begin{aligned}
[\alpha]_{\mathcal{S}}(t) &= \{f_A(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in [\alpha]_{\mathcal{S}}(t_1) \times \cdots \times [\alpha]_{\mathcal{S}}(t_n)\} \\
&\subseteq \{f_A([\beta]_{\mathcal{A}}(t_1), \ldots, [\beta]_{\mathcal{A}}(t_n)) \mid \beta \in \alpha\} \\
&= \{[\beta]_{\mathcal{A}}(t) \mid \beta \in \alpha\}
\end{aligned}$$

□

Set equality in the above lemma is not guaranteed without the linearity of t. This can be seen from Example 37 where $[\alpha]_{\mathcal{S}}(\mathsf{f}^\sharp(x, x, x)) = \{0, 1\}$ and $\{[\beta]_{\mathcal{A}}(\mathsf{f}^\sharp(x, x, x)) \mid \beta \in \alpha\} = \{0\}$.

The following lemma relates interpretations in $\mathcal{A}$ to interpretations in $\mathcal{S}$.

Lemma 39 *Let $l \to r$ be an $\mathcal{A}$-right-linear rewrite rule.*

- *If $l =_{\mathcal{A}} r$ then $l \trianglerighteq_{\mathcal{S}} r$.*
- *If $l \geqslant_{\mathcal{A}} r$ then $l \geqslant_{\mathcal{S}} r$.*
- *If $l >_{\mathcal{A}} r$ then $l >_{\mathcal{S}} r$.*

PROOF. Let r' be a linearization of r such that $r' =_{\mathcal{A}} r$ and let σ be a substitution such that $r'\sigma = r$. We may assume that σ only affects the (fresh) variables in $\mathcal{V}\mathsf{ar}(r') \setminus \mathcal{V}\mathsf{ar}(r)$. In particular, $l = l\sigma$. Since the relations $\trianglerighteq_{\mathcal{S}}$, $\geqslant_{\mathcal{S}}$, and $>_{\mathcal{S}}$ are closed under substitutions it is sufficient to show $l \trianglerighteq_{\mathcal{S}} r'$, $l \geqslant_{\mathcal{S}} r'$, and $l >_{\mathcal{S}} r'$ under the stated conditions. We have $[\alpha]_{\mathcal{S}}(l) \supseteq \{[\beta]_{\mathcal{A}}(l) \mid \beta \in \alpha\}$ and $[\alpha]_{\mathcal{S}}(r') = \{[\beta]_{\mathcal{A}}(r') \mid \beta \in \alpha\} = \{[\beta]_{\mathcal{A}}(r) \mid \beta \in \alpha\}$ according to Lemma 38. Now, if $l =_{\mathcal{A}} r$ then $[\beta]_{\mathcal{A}}(l) = [\beta]_{\mathcal{A}}(r)$ for all assignments β and thus $[\alpha]_{\mathcal{S}}(l) \supseteq [\alpha]_{\mathcal{S}}(r')$. Hence $l \trianglerighteq_{\mathcal{S}} r'$ by definition. Next, if $l \geqslant_{\mathcal{A}} r$ then $[\beta]_{\mathcal{A}}(l) \geqslant [\beta]_{\mathcal{A}}(r)$ for

all assignments β. Hence

$$\begin{aligned}\max([\alpha]_{\mathcal{S}}(l)) &\geqslant \max\{[\beta]_{\mathcal{A}}(l) \mid \beta \in \alpha\}\\ &\geqslant \max\{[\beta]_{\mathcal{A}}(r) \mid \beta \in \alpha\}\\ &= \max([\alpha]_{\mathcal{S}}(r'))\end{aligned}$$

and thus $l \geqslant_{\mathcal{S}} r'$ by definition. The proof that $l >_{\mathcal{S}} r$ whenever $l >_{\mathcal{A}} r$ follows in exactly the same way. □

We are now ready for the usable rules criterion announced earlier.

Theorem 40 *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in its dependency graph. If there exists an algebra $\mathcal{A}$ equipped with a well-order $>$ such that $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ is $\mathcal{A}$-right-linear, $\mathcal{U}(\mathcal{C}) \subseteq =_{\mathcal{A}}$, $\mathcal{C} \subseteq \geqslant_{\mathcal{A}}$, and $\mathcal{C} \cap >_{\mathcal{A}} \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.*

PROOF. From Lemma 39 we obtain $\mathcal{U}(\mathcal{C}) \subseteq \trianglerighteq_{\mathcal{S}}$, $\mathcal{C} \subseteq \geqslant_{\mathcal{S}}$, and $\mathcal{C} \cap >_{\mathcal{S}} \neq \varnothing$. Lemma 36 states that $(\trianglerighteq_{\mathcal{S}}, \geqslant_{\mathcal{S}}, >_{\mathcal{S}})$ is a reduction triple and $\mathcal{P} \subseteq \trianglerighteq_{\mathcal{S}}$. Hence the conditions of Theorem 20 are satisfied and the result follows. □

Note that the set extension is only used in the proof of Theorem 40. One can formulate a version of Theorem 40 that uses the reduction triple $(\trianglerighteq_{\mathcal{S}}, \geqslant_{\mathcal{S}}, >_{\mathcal{S}})$ instead of $(=_{\mathcal{A}}, \geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ in its statement. This may result in a stronger result (as the $\mathcal{A}$-right-linearity condition disappears from sight) but we expect that verifying the constraints $\mathcal{U}(\mathcal{C}) \subseteq \trianglerighteq_{\mathcal{S}}$, $\mathcal{C} \subseteq \geqslant_{\mathcal{S}}$, and $\mathcal{C} \cap >_{\mathcal{S}} \neq \varnothing$ will be much harder.

Example 41 *Consider again the problematic SCC $\mathcal{C} = \{15\}$ of Example 25. In Example 34 we observed that the induced algebra over $\mathbb{N}$ regards $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ as right-linear. Hence Theorem 40 is applicable. The induced ordering constraints*

$$\begin{array}{lll} 7\colon & x \geqslant x & 15\colon\ x+1 > \max\{0, x-y\} \\ 8\colon & \max\{0, x-y\} \geqslant \max\{0, x-y\} & \end{array}$$

are satisfied and therefore we can finally conclude the termination of the TRS in Example 25.

4.2 Towards Automation

How can an inequality like $x + 1 \geqslant \max\{0, x - 1\} + 1$ in Example 29 or $x + 1 > \max\{0, x - y\}$ in Example 41 be verified automatically? Because inequalities resulting from interpretations with negative coefficients may contain

the max operator, we cannot use standard techniques (cf. [28]) for comparing polynomial expressions. In order to avoid reasoning by case analysis ($x-1>0$ or $x-1\leqslant 0$ for the above inequality), we approximate the evaluation function of the induced algebra.

First we present an approach for weakly monotone polynomial interpretations in connection with Corollary 28.

Definition 42 *Given a polynomial P with coefficients in $\mathbb{Z}$, we denote the constant part by $c(P)$ and the non-constant part $P-c(P)$ by $n(P)$. Let $\mathcal{Z}$ be an $\mathcal{F}$-algebra such that every $f_{\mathbb{Z}}$ is a weakly monotone polynomial. With every term t we associate polynomials $P_{left}(t)$ and $P_{right}(t)$ with coefficients in $\mathbb{Z}$ and variables in t as indeterminates:*

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

where $P_1=f_{\mathbb{Z}}(P_{left}(t_1),\ldots,P_{left}(t_n))$ and

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

where $P_2=f_{\mathbb{Z}}(P_{right}(t_1),\ldots,P_{right}(t_n))$.

The mapping P_{right} over-approximates the evaluation function of the induced algebra by removing negative constants. The mapping P_{left} provides an under-approximation by estimating $\max\{0,P\}$ with P provided P is not a negative constant.

Example 43 *Consider again the TRS of Example 24. Since $\mathsf{half}_{\mathcal{Z}}(0)=-1$ and both $n(-1)=0$ and $c(-1)<0$, we have $P_{left}(\mathsf{half}(0))=0$. By applying P_{left} to the left-hand sides and P_{right} to the right-hand sides of the rules in $\{1,2,3,7\}$, the following ordering constraints are obtained:*

$$1,2\colon\ 0\geqslant 0 \qquad 3\colon\ x+1\geqslant x+1 \qquad 7\colon\ x+1>x$$

The only difference with the constraints in Example 29 is the interpretation of the term $\mathsf{s}(\mathsf{half}(x))$ on the right-hand side of rule 3. We have $P_{right}(\mathsf{half}(x))=n(x-1)=x$ and thus $P_{right}(\mathsf{s}(\mathsf{half}(x)))=x+1$. Although $x+1$ is less precise than $\max\{0,x-1\}+1$, it is accurate enough to solve the ordering constraint resulting from rule 3.

Although $P_{left}(t)$ may have negative coefficients, we always assume that the indeterminates in $P_{left}(t)$ (and in $P_{right}(t)$) range over the natural numbers.

According the following lemma, $P_{left}(t)$ provides an under-approximation and $P_{right}(t)$ an over-approximation of the interpretation of t in the induced algebra.

Lemma 44 *Let $\mathcal{Z}$ be an $\mathcal{F}$-algebra such that every interpretation function is a weakly monotone polynomial. Let t be a term. For every assignment $\alpha\colon \mathcal{V} \to \mathbb{N}$ we have $\alpha(P_{right}(t)) \geqslant [\alpha]_{\mathcal{N}}(t) \geqslant \alpha(P_{left}(t))$.*

PROOF. By induction on the structure of t. If $t \in \mathcal{V}$ then $\alpha(P_{right}(t)) = \alpha(P_{left}(t)) = [\alpha]_{\mathcal{N}}(t) = \alpha(t)$. Suppose $t = f(t_1, \ldots, t_n)$. Let P_1 and P_2 as in Definition 42. We have $\alpha(P_1) = f_{\mathbb{Z}}(\alpha(P_{left}(t_1)), \ldots, \alpha(P_{left}(t_n)))$ and $\alpha(P_2) = f_{\mathbb{Z}}(\alpha(P_{right}(t_1)), \ldots, \alpha(P_{right}(t_n)))$. According to the induction hypothesis,

$$\alpha(P_{right}(t_i)) \geqslant [\alpha]_{\mathcal{N}}(t_i) \geqslant \alpha(P_{left}(t_i))$$

for all i. Since $f_{\mathbb{Z}}$ is weakly monotone,

$$\alpha(P_2) \geqslant f_{\mathbb{Z}}([\alpha]_{\mathcal{N}}(t_1), \ldots, [\alpha]_{\mathcal{N}}(t_n)) \geqslant \alpha(P_1)$$

By applying the weakly monotone function $\max\{0, \cdot\}$ we obtain

$$\max\{0, \alpha(P_2)\} \geqslant [\alpha]_{\mathcal{N}}(t) \geqslant \max\{0, \alpha(P_1)\}$$

We have

$$\alpha(P_{left}(t)) = \begin{cases} 0 & \text{if } n(P_1) = 0 \text{ and } c(P_1) < 0 \\ \alpha(P_1) & \text{otherwise} \end{cases}$$

and thus $\alpha(P_{left}(t)) \leqslant \max\{0, \alpha(P_1)\}$. Likewise,

$$\alpha(P_{right}(t)) = \begin{cases} \alpha(n(P_2)) & \text{if } c(P_2) < 0 \\ \alpha(P_2) & \text{otherwise} \end{cases}$$

In the former case, $\alpha(n(P_2)) = \alpha(P_2) - c(P_2) > \alpha(P_2)$ and $\alpha(n(P_2)) \geqslant 0$. The latter inequality is a consequence of the fact that $P_{right}(t)$ has no negative coefficients, which is easily proved by induction on the structure of t. In the latter case $\alpha(P_2) \geqslant 0$. So in both cases we have $\alpha(P_{right}(t)) \geqslant \max\{0, \alpha(P_2)\}$. Hence we obtain the desired inequalities. □

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

Corollary 45 *Let $\mathcal{Z}$ be an $\mathcal{F}$-algebra such that every $f_{\mathbb{Z}}$ is a weakly monotone polynomial. Let s and t be terms. If $P_{left}(s) - P_{right}(t) > 0$ then $s >_{\mathcal{N}} t$. If $P_{left}(s) - P_{right}(t) \geqslant 0$ then $s \geqslant_{\mathcal{N}} t$.* □

The remaining question is how to find suitable interpretations for the function symbols. This problem will be discussed in Section 5.

The approximation for negative constants is typically useful for handling destructors like predecessor and the tail function on lists:

$$\begin{array}{rr} \mathsf{p}(\mathsf{0}) \to \mathsf{0} & \mathsf{tail}([\,]) \to [\,] \\ \mathsf{p}(\mathsf{s}(x)) \to x & \mathsf{tail}(x : xs) \to xs \end{array}$$

The limitation of the approximation becomes clear when trying to prove $\mathsf{p}(x) \geqslant_{\mathcal{N}} \mathsf{p}(x)$ or $\mathsf{p}(x) \geqslant_{\mathcal{N}} \mathsf{0}$. More importantly, the approach developed so far cannot handle inequalities like $x + 1 > \max\{0, x - y\}$ in Example 41 that originate from polynomial interpretations with negative coefficients. The reason is that Lemma 44 relies essentially on weak monotonicity of the polynomial interpretations. Suppose that we would modify the definition of P_{right} such that on input $t = f(t_1, \ldots, t_n)$ it returns the non-negative part $N(P_2)$ of P_2 when P_2 contains negative coefficients. Then we would wrongly conclude $\mathsf{0} \geqslant_{\mathcal{N}} \mathsf{s}(\mathsf{0}) - (\mathsf{s}(\mathsf{0}) - x)$ as

$$P_{\mathit{left}}(\mathsf{0}) = 0 = N(1 - 1) = N(1 - N(1 - x)) = P_{\mathit{right}}(\mathsf{s}(\mathsf{0}) - (\mathsf{s}(\mathsf{0}) - x))$$

Because P_{left} and P_{right} are approximations, they also cannot be used for checking equalities.

We now present an approach that can be used checking equalities and inequalities in connection with Corollary 31 and Theorem 40 for polynomial interpretations with negative coefficients. This new approach does not subsume the approximation introduced above for polynomial interpretations with negative constants. In particular, it cannot cope with a constraint like $x \geqslant_{\mathcal{N}} \mathsf{p}(x)$.

Let $\mathcal{P}_{\geqslant 0}$ be a subset of the set of polynomials with integral coefficients such that $\alpha(P) \geqslant 0$ for all $P \in \mathcal{P}_{\geqslant 0}$ and all $\alpha \colon \mathcal{V} \to \mathbb{N}$ and such that membership in $\mathcal{P}_{\geqslant 0}$ is decidable. For instance, $\mathcal{P}_{\geqslant 0}$ could be the set of polynomials without negative coefficients. We define $\mathcal{P}_{<0}$ in the same way.

In the following definition we define a polynomial $Q(t)$ for every term t. Unlike $P_{\mathit{left}}(t)$ and $P_{\mathit{right}}(t)$, $Q(t)$ contains enough information to compute the exact value of $[\alpha]_{\mathcal{N}}(t)$. This latter property is formally expressed in Theorem 48.

Definition 46 *Let $\mathcal{Z}$ be an algebra. With every term t we associate a polynomial $Q(t)$ as follows:*

$$Q(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ P & \text{if } t = f(t_1, \ldots, t_n) \text{ and } P \in \mathcal{P}_{\geqslant 0} \\ 0 & \text{if } t = f(t_1, \ldots, t_n) \text{ and } P \in \mathcal{P}_{<0} \\ v(P) & \text{otherwise} \end{cases}$$

where $P = f_{\mathbb{Z}}(Q(t_1), \ldots, Q(t_n))$. *In the last clause* $v(P)$ *denotes a fresh* abstract variable *that we uniquely associate with* P.

The polynomial $Q(t)$ is computed by recursively interpreting t in the given algebra. If at any stage a polynomial is encountered that belongs to $\mathcal{P}_{<0}$, it is replaced by 0. A polynomial whose status cannot be determined is replaced by an abstract variable. There are two kinds of indeterminates in $Q(t)$: ordinary variables occurring in t and abstract variables. The intuitive meaning of an abstract variable $v(P)$ is $\max\{0, P\}$. The latter quantity is always non-negative, like an ordinary variable ranging over the natural numbers, but from $v(P)$ we can extract the original polynomial P and this information may be crucial for a comparison between two polynomial expressions to succeed. Note that the polynomial P associated with an abstract variable $v(P)$ may contain other abstract variables. However, because $v(P)$ is different from previously selected abstract variables (in recursive calls to Q), there are no spurious loops like $P_1 = v(x - v(P_2))$ and $P_2 = v(x - v(P_1))$.

The reason for using $\mathcal{P}_{\geqslant 0}$ and $\mathcal{P}_{<0}$ in the above definition is to make our approach independent of the particular method that is used to test non-negativeness or negativeness of polynomials.

Definition 47 *With every assignment* $\alpha\colon \mathcal{V} \to \mathbb{N}$ *we associate an assignment* $\alpha^*\colon \mathcal{V} \to \mathbb{N}$ *defined as follows:*

$$\alpha^*(x) = \begin{cases} \max\{0, \alpha^*(P)\} & \text{if } x \text{ is an abstract variable } v(P) \\ \alpha(x) & \text{otherwise} \end{cases}$$

The above definition is recursive because P may contain abstract variables. However, since $v(P)$ is different from previously selected abstract variables, the recursion terminates and it follows that α^* is well-defined.

Theorem 48 *Let* $\mathcal{Z}$ *be an algebra such that every* $f_{\mathbb{Z}}$ *is a polynomial. Let* t *be a term. For every assignment* $\alpha\colon \mathcal{V} \to \mathbb{N}$ *we have* $[\alpha]_{\mathcal{N}}(t) = \alpha^*(Q(t))$.

PROOF. We show that $[\alpha]_{\mathcal{N}}(t) = \alpha^*(Q(t))$ by induction on t. If t is a variable then $[\alpha]_{\mathcal{N}}(t) = \alpha(t) = \alpha^*(t) = \alpha^*(Q(t))$. Suppose $t = f(t_1, \ldots, t_n)$. Let $P = f_{\mathbb{Z}}(Q(t_1), \ldots, Q(t_n))$. The induction hypothesis yields $[\alpha]_{\mathcal{N}}(t_i) = \alpha^*(Q(t_i))$ for all i and thus

$$\begin{aligned} [\alpha]_{\mathcal{N}}(t) &= f_{\mathbb{N}}(\alpha^*(Q(t_1)), \ldots, \alpha^*(Q(t_n))) \\ &= \max\{0, f_{\mathbb{Z}}(\alpha^*(Q(t_1)), \ldots, \alpha^*(Q(t_n)))\} = \max\{0, \alpha^*(P)\} \end{aligned}$$

We distinguish three cases, corresponding to the definition of $Q(t)$.

- First suppose that $P \in \mathcal{P}_{\geqslant 0}$. This implies that $\alpha^*(P) \geqslant 0$ and thus we have $\max\{0, \alpha^*(P)\} = \alpha^*(P)$. Hence $[\alpha]_{\mathcal{N}}(t) = \alpha^*(P) = \alpha^*(Q(t))$.
- Next suppose that $P \in \mathcal{P}_{<0}$. So $\alpha^*(P) < 0$ and thus $\max\{0, \alpha^*(P)\} = 0$. Hence $[\alpha]_{\mathcal{N}}(t) = 0 = \alpha^*(Q(t))$.
- In the remaining case we do not know the status of P. We have $Q(t) = v(P)$ and thus $\alpha^*(Q(t)) = \max\{0, \alpha^*(P)\}$ which immediately yields the desired identity $[\alpha]_{\mathcal{N}}(t) = \alpha^*(Q(t))$. $\Box$

Corollary 49 *Let $\mathcal{Z}$ be an algebra such that every $f_{\mathbb{Z}}$ is a polynomial. Let s and t be terms. If $Q(s) = Q(t)$ then $s =_{\mathcal{N}} t$. If $\alpha^*(Q(s) - Q(t)) > 0$ for all assignments $\alpha\colon \mathcal{V} \to \mathbb{N}$ then $s >_{\mathcal{N}} t$. If $\alpha^*(Q(s)-Q(t)) \geqslant 0$ for all assignments $\alpha\colon \mathcal{V} \to \mathbb{N}$ then $s \geqslant_{\mathcal{N}} t$.* $\Box$

Corollary 49 can be used to verify equalities.

Example 50 *Consider rule 8 in Example 25. We have $\mathsf{s}(x) - \mathsf{s}(y) =_{\mathcal{N}} x - y$ because $Q(\mathsf{s}(x)-\mathsf{s}(y)) = v(x-y) = Q(x-y)$. Next consider dependency pair 15. We have $Q(\mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y))) = x+1$ and $Q(\mathsf{mod}^\sharp(\mathsf{s}(x)-\mathsf{s}(y), \mathsf{s}(y))) = v(x-y)$. Since $x+1-v(x-y)$ may be negative (when interpreting $v(x-y)$ as a variable), the above corollary cannot be used to conclude that 15 is strictly decreasing. However, if we estimate $v(x - y)$ by x, the non-negative part of $x - y$, then we obtain $x + 1 - x = 1$ which is clearly positive.*

If non-negativeness of $Q = Q(s)-Q(t)$ cannot be shown, we select an abstract variable in Q and apply the following lemma.

Given a polynomial P with coefficients in $\mathbb{Z}$, we denote the non-negative part of P by $N(P)$.

Lemma 51 *Let Q be a polynomial with integer coefficients. Suppose $v(P)$ is an abstract variable that occurs in Q but not in $N(Q)$. If Q' is the polynomial obtained from Q by replacing $v(P)$ with $N(P)$ then $\alpha^*(Q) \geqslant \alpha^*(Q')$ for all assignments $\alpha\colon \mathcal{V} \to \mathbb{N}$.*

PROOF. Let $\alpha\colon \mathcal{V} \to \mathbb{N}$ be an arbitrary assignment. In $\alpha^*(Q)$ every occurrence of $v(P)$ is assigned the value $\alpha^*(v(P)) = \max\{0, \alpha^*(P)\}$. We have $\alpha^*(N(P)) \geqslant \alpha^*(P)$ and $\alpha^*(N(P)) \geqslant 0$. These two facts imply $\alpha^*(N(P)) \geqslant \alpha^*(v(P))$. By assumption, $v(P)$ occurs only in the negative part of Q. Hence Q is (strictly) anti-monotone in $v(P)$ and therefore $\alpha^*(Q) \geqslant \alpha^*(Q')$. $\Box$

In order to determine whether $s \geqslant_{\mathcal{N}} t$ (or $s >_{\mathcal{N}} t$) holds, the idea now is to first use standard techniques to test the non-negativeness of $Q = Q(s) - Q(t)$ (i.e., we determine whether $\alpha(Q) \geqslant 0$ for all assignments α by checking whether $Q \in \mathcal{P}_{\geqslant 0}$). If Q is non-negative then we certainly have $\alpha^*(Q) \geqslant 0$ for all

assignments α and thus $s \geqslant_{\mathcal{N}} t$ follows from Corollary 49. If non-negativeness cannot be shown then we apply the previous lemma to replace an abstract variable that occurs only in the negative part of Q. The resulting polynomial Q' is tested for non-negativeness. If the test succeeds then for all assignments α we have $\alpha^*(Q') \geqslant 0$ and thus also $\alpha^*(Q) \geqslant 0$ by the previous lemma. According to Corollary 49 this is sufficient to conclude $s \geqslant_{\mathcal{N}} t$. Otherwise we repeat the above process with Q'. The process terminates when there are no more abstract variables left that appear only in the negative part of the current polynomial.

5 Tyrolean Termination Tool

The techniques introduced in the preceding sections have been implemented in the Tyrolean Termination Tool. T$_{\text{T}}$T produces textbook quality output and has a convenient web interface. The tool is available at

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

In contrast to its predecessor, the Tsukuba Termination Tool [23], it is possible to run the tool in *fully automatic mode* on a *collection* of rewrite systems. Moreover, besides ordinary (first-order) rewrite systems, the tool accepts simply-typed applicative rewrite systems which are transformed into ordinary rewrite systems by the recent method of Aoto and Yamada [3].

In the next subsection we describe the fully automatic mode. Section 5.2 describes the differences between the semi automatic mode and the Tsukuba Termination Tool. In Section 5.3 we show a termination proof of a simply-typed applicative system obtained by T$_{\text{T}}$T. In Section 5.4 we describe how to input a collection of rewrite systems and how to interpret the resulting output. Some implementation details are given in Section 5.5.

5.1 Fully Automatic Mode

In this mode T$_{\text{T}}$T uses a simple strategy to solve (recursively) the ordering constraints for each SCC of the approximated dependency graph. The strategy is based on the new features described in the previous sections and uses LPO (both with strict and quasi-precedence) with *some* argument filterings [26] and mostly linear polynomial interpretations with coefficients from $\{-1, 0, 1\}$ as base orders.

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

(1) First we check whether the *subterm criterion* is applicable.
(2) If the subterm criterion was unsuccessful, we compute the *usable rules*.
(3) The resulting (usable rules and dependency pairs) constraints are subjected to the *natural* (see below) polynomial interpretation.
(4) If the constraints could not be solved in step 3, we employ the *divide and conquer* algorithm for computing suitable argument filterings with respect to the *some* heuristic [26] and the lexicographic path order (LPO) with *strict* precedence.
(5) If the previous step was unsuccessful, we repeat step 3 with *arbitrary* linear polynomial interpretations with coefficients from $\{0, 1\}$.
(6) Next we repeat step 4 with the variant of LPO based on *quasi-precedences* and a small increase in the search space for argument filterings (see below).
(7) If the constraints could still not be solved, we try polynomial interpretations with negative constants.
(8) As a last resort, we use polynomial interpretations with coefficients from $\{-1, 0, 1\}$ in connection with Theorems 40 and 23. If the latter fails, due to the $\mathcal{A}$-right-linearity restriction, we give Corollary 31 a try.

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

Taking the following polynomial interpretations for certain function symbols that appear in many example TRSs is what we call *natural* (for other function symbols we take linear interpretations with coefficients from $\{0, 1\}$):

$$\begin{array}{cccc} \mathsf{0}_{\mathbb{Z}} = 0 & \mathsf{1}_{\mathbb{Z}} = 1 & \mathsf{2}_{\mathbb{Z}} = 2 & \cdots \\ \mathsf{s}_{\mathbb{Z}}(x) = x + 1 & \mathsf{p}_{\mathbb{Z}}(x) = x - 1 & +_{\mathbb{Z}}(x, y) = x + y & \times_{\mathbb{Z}}(x, y) = xy \end{array}$$

If the current set of constraints can be solved in step 4 or 5, then they can also be solved in step 6 or 7, respectively, but the reverse is not true. The sole reason for adopting LPO and polynomial interpretations in alternating layers is efficiency; the search space in steps 3 and 4 is significantly smaller than in steps 5 and 6. Needless to say, the search space in step 5 is much smaller than in step 7 which in turn is much smaller than in step 8. The reason for putting the subterm criterion first is that with this criterion many SCCs can be eliminated very quickly. The extension of the search space for argument filterings mentioned in step 6 is obtained by also considering the full *reverse* argument filtering $[n, \ldots, 1]$ for every n-ary function symbol. The advantage of this extension is that there is no need for a version of LPO with right-to-left status.

The effectiveness of the automatic strategy can be seen from the data presented in Figure 1, which were obtained by running T$_{\mathsf{T}}$T in fully automatic mode on the 89 terminating TRSs (66 in Section 3 and 23 in Section 4) of [6]. An

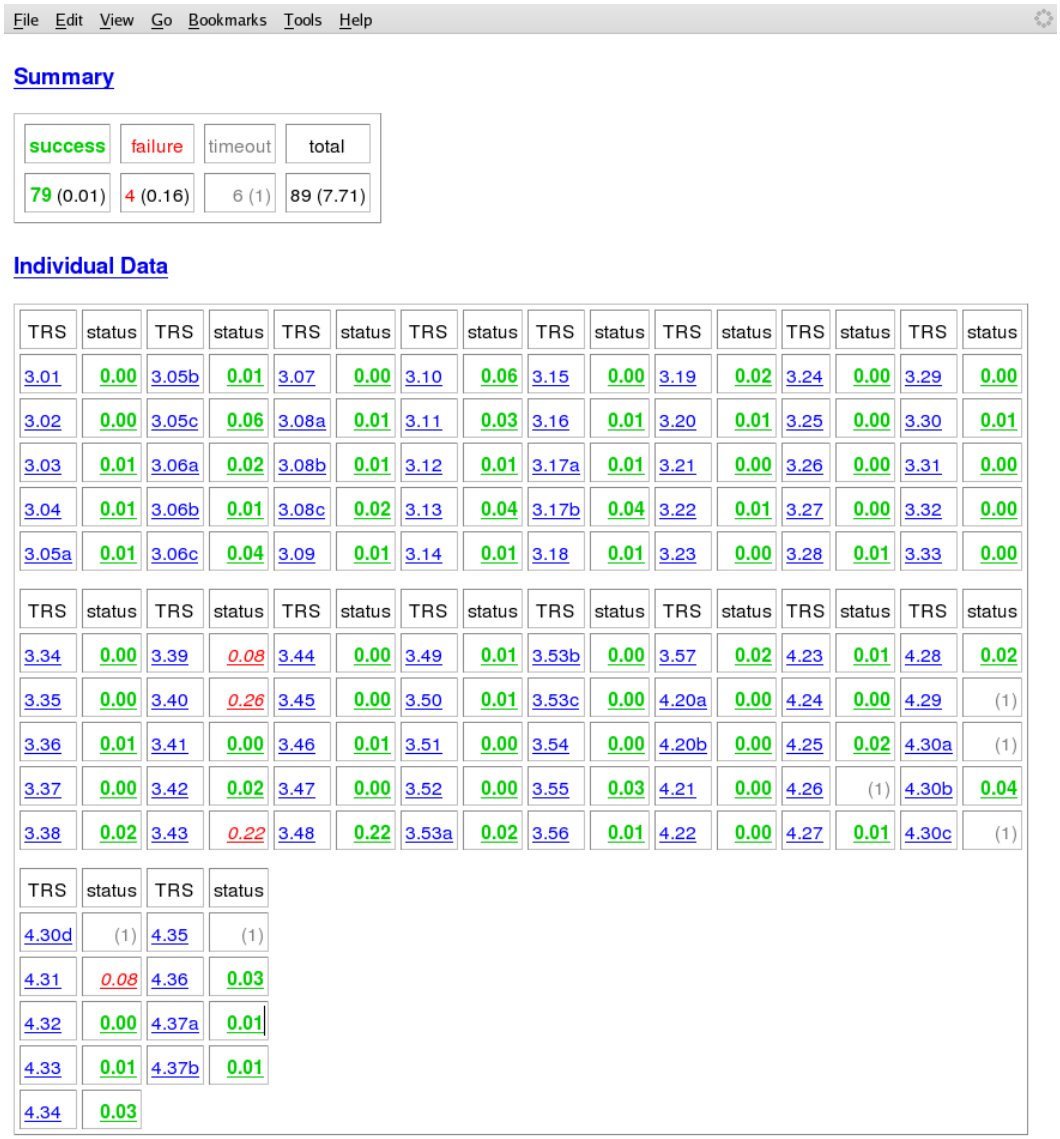


Fig. 1. Output produced by TTT.

explanation of the data is given in Section 5.4.

5.2 Semi Automatic Mode

Figure 2 shows the web interface for the semi-automatic mode.

This menu corresponds to the options that were available in the Tsukuba Termination Tool. A first difference is that we now support the dependency pair method for innermost termination [5]. A second difference is that dependency pairs that are covered by the criterion of Dershowitz [13] are excluded (cf. the remark following Definition 2). The other differences are described in the

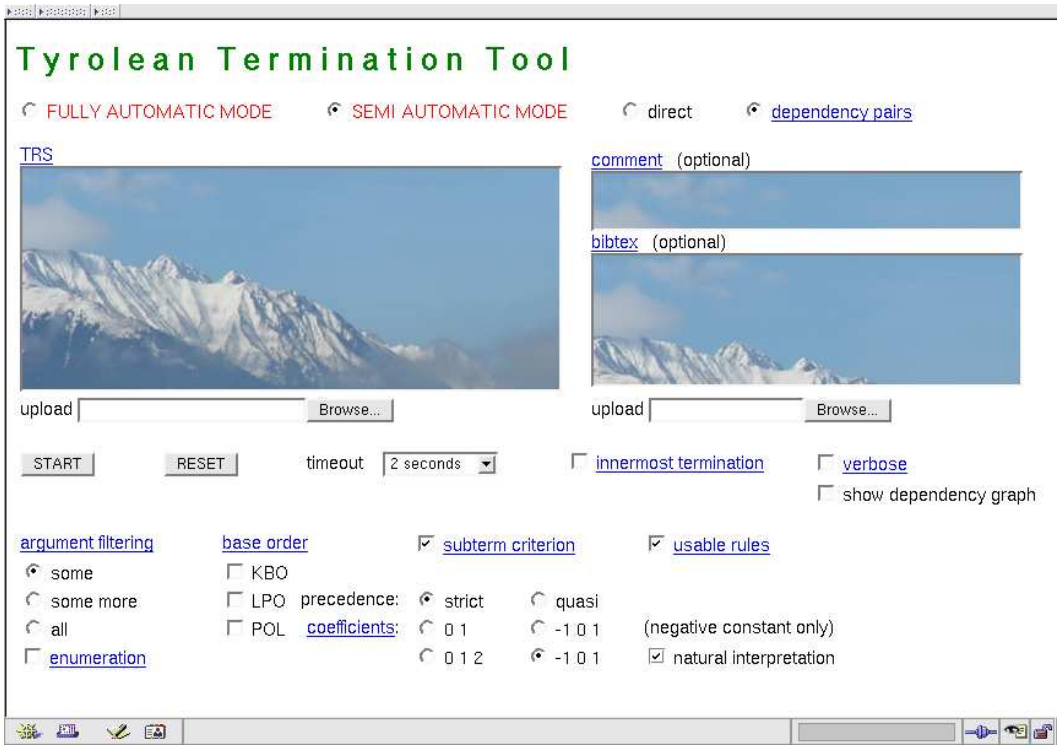


Fig. 2. A screen shot of the semi automatic mode of TTT.

following paragraphs.

First of all, when approximating the (innermost) dependency graph the original estimations of [5] are no longer available since the approximations described in [26] generally produce smaller graphs while the computational overhead is negligible.

Secondly, the user can no longer select the cycle analysis method (all cycles separately, all strongly connected components separately, or the recursive SCC algorithm of [26]). Extensive experiments reveal that the latter method outperforms the other two, so this is now the only supported method in TTT.

Most of the boxes and buttons are self-explanatory. Many correspond to settings described in Section 5.1. By clicking the *enumerate* box, TTT searches for a suitable argument filtering by enumerating all possible argument filterings. For most examples the divide and conquer method is more efficient than the straightforward enumeration method, but still, there are TRSs where enumeration is more effective (cf. [26]), so the user has the option to change the search strategy.

Besides ordinary first-order TRSs, T$_{\text{T}}$T accepts *simply-typed applicative rewrite systems* (STARSs) [2]. Applicative terms are built from variables, constants, and a single binary operator $\cdot$, called application. Constants and variables are equipped with a simple type such that the rewrite rules typecheck. A typical example is provided by the following rules for the map function

$$\begin{aligned}(\mathsf{map} \cdot f) \cdot \mathsf{nil} &\to \mathsf{nil}\\ (\mathsf{map} \cdot f) \cdot ((\mathsf{cons} \cdot x) \cdot y) &\to (\mathsf{cons} \cdot (f \cdot x)) \cdot ((\mathsf{map} \cdot f) \cdot y)\end{aligned}$$

with the type declaration $\mathsf{nil}\colon \alpha$, $\mathsf{cons}\colon \beta \to \alpha \to \alpha$, $\mathsf{map}\colon (\beta \to \beta) \to \alpha \to \alpha$, $f\colon \beta \to \beta$, $x\colon \beta$, and $y\colon \alpha$. Here α is the list type and β the type of elements of lists. STARSs are useful to model higher-order functions in a first-order setting. As usual, the application operator $\cdot$ is suppressed in the notation and parentheses are removed under the "association to the left" rule. The above rules then become

$$\begin{aligned}\mathsf{map}\ f\ \mathsf{nil} &\to \mathsf{nil}\\ \mathsf{map}\ f\ (\mathsf{cons}\ x\ y) &\to \mathsf{cons}\ (f\ x)\ (\mathsf{map}\ f\ y)\end{aligned}$$

This corresponds to the syntax of STARSs in T$_{\text{T}}$T. The types of constants must be declared by the keyword `TYPES`. The types of variables is automatically inferred when typechecking the rules, which follow the `RULES` keyword. So the above STARS would be inputted to T$_{\text{T}}$T as

```
TYPES
 nil : a                                  ;
cons : b => a => a                        ;
 map : (b => b) => a => a                 ;

RULES
map f nil         -> nil                  ;
map f (cons x y) -> cons (f x) (map f y)  ;
```

In order to prove termination of STARSs, T$_{\text{T}}$T uses the two-phase transformation developed by Aoto and Yamada [3]. In the first phase all head variables (e.g. `f` in `f x`) are removed by the *head variable instantiation* technique. The soundness of this phase relies on the *ground term existence condition*, which basically states that all simple types are inhabited by at least one ground term. Users need not be concerned about this technicality as T$_{\text{T}}$T automatically adds fresh constants of the appropriate types to the signature so that the ground term existence condition is satisfied. (Moreover, the termination status of the original STARS is not affected by adding fresh constants.) After the first phase an ordinary TRS is obtained in which the application symbol

is the only non-constant symbol. Such TRSs are not easily proved terminating since the root symbol of every term that has at least two symbols is the application symbol and thus provides no information which could be put to good use. In the second phase applicative terms are transformed into ordinary terms by the *translation to functional form* technique. This technique removes all occurrences of the application symbol. We refer to [3] for a complete description of the transformation. We contend ourselves with showing the Postscript output (in Fig. 3) produced by TTT on the following variation of combinatory logic (inspired by a recent question posted on the TYPES Forum by Jeremy Dawson):

```
TYPES
I : o => o                                  ;
W : (o => o => o) => o => o                 ;
S : (o => o => o) => (o => o) => o => o ;

RULES
I x     -> x          ;
W f x   -> f x x      ;
S x y z -> x z (y z) ;
```

Note that the types are crucial for termination; the untyped version admits the cyclic rewrite step W W W → W W W.

5.4 A Collection of Rewrite Systems as Input

Single TRSs (or STARSs) are inputted by typing (the type declarations and) the rules into the upper left text area or by uploading a file via the browse button. Besides the original TTT syntax (which is obtained by clicking the TRS link), TTT supports the official format of the Termination Problem Data Base [38]. The user can also upload a zip archive. All files ending in `.trs` are extracted from the archive and the termination prover runs on each of these files in turn. The results are collected and presented in two tables. The first table gives the number of successes and failures, both with the average time spent on each TRS, the number of timeouts, and the total number of TRSs extracted from the zip archive together with the total execution time. The second table lists for each TRS the execution time in seconds together with the status: **bold green** indicates success, *red italics* indicates failure, and gray indicates timeout. By clicking **green** (*red*) entries the user can view the termination proof (attempt) in HTML or Postscript format. The tables are regularly updated during the termination proving process, enabling the user to access generated termination proofs without having to wait for the overall process to terminate. Figure 1 shows the two tables for the 89 terminating

Termination Proof Script [a]

Consider the simply-typed applicative TRS

$$\begin{aligned} \mathsf{I}\ x &\to x \\ \mathsf{W}\ f\ x &\to f\ x\ x \\ \mathsf{S}\ x\ y\ z &\to x\ z\ (y\ z) \end{aligned}$$

over the signature $\mathsf{I}\colon \mathsf{o} \Rightarrow \mathsf{o}$, $\mathsf{W}\colon (\mathsf{o} \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}) \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}$, and $\mathsf{S}\colon (\mathsf{o} \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}) \Rightarrow (\mathsf{o} \Rightarrow \mathsf{o}) \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}$. In order to satisfy the ground term existence condition we extend the signature by $\mathsf{c}\colon \mathsf{o} \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}$ and $\mathsf{c}'\colon \mathsf{o}$. Instantiating all head variables yields the following rules:

$$\begin{aligned} \mathsf{I}\ x &\to x \\ \mathsf{W}\ \mathsf{c}\ x &\to \mathsf{c}\ x\ x \\ \mathsf{S}\ \mathsf{c}\ \mathsf{I}\ z &\to \mathsf{c}\ z\ (\mathsf{I}\ z) \\ \mathsf{S}\ \mathsf{c}\ (\mathsf{W}\ w)\ z &\to \mathsf{c}\ z\ (\mathsf{W}\ w\ z) \\ \mathsf{S}\ \mathsf{c}\ (\mathsf{S}\ w\ v)\ z &\to \mathsf{c}\ z\ (\mathsf{S}\ w\ v\ z) \\ \mathsf{S}\ \mathsf{c}\ (\mathsf{c}\ w)\ z &\to \mathsf{c}\ z\ (\mathsf{c}\ w\ z) \end{aligned}$$

By transforming terms into functional form the TRS

$$\begin{aligned} 1:&\quad \mathsf{I}_1(x) &&\to x \\ 2:&\quad \mathsf{W}_2(\mathsf{c}, x) &&\to \mathsf{c}_2(x, x) \\ 3:&\quad \mathsf{S}_3(\mathsf{c}, \mathsf{I}, z) &&\to \mathsf{c}_2(z, \mathsf{I}_1(z)) \\ 4:&\quad \mathsf{S}_3(\mathsf{c}, \mathsf{W}_1(w), z) &&\to \mathsf{c}_2(z, \mathsf{W}_2(w, z)) \\ 5:&\quad \mathsf{S}_3(\mathsf{c}, \mathsf{S}_2(w, v), z) &&\to \mathsf{c}_2(z, \mathsf{S}_3(w, v, z)) \\ 6:&\quad \mathsf{S}_3(\mathsf{c}, \mathsf{c}_1(w), z) &&\to \mathsf{c}_2(z, \mathsf{c}_2(w, z)) \end{aligned}$$

is obtained. There are 3 dependency pairs:

$$\begin{aligned} 7:&\quad \mathsf{S}^\sharp_3(\mathsf{c}, \mathsf{I}, z) &&\to \mathsf{I}^\sharp_1(z) \\ 8:&\quad \mathsf{S}^\sharp_3(\mathsf{c}, \mathsf{W}_1(w), z) &&\to \mathsf{W}^\sharp_2(w, z) \\ 9:&\quad \mathsf{S}^\sharp_3(\mathsf{c}, \mathsf{S}_2(w, v), z) &&\to \mathsf{S}^\sharp_3(w, v, z) \end{aligned}$$

The approximated dependency graph contains one SCC: $\{9\}$.

- Consider the SCC $\{9\}$. By taking the simple projection π with $\pi(\mathsf{S}^\sharp_3) = 2$, the dependency pair simplifies to

$$9:\quad \mathsf{S}_2(w, v) \to v$$

  and is compatible with the proper subterm relation.

Hence the TRS is terminating.

[a] Tyrolean Termination Tool (0.01 seconds) — September 21, 2005

Fig. 3. Example output.

TRSs in Sections 3 and 4 of [6]. Here we used T$_\top$T's fully automatic mode with a timeout of 1 second (for each TRS). The experiment was performed on a PC equipped with a 1.80 GHz Intel Pentium Processor - M and 1 GB of memory, using native-compiled code for Linux/Fedora.

5.5 Some Implementation Details

The web interface of T$_\top$T is written in Ruby[6] and the termination prover underlying T$_\top$T is written in Objective Caml (OCaml),[7] using the third-party libraries[8] `findlib`, `extlib`, and `pcre-ocaml`.

The termination prover consists of about 13,000 lines of OCaml code. About 20% is used for the manipulation of terms and rules. Another 15% is devoted to graph manipulations. This part of the code is not only used to compute dependency graph approximations, but also for precedences in KBO and LPO, and for the relation $\rhd_{\mathrm{d}}$ in Definition 13 which is used to compute the usable rules. The various termination methods that are provided by T$_\top$T account for less than 10% each. Most of the remaining code deals with I/O: parsing the input and producing HTML and Postscript output. For the official Termination Problem Data Base format we use parsers written in OCaml by Claude Marché.

It is interesting to note that two of the original techniques that make T$_\top$T fast, the recursive SCC algorithm and the subterm criterion, account for just 13 and 20 lines, respectively. Actually, we implemented the subterm criterion twice. The number 20 refers to a straightforward encoding that generates all simple projections until a suitable one is found. This encoding works fine on all examples we tested (see Section 6), with one exception (`AProVE/AAECC-ring`). The reason is that the dependency graph of that TRS contains an SCC consisting of nine 7-ary and one 6-ary dependency pair symbols, amounting to $7^9 \times 6 = 242121642$ simple projections. Specializing the divide and conquer algorithm developed in [26] for arbitrary argument filterings amounts to 11 additional lines of code. (The generic divide and conquer algorithm is implemented in 209 lines of OCaml code.) This latter implementation is the one used in the experiments described in Section 6 and the only one available from the web interface.

Concerning the implementation of simply-typed applicative rewrite systems, we use the Damas-Milner type reconstruction algorithm (see e.g. [35]) to infer the types of variables.

[6] `http://www.ruby-lang.org/`
[7] `http://www.ocaml.org/`
[8] `http://caml.inria.fr/humps/`

We conclude this section with some remarks on the implementation of base orders in TTT. The implementation of LPO follows [23] but we first check whether the current pair of terms can be oriented by the embedding order in every recursive call to LPO. This improves the efficiency by about 20%. The implementation of KBO is based on [30]. We use the "method for complete description" [14] to compute a suitable weight function. The implementation of polynomial interpretations with coefficients from $\{0, 1\}$ is based on [11, Figure 1] together with the simplification rules described in Section 4.4.1 of the same paper. The current implementation of polynomial interpretations with coefficients from $\{-1, 0, 1\}$ in TTT is rather naive. In particular, we do not backtrack when the choice of the abstract variable in Lemma 51 does not lead to a positive conclusion. We anticipate that the recent techniques of [11] can be extended to handle negative coefficients.

6 Experiments

In this section we assess the techniques introduced in the preceding sections on the 773 TRSs (at least 94 of which are non-terminating) in version 2.0 of the Termination Problem Data Base [38].

In all experiments we used the EDG* approximation [33] of the dependency graph. Moreover, we adopted the recursive SCC algorithm in [26] for handling cycles in the graph. When the lexicographic path order or Knuth-Bendix order is used, the divide and conquer algorithm is used to search for suitable argument filterings. The experiments were conducted in the same environment as described in Section 5.4.

We tested individual methods and combinations of them. The results are summarized in Tables 1–3. The letters in the column headings have the following meaning:

- `s` the subterm criterion of Section 2,
- `u` the usable rule criterion of Sections 3 (for Tables 1 and 2) and 4 (for Table 3),
- `l` lexicographic path order in combination with *some* [26] argument filterings,
- `k` Knuth-Bendix order in combination with *some* argument filterings,
- `p` polynomial interpretation restricted to linear polynomials with coefficients and constants indicated in the table headings.

We list the number of successful termination attempts, the number of failures (which means that no termination proof was found while fully exploring the search space implied by the options), and the number of timeouts. The figures below the number of successes and failures denote the average time in seconds.

Table 1
Experiments I.

	dg	s	l	ul	sul	k	uk	suk
success	51 (44)	206	206	244	260	162	262	290
	0.00 (0.00)	0.00	0.06	0.15	0.14	0.29	0.29	0.24
failure	722 (729)	567	540	504	488	548	465	437
	0.01 (0.01)	0.01	0.57	0.42	0.43	0.96	0.75	0.79
timeout (*30* s)	0 (0)	0	27	25	25	63	46	46
total time	8 (9)	9	1133	998	996	2464	1804	1795

In Table 1 the combination of the subterm criterion and the usable rule criterion in connection with traditional simplification orders is examined. As can be seen from column s, the subterm criterion is extremely fast. This is even more apparent from a comparison with column dg, where termination is concluded if the approximated dependency graph contains no cycles. In parentheses we provide the numbers when the original definition of dependency pairs is used (cf. the paragraph following Definition 2). The difference disappears as soon as the subterm criterion is applied. It is interesting to note that the subterm criterion could handle 1052 of the 1589 generated SCCs, resulting in termination proofs for 206 of the 773 TRSs. Clearly, the subterm criterion is very suitable as a method of first choice in any termination prover incorporating the dependency pair technique.

Table 2 shows the effect of the usable rule criterion in combination with linear polynomial interpretations possibly with negative constants. Enlarging the coefficient domain $\{0, 1\}$ with the value 2 gives very few additional examples (when using the usable rule criterion). The effect of allowing -1 as constant is more apparent. An interesting remark is that there is no overlap between the additional TRSs that can be proved terminating by allowing 2 and by allowing -1.

In Table 3 we use the negative coefficient method developed in Section 4 in connection with Corollary 31 (p) and Theorems 23 (u1p) and 40 (u2p and u3p). The difference between columns u2p and u3p is that for the latter we revert to Corollary 31 if the $\mathcal{A}$-right-linearity condition in Theorem 40 cannot be satisfied. In column u4p we first check whether the SCC $\mathcal{C}$ under consideration contains a right-ground dependency pair or $\mathcal{C} \cup \mathcal{U}(\mathcal{C})$ is non-duplicating. If one of these conditions is fulfilled then we use $\mathcal{U}(\mathcal{C})$ and ignore the $\mathcal{A}$-right-linearity constraint, otherwise we proceed as in column u3p.

Comparing the p and u1p columns, the usefulness of Theorem 23 is clear. Theorem 40 (u2p) is even more powerful. However, as can be inferred from columns

Table 2
Experiments II.

coefficients	$0,1$			$0,1,2$			$0,1$		
constants	$0,1$			$0,1,2$			$-1,0,1$		
	p	up	sup	p	up	sup	p	up	sup
success	247	340	364	262	348	369	256	355	381
	0.28	0.31	0.29	0.32	0.29	0.29	0.31	0.29	0.29
failure	458	395	371	372	347	326	348	325	300
	1.12	0.82	0.85	0.98	0.64	0.68	1.71	1.22	1.22
timeout (*30* s)	68	38	38	139	78	78	169	93	92
total time	2622	1571	1562	4619	2664	2667	5745	3292	3238

Table 3
Polynomial interpretations with negative coefficients from $\{-1,0,1\}$.

	p	u1p	u2p	u3p	u4p	su1p	su2p	su3p	su4p
success	151	232	264	270	270	282	303	309	309
	0.46	0.26	0.36	0.36	0.36	0.20	0.30	0.30	0.30
failure	400	369	357	317	334	334	324	286	302
	2.35	1.54	1.39	2.24	1.47	1.54	1.27	2.23	1.39
timeout (*30* s)	222	172	152	186	169	157	146	178	162
total time	7670	5788	5149	6388	5657	5280	4884	6071	5374

u3p and u4p, six TRSs handled by Theorem 23 and Corollary 31 violate the $\mathcal{A}$-right-linearity condition in Theorem 40. Four of them are variations of Toyama's example, the remaining two are the one-rule system $x \times ((-y) \times y) \to (-(y \times y)) \times x$ and the two-rules system consisting of $\mathsf{f}(x,x) \to \mathsf{f}(\mathsf{g}(x),x)$ and $\mathsf{g}(x) \to \mathsf{s}(x)$.

Comparing the p, u4p, and su4p columns in Table 3 with the final three columns in Table 2, one might wrongly conclude that polynomial interpretations with negative coefficients make TTT both slower and less powerful. Therefore, in Table 4 we provide in columns p′, u4p′, and su4p′ the results for the strategy where polynomial interpretations with negative coefficients are attempted only if polynomial interpretations with negative constants fail.

Table 4

	(p)	p′	(u4p)	u4p′	(su4p)	su4p′
success	(151)	270	(270)	372	(309)	400
	(0.46)	0.44	(0.36)	0.30	(0.30)	0.39
failure	(400)	312	(334)	276	(302)	255
	(2.35)	2.62	(1.47)	1.55	(1.39)	1.49
timeout (*30* s)	(222)	191	(169)	125	(162)	118
total time	(7670)	6668	(5657)	4290	(5374)	4077

Table 5
Fully automatic mode.

t	*1*	*2*	*10*	*30*
success	390	394	403	411
	0.03	0.05	0.16	0.54
failure	184	202	224	240
	0.18	0.28	0.68	1.80
timeout (t s)	199	177	146	122
total time	244	430	1678	4314

Table 5 shows the power of the fully automatic mode of T$\!\top\!$T. A remarkable 95% of the successful termination proofs obtained with a 30 seconds timeout are also obtained when setting the timeout to 1 second. Most of the remaining 5% are due to polynomial interpretations.

Although the numbers presented in Table 5 may suggest otherwise, the fully automatic mode does not encompass the full power of T$\!\top\!$T. For instance, the fully automatic mode misses out on two TRSs in the $\{0, 1, 2\}$ columns of Table 2. Another such TRS, which can be handled by appropriately setting the options in the semi-automatic mode, is given in [26, Example 43].

7 Related Work

Needless to say, TTT is not the only tool available for proving termination of rewrite systems and the dependency pair method is not the only successful termination technique. We start this final section by briefly discussing some of the other tools that participate in the TRS category of the annual termination competition [9] and the techniques they implement. We do not claim completeness of the description below; because of the rapid developments in the field, by the time this paper appears in print, any description is likely to be outdated.

- We start our discussion with CiME [10], the very first tool for automatically proving termination of rewrite systems that is still available. CiME is a tool with powerful techniques for finding termination proofs based on polynomial interpretations in connection with the dependency pair method. In contrast to TTT, CiME can handle rewrite systems with AC operators. As a matter of fact, termination is only a side-issue in CiME. Its main strength lies in completing equational theories modulo theories like AC and C.
- Matchbox [43] is a tool that is largely based on methods from formal language theory. These methods are especially useful for proving termination of string rewrite systems. The latest version of Matchbox tries to establish termination or non-termination using results on match-bounded rewriting [17] in combination with matrix interpretations [15].
- TPA [29] is a tool based on semantic labeling [44]. Besides a two-valued domain it also uses natural numbers as labels, which is surprisingly powerful. Polynomial interpretations and recursive path orders are available as basic techniques. TPA can prove relative termination (Geser [16]). Because of semantic labeling, the tool is capable of proving termination of rewrite systems that are not handled by any current tool based on dependency pairs.
- Last but not least, AProVE, a very powerful tool for proving termination and non-termination of ordinary rewrite systems (possibly modulo AC), logic programs, conditional rewrite systems, context-sensitive rewrite systems. Version 1.2 of AProVE is described in [19]. Of all existing tools, AProVE supports the most base orders and even offers several different algorithms implementing these. It incorporates virtually all recent refinements of the dependency pair method. AProVE has several methods that are not available in any other tool. We mention here the size-change principle [39], transformations for dependency pairs like narrowing and instantiation, and a modular refinement where the set of usable rules is determined after a suitable argument filtering has been computed [40].

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

With respect to the results presented in this paper, most tools that incorporate dependency pairs use the subterm criterion and compute usable rules. The latest version of AProVE combines the subterm criterion with the size-change principle. The latter tool also contains an implementation of polynomial interpretations with negative coefficients. Aoto and Yamada [4] extended the subterm criterion to simply-typed applicative rewrite systems. Very recently, Alarcón *et al.* [1] extended the subterm criterion to context-sensitive rewrite systems.

We conclude with mentioning related work on the use of polynomial interpretations with negative coefficients for proving termination. Lucas [31,32] considers polynomials with *real* coefficients for automatically proving termination of (context-sensitive) rewrite systems are considered. He solves the problem of well-foundedness by replacing the standard order on $\mathbb{R}$ with $>_\delta$ for some fixed positive $\delta \in \mathbb{R}$: $x >_\delta y$ if and only if $x - y \geqslant \delta$. In addition, he demands that interpretations are uniformly bounded from below (i.e., there exists an $m \in \mathbb{R}$ such that $f_{\mathbb{R}}(x_1, \ldots, x_n) \geqslant m$ for all function symbols f and $x_1, \ldots, x_n \geqslant m$). While this method allows one to use finer positive polynomial like $x^2 - \frac{1}{2}x + 1$, the latter requirement entails that interpretations like $x - 1$ or $x - y + 1$ cannot be handled. This contrasts our approach in which a given algebra (on $\mathbb{Z}$) is replaced by an induced algebra (on $\mathbb{N}$). We anticipate that by combining both approaches, unrestricted polynomial interpretations with real coefficients like $x - \frac{1}{2}y$ can be used for termination proofs.

Gramlich and Lucas [22] use polynomial interpretations with integral coefficients for proving termination of context-sensitive rewriting. Negative coefficients are allowed, but only for argument positions where no rewriting is permitted. Consequently, the lack of monotonicity does not pose a problem.

Acknowledgements

We are grateful to Daniel Laimer for improving the web interface of TTT. We thank the anonymous referees for providing numerous suggestions which helped to improve the presentation.

References

[1] B. Alarcón, R. Gutiérrez, and S. Lucas. Context-sensitive dependency pairs. In *Proceedings of the 26th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 4337 of *Lecture Notes in Computer Science*, pages 298–309, 2006.

[2] T. Aoto and T. Yamada. Termination of simply typed term rewriting by translation and labelling. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 380–394, 2003.

[3] T. Aoto and T. Yamada. Termination of simply-typed applicative term rewriting systems. In *Proceedings of the 2nd International Workshop on Higher-Order Rewriting*, Technical Report AIB-2004-03, RWTH Aachen, pages 61–65, 2004.

[4] T. Aoto and T. Yamada. Dependency pairs for simply typed term rewriting. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3467 of *Lecture Notes in Computer Science*, pages 120–134, 2005.

[5] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[6] T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, 2001.

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

[8] L. Bachmair and N. Dershowitz. Commutation, transformation, and termination. In *Proceedings of the 8th International Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20, 1986.

[9] C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 346–364, 2000.

[10] E. Contejean, C. Marché, B. Monate, and X. Urbain. C*i*ME version 2, 2000. Available at `http://cime.lri.fr/`.

[11] E. Contejean, C. Marché, A.-P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 34(4):325–363, 2005.

[12] N. Dershowitz. 33 Examples of termination. In *French Spring School of Theoretical Computer Science*, volume 909 of *Lecture Notes in Computer Science*, pages 16–26, 1995.

[13] N. Dershowitz. Termination by abstraction. In *Proceedings of the 20th International Conference on Logic Programming*, volume 3132 of *Lecture Notes in Computer Science*, pages 1–18, 2004.

[14] J. Dick, J. Kalmus, and U. Martin. Automating the Knuth-Bendix ordering. *Acta Infomatica*, 28:95–119, 1990.

[15] J. Endrullis, J. Waldmann, and H. Zantema. Matrix interpretations for proving termination of term rewriting. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 574–588, 2006.

[16] A. Geser. *Relative Termination*. PhD thesis, University of Passau, 1990.

[17] A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3467, pages 353–267, 2005.

[18] J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.

[19] J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic termination proofs in the dependency pair framework. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 281–286, 2006.

[20] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2850 of *Lecture Notes in Artificial Intelligence*, pages 165–179, 2003.

[21] B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994.

[22] B. Gramlich and S. Lucas. Simple termination of context-sensitive rewriting. In *Proceedings of the 3rd ACM Sigplan Workshop on Rule-based Programming*, pages 29–41. ACM Press, 2002.

[23] N. Hirokawa and A. Middeldorp. Tsukuba Termination Tool. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 311–320, 2003.

[24] N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 249–268, 2004.

[25] N. Hirokawa and A. Middeldorp. Polynomial interpretations with negative coefficients. In *Proceedings of the 7th International Conference on Artificial Intelligence and Symbolic Computation*, volume 3249 of *Lecture Notes in Artificial Intelligence*, pages 185–198, 2004.

[26] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005.

[27] N. Hirokawa and A. Middeldorp. Tyrolean Termination Tool. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3467 of *Lecture Notes in Computer Science*, pages 175–184, 2005.

[28] H. Hong and D. Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21:23–28, 1998.

[29] A. Koprowski. TPA: Termination proved automatically. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, volume 4098 of *Lecture Notes in Computer Science*, pages 257–266, 2006.

[30] K. Korovin and A. Voronkov. Orienting rewrite rules with the Knuth-Bendix order. *Information and Computation*, 183:165–186, 2003.

[31] S. Lucas. Polynomials for proving termination of context-sensitive rewriting. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures*, volume 2987 of *Lecture Notes in Computer Science*, pages 318–332, 2004.

[32] S. Lucas. Polynomials over the reals in proof of termination: From theory to practice. *RAIRO Theoretical Informatics and Applications*, 39(3):547–586, 2005.

[33] A. Middeldorp. Approximations for strategies and termination. In *Proceedings of the 2nd International Workshop on Reduction Strategies in Rewriting and Programming*, volume 70(6) of *Electronic Notes in Theoretical Computer Science*, 2002.

[34] A. Middeldorp and H. Ohsaki. Type introduction for equational rewriting. *Acta Informatica*, 36(12):1007–1029, 2000.

[35] B.C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[36] J. Steinbach and U. Kühler. Check your ordering – termination proofs and open problems. Technical Report SR-90-25, Universität Kaiserslautern, 1990.

[37] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[38] Termination Problem Data Base. Available at `http://www.lri.fr/~marche/tpdb`.

[39] R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 16(4):229–270, 2005.

[40] R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, volume 3097 of *Lecture Notes in Artificial Intelligence*, pages 75–90, 2004.

[41] Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.

[42] X. Urbain. Modular & incremental automated termination proofs. *Journal of Automated Reasoning*, 32:315–355, 2004.

[43] J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 85–94, 2004.

[44] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.