Termination of term rewriting using dependency pairs*

Thomas Arts[†] Jürgen Giesl[‡]

Abstract

We present techniques to prove termination and innermost termination of term rewriting systems automatically. In contrast to previous approaches, we do not compare left- and right-hand sides of rewrite rules, but introduce the notion of *dependency pairs* to compare left-hand sides with special subterms of the right-hand sides. This results in a technique which allows to apply existing methods for automated termination proofs to term rewriting systems where they failed up to now. In particular, there are numerous term rewriting systems where a *direct* termination proof with simplification orderings is not possible, but in combination with our technique, well-known simplification orderings (such as the recursive path ordering, polynomial orderings, or the Knuth-Bendix ordering) can now be used to prove termination automatically.

Unlike previous methods, our technique for proving *innermost* termination automatically can also be applied to prove innermost termination of term rewriting systems that are not terminating. Moreover, as innermost termination implies termination for certain classes of term rewriting systems, this technique can also be used for termination proofs of such systems.

1 Introduction

Termination is one of the most fundamental properties of a term rewriting system (TRS), cf. e.g. [21]. While in general this problem is undecidable [37], several methods for proving termination have been developed (e.g. path orderings [17, 20, 40, 51, 55], Knuth-Bendix orderings [22, 42], forward closures

*Technical Report IBN 97/46, Darmstadt University of Technology. This is a preliminary version of an article which appeared in *Theoretical Computer Science*.

†Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands, E-mail: `thomas@cs.ruu.nl`

‡FB Informatik, Darmstadt University of Technology, Alexanderstr. 10, 64283 Darmstadt, Germany, E-mail: `giesl@informatik.th-darmstadt.de`

[20, 47], semantic interpretations [12, 13, 31, 46, 53, 59], transformation orderings [8, 11, 54], distribution elimination [59], dummy elimination [26], semantic labelling [60] etc. — for surveys see e.g. [18, 55]).

We present a new approach for the *automation* of termination proofs. Most well-known techniques for proving termination automatically try to find a well-founded ordering such that for all rules of the TRS the left-hand sides are greater than the corresponding right-hand sides. In most practical applications the synthesized orderings are total on ground terms [25] and therefore virtually all orderings used are *simplification orderings* [17, 18, 50, 55]. However, numerous TRSs are not simply terminating, i.e. not compatible with a simplification ordering. Hence, standard techniques like the recursive path ordering, polynomial interpretations, and the Knuth-Bendix ordering fail in proving termination of these TRSs.

In Sect. 2 we introduce a new criterion for termination based on the notion of *dependency pairs*. The main advantage of our termination criterion is that it is especially well suited for automation. To check the criterion automatically, we have developed a procedure which generates a set of constraints for every TRS. If there exists a well-founded ordering satisfying these constraints, then the TRS is terminating. For the synthesis of suitable orderings existing techniques, such as the recursive path ordering or polynomial interpretations, may be used. It turns out that for many TRSs where a *direct* application of simplification orderings fails, the constraints generated by our technique are nevertheless satisfied by an automatically generated simplification ordering. Moreover, all TRSs that can be proved terminating directly by synthesizing a simplification ordering automatically, can automatically be proved terminating by this new technique, too.

Rewriting under strategies is often used for modelling certain programming paradigms. For example, *innermost* rewriting, i.e. rewriting where only innermost redeces are contracted, can be used to model call-by-value computation semantics. For that reason, there has been an increasing interest in research on properties of rewriting under strategies. In particular, the study of termination is important when regarding such restricted versions of rewriting [35, 36, 45]. To prove *innermost termination* (also called (strong) innermost normalisation), one has to show that the length of every innermost reduction is finite. Techniques for proving innermost termination can for example be utilized for termination proofs of functional programs (modelled by TRSs) with eager reduction strategy or of logic programs. (When transforming well-moded logic programs into TRSs, innermost termination of the TRS is sufficient for left-termination of the logic program [7].) Up to now, the only way to prove innermost termination *automatically* was by showing termination of the TRS. Therefore, none of the existing techniques could prove innermost termination of non-terminating systems. However, in Sect. 3 we show that after some modification, the dependency pair technique can be used as the first *specific* method for *innermost* termination. In Sect. 4 we conclude and give some comments on related work. Sect. 5

contains a collection of examples to illustrate the power of our method.

2 Proving termination

In this section we present a new approach for automated termination proofs. In Sect. 2.1, we state our termination criterion and prove that it is a necessary and sufficient criterion for termination. Sect. 2.2 shows how this criterion can be checked automatically by generating a set of constraints that are satisfied by a well-founded ordering if and only if the criterion is fulfilled. The generation of suitable well-founded orderings is described in Sect. 2.3. To increase the power of our method we introduce a refined approach for its automation in Sect. 2.4 and an additional refinement in Sect. 2.5. In this way we obtain a very powerful technique which performs automated termination proofs for many TRSs where termination could not be proved automatically before. An overview of this technique is given in Sect. 2.6.

2.1 Termination criterion

For *constructor systems* it is common to split the signature into two disjoint sets, the *defined symbols* and the *constructors*. The following definition extends these notions to arbitrary term rewriting systems $\mathcal{R}(\mathcal{F}, R)$ (with the rules R over a signature $\mathcal{F}$). For an introduction to term rewriting and its notations, we refer to Dershowitz and Jouannaud [21] and Klop [41]. Here, the *root* of a term $f(\ldots)$ is the leading function symbol f.

Definition 1 (Defined symbols and constructors) *Let $\mathcal{R}(\mathcal{F}, R)$ be a TRS. The set $D_\mathcal{R}$ of* defined symbols *of $\mathcal{R}$ is defined as $\{root(l) \mid l \to r \in R\}$ and the set $C_\mathcal{R}$ of* constructors *of $\mathcal{R}$ is defined as $\mathcal{F} \setminus D_\mathcal{R}$.*

To refer to the defined symbols and constructors explicitly, a rewrite system is written as $\mathcal{R}(D_\mathcal{R}, C_\mathcal{R}, R)$ and the subscripts are omitted if $\mathcal{R}$ is clear from the context.

Example 2 *The following TRS has two defined symbols, viz.* minus *and* quot*, and two constructors, viz.* 0 *and* s*.*

$$\begin{array}{rcl}
\mathsf{minus}(x, \mathsf{0}) & \to & x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \to & \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y)))
\end{array}$$

Most techniques for automated termination proofs are restricted to simplification orderings. However, the TRS above is not compatible with a simplification ordering, because the left-hand side of the last quot-rule is embedded in its right-hand side if y is instantiated with $\mathsf{s}(x)$. Therefore these techniques cannot prove termination of this TRS.

In contrast to previous methods which compare left- and right-hand sides of rules, the central idea of our approach is to compare left-hand sides of rules only with those *subterms* of the right-hand sides that may possibly start a new reduction.

The motivation for this approach is to regard TRSs as 'programs'. Intuitively, such a program is terminating if the arguments are decreasing in each recursive call. For example, to prove termination of quot, instead of comparing both sides of the *rules*, one only has to compare the input *arguments* $\mathsf{s}(x), \mathsf{s}(y)$ with the arguments $\mathsf{minus}(x, y), \mathsf{s}(y)$ of the corresponding recursive call. This way of looking at termination of TRSs motivates that only those subterms of the right-hand sides that have a defined root symbol are considered for the examination of the termination behaviour.

More precisely, if a term $f(s_1, \ldots, s_n)$ rewrites to a term $C[g(t_1, \ldots, t_m)]$ (where g is a defined symbol and C denotes some context), then to prove termination the argument tuples $s_1, \ldots, s_n$ and $t_1, \ldots, t_m$ are compared. In order to avoid the handling of tuples, a special *tuple symbol* F, not occurring in the signature of the TRS, is introduced for every defined symbol f in D. Instead of comparing *tuples*, now the *terms* $F(s_1, \ldots, s_n)$ and $G(t_1, \ldots, t_m)$ are compared. To ease readability, in this paper we assume that the original signature $\mathcal{F}$ consists of lower case function symbols only, whereas the tuple symbols are denoted by the corresponding upper case symbols.

Definition 3 (Dependency pair) *Let $\mathcal{R}(D, C, R)$ be a TRS. If*

$$f(s_1, \ldots, s_n) \to C[g(t_1, \ldots, t_m)]$$

is a rewrite rule of R with $g \in D$, then $\langle F(s_1, \ldots, s_n), G(t_1, \ldots, t_m)\rangle$ is called a dependency pair *of $\mathcal{R}$.*

The dependency pairs of a TRS are easily determined and if the TRS is finite, then only finitely many dependency pairs exist.

Example 4 *The dependency pairs of the TRS in Ex. 2 are*

$$\langle \mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{M}(x, y)\rangle \tag{1}$$
$$\langle \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{M}(x, y)\rangle \tag{2}$$
$$\langle \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{s}(y))\rangle \tag{3}$$

where M *and* Q *denote the tuple symbols for* minus *and* quot *respectively.*

The notion of dependency pairs is the basis for our termination criterion. Since every left-hand side has a defined root symbol, no rule matches a term without defined symbols, hence such a term is a normal form. Thus, infinite reductions originate from the fact that defined symbols are introduced by the right-hand sides of rewrite rules. By tracing the introduction of these defined symbols, information is obtained about the termination behaviour of the TRS. For that purpose we consider special sequences of dependency pairs, so-called *chains*, such that the right-hand side of every dependency pair in a chain corresponds to the newly introduced redex that should be traced.

Definition 5 (Chain) *Let $\mathcal{R}(D, C, R)$ be a TRS. A sequence of dependency pairs $\langle s_1, t_1\rangle\,\langle s_2, t_2\rangle \ldots$ is an* $\mathcal{R}$-chain *if there exists a substitution σ such that $t_j\sigma \to_{\mathcal{R}}^* s_{j+1}\sigma$ holds for every two consecutive pairs $\langle s_j, t_j\rangle$ and $\langle s_{j+1}, t_{j+1}\rangle$ in the sequence.*

If $\mathcal{R}$ is clear from the context we often write 'chain' instead of '$\mathcal{R}$-chain'. We always assume that different (occurrences of) dependency pairs have disjoint sets of variables and we always regard substitutions whose domain may be infinite. Hence, in our example we have the chain

$$\langle \mathsf{Q}(\mathsf{s}(x_1), \mathsf{s}(y_1)), \mathsf{Q}(\mathsf{minus}(x_1, y_1), \mathsf{s}(y_1))\rangle\ \ \langle \mathsf{Q}(\mathsf{s}(x_2), \mathsf{s}(y_2)), \mathsf{Q}(\mathsf{minus}(x_2, y_2), \mathsf{s}(y_2))\rangle,$$

because $\mathsf{Q}(\mathsf{minus}(x_1, y_1), \mathsf{s}(y_1))\sigma \to_{\mathcal{R}}^* \mathsf{Q}(\mathsf{s}(x_2), \mathsf{s}(y_2))\sigma$ holds for the substitution σ that replaces x_1 by $\mathsf{s}(\mathsf{0})$, x_2 by $\mathsf{0}$, and both y_1 and y_2 by $\mathsf{0}$. In fact any finite sequence of the dependency pair (3) in Ex. 4 is a chain. However, in the next section we show that the above TRS has no *infinite* chain. The following theorem proves that the absence of infinite chains is a sufficient and necessary criterion for termination.

Theorem 6 (Termination criterion) *A TRS $\mathcal{R}(D, C, R)$ is terminating if and only if no infinite $\mathcal{R}$-chain exists.*

Proof. We first prove that the above criterion is *sufficient* for termination, i.e. we show that for any infinite reduction we can construct an infinite $\mathcal{R}$-chain.

Let t be a term that starts an infinite reduction. By a minimality argument, the term t contains a subterm[1] $f_1(\vec{u}_1)$ that starts an infinite reduction, but none of the terms $\vec{u}_1$ starts an infinite reduction, i.e. the terms $\vec{u}_1$ are terminating.

Let us consider an infinite reduction starting with $f_1(\vec{u}_1)$. First, the arguments $\vec{u}_1$ are reduced in zero or more steps to arguments $\vec{v}_1$ and then a rewrite rule $f_1(\vec{w}_1) \to r_1$ is applied to $f_1(\vec{v}_1)$, i.e. a substitution σ_1 exists such that $f_1(\vec{v}_1) = f_1(\vec{w}_1)\sigma_1 \to_{\mathcal{R}} r_1\sigma_1$. Now the infinite reduction continues with $r_1\sigma_1$, i.e. the term $r_1\sigma_1$ starts an infinite reduction, too.

By assumption there exists no infinite reduction beginning with one of the terms $\vec{v}_1 = \vec{w}_1\sigma_1$. Hence, for all variables x occurring in $f_1(\vec{w}_1)$ the terms $\sigma_1(x)$ are terminating. Thus, since $r_1\sigma_1$ starts an infinite reduction, there occurs a subterm $f_2(\vec{u}_2)$ in r_1, i.e. $r_1 = C[f_2(\vec{u}_2)]$ for some context C, such that

[1] Tuples of terms $t_1, \ldots, t_n$ are denoted by $\vec{t}$.

- $f_2(\vec{u}_2)\sigma_1$ starts an infinite reduction and
- $\vec{u}_2\sigma_1$ are terminating terms.

The first dependency pair of the infinite $\mathcal{R}$-chain that we construct is $\langle F_1(\vec{w}_1), F_2(\vec{u}_2)\rangle$ corresponding to the rewrite rule $f_1(\vec{w}_1) \to C[f_2(\vec{u}_2)]$. The other dependency pairs of the infinite $\mathcal{R}$-chain are determined in the same way: Let $\langle F_{j-1}(\vec{w}_{j-1}), F_j(\vec{u}_j)\rangle$ be a dependency pair such that $f_j(\vec{u}_j)\sigma_{j-1}$ starts an infinite reduction and the terms $\vec{u}_j\sigma_{j-1}$ are terminating. Again, in zero or more steps $f_j(\vec{u}_j)\sigma_{j-1}$ reduces to $f_j(\vec{v}_j)$ to which a rewrite rule $f_j(\vec{w}_j) \to r_j$ can be applied such that $r_j\sigma_j$ starts an infinite reduction for some substitution σ_j with $\vec{v}_j = \vec{w}_j\sigma_j$.

Similar to the observations above, since $r_j\sigma_j$ starts an infinite reduction, there must be a subterm $f_{j+1}(\vec{u}_{j+1})$ in r_j such that

- $f_{j+1}(\vec{u}_{j+1})\sigma_j$ starts an infinite reduction and
- $\vec{u}_{j+1}\sigma_j$ are terminating terms.

This results in the j-th dependency pair of the chain, viz. $\langle F_j(\vec{w}_j), F_{j+1}(\vec{u}_{j+1})\rangle$. In this way, one obtains the infinite sequence

$$\langle F_1(\vec{w}_1), F_2(\vec{u}_2)\rangle\ \langle F_2(\vec{w}_2), F_3(\vec{u}_3)\rangle\ \langle F_3(\vec{w}_3), F_4(\vec{u}_4)\rangle \ldots$$

It remains to prove that this sequence is really an $\mathcal{R}$-chain.

Note that $F_j(\vec{u}_j\sigma_{j-1}) \to^*_{\mathcal{R}} F_j(\vec{v}_j)$ and $\vec{v}_j = \vec{w}_j\sigma_j$. Since we assume, without loss of generality, that the variables of different occurrences of dependency pairs are disjoint, we obtain one substitution $\sigma = \sigma_1 \circ \sigma_2 \circ \ldots$ (which is the disjoint union of $\sigma_1, \sigma_2, \ldots$) such that $F_j(\vec{u}_j)\sigma \to^*_{\mathcal{R}} F_j(\vec{w}_j)\sigma$ for all j. Thus, we have in fact constructed an infinite $\mathcal{R}$-chain.

Now we show that our criterion is even *necessary* for termination, i.e. we prove that any infinite $\mathcal{R}$-chain corresponds to an infinite reduction. Assume there exists an infinite $\mathcal{R}$-chain.

$$\langle F_1(\vec{s}_1), F_2(\vec{t}_2)\rangle\ \langle F_2(\vec{s}_2), F_3(\vec{t}_3)\rangle\ \langle F_3(\vec{s}_3), F_4(\vec{t}_4)\rangle\ \ldots$$

Hence, there is a substitution σ such that

$$F_2(\vec{t}_2)\sigma \to^*_{\mathcal{R}} F_2(\vec{s}_2)\sigma,\ F_3(\vec{t}_3)\sigma \to^*_{\mathcal{R}} F_3(\vec{s}_3)\sigma,\ \ \ldots$$

thus also

$$f_2(\vec{t}_2)\sigma \to^*_{\mathcal{R}} f_2(\vec{s}_2)\sigma,\ f_3(\vec{t}_3)\sigma \to^*_{\mathcal{R}} f_3(\vec{s}_3)\sigma,\ \ \ldots$$

as the tuple symbols $F_2, F_3, \ldots$ are no defined symbols.

Note that every dependency pair $\langle F(\vec{s}), G(\vec{t})\rangle$ corresponds to a rewrite rule $f(\vec{s}) \to C[g(\vec{t})]$ for some context C. Therefore, this results in the reduction

$$\begin{array}{lllll}
f_1(\vec{s}_1)\sigma & \to & C_1[f_2(\vec{t}_2)]\sigma & & \\
 & & \quad\downarrow_* & & \\
 & & C_1[f_2(\vec{s}_2)]\sigma & \to & C_1[C_2[f_3(\vec{t}_3)]]\sigma \\
 & & & & \quad\quad\downarrow_* \\
 & & & & C_1[C_2[f_3(\vec{s}_3)]]\sigma \quad \to \ldots
\end{array}$$

which is infinite. □

2.2 Checking the termination criterion automatically

The advantage of our termination criterion is that it is particularly well suited for automation. In this section we present a method for proving the absence of infinite chains automatically. For that purpose, we introduce a procedure which, given a TRS, generates a set of inequalities such that the existence of a well-founded ordering satisfying these inequalities is sufficient for termination of the TRS. A well-founded ordering satisfying the generated inequalities can often be synthesized by standard techniques, even if a *direct* termination proof is not possible with these techniques (i.e. even if a well-founded ordering orienting the rules of the TRS cannot be synthesized). For the automation of our method we assume the TRSs to be finite, such that only finitely many dependency pairs have to be considered.

Note that if all chains correspond to a decreasing sequence w.r.t. some well-founded ordering, then all chains must be finite. Hence, to prove the absence of infinite chains, we try to synthesize a well-founded ordering $>$ such that all dependency pairs are decreasing w.r.t. this ordering. More precisely, if for any sequence of dependency pairs $\langle s_1, t_1\rangle\langle s_2, t_2\rangle\langle s_3, t_3\rangle \ldots$ and for any substitution σ with $t_j\sigma \to_{\mathcal{R}}^* s_{j+1}\sigma$ we have

$$s_1\sigma > t_1\sigma > s_2\sigma > t_2\sigma > s_3\sigma > t_3\sigma > \ldots,$$

then no infinite chain exists.

However, for most TRSs, the above inequalities are not satisfied by any well-founded ordering $>$, because the terms $t_j\sigma$ and $s_{j+1}\sigma$ of consecutive dependency pairs in chains are often identical and therefore $t_j\sigma > s_{j+1}\sigma$ does not hold.

But obviously not *all* of the inequalities $s_j\sigma > t_j\sigma$ and $t_j\sigma > s_{j+1}\sigma$ have to be *strict*. For instance, to guarantee the absence of infinite chains it is sufficient if there exists a well-founded *quasi*-ordering[2] $\geq$ such that terms *in* dependency pairs are strictly decreasing (i.e. $s_j\sigma > t_j\sigma$) and terms *in between* dependency

[2] A quasi-ordering $\geq$ is a reflexive and transitive relation and $\geq$ is called *well-founded* if its strict part $>$ is well founded.

pairs are only weakly decreasing (i.e. $t_j\sigma \geq s_{j+1}\sigma$). So for each sequence of dependency pairs as above we only demand

$$s_1\sigma > t_1\sigma \geq s_2\sigma > t_2\sigma \geq s_3\sigma > t_3\sigma \geq \ldots \qquad (4)$$

Note that we cannot determine automatically for which substitutions σ we have $t_j\sigma \to_{\mathcal{R}}^{*} s_{j+1}\sigma$ and moreover, it is practically impossible to examine infinite sequences of dependency pairs. Therefore, in the following we restrict ourselves to *weakly monotonic* quasi-orderings $\geq$ where both $\geq$ and its strict part $>$ are *closed under substitution*. (A quasi-ordering $\geq$ is *weakly monotonic* if $s \geq t$ implies $f(\ldots s \ldots) \geq f(\ldots t \ldots)$.) Then, to guarantee $t_j\sigma \geq s_{j+1}\sigma$ whenever $t_j\sigma \to_{\mathcal{R}}^{*} s_{j+1}\sigma$ holds, it is sufficient to demand $l \geq r$ for all rewrite rules $l \to r$ of the TRS. To ensure $s_j\sigma > t_j\sigma$ for those dependency pairs occurring in possibly infinite chains, we demand $s > t$ for *all* dependency pairs $\langle s, t\rangle$. In fact the existence of such a well-founded ordering is not only sufficient, but even necessary to ensure the absence of infinite chains.

Theorem 7 (Proving termination) *A TRS $\mathcal{R}(D, C, R)$ is terminating iff there exists a well-founded weakly monotonic quasi-ordering $\geq$, where both $\geq$ and $>$ are closed under substitution, such that*

- *$l \geq r$ for all rules $l \to r$ in R and*
- *$s > t$ for all dependency pairs $\langle s, t\rangle$.*

Proof. We first prove that the above conditions are sufficient, i.e. that the existence of such a quasi-ordering implies termination of $\mathcal{R}$. Note that as $l \geq r$ holds for all rules $l \to r$ in R and as $\geq$ is weakly monotonic and closed under substitution, we have $\to_{\mathcal{R}}^{*} \subseteq \geq$, i.e. if $t \to_{\mathcal{R}}^{*} s$ then $t \geq s$.

Suppose there is an infinite $\mathcal{R}$-chain $\langle s_1, t_1\rangle \langle s_2, t_2\rangle \ldots$ Then there exists a substitution σ such that $t_j\sigma \to_{\mathcal{R}}^{*} s_{j+1}\sigma$ holds for all j. As $\to_{\mathcal{R}}^{*} \subseteq \geq$, this implies $t_j\sigma \geq s_{j+1}\sigma$.

Since we have $s_j > t_j$ for all dependency pairs, we obtain the infinite descending sequence

$$s_1\sigma > t_1\sigma \geq s_2\sigma > t_2\sigma \geq s_3\sigma > \ldots$$

which is a contradiction to the well-foundedness of $>$. Therefore, no infinite $\mathcal{R}$-chain exists and hence by Thm. 6, $\mathcal{R}$ is terminating.

Now we prove that the above conditions are even necessary for termination. In fact, we prove a stronger result, i.e. that termination of $\mathcal{R}$ implies termination of the system $\mathcal{R}'$ with the rules

$$R' = R \cup \{s \to t \mid \langle s, t\rangle \text{ is a dependency pair of } \mathcal{R}\}.$$

Hence, the rewrite ordering of $\mathcal{R}'$ is (even) a well-founded *strongly* monotonic ordering $>$ (closed under substitution) satisfying $s > t$ and the *strict* inequalities $l > r$ for all rules of R.

Assume that $\mathcal{R}'$ is not terminating. Hence, there exists a term q_1 starting an infinite $\mathcal{R}'$-reduction.

$$q_1 \to_{\mathcal{R}'} q_2 \to_{\mathcal{R}'} \dots \to_{\mathcal{R}'} q_k \to_{\mathcal{R}'} \dots$$

Clearly, q_1 must contain tuple symbols, because $\mathcal{R}$ is terminating. Without loss of generality we may assume that q_1 is 'minimal', i.e. that none of the proper subterms of q_1 starts an infinite reduction. We show that this implies that the *root* of q_1 is a tuple symbol.

For any term q, let $[\![\, q \,]\!]$ denote the result of replacing all subterms with a root tuple symbol by one and the same new variable y. Note that tuple symbols do not occur in rewrite rules of $\mathcal{R}$. Therefore, $q_j \to_{\mathcal{R}'} q_{j+1}$ implies $[\![\, q_j \,]\!] \to_{\mathcal{R}} [\![\, q_{j+1} \,]\!]$, if the contracted redex in q_j is on a position *above* all tuple symbols. If the contracted redex is below a tuple symbol, then $q_j \to_{\mathcal{R}'} q_{j+1}$ implies $[\![\, q_j \,]\!] = [\![\, q_{j+1} \,]\!]$. If the contracted redex has a tuple root symbol, then $q_j \to_{\mathcal{R}'} q_{j+1}$ also implies $[\![\, q_j \,]\!] = [\![\, q_{j+1} \,]\!]$. The reason is that in this case the reduct also has a tuple root symbol, since all rewrite rules of $\mathcal{R}'$ that have a tuple symbol as root of the left-hand side also have a tuple symbol as root of the right-hand side. Hence, as $\mathcal{R}$ is terminating, after a finite number of steps (say k) all contracted redeces in the infinite reduction are below a tuple symbol or have a tuple root symbol (otherwise $[\![\, q_1 \,]\!]$ would start an infinite $\mathcal{R}$-reduction).

Let q_k have the form $C_k[t_{k,1}, \dots, t_{k,n_k}]$, where C_k is a context without tuple symbols and $t_{k,j}$ are terms with tuple root symbols. Then one of the $t_{k,j}$ starts an infinite reduction. Now assume that the root symbol of q_1 is not a tuple symbol, i.e. q_1 has the form $C_1[t_{1,1}, \dots, t_{1,n_1}]$, where C_1 is a (non-empty) context without tuple symbols and $t_{1,1}, \dots, t_{1,n_1}$ have tuple symbols on their root positions. By induction on the length k of the reduction, one shows that for each $t_{k,j}$ there exists a $t_{1,i}$ such that $t_{1,i} \to^*_{\mathcal{R}'} t_{k,j}$. Thus, q_1 has a proper subterm $t_{1,i}$ which starts an infinite reduction. This is a contradiction to the minimality of q_1.

Hence, q_1 has the form $F_1(\vec{u}_1)$ where $\vec{u}_1$ are terminating terms. So in the infinite reduction, first the arguments $\vec{u}_1$ are reduced in zero or more steps to $\vec{v}_1$, and then $F_1(\vec{v}_1)$ is reduced to $F_2(\vec{u}_2)$, i.e. $\langle F_1(\vec{v}_1), F_2(\vec{u}_2)\rangle$ is an instantiation of a dependency pair. Note that $\vec{u}_2$ are again terminating terms (this is due to the above observations, because all subterms of $\vec{u}_2$ with tuple root symbols already occur in $\vec{v}_1$). So the infinite reduction has the form

$$F_1(\vec{u}_1) \to^*_{\mathcal{R}'} F_1(\vec{v}_1) \to_{\mathcal{R}'} F_2(\vec{u}_2) \to^*_{\mathcal{R}'} F_2(\vec{v}_2) \to_{\mathcal{R}'} F_3(\vec{u}_3) \to^*_{\mathcal{R}'} \dots,$$

where $\vec{u}_j \to^*_{\mathcal{R}'} \vec{v}_j$ holds for all j and $\langle F_j(\vec{v}_j), F_{j+1}(\vec{u}_{j+1})\rangle$ is an instantiation of a dependency pair of $\mathcal{R}$. Let

$$\langle F_1(\vec{s}_1), F_2(\vec{t}_2)\rangle \ \langle F_2(\vec{s}_2), F_3(\vec{t}_3)\rangle \dots$$

be the sequence of these dependency pairs and let $\vec{s}_j\sigma = \vec{v}_j$ and $\vec{t}_j\sigma = \vec{u}_j$. If $\sigma'(x)$ is defined to be $[\![\, \sigma(x) \,]\!]$, then $F_j(\vec{t}_j)\sigma' \to^*_{\mathcal{R}} F_j(\vec{s}_j)\sigma'$ holds for all j.

The reason is that in dependency pairs, tuple symbols occur on root positions only (i.e. $\vec{s}_j$ and $\vec{t}_j$ do not contain tuple symbols). Therefore, $\vec{s}_j\sigma' = [\![\vec{v}_j]\!]$, $\vec{t}_j\sigma' = [\![\vec{u}_j]\!]$ and again $\vec{u}_j \to^*_{\mathcal{R}'} \vec{v}_j$ implies $[\![\vec{u}_j]\!] \to^*_{\mathcal{R}} [\![\vec{v}_j]\!]$. So the above sequence of dependency pairs is an infinite $\mathcal{R}$-chain. By Thm. 6, this is a contradiction to the termination of $\mathcal{R}$. Hence, $\mathcal{R}'$ must also be terminating. $\square$

By the above theorem, termination proofs are now reduced to the search for quasi-orderings satisfying certain constraints. Therefore, the technique of Thm. 7 is very useful to apply standard methods like the recursive path ordering or polynomial interpretations to TRSs where they are not directly applicable.

Example 8 *For instance, in our example we have to find a quasi-ordering satisfying the following inequalities.*

$$
\begin{aligned}
\mathsf{minus}(x, \mathsf{0}) &\geq x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) &\geq \mathsf{minus}(x, y) \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) &\geq \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) &\geq \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \\
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{M}(x, y) \\
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{M}(x, y) \\
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{s}(y))
\end{aligned}
$$

In the next section we show how quasi-orderings satisfying such sets of inequalities can be synthesized automatically using standard techniques.

2.3 Generating suitable quasi-orderings

A well-founded ordering satisfying the constraints in Ex. 8 can for instance be generated by the well-known techniques of *polynomial interpretations* [46]. However, when using polynomial interpretations for *direct* termination proofs of TRSs, the polynomials have to be (strongly) monotonic in all their arguments, i.e. $s > t$ implies $f(\ldots s \ldots) > f(\ldots t \ldots)$. But for the approach of this paper, we only need a *weakly* monotonic quasi-ordering satisfying the inequalities. Thus, $s > t$ only implies $f(\ldots s \ldots) \geq f(\ldots t \ldots)$. Hence, when using our method it suffices to find a polynomial interpretation with weakly monotonic polynomials, which do not necessarily depend on all their arguments. For example, we may map $\mathsf{minus}(x, y)$ to the polynomial x which does not depend on the second argument y.

Then the inequalities in Ex. 8 are satisfied by a polynomial ordering where $\mathsf{0}$ is mapped to 0, $\mathsf{s}(x)$ is mapped to $x+1$, and $\mathsf{minus}(x, y)$, $\mathsf{quot}(x, y)$, $\mathsf{M}(x, y)$ and $\mathsf{Q}(x, y)$ are all mapped to x. Methods for the automated synthesis of polynomial orderings have for instance been developed in [31, 53]. In this way, termination of this TRS can be proved fully automatically, although a direct termination proof with simplification orderings was not possible.

Instead of polynomial orderings one can also use path orderings, which can easily be generated automatically. However, these path orderings are always strongly monotonic, whereas in our method we only need a weakly monotonic ordering. For that reason, before synthesizing a suitable path ordering some of the arguments of function symbols may be eliminated. For instance, one may eliminate the second argument of the function symbol minus. Then every term $\mathsf{minus}(s,t)$ in the inequalities is replaced by $\mathsf{m}(s)$ (where m is a new unary function symbol). By comparing the terms resulting from this replacement (instead of the original terms) we can take advantage of the fact that minus does not have to be strongly monotonic in its second argument.

Example 9 *In this way, the inequalities of Ex. 8 are transformed into*

$$\begin{array}{rcl}
\mathsf{m}(x) & \geq & x \\
\mathsf{m}(\mathsf{s}(x)) & \geq & \mathsf{m}(x) \\
\mathsf{quot}(\mathsf{0},\mathsf{s}(y)) & \geq & \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x),\mathsf{s}(y)) & \geq & \mathsf{s}(\mathsf{quot}(\mathsf{m}(x),\mathsf{s}(y))) \\
\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) & > & \mathsf{M}(x,y) \\
\mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)) & > & \mathsf{M}(x,y) \\
\mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)) & > & \mathsf{Q}(\mathsf{m}(x),\mathsf{s}(y)).
\end{array}$$

These inequalities are satisfied by the recursive path ordering using the precedence $\mathsf{quot} \rhd \mathsf{s} \rhd \mathsf{m}$ *and* $\mathsf{Q} \rhd \mathsf{M}$.

Apart from eliminating arguments of function symbols, another possibility is to replace functions by one of their arguments. So instead of deleting the second argument of minus one could replace all terms $\mathsf{minus}(s,t)$ by minus' first argument s. Then the resulting inequalities are again satisfied by the recursive path ordering. To perform this elimination of arguments resp. of function symbols we introduce the following concept.

Definition 10 (Argument filtering TRS) *An* argument filtering TRS[3] *for the signature $\mathcal{F}$ (AFS for short) is a TRS whose rewrite rules are of the form*

$$\begin{array}{l}
f(x_1,\ldots,x_n) \to g(y_1,\ldots y_k) \quad \textit{or} \\
f(x_1,\ldots,x_n) \to x_i
\end{array}$$

where $x_1,\ldots,x_n$ are pairwise different variables, $y_1,\ldots,y_k$ are pairwise different variables out of $x_1,\ldots,x_n$, $g \notin \mathcal{F}$, and for every function symbol $f \in \mathcal{F}$ there is at most one f-rule in the AFS.

[3]Argument filtering TRSs are a special form of recursive program schemes [15, 41].

From a rewriting point of view AFSs are quite simple, because every AFS is complete. Hence, for any term t the normal form $t\downarrow_{\mathcal{A}}$ w.r.t. an AFS $\mathcal{A}$ is unique.

The following theorem states that in order to find a quasi-ordering satisfying a particular set of inequalities, one may first normalize the terms in the inequalities with respect to an AFS. Subsequently, one only has to find a quasi-ordering that satisfies these modified inequalities. Note that for a finite signature there are only finitely many AFSs (up to renaming of the symbols). Hence, by combining the synthesis of a suitable AFS with well-known techniques for the generation of (*strongly* monotonic) simplification orderings, now the search for a *weakly* monotonic ordering satisfying the constraints can be automated.

Theorem 11 (Preservation under argument filtering) *Let $\mathcal{A}$ be an AFS and let IN be a set of inequalities. If the inequalities*

$$\{s\downarrow_{\mathcal{A}} > t\downarrow_{\mathcal{A}} \mid s > t \in IN\} \cup \{s\downarrow_{\mathcal{A}} \geq t\downarrow_{\mathcal{A}} \mid s \geq t \in IN\}$$

are satisfied by a well-founded weakly monotonic quasi-ordering (where both the quasi-ordering and its strict part are closed under substitution), then there also exists such a quasi-ordering satisfying the inequalities IN.

Proof. Assuming that the normalized inequalities are satisfied by a quasi-ordering $\geq$, a relation $\geq'$ on terms is defined where the terms are first normalized w.r.t. $\mathcal{A}$ and then compared w.r.t. the quasi-ordering $\geq$ (i.e. $s \geq' t$ iff $s\downarrow_{\mathcal{A}} \geq t\downarrow_{\mathcal{A}}$). It is straightforward to see that $\geq'$ is a well-founded quasi-ordering satisfying the inequalities IN.

For any substitution σ, let $\sigma\downarrow_{\mathcal{A}}$ denote the substitution which results from σ by normalizing all terms in the range of σ w.r.t. $\mathcal{A}$. Then, for all terms t and all substitutions σ we have $(t\sigma)\downarrow_{\mathcal{A}}= (t\downarrow_{\mathcal{A}})(\sigma\downarrow_{\mathcal{A}})$. Hence, both $\geq'$ and its strict part $>'$ are closed under substitution. Moreover, $\geq'$ is weakly monotonic, because $s\downarrow_{\mathcal{A}} \geq t\downarrow_{\mathcal{A}}$ implies $f(\ldots x \ldots)\downarrow_{\mathcal{A}} [x/s\downarrow_{\mathcal{A}}] \geq f(\ldots x \ldots)\downarrow_{\mathcal{A}} [x/t\downarrow_{\mathcal{A}}]$ resp. $f(\ldots s \ldots)\downarrow_{\mathcal{A}} \geq f(\ldots t \ldots)\downarrow_{\mathcal{A}}$ (here, x is a variable occurring just once in $f(\ldots x \ldots)$). $\square$

By the above theorem in combination with Thm. 7 it is now possible to prove termination of the TRS in Ex. 2 automatically using the recursive path ordering. After normalizing all inequalities in Ex. 8 w.r.t. the one-rule AFS

$$\mathsf{minus}(x,y) \to \mathsf{m}(x),$$

one obtains the inequalities in Ex. 9 which are satisfied by the recursive path ordering.

2.4 Refinement using dependency graphs

While the method of Thm. 7 can be successfully used for automated termination proofs, in this section we introduce a refinement of this approach, i.e. we show how the constraints obtained can be weakened. By this weakening, the (automatic) search for a suitable quasi-ordering satisfying these constraints can be eased significantly.

In order to ensure that every possible infinite chain results in an infinite decreasing sequence of terms, in Thm. 7 we demanded $s > t$ for *all* dependency pairs $\langle s, t\rangle$. However, in many examples several dependency pairs can occur at most once in any chain and therefore they do not have to be considered at all. Moreover, for the other dependency pairs it is often sufficient if just *some* of them are strictly decreasing, whereas others may be weakly decreasing.

Example 12 *The dependency pair* $\langle \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{M}(x, y)\rangle$ *occurs at most* once *in any chain: Recall that a dependency pair* $\langle v, w\rangle$ *may only follow a pair* $\langle s, t\rangle$ *in a chain, if there exists a substitution* σ *such that* $t\sigma \to_{\mathcal{R}}^{*} v\sigma$. *As the tuple symbol* M *is not a defined symbol,* $\mathsf{M}(x, y)\sigma$ *can only be reduced to terms with the same root symbol* M. *Hence, the dependency pair (2) can only be succeeded by the dependency pair (1) which in turn can only be succeeded by itself, i.e. (2) can never occur twice in a chain. Therefore, any possibly infinite chain has an infinite tail in which the dependency pair* $\langle \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{M}(x, y)\rangle$ *does not occur. Therefore it suffices to show that no infinite chain exists consisting of the other dependency pairs.*

For the TRS of Ex. 2 it is not necessary to reduce the number of constraints in order to prove termination automatically. However, for the following TRS we have to get rid of a constraint in order to use a simplification ordering for satisfying the inequalities.

Example 13 *Let us extend the TRS of Ex. 2 by three additional rules. We now write infix operators for the defined symbols* minus *and* plus *to ease readability.*

$$\begin{array}{rcl}
x - \mathsf{0} & \to & x \\
\mathsf{s}(x) - \mathsf{s}(y) & \to & x - y \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \to & \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(x - y, \mathsf{s}(y))) \\
\mathsf{0} + y & \to & y \\
\mathsf{s}(x) + y & \to & \mathsf{s}(x + y) \\
(x - y) - z & \to & x - (y + z)
\end{array}$$

The dependency pairs of this TRS are the dependency pairs as given in Ex. 4 together with the dependency pairs

$$\langle \mathsf{P}(\mathsf{s}(x), y), \mathsf{P}(x, y)\rangle \tag{5}$$

$$\langle \mathsf{M}(x - y, z), \mathsf{P}(y, z)\rangle \tag{6}$$

$$\langle \mathsf{M}(x - y, z), \mathsf{M}(x, y + z)\rangle \tag{7}$$

where P *is the tuple symbol for the defined symbol '+'. To prove termination according to Thm. 7 we now obtain the following inequalities.*

$$
\begin{array}{rclrcl}
x - 0 & \geq & x & \mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{M}(x, y) \\
\mathsf{s}(x) - \mathsf{s}(y) & \geq & x - y & \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{M}(x, y) \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \geq & \mathsf{0} & \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{Q}(x - y, \mathsf{s}(y)) \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \geq & \mathsf{s}(\mathsf{quot}(x - y, \mathsf{s}(y))) & \mathsf{P}(\mathsf{s}(x), y) & > & \mathsf{P}(x, y) \\
\mathsf{0} + y & \geq & y & \mathsf{M}(x - y, z) & > & \mathsf{P}(y, z) \\
\mathsf{s}(x) + y & \geq & \mathsf{s}(x + y) & \mathsf{M}(x - y, z) & > & \mathsf{M}(x, y + z) \\
(x - y) - z & \geq & x - (y + z) & & &
\end{array}
$$

Since the inequality $\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) > \mathsf{Q}(x - y, \mathsf{s}(y))$ *has an instantiation that is self-embedding, no simplification ordering satisfies these inequalities directly. In order to apply techniques for the automated generation of simplification orderings, therefore Thm. 11 has to be used first. We have to normalize the inequalities w.r.t. an AFS* $\mathcal{A}$ *that rewrites* $x - y$ *to* $\mathsf{m}(x)$ *or to* x *(this is forced by the inequalities). But thereafter, the inequality*

$$\mathsf{M}(x - y, z)\!\downarrow_{\mathcal{A}} > \mathsf{P}(y, z)\!\downarrow_{\mathcal{A}}$$

in combination with the other remaining inequalities cannot be satisfied by any well-founded monotonic ordering closed under substitution. (The reason is that y *does not occur in* $\mathsf{M}(x - y, z)\!\downarrow_{\mathcal{A}}$ *any more, whereas* $\mathsf{P}(y, z)\!\downarrow_{\mathcal{A}}$ *still depends on* y*, as* $\mathcal{A}$ *must not eliminate the first argument of* P*.) Hence, an automatic termination proof fails at this point.*

Recall that one may delete all dependency pairs which occur at most once in any chain. In the example above, this elimination of constraints results in a set of inequalities for which a suitable quasi-ordering can be generated automatically, whereas this was not possible for the original set of constraints.

Example 14 *For the TRS of Ex. 13, the constraint* $\mathsf{M}(\ldots) > \mathsf{P}(\ldots)$ *is unnecessary to ensure the absence of infinite chains. The reason is that in any chain the dependency pair (6) can occur at most once, since the only dependency pair following (6) can be (5) and (5) can only be followed by itself.*

To determine those dependency pairs which may occur infinitely often in a chain we define a graph of dependency pairs where those dependency pairs that possibly occur consecutive in a chain are connected. In this way, any infinite chain corresponds to a cycle in the graph (as we restricted ourselves to finite TRSs).

Definition 15 (Dependency graph) *The* dependency graph *of a TRS* $\mathcal{R}$ *is the directed graph whose nodes are the dependency pairs and there is an arc from* $\langle s, t\rangle$ *to* $\langle v, w\rangle$ *if* $\langle s, t\rangle\langle v, w\rangle$ *is an* $\mathcal{R}$*-chain.*

Thus, the dependency graph connects dependency pairs that form a chain, i.e. for some instantiation the right-hand side of one pair reduces to the left-hand side of the other pair. Every chain corresponds to a path in the dependency graph. Note however that the converse does not hold, i.e. a path in this graph does not necessarily correspond to a chain, since instead of using one 'global' substitution for all dependency pairs in a chain, here one may use different 'local' substitutions for consecutive dependency pairs.

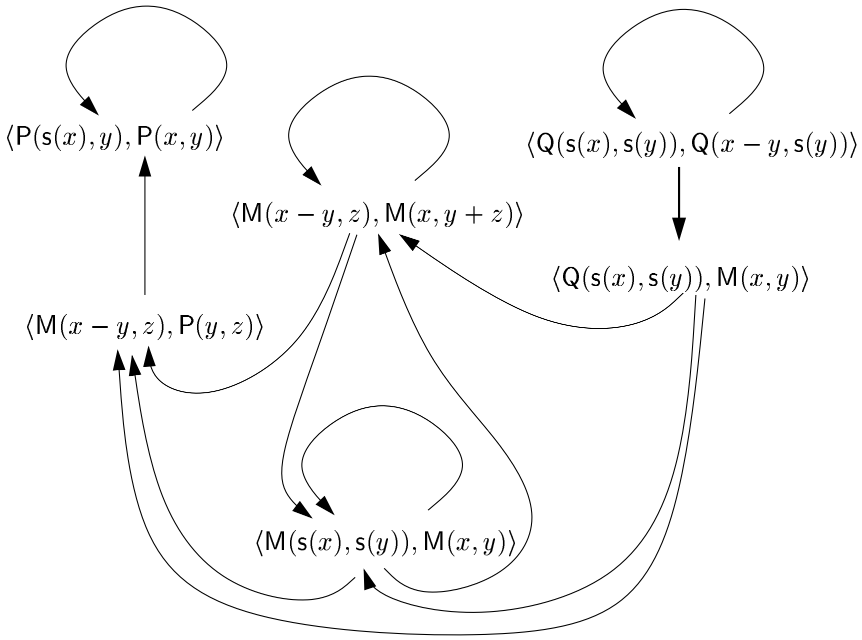

Figure 1: The dependency graph for the TRS of Ex. 13.

Now to prove termination of a TRS it is sufficient if $s > t$ holds for at least one dependency pair $\langle s, t\rangle$ on each cycle of the dependency graph and if $s \geq t$ holds for all other dependency pairs on cycles. Dependency pairs that do not occur on a cycle can be ignored.

Example 16 *For the TRS of Ex. 13 we obtain the dependency graph in Fig. 1. Hence, this results in the following set of inequalities.*

$$
\begin{array}{rcl}
x - \mathsf{0} & \geq & x \\
\mathsf{s}(x) - \mathsf{s}(y) & \geq & x - y \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \geq & \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \geq & \mathsf{s}(\mathsf{quot}(x - y, \mathsf{s}(y))) \\
\mathsf{0} + y & \geq & y \\
\mathsf{s}(x) + y & \geq & \mathsf{s}(x + y) \\
(x - y) - z & \geq & x - (y + z)
\end{array}
\qquad
\begin{array}{rcl}
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{M}(x, y) \\
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{Q}(x - y, \mathsf{s}(y)) \\
\mathsf{P}(\mathsf{s}(x), y) & > & \mathsf{P}(x, y) \\
\mathsf{M}(x - y, z) & > & \mathsf{M}(x, y + z)
\end{array}
$$

The inequalities obtained are satisfied by the polynomial ordering where $\mathsf{0}$ *is mapped to* 0, $\mathsf{s}(x)$ *is mapped to* $x + 2$, $x - y$ *is mapped to* $x + 1$, $\mathsf{quot}(x, y)$ *is mapped to* $2x$, $\mathsf{M}(x, y)$ *and* $\mathsf{Q}(x, y)$ *are mapped to* x, *and both* $+$ *and* P *are mapped to addition. By normalizing the inequalities with respect to the argument filtering TRS*

$$
\begin{array}{rcl}
x - y & \to & \mathsf{m}(x) \\
\mathsf{M}(x, y) & \to & x
\end{array}
$$

the resulting inequalities are also satisfied by the recursive path ordering. Thus, by the following theorem, termination of the TRS is proved.

Theorem 17 (Dependency graph refinement) *A TRS* $\mathcal{R}(D, C, R)$ *is terminating iff there exists a well-founded weakly monotonic quasi-ordering* $\geq$, *where both* $\geq$ *and* $>$ *are closed under substitution, such that*

- $l \geq r$ *for all rules* $l \to r$ *in* R,
- $s \geq t$ *for all dependency pairs* $\langle s, t \rangle$ *on a cycle of the dependency graph, and*
- $s > t$ *for at least one dependency pair* $\langle s, t \rangle$ *on each cycle of the dependency graph.*

Proof. The proof is similar to the proof of Thm. 7 with the additional observation that any infinite $\mathcal{R}$-chain corresponds to an infinite path in the dependency graph. This infinite path traverses at least one cycle infinitely many times, since there are only finitely many dependency pairs. Since at least one dependency pair in this cycle corresponds to a strict inequality, the chain corresponds to a descending sequence of terms containing infinitely many strict inequalities.

Thm. 7 directly implies that the above conditions are also *necessary* for the termination of $\mathcal{R}$. $\square$

However, to perform termination proofs according to Thm. 17, we have to construct the dependency graph automatically. Unfortunately, in general this is not possible, since for two dependency pairs $\langle s, t\rangle, \langle v, w\rangle$ it is undecidable whether they form a chain (i.e. whether there exists a substitution σ such that $t\sigma \to_{\mathcal{R}}^* v\sigma$).

Therefore, we introduce a technique to approximate the dependency graph, i.e. the technique computes a *superset* of those terms t, v where $t\sigma \to_{\mathcal{R}}^* v\sigma$ holds for some substitution σ. We call terms t, v suggested by our technique *connectable terms*. In this way, (at least) all cycles that occur in the dependency graph and hence all possibly infinite chains can be determined. So by computing a graph *containing* the dependency graph we can indeed apply the method of Thm. 17 for automated termination proofs.

For the computation of connectable terms we use syntactic unification. This unification is not performed on the terms of the dependency pairs directly, but one of the terms is modified first. If t is a term with a constructor root symbol c, then $t\sigma$ can only be reduced to terms which have the same root symbol c. If the root symbol of t is defined, then this does not give us any direct information about those terms $t\sigma$ can be reduced to. For that reason, to determine whether the term t is connectable to v, we replace all subterms in t that have a defined root symbol by a new variable and check whether this modification of t unifies with v.

For example, $\mathsf{P}(\ldots)$ is not connectable to $\mathsf{M}(\ldots)$. On the other hand, the term $\mathsf{Q}(x - y, \mathsf{s}(y))$ is connectable to $\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y))$, because before unification, the subterm $x - y$ is replaced by a new variable.

In order to ensure that t is connectable to v whenever there exists a substitution σ such that $t\sigma \to_{\mathcal{R}}^* v\sigma$, before unification we also have to *rename* multiple occurrences of the same variable x in t. (The reason is that different occurrences of $x\sigma$ can reduce to different terms.) As an example consider the following TRS from Toyama [56].

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

The only dependency pair, viz. $\langle \mathsf{F}(0, 1, x), \mathsf{F}(x, x, x)\rangle$, is on a cycle of the dependency graph, because $\mathsf{F}(x, x, x)\sigma$ reduces to $\mathsf{F}(0, 1, x')\sigma$, if σ replaces x and x' by $\mathsf{g}(0, 1)$. Note however that $\mathsf{F}(x, x, x)$ does not unify with $\mathsf{F}(0, 1, x')$, i.e. if we would not rename $\mathsf{F}(x, x, x)$ to $\mathsf{F}(x_1, x_2, x_3)$ before the unification, then we could not determine this cycle of the dependency graph and we would falsely conclude termination of this (non-terminating) TRS.

To perform the required modification on the term t, two functions CAP and REN are introduced. For any term t, $\mathrm{CAP}(t)$ results from replacing all subterms of t that have a defined root symbol by different new variables and $\mathrm{REN}(t)$ results from replacing all variables in t by different fresh variables. In particular, different occurrences of the same variable are also replaced by different new variables.

Definition 18 (Connectable terms) *Let D be the set of defined symbols. Then the functions* CAP *and* REN *from terms to terms are inductively defined as*

$$\begin{array}{rcll} \textsc{cap}(x) & = & x, & \textit{for variables } x \\ \textsc{cap}(f(t_1,\ldots,t_n)) & = & \left\{\begin{array}{l} y, \\ f(\textsc{cap}(t_1),\ldots,\textsc{cap}(t_n)), \end{array}\right. & \begin{array}{l} \textit{if } f \in D \\ \textit{if } f \notin D \end{array} \\ \\ \textsc{ren}(x) & = & y, & \textit{for variables } x \\ \textsc{ren}(f(t_1,\ldots,t_n)) & = & f(\textsc{ren}(t_1),\ldots,\textsc{ren}(t_n)) & \end{array}$$

where y is the next variable in an infinite list of fresh variables $y_1, y_2, \ldots$

For any terms t and v, the term t is connectable *to v if* REN(CAP(t)) *and v are unifiable.*

Strictly speaking, neither CAP nor REN are proper functions, because one time we have $\textsc{ren}(x) = y_1$ and the next time we obtain $\textsc{ren}(x) = y_2$. Of course, CAP and REN can easily be transformed into proper functions by giving CAP and REN a second argument which contains the next fresh variable that has not yet been used. However, we omitted this second argument to ease readability. For example, we have

$$\textsc{ren}(\textsc{cap}(\mathsf{Q}(x - y, \mathsf{s}(y)))) = \textsc{ren}(\mathsf{Q}(y_1, \mathsf{s}(y))) = \mathsf{Q}(y_2, \mathsf{s}(y_3))$$

and

$$\textsc{ren}(\textsc{cap}(\mathsf{Q}(x, x))) = \textsc{ren}(\mathsf{Q}(x, x)) = \mathsf{Q}(y_4, y_5).$$

As $\textsc{ren}(t)$ is always a linear term, to check whether two terms are connectable we can even use a unification algorithm without occur check.

To approximate the dependency graph, we draw an arc from a dependency pair $\langle s, t\rangle$ to $\langle v, w\rangle$ whenever t is connectable to v. In this way, for our example the dependency graph of Fig. 1 is constructed automatically. So termination of the TRS in Ex. 13 can be proved automatically.

The following theorem proves the soundness of this approach: by computing connectable terms we in fact obtain a supergraph of the dependency graph. Using this supergraph, we can now prove termination according to Thm. 17.

Theorem 19 (Computing dependency graphs) *Let $\mathcal{R}$ be a TRS and let $\langle s, t\rangle, \langle v, w\rangle$ be dependency pairs. If there is an arc from $\langle s, t\rangle$ to $\langle v, w\rangle$ in the dependency graph, then t is connectable to v.*

Proof. By induction on the structure of t we prove that if there exists a substitution σ with $t\sigma \to_{\mathcal{R}}^* u$ for some term u, then $\textsc{ren}(\textsc{cap}(t))$ matches u. So in particular, if $t\sigma \to_{\mathcal{R}}^* v\sigma$, then $\textsc{ren}(\textsc{cap}(t))$ matches $v\sigma$. As $\textsc{ren}(\textsc{cap}(t))$ only contains new variables, this implies that $\textsc{ren}(\textsc{cap}(t))$ and v are unifiable.

Assume that $t\sigma \to_{\mathcal{R}}^* u$ for some term u. If t is a variable or if $t = f(t_1, \ldots, t_k)$ for a defined symbol f, then $\textsc{ren}(\textsc{cap}(t))$ is a variable, hence it matches u.

If $t = c(t_1, \ldots, t_k)$ for some constructor c, then

$$\text{REN}(\text{CAP}(t)) = c(\text{REN}(\text{CAP}(t_1)), \ldots, \text{REN}(\text{CAP}(t_k))).$$

In this case, u has to be of the form $c(u_1, \ldots, u_k)$ and $t_j\sigma \to_{\mathcal{R}}^* u_j$ holds for all j. By the induction hypothesis we obtain that $\text{REN}(\text{CAP}(t_j))$ matches u_j. Since the variables in $\text{REN}(\text{CAP}(t_j))$ are disjoint from the variables in $\text{REN}(\text{CAP}(t_i))$ for all $i \neq j$, $\text{REN}(\text{CAP}(t))$ also matches u. □

2.5 Refined termination proofs by narrowing dependency pairs

By the refinement of dependency graphs, Thm. 17 provides us with a powerful technique to prove that there exists no infinite chain of dependency pairs. However, there are still examples where the automation of our method fails.

Example 20 *For instance, let us replace the last rule of the TRS in Ex. 13 by a 'commutativity' rule (here,* $\mathsf{s0}$ *abbreviates* $\mathsf{s(0)}$ *etc.).*

$$\begin{array}{rcl}
x - \mathsf{0} & \to & x \\
\mathsf{s}(x) - \mathsf{s}(y) & \to & x - y \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \to & \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(x - y, \mathsf{s}(y))) \\
\mathsf{0} + y & \to & y \\
\mathsf{s}(x) + y & \to & \mathsf{s}(x + y) \\
(x - \mathsf{s0}) + (y - \mathsf{ss}z) & \to & (y - \mathsf{ss}z) + (x - \mathsf{s0})
\end{array}$$

One of the new dependency pairs, viz.

$$\langle \mathsf{P}(x - \mathsf{s0}, y - \mathsf{ss}z), \mathsf{P}(y - \mathsf{ss}z, x - \mathsf{s0}) \rangle, \tag{8}$$

forms a cycle of the dependency graph. Hence, due to Thm. 17 we have to find an ordering such that the dependency pair (8) is strictly decreasing, i.e.

$$\mathsf{P}(x - \mathsf{s0}, y - \mathsf{ss}z) > \mathsf{P}(y - \mathsf{ss}z, x - \mathsf{s0}).$$

In order to apply techniques for the synthesis of simplification orderings, we have to normalize the inequalities w.r.t. an AFS again which rewrites $x - y$ *to* $\mathsf{m}(x)$ *(or to* x*). However, the resulting constraint*

$$\mathsf{P}(\mathsf{m}(x), \mathsf{m}(y)) > \mathsf{P}(\mathsf{m}(y), \mathsf{m}(x))$$

is not satisfied by any well-founded ordering closed under substitution. Hence, in this way termination of the TRS cannot be proved automatically.

Up to now we demanded constraints which ensure that in any sequence of dependency pairs $\langle s_1, t_1\rangle\,\langle s_2, t_2\rangle \ldots$ and for all substitutions σ with $t_j\sigma \to_{\mathcal{R}}^* s_{j+1}\sigma$ we have[4]

$$s_1\sigma \;>\; t_1\sigma \;\geq\; s_2\sigma \;>\; t_2\sigma \;\geq\; \ldots \tag{4}$$

So we demanded $s > t$ for the dependency pairs $\langle s, t\rangle$. But instead of the requirement that there should be a strict decrease *in* dependency pairs, it would also be sufficient if the ordering is strict *between* two dependency pairs. Thus, if $\langle s, t\rangle$ and $\langle v, w\rangle$ are consecutive in a chain, then instead of $s\sigma > t\sigma \geq v\sigma$ one could demand $s\sigma \geq t\sigma$ and $t\sigma > v\sigma$ for all substitutions σ with $t\sigma \to_{\mathcal{R}}^* v\sigma$.

To achieve this effect we replace the original dependency pairs by new pairs of terms. Subsequently, we demand that these *new* pairs of terms are *strictly* decreasing. Note that if the reduction from $t\sigma$ to $v\sigma$ is always of the form

$$t\sigma \to_{\mathcal{R}} t'\sigma \to_{\mathcal{R}}^* v\sigma,$$

then instead of $s\sigma > t\sigma \geq v\sigma$ we may also require $s\sigma > t'\sigma \geq v\sigma$. To compute the terms t' we use *narrowing* (cf. e.g. [39]).

Definition 21 (Narrowing) *Let $\mathcal{R}$ be a TRS. A term t* narrows *to a term t' via the substitution μ (denoted by $t \rightsquigarrow_{\mathcal{R}} t'$), if there exists a non-variable position p in t, μ is the most general unifier of $t|_p$ and l for some rewrite rule $l \to r$ of $\mathcal{R}$, and $t' = t\mu[r\mu]_p$. (Here, the variables of $l \to r$ must have been renamed to fresh variables.)*

If a dependency pair $\langle s, t\rangle$ is followed by another dependency pair $\langle v, w\rangle$ in a chain, and if t is not already unifiable with v (i.e. at least one rule of $\mathcal{R}$ is needed to reduce $t\sigma$ to $v\sigma$), then we may perform all possible narrowing steps on t (resulting in new terms $t_1, \ldots, t_n$) in order to examine the reduction from $t\sigma$ to $v\sigma$.

However, instead of only narrowing right-hand sides of dependency pairs $\langle s, t\rangle$, the substitutions derived from narrowing the term t should also be applied on the left-hand side s of the pair $\langle s, t\rangle$. Thus, if $t \rightsquigarrow_{\mathcal{R}} t_1, \ldots, t \rightsquigarrow_{\mathcal{R}} t_n$ are *all* possible narrowings of t (via the substitutions $\mu_1, \ldots, \mu_n$), then instead of

$$s\sigma \;\; > t\sigma \;\; \geq v\sigma \qquad\qquad \text{for all } \sigma \text{ with } t\sigma \;\; \to_{\mathcal{R}}^* v\sigma$$

it is sufficient to demand

$$\begin{array}{ll} s\mu_1\sigma > t_1\sigma \geq v\sigma & \qquad\text{for all } \sigma \text{ with } t_1\sigma \to_{\mathcal{R}}^* v\sigma, \\ \vdots & \\ s\mu_n\sigma > t_n\sigma \geq v\sigma & \qquad\text{for all } \sigma \text{ with } t_n\sigma \to_{\mathcal{R}}^* v\sigma. \end{array}$$

[4]By taking the dependency graph into account, this requirement has been weakened, i.e. it is sufficient if just an infinite *subset* of dependency pairs is strictly decreasing in any possibly infinite chain.

Hence, we may replace the dependency pair $\langle s,t\rangle$ by the n new pairs $\langle s\mu_1,t_1\rangle$, $\ldots,\langle s\mu_n,t_n\rangle$. For that purpose instead of narrowing *terms* we introduce the concept of narrowing *pairs* of terms.

Definition 22 (Narrowing pairs) *Let $\mathcal{R}$ be a TRS. If a term t narrows to a term t' via the substitution μ, then we say that the* pair *of terms $\langle s,t\rangle$ narrows to the pair $\langle s\mu,t'\rangle$.*

Example 23 *For Ex. 20, the instantiated right-hand side*

$$\mathsf{P}(y-\mathsf{ss}z,x-\mathsf{s0})\sigma$$

of dependency pair (8) can only reduce to an instantiation of a left-hand side of a dependency pair if one of the minus*-rules is applied to $(y-\mathsf{ss}z)\sigma$ or $(x-\mathsf{s0})\sigma$. So instead of the dependency pair (8) we may regard its two narrowings*

$$\langle\mathsf{P}(x-\mathsf{s0},\mathsf{s}y-\mathsf{ss}z),\mathsf{P}(y-\mathsf{s}z,x-\mathsf{s0})\rangle \tag{9}$$

$$\langle\mathsf{P}(\mathsf{s}x-\mathsf{s0},y-\mathsf{ss}z),\mathsf{P}(y-\mathsf{ss}z,x-\mathsf{0})\rangle. \tag{10}$$

Now the constraints that the left-hand sides of the new pairs (9) and (10) should be greater than their right-hand sides (together with the remaining constraints for this system) are again satisfied by the orderings mentioned in Ex. 16. Hence, in this way termination of the TRS can be proved automatically.

If $\mathcal{P}$ is the set of all dependency pairs of $\mathcal{R}$, then instead of checking whether there exists an infinite $\mathcal{R}$-chain of pairs from $\mathcal{P}$ now it suffices to show that there is no infinite $\mathcal{R}$-chain of pairs from $\mathcal{P}\setminus\{\langle s,t\rangle\}\cup\{\langle s\mu_1,t_1\rangle,\ldots,\langle s\mu_n,t_n\rangle\}$, where $\langle s\mu_1,t_1\rangle,\ldots,\langle s\mu_n,t_n\rangle$ are all narrowings of $\langle s,t\rangle$. (So with this refinement we have to regard chains of pairs of terms which are no dependency pairs any more.) Note that any pair $\langle s,t\rangle$ can only be narrowed (in one step) to *finitely many* pairs $\langle s',t'\rangle$ (up to variable renaming) and these pairs $\langle s',t'\rangle$ can easily be computed automatically. In particular, if a dependency pair $\langle s,t\rangle$ has *no* narrowings, then it does not have to be considered any more for the termination proof.

However, the following two examples demonstrate that a pair $\langle s,t\rangle$ in $\mathcal{P}$ may only be replaced by its narrowings, if t does not unify with any left-hand side of a pair in $\mathcal{P}$ and if t is a *linear* term.

Example 24 *The following non-terminating TRS*

$$\begin{array}{rcl}\mathsf{f}(\mathsf{0}) & \to & \mathsf{f}(\mathsf{0})\\ \mathsf{0} & \to & \mathsf{1}\end{array}$$

has one cycle in the dependency graph formed by an arc from the dependency pair $\langle\mathsf{F}(\mathsf{0}),\mathsf{F}(\mathsf{0})\rangle$ to itself. Narrowing this pair, although its right-hand side unifies with its left-hand side, results in $\langle\mathsf{F}(\mathsf{0}),\mathsf{F}(\mathsf{1})\rangle$. Now the new right-hand side

$\mathsf{F}(1)$ *is not connectable to* $\mathsf{F}(0)$ *any more. Hence, by ignoring the unification condition, the only cycle in the dependency graph would be erroneously removed and therefore termination of this TRS could falsely be concluded.*

Similarly, the linearity of the right-hand side plays a crucial role, as can be seen from the non-terminating TRS

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

where $\langle \mathsf{F}(\mathsf{s}(x)), \mathsf{F}(\mathsf{g}(x,x))\rangle$ *forms the only cycle of the dependency graph. However, by ignoring the linearity condition, this dependency pair could be deleted, as the term* $\mathsf{F}(\mathsf{g}(x,x))$ *cannot be narrowed. Hence, no cycle exists in the new dependency graph and therefore termination of the TRS would falsely be concluded*[5]*.*

The following theorem proves that under the above conditions the replacement of dependency pairs by their narrowings maintains the sufficiency and necessity of our termination criterion.

Theorem 25 (Narrowing refinement for termination) *Let $\mathcal{R}$ be a TRS and let $\mathcal{P}$ be a set of pairs of terms. Let $\langle s,t\rangle \in \mathcal{P}$ such that t is linear and for all $\langle v,w\rangle \in \mathcal{P}$ the terms t and v are not unifiable (after renaming the variables). Let*

$$\mathcal{P}' = \mathcal{P} \setminus \{\langle s,t\rangle\} \cup \{\langle s',t'\rangle \mid \langle s',t'\rangle \textit{ is a narrowing of } \langle s,t\rangle\}.$$

There exists an infinite $\mathcal{R}$-chain of pairs from $\mathcal{P}$ iff there exists an infinite $\mathcal{R}$-chain of pairs from $\mathcal{P}'$.

Proof. It suffices to prove that for every $\langle s,t\rangle \in \mathcal{P}$ the sequence

$$\ldots\ \langle v_1,w_1\rangle\ \langle s,t\rangle\ \langle v_2,w_2\rangle\ \ldots$$

(of pairs from $\mathcal{P}$ or $\mathcal{P}'$) is an $\mathcal{R}$-chain iff there exists a narrowing $\langle s',t'\rangle$ of $\langle s,t\rangle$ such that $\ldots \langle v_1,w_1\rangle\ \langle s',t'\rangle\ \langle v_2,w_2\rangle \ldots$ is an $\mathcal{R}$-chain. Here, $\langle s,t\rangle$ resp. $\langle s',t'\rangle$ may also be the first pair in the chain (i.e. $\langle v_1,w_1\rangle$ may be missing).

If this has been proved then all occurrences of $\langle s,t\rangle$ in an infinite chain may be replaced by pairs from $\mathcal{P}'$. In an analogous way, every infinite chain of pairs from $\mathcal{P}'$ can also be transformed into an infinite chain of pairs from $\mathcal{P}$.

For the first direction, let $\ldots \langle v_1,w_1\rangle\ \langle s,t\rangle\ \langle v_2,w_2\rangle \ldots$ be an $\mathcal{R}$-chain. Hence, there must be a substitution such that for all pairs, the instantiated

[5]The problem is that the first reduction step from $\mathsf{F}(\mathsf{g}(x,x))\sigma$ to $\mathsf{F}(\mathsf{s}(x'))\sigma$ takes place 'in σ' and therefore it cannot be captured by narrowing. For linear terms, this effect could be simulated by choosing another suitable σ', but in the above example this is not possible, because here two different occurrences of $x\sigma$ are reduced to different terms.

right-hand side reduces to the instantiated left-hand side of the next pair in the chain. Let σ be such a substitution where the length of the reduction

$$t\sigma \to_{\mathcal{R}}^* v_2\sigma$$

is *minimal.* Note that the length of this reduction cannot be zero, as t and v_2 do not unify. Hence, we have $t\sigma \to_{\mathcal{R}} q \to_{\mathcal{R}}^* v_2\sigma$ for some term q.

There are two possibilities for the reduction $t\sigma \to_{\mathcal{R}} q$. Let us first assume that this reduction takes place 'in σ'. Hence, there is a variable x in t (i.e. $t|_p = x$ for some position p) such that $\sigma(x) \to_{\mathcal{R}} r$ and $q = t[r]_p$. The variable x only occurs *once* in t (as t is linear) and therefore, we have $q = t\sigma'$, where σ' is the substitution with $\sigma'(x) = r$ and $\sigma'(y) = \sigma(y)$ for all $y \neq x$. As all (occurrences of) dependency pairs are variable disjoint, σ' behaves like σ for all pairs except $\langle s, t\rangle$. For this pair, we have

$$w_1\sigma' = w_1\sigma \to_{\mathcal{R}}^* s\sigma \to_{\mathcal{R}}^* s\sigma' \quad \text{and}$$

$$t\sigma' = q \to_{\mathcal{R}}^* v_2\sigma = v_2\sigma'.$$

Hence, σ' is also a substitution where each instantiated right-hand side reduces to the instantiation of the left-hand side of the following pair in the chain. But as the reduction from $t\sigma'$ to $v_2\sigma'$ is shorter than the reduction from $t\sigma$ to $v_2\sigma$, this is a contradiction to the minimality of σ.

So the reduction $t\sigma \to_{\mathcal{R}} q$ cannot take place 'in σ'. Hence, t contains some subterm $f(\vec{u})$ such that a rule $l \to r$ has been applied to $f(\vec{u})\sigma$. In other words, l matches $f(\vec{u})\sigma$ (i.e. $l\rho = f(\vec{u})\sigma$). Hence, the reduction has the following form:

$$t\sigma = t\sigma[f(\vec{u})\sigma]_p = t\sigma[l\rho]_p \to_{\mathcal{R}} t\sigma[r\rho]_p = q.$$

Similar to Def. 21 we assume that the variables of $l \to r$ have been renamed to fresh ones. Therefore we can extend σ to 'behave' like ρ on the variables of l and r (but it still remains the same on the variables of all pairs in the chain). Now σ is a unifier of l and $f(\vec{u})$ and hence, there also exists a most general unifier μ. By the definition of most general unifiers, then there must be a substitution τ such that $\sigma = \mu\tau$.

Let t' be the term $t\mu[r\mu]_p$ and let s' be $s\mu$. Then $\langle s, t\rangle$ narrows to $\langle s', t'\rangle$. As we may assume s' and t' to be variable disjoint from all other pairs, we may extend σ to behave like τ on the variables of s' and t'. Then we have

$$w_1\sigma \to_{\mathcal{R}}^* s\sigma = s\mu\tau = s'\tau = s'\sigma \quad \text{and}$$

$$t'\sigma = t'\tau = t\mu\tau[r\mu\tau]_p = t\sigma[r\sigma]_p = t\sigma[r\rho]_p = q \to_{\mathcal{R}}^* v_2\sigma.$$

Hence, $\ldots \langle v_1, w_1\rangle\ \langle s', t'\rangle\ \langle v_2, w_2\rangle \ldots$ is also an $\mathcal{R}$-chain.

For the other direction of the theorem, let $\ldots \langle v_1, w_1\rangle\ \langle s', t'\rangle\ \langle v_2, w_2\rangle \ldots$ be an $\mathcal{R}$-chain. Hence, there is a substitution σ such that for all pairs the instantiated right-hand side reduces to the instantiated left-hand side of the next pair in the chain. So in particular we have

$$w_1\sigma \to_{\mathcal{R}}^* s'\sigma \quad \text{and} \quad t'\sigma \to_{\mathcal{R}}^* v_2\sigma.$$

We know that $\langle s, t\rangle$ narrows to $\langle s', t'\rangle$ via some substitution μ. As the variables in $\langle s, t\rangle$ are disjoint from all other occurring variables, we may extend σ to 'behave' like $\mu\sigma$ on the variables of s and t. Then we have $s\sigma = s\mu\sigma = s'\sigma$ and hence,

$$w_1\sigma \to_{\mathcal{R}}^* s\sigma.$$

Moreover, by the definition of narrowing, $t\mu \to_{\mathcal{R}} t'$. This implies $t\mu\sigma \to_{\mathcal{R}} t'\sigma$ and as $t\sigma = t\mu\sigma$, we have

$$t\sigma \to_{\mathcal{R}}^* v_2\sigma.$$

Hence, $\ldots \langle v_1, w_1\rangle \; \langle s, t\rangle \; \langle v_2, w_2\rangle \ldots$ is also an $\mathcal{R}$-chain. □

Note that while *dependency pairs* may indeed be replaced by their narrowings, in general a similar replacement of *rules* by their narrowings is unsound, i.e. it does not preserve the termination behaviour. For example, in the TRS with the rules $\mathsf{f}(1) \to \mathsf{f}(0)$ and $0 \to 1$, the right-hand side 1 of the second rule cannot be narrowed. However, deleting this second rule transforms the non-terminating TRS into a terminating one. So narrowing of dependency pairs is different from narrowing of rules, because even if some dependency pairs are eliminated, still *all* rules can be used for the reductions *between* two dependency pairs.

Example 26 *Narrowing pairs can be repeated several times if appropriate. So instead of replacing the dependency pair (8) by (9) and (10) we could also apply narrowing again and replace (9) and (10) by those pairs they narrow to. For example, the pair (9) has a linear right-hand side which does not unify with the left-hand side of any pair. Thus it may be replaced by its narrowings*

$$\langle \mathsf{P}(x - \mathsf{s}0, \mathsf{ss}y - \mathsf{ss}z), \mathsf{P}(y - z, x - \mathsf{s}0)\rangle$$
$$\langle \mathsf{P}(\mathsf{s}x - \mathsf{s}0, \mathsf{s}y - \mathsf{ss}z), \mathsf{P}(y - \mathsf{s}z, x - 0)\rangle.$$

In general, before application of Thm. 17 one can apply an *arbitrary* number of narrowing steps to the dependency pairs. Subsequently, the resulting set of pairs is considered to be the 'set of dependency pairs' and the techniques presented to approximate the dependency graph and to synthesize the inequalities are applied. Finally, standard techniques are used to find an ordering satisfying the generated inequalities.

By the use of narrowing the automation of our method can be improved significantly. For instance, if in our example we perform at least one narrowing step, then termination can again be verified automatically.

Note that if an ordering can be found that satisfies the set of inequalities obtained without narrowing any of the pairs, then the inequalities obtained after narrowing are also satisfied by the same ordering. (If the ordering satisfies $s > t$ and $l \geq r$, then it also satisfies $s\mu > t\mu \geq t'$, provided that $t \rightsquigarrow_{\mathcal{R}} t'$ via the substitution μ. Hence, $s > t$ resp. $s \geq t$ implies $s' > t'$ resp. $s' \geq t'$ for any narrowing $\langle s', t' \rangle$ of $\langle s, t \rangle$. Moreover, if $\langle s', t' \rangle$ and $\langle v', w' \rangle$ are narrowings of $\langle s, t \rangle$ and $\langle v, w \rangle$ respectively, then there can only be an arc from $\langle s', t' \rangle$ to $\langle v', w' \rangle$ in the new dependency graph if there already was an arc from $\langle s, t \rangle$ to $\langle v, w \rangle$ in the original dependency graph. The corresponding statement also holds for our approximation of dependency graphs, i.e. if t' is connectable to v', then t is also connectable to v.) Thus, replacing pairs by their narrowings can only extend the set of TRSs for which termination can be proved automatically.

2.6 Summary

Combining all refinements, we obtain the following technique to prove termination automatically using the dependency pair approach:

- Determine the dependency pairs (this can be automated in a straightforward way).
- Replace some (dependency) pairs by all their narrowings. This step may be repeated several times.
- Approximate the dependency graph by estimating for all (dependency) pairs whether an arc exists between them. For this purpose, the functions CAP and REN are introduced. The pairs that occur on a cycle in the approximated dependency graph are computed by standard graph algorithms. Pairs which are not on a cycle in the approximated dependency graph can be ignored.
- Transform the rules and the (dependency) pairs on cycles into inequalities.
- Find a well-founded weakly monotonic quasi-ordering satisfying the inequalities after normalizing them with respect to one of the possible AFSs.

For finding suitable orderings standard techniques like the recursive path ordering or polynomial interpretations may be used. In this way, standard techniques can now be applied to prove termination of TRSs whose termination could not be proved automatically before. For a collection of examples to demonstrate the power of our approach see Sect. 5.

3 Proving innermost termination

Similar to our approach for termination we now introduce a method to prove *innermost termination* of TRSs. Several ideas and notions can be transferred from the termination case to the innermost termination case. Therefore many theorems in this section look similar to the theorems in the previous section and in their proofs we only indicate the differences to the previous approach.

In Sect. 3.1 we present a criterion for innermost termination corresponding to the termination criterion of Sect. 2. We show in Sect. 3.2 that this criterion is also suitable for automation and that similar refinements for improving the technique can be developed (Sect. 3.3 and Sect. 3.4). The automated checking of this criterion enables us to prove innermost termination automatically, even if the TRS is not terminating. Additionally, for several classes of TRSs innermost termination already suffices for termination [35, 36]. Moreover, numerous modularity results exist for innermost termination [4, 5, 6, 35, 44, 45], which do not hold for termination. Therefore, for those classes of TRSs *termination* can be proved by splitting the TRS and proving *innermost termination* of the subsystems separately. The advantage of this approach is that there are several interesting TRSs where a direct termination proof is not possible with the existing automatic techniques (including the technique of Sect. 2). However in many of these examples, a suitable ordering satisfying the constraints generated by our technique for proving *innermost* termination can nevertheless be synthesized automatically. So for many TRSs proving innermost termination automatically is essentially easier than proving termination. In this way, innermost termination (and thereby, termination) of many also non-simply terminating systems can now be verified automatically. An overview of the technique is given in Sect. 3.5.

3.1 Innermost termination criterion

In contrast to the approach in the previous section, now our aim is to prove that the length of every *innermost* reduction is finite (where innermost redeces have to be contracted first). Of course, termination implies innermost termination, but in general the converse does not hold.

Example 27 *As an example consider the following TRS with the defined symbols* f *and* g *and the constructors* $\mathsf{0}$ *and* s.

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

In this example, we have the following infinite (cycling) reduction.

$$\mathsf{f}(\mathsf{gs0},\mathsf{s0},\mathsf{gs0}) \to \mathsf{f}(\mathsf{gs0},\mathsf{gs0},\mathsf{gs0}) \to \mathsf{f}(\mathsf{gs0},\mathsf{sg0},\mathsf{gs0}) \to \mathsf{f}(\mathsf{gs0},\mathsf{s0},\mathsf{gs0}) \to \ldots$$

However, this reduction is not an innermost reduction, because in the first reduction step the subterm $\mathsf{gs0}$ *is a redex and would have to be reduced first. Although this TRS is not terminating, it nevertheless turns out to be innermost terminating.*

The aim of this section is to develop a criterion for innermost termination similar to our termination criterion of Sect. 2. In the above example we obtain the following dependency pairs.

$$\langle \mathsf{F}(\mathsf{g}(x), \mathsf{s}(0), y), \mathsf{G}(x)\rangle \tag{11}$$
$$\langle \mathsf{F}(\mathsf{g}(x), \mathsf{s}(0), y), \mathsf{F}(y, y, \mathsf{g}(x))\rangle \tag{12}$$
$$\langle \mathsf{G}(\mathsf{s}(x)), \mathsf{G}(x)\rangle \tag{13}$$

Recall that a sequence of dependency pairs $\langle s_1, t_1\rangle\ \langle s_2, t_2\rangle \ldots$ is a *chain*, if there exists a substitution σ such that $t_j\sigma$ reduces to $s_{j+1}\sigma$ for all j. Here, the right-hand side of each dependency pair corresponds to a newly introduced redex and the reductions $t_j\sigma \to_{\mathcal{R}}^* s_{j+1}\sigma$ are used to contract the arguments of the redex that is traced. However, *chains* correspond to *arbitrary* reductions, whereas now we are only interested in *innermost* reductions. Therefore, we have to restrict the definition of chains in order to obtain a notion which corresponds to the innermost reduction strategy.

The first restriction is motivated by the fact that when regarding innermost reductions, arguments of a redex should be in normal form before the redex is contracted. Therefore we demand that all $s_j\sigma$ should be normal forms. Additionally, when concentrating on innermost reductions, the reductions of the arguments to normal form should also be innermost reductions. This results in the following restricted notion of a chain (where innermost reductions are denoted by '$\stackrel{\mathsf{i}}{\to}$').

Definition 28 (Innermost chain) *Let $\mathcal{R}(D, C, R)$ be a TRS. A sequence of dependency pairs $\langle s_1, t_1\rangle\ \langle s_2, t_2\rangle \ldots$ is an* innermost $\mathcal{R}$-chain *if there exists a substitution σ such that all $s_j\sigma$ are normal forms and such that $t_j\sigma \stackrel{\mathsf{i}}{\to}_{\mathcal{R}}^* s_{j+1}\sigma$ holds for every two consecutive pairs $\langle s_j, t_j\rangle$ and $\langle s_{j+1}, t_{j+1}\rangle$ in the sequence.*

In our example we have the innermost chain

$$\langle \mathsf{G}(\mathsf{s}(x_1)), \mathsf{G}(x_1)\rangle\ \langle \mathsf{G}(\mathsf{s}(x_2)), \mathsf{G}(x_2)\rangle\ \langle \mathsf{G}(\mathsf{s}(x_3)), \mathsf{G}(x_3)\rangle$$

because $\mathsf{G}(x_1)\sigma \stackrel{\mathsf{i}}{\to}_{\mathcal{R}}^* \mathsf{G}(\mathsf{s}(x_2))\sigma$ and $\mathsf{G}(x_2)\sigma \stackrel{\mathsf{i}}{\to}_{\mathcal{R}}^* \mathsf{G}(\mathsf{s}(x_3))\sigma$ holds for the substitution σ that replaces x_1 by $\mathsf{s}(\mathsf{s}(x_3))$ and x_2 by $\mathsf{s}(x_3)$.

Of course, every innermost chain is also a chain, but not vice versa. For instance, the infinite sequence consisting of the second dependency pair (12) only is an infinite chain, because

$$\mathsf{F}(y_1, y_1, \mathsf{g}(x_1))\sigma \to_{\mathcal{R}}^* \mathsf{F}(\mathsf{g}(x_2), \mathsf{s}(0), y_2)\sigma \tag{14}$$

holds if $\sigma(x_j) = \mathsf{s}(0)$ and $\sigma(y_j) = \mathsf{g}(\mathsf{s}(0))$. However, this infinite chain is not an innermost chain, because for every substitution σ satisfying (14), the term $\mathsf{F}(\mathsf{g}(x_2), \mathsf{s}(0), y_2)\sigma$ is not a normal form. The following theorem proves that the absence of infinite innermost chains is a sufficient and necessary criterion for innermost termination. (Hence, the restriction of *chains* to *innermost chains* in fact corresponds to the restriction of *reductions* to *innermost reductions*.)

Theorem 29 (Innermost termination criterion) *A TRS $\mathcal{R}(D, C, R)$ is innermost terminating if and only if no infinite innermost $\mathcal{R}$-chain exists.*

Proof. The proof of this theorem is very similar to the proof of Thm. 6. In the same way as in the latter proof, an infinite sequence of dependency pairs can be constructed, whenever an infinite innermost reduction exists. The difference, however, is that now the arguments of the terms are innermost reduced to normal form before building the next dependency pair, whereas in the proof of Thm. 6 the arguments were reduced an arbitrary number of steps. The sequence constructed in this way is in fact an innermost chain.

For the other direction, similar to the corresponding proof of Thm. 6 one can show that any infinite innermost chain corresponds to an infinite innermost reduction. $\square$

3.2 Checking the innermost termination criterion automatically

In this section we present an automatic approach for innermost termination proofs using the criterion of Thm. 29, i.e. we develop a method to prove the absence of infinite innermost chains automatically.

Assume that there is a sequence $\langle s_1, t_1\rangle\langle s_2, t_2\rangle\langle s_3, t_3\rangle \ldots$ of dependency pairs and a substitution σ such that all terms $s_j\sigma$ are in normal form and such that $t_j\sigma$ reduces innermost to $s_{j+1}\sigma$ for all j. Then to prove that this sequence is finite, it suffices again to find a well-founded quasi-ordering $\geq$ such that the following inequalities are satisfied.

$$s_1\sigma \;>\; t_1\sigma \;\geq\; s_2\sigma \;>\; t_2\sigma \;\geq\; s_3\sigma \;>\; t_3\sigma \;\geq\; \ldots \tag{4}$$

To ensure that all dependency pairs are decreasing, we again demand $s > t$ for all dependency pairs $\langle s, t\rangle$. In our example this results in the following constraints, cf. (11), (12), (13):

$$\mathsf{F}(\mathsf{g}(x), \mathsf{s}(\mathsf{0}), y) \;>\; \mathsf{G}(x) \tag{15}$$

$$\mathsf{F}(\mathsf{g}(x), \mathsf{s}(\mathsf{0}), y) \;>\; \mathsf{F}(y, y, \mathsf{g}(x)) \tag{16}$$

$$\mathsf{G}(\mathsf{s}(x)) \;>\; \mathsf{G}(x). \tag{17}$$

Moreover, we have to ensure $t_j\sigma \geq s_{j+1}\sigma$ whenever $t_j\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} s_{j+1}\sigma$ holds. Recall that to prove termination we demanded that *all* rules were weakly decreasing. This was necessary, because in chains, σ may be an arbitrary substitution and hence, every rule can be used in the reduction from $t_j\sigma$ to $s_{j+1}\sigma$[6]. However, in contrast to the situation for chains, in an innermost chain only a subset of the rewrite rules of the TRS can be applied in the reduction in between the dependency pairs. Therefore, to prove innermost termination we only demand the constraints $l \geq r$ for those rules $l \to r$ that can be used in an innermost reduction of $t_j\sigma$. Note that as all terms $s_j\sigma$ are normal, σ is a *normal substitution* (i.e. it instantiates all variables with normal forms). Hence, for the dependency pairs (11) and (13) we directly obtain that *no* rule can ever be used to reduce a normal instantiation of $\mathsf{G}(x)$ (because G is no defined symbol). The only dependency pair whose right-hand side can be reduced if its variables are instantiated with normal forms is (12), because this is a dependency pair with *defined symbols* in the right-hand side. As the only defined symbol in $\mathsf{F}(y, y, \mathsf{g}(x))$ is g, the only rules that may be applied on normal instantiations of this term are the two g-rules of the TRS. Since these g-rules can never introduce a new redex with root symbol f, the two g-rules are the only rules that can be used to reduce any normal instantiation of $\mathsf{F}(y, y, \mathsf{g}(x))$. Hence, in this example we only have to demand that these rules should be weakly decreasing.

$$\mathsf{g}(\mathsf{s}(x)) \geq \mathsf{s}(\mathsf{g}(x)), \quad \mathsf{g}(\mathsf{0}) \geq \mathsf{0} \tag{18}$$

In general, to determine the *usable rules*, i.e. (a superset of) those rules that may possibly be used in an innermost reduction of a normal instantiation of a term t, we proceed as follows. If t contains a defined symbol f, then all f-rules are *usable* and furthermore, all rules that are *usable* for right-hand sides of f-rules are also *usable* for t. However, if one of these rules contains a redex as a proper subterm of the left-hand side, then we do not have to include it in the usable rules, since this rule can never be applied in any innermost reduction.

Definition 30 (Usable rules) *Let $\mathcal{R}(D, C, R)$ be a TRS. For any symbol f let $Rules(R, f) = \{l \to r \text{ in } R \mid root(l) = f,\ l \text{ has no redex as proper subterm}\}$. For any term t, the set of usable rules $\mathbf{U}(R, t)$ is inductively defined as*

$$\begin{aligned}
\mathbf{U}(R, x) &= \emptyset \\
\mathbf{U}(R, f(t_1, \ldots, t_n)) &= Rules(R, f) \ \cup\ \textstyle\bigcup_{j=1}^{n} \mathbf{U}(R', t_j) \cup \\
&\quad\ \textstyle\bigcup_{l \to r \in Rules(R,f)} \mathbf{U}(R', r),
\end{aligned}$$

where[7] $R' = R \setminus Rules(R, f)$.

Hence, in our example we have

[6] Provided that a variable occurs in t_j, but termination is decidable for TRSs with ground right-hand sides [16].

[7] $\mathbf{U}(R, t)$ is well-defined, because its first argument R is decreasing.

$$
\begin{aligned}
\mathbf{U}(R, \mathsf{F}(y, y, \mathsf{g}(x))) &= \mathit{Rules}(R, \mathsf{F}) \cup \mathbf{U}(R, y) \cup \mathbf{U}(R, \mathsf{g}(x)) \cup \emptyset \\
&= \mathbf{U}(R, \mathsf{g}(x)) \\
&= \mathit{Rules}(R, \mathsf{g}) \cup \\
&\quad\ \mathbf{U}(\{\mathsf{f}(\ldots) \to \mathsf{f}(\ldots)\}, x) \cup \\
&\quad\ \mathbf{U}(\{\mathsf{f}(\ldots) \to \mathsf{f}(\ldots)\}, \mathsf{s}(\mathsf{g}(x))) \cup \\
&\quad\ \mathbf{U}(\{\mathsf{f}(\ldots) \to \mathsf{f}(\ldots)\}, \mathsf{0}) \\
&= \{\mathsf{g}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{g}(x)), \mathsf{g}(\mathsf{0}) \to \mathsf{0}\}.
\end{aligned}
$$

Observe that by the above definition $\mathit{Rules}(R, f) = \emptyset$ for any constructor f.

When proving termination we had to search for a *weakly monotonic* quasi-ordering satisfying the constraints obtained. The reason for demanding weak monotonicity was that $l \geq r$ for all rules had to ensure $t_j\sigma \geq s_{j+1}\sigma$ whenever $t_j\sigma$ could be reduced to $s_{j+1}\sigma$. However, now for the tuple symbols we do not need weak monotonicity on all positions any more. For example, for the tuple symbol F we only have to ensure that all reductions starting from $\mathsf{F}(y, y, \mathsf{g}(x))\sigma$ are weakly decreasing (where σ is a normal substitution). Obviously, such reductions can never take place in the first two arguments of F and hence, F does not have to be weakly monotonic in these arguments.

The constraints (18) already ensure that during reductions of $\mathsf{F}(y, y, \mathsf{g}(x))\sigma$ the value of the *subterm* $\mathsf{g}(x)\sigma$ can only be decreased. Of course, we have to guarantee that the value of the *whole* term $\mathsf{F}(y, y, \mathsf{g}(x))$ is weakly decreasing if an instantiation of $\mathsf{g}(x)$ is replaced by a smaller term. For that purpose, we demand that $\mathsf{F}(y, y, \mathsf{g}(x))$ must be weakly monotonic on the position of its subterm $\mathsf{g}(x)$, i.e. for the tuple symbol F we only have to demand the following monotonicity constraint:

$$
x_1 \geq x_2 \;\Rightarrow\; \mathsf{F}(y, y, x_1) \geq \mathsf{F}(y, y, x_2). \tag{19}
$$

We only compute such monotonicity constraints for the tuple symbols and for all other (lower case) symbols we demand weak monotonicity in all of their arguments. In general, we obtain the following procedure for the generation of constraints.

Theorem 31 (Proving innermost termination) *Let $\mathcal{R}(D, C, R)$ be a TRS and let $\geq$ be a well-founded quasi-ordering where both $\geq$ and $>$ are closed under substitution. If $\geq$ is weakly monotonic on all symbols apart from the tuple symbols and if $\geq$ satisfies the following constraints for all dependency pairs $\langle s, t \rangle$*

- *$l \geq r$ for all usable rules $l \to r$ in $\mathbf{U}(R, t)$,*
- *$s > t$,*

- $x_1 \geq y_1 \wedge \ldots \wedge x_n \geq y_n \Rightarrow C[x_1, \ldots, x_n] \geq C[y_1, \ldots, y_n]$,
  if $t = C[f_1(\vec{u}_1), \ldots, f_n(\vec{u}_n)]$, *where* C *is a context without defined symbols and* $f_1, \ldots, f_n$ *are defined symbols,*

then $\mathcal{R}$ *is innermost terminating.*

Proof. The proof of this theorem corresponds to the proof of Thm. 7. Suppose that $\langle s_1, t_1 \rangle \langle s_2, t_2 \rangle \ldots$ is an infinite innermost $\mathcal{R}$-chain. Then there exists a substitution σ such that $s_j\sigma$ is in normal form and $t_j\sigma$ reduces innermost to $s_{j+1}\sigma$ for all j. Hence, σ replaces all variables by normal forms and therefore, the only rules that can be applied in this reduction are the usable rules $\mathbf{U}(R, t_j)$. All usable rules are weakly decreasing and the terms t_j are weakly monotonic on all positions where reductions are applied. (The reason is that lower case symbols are weakly monotonic and without loss of generality we can assume that σ does not introduce any tuple symbols, i.e. the only tuple symbol in $t_j\sigma$ is on the root position.) Hence, we have $t_j\sigma \geq s_{j+1}\sigma$. This results in an infinite decreasing sequence $s_1\sigma > t_1\sigma \geq s_2\sigma > t_2\sigma \geq \ldots$ which is a contradiction to the well-foundedness of $\geq$. Thus, no infinite innermost $\mathcal{R}$-chain exists and by Thm. 29, the TRS is innermost terminating. $\square$

So there are two main differences between the termination approach and the approach for innermost termination. The first difference is in the set of inequalities that is generated. As we restrict ourselves to innermost reductions and to terms $s_j\sigma$ that are normal forms, several inequalities that have to be demanded when proving termination are unnecessary when proving innermost termination (i.e. we do not have to demand $l \geq r$ for all rules any more, but it suffices if just the usable rules are weakly decreasing). After generating the inequalities, the second difference is that the quasi-ordering satisfying the inequalities does not have to be weakly monotonic for all function symbols (i.e. tuple symbols only have to satisfy the monotonicity constraints that are stated explicitly).

Hence, in Ex. 27 to prove innermost termination it is sufficient to find a well-founded quasi-ordering satisfying the constraints in (15) – (19). For the synthesis of suitable quasi-orderings we proceed in the same way as it has been done for termination (Sect. 2.3) where for polynomial interpretations the difference is that the polynomials do not have to be weakly monotonic in all arguments.

For example, these constraints are fulfilled by the polynomial ordering where the constant $\mathsf{0}$ is mapped to the number 0, $\mathsf{s}(x)$ is mapped to $x + 1$, $\mathsf{g}(x)$ is mapped to $x + 2$, $\mathsf{F}(x, y, z)$ is mapped to $(x - y)^2 + 1$, and $\mathsf{G}(x)$ is mapped to x. Note that this quasi-ordering is not weakly monotonic on the tuple symbol F. The only monotonicity constraint in our example is (19), which is obviously satisfied as $\mathsf{F}(x, y, z)$ is mapped to a polynomial which is weakly monotonic[8] in

[8] When using polynomial interpretations, monotonicity constraints like (19) can also be represented as inequalities. For instance, if F is mapped to some polynomial $[\mathsf{F}]$, then instead of (19) one could demand that the *partial derivative* of $[\mathsf{F}](y, y, x)$ with respect to x should

its third argument z. However, this polynomial is not weakly monotonic in x or y.

Unlike Thm. 7 for termination proofs, the existence of a quasi-ordering satisfying the conditions of Thm. 31 is sufficient, but not necessary for innermost termination. The reason is that demanding the constraints of Thm. 31 for *all* instantiations may be too strong, since for *innermost* chains sometimes it would be sufficient to regard *certain* instantiations only.

Example 32 *For example, consider the innermost terminating TRS*

$$\begin{array}{rcl} \mathsf{f}(\mathsf{s}(x)) & \to & \mathsf{f}(\mathsf{g}(\mathsf{h}(x))) \\ \mathsf{g}(\mathsf{h}(x)) & \to & \mathsf{g}(x) \\ \mathsf{g}(\mathsf{s}(x)) & \to & \mathsf{s}(x) \\ \mathsf{g}(\mathsf{0}) & \to & \mathsf{s}(\mathsf{0}) \\ \mathsf{h}(\mathsf{0}) & \to & \mathsf{a}. \end{array}$$

In this example there are no infinite innermost chains. However, the constraints according to Thm. 31 include the inequalities

$$\begin{array}{rcl} \mathsf{F}(\mathsf{s}(x)) & > & \mathsf{F}(\mathsf{g}(\mathsf{h}(x))) \\ \mathsf{g}(\mathsf{h}(x)) & \geq & \mathsf{g}(x) \\ \mathsf{g}(\mathsf{0}) & \geq & \mathsf{s}(\mathsf{0}) \\ x_1 \geq x_2 \Rightarrow \mathsf{F}(x_1) & \geq & \mathsf{F}(x_2). \end{array}$$

These constraints imply $\mathsf{F}(\mathsf{s}(\mathsf{0})) > \mathsf{F}(\mathsf{g}(\mathsf{h}(\mathsf{0}))) \geq \mathsf{F}(\mathsf{g}(\mathsf{0})) \geq \mathsf{F}(\mathsf{s}(\mathsf{0}))$. *Therefore they cannot be satisfied by any well-founded quasi-ordering closed under substitution.*

However, the approach of Thm. 31 suffices to prove *innermost termination* of numerous important examples and challenge problems (including the TRS in Ex. 27) automatically, i.e. this technique allows the application of standard techniques for innermost termination proofs, even if the TRS is not terminating. Moreover, using the results of Gramlich [35, 36], Thm. 31 can also be applied to prove *termination* of TRSs that are non-overlapping (or for locally confluent overlay systems).

Example 33 *As an example regard the following TRS by Kolbe [43] where* $\mathsf{quot}(x, y, z)$ *is used to compute* $1 + \left\lfloor \frac{x-y}{z} \right\rfloor$, *if* $x \geq y$ *and* $z \neq 0$ *(i.e.* $\mathsf{quot}(x, y, y)$ *computes* $\left\lfloor \frac{x}{y} \right\rfloor$*).*

---

be non-negative, i.e. $\frac{\partial [\mathsf{F}](y,y,x)}{\partial x} \geq 0$, cf. [31].
If one uses other techniques (e.g. path orderings) which can only generate monotonic orderings, then of course one may drop monotonicity constraints like (19).

$$\begin{aligned}
\mathsf{quot}(\mathsf{0},\mathsf{s}(y),\mathsf{s}(z)) &\to \mathsf{0}\\
\mathsf{quot}(\mathsf{s}(x),\mathsf{s}(y),z) &\to \mathsf{quot}(x,y,z)\\
\mathsf{quot}(x,\mathsf{0},\mathsf{s}(z)) &\to \mathsf{s}(\mathsf{quot}(x,\mathsf{s}(z),\mathsf{s}(z)))
\end{aligned}$$

The above system is not simply terminating (the left-hand side of the last rule is embedded in the right-hand side if z is instantiated with $\mathsf{0}$*) and therefore most automatic approaches for termination proofs (which are restricted to simplification orderings) fail.*

Nevertheless, with our technique we can prove innermost termination and therefore termination of this system automatically. As quot *is the only defined symbol of this system, we obtain the following dependency pairs.*

$$\langle \mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y),z),\mathsf{Q}(x,y,z)\rangle \tag{20}$$

$$\langle \mathsf{Q}(x,\mathsf{0},\mathsf{s}(z)),\mathsf{Q}(x,\mathsf{s}(z),\mathsf{s}(z))\rangle \tag{21}$$

In this example there are no usable rules, as in the right-hand sides of the dependency pairs no defined symbols occur. Hence, due to Thm. 31 we only have to find a well-founded quasi-ordering such that both dependency pairs are decreasing. These constraints are for instance satisfied by the polynomial ordering where $\mathsf{0}$ *is mapped to the number 0,* $\mathsf{s}(x)$ *is mapped to* $x+1$*, and* $\mathsf{Q}(x,y,z)$ *is mapped to* $x+(x-y+z)^2$*. Hence, innermost termination and thereby also termination of this TRS is proved (as it is non-overlapping).*

Note that again we benefit from the fact that the tuple symbol Q *need not be weakly monotonic in its arguments. Therefore, an interpretation like the polynomial* $x+(x-y+z)^2$ *may be used, which is not weakly monotonic in any of its arguments. In fact, if the set of usable rules is empty, then the quasi-ordering need not even be weakly monotonic for any symbol. The termination approach of Sect. 2 cannot be used to prove termination of this TRS automatically, since the generated inequalities are not satisfied by any well-founded weakly monotonic total quasi-ordering or any quasi-simplification ordering (not even after normalisation by a suitable AFS).*

3.3 Refinement using innermost dependency graphs

To prove innermost termination of a TRS according to Thm. 31 we have to find an ordering such that $s > t$ holds for *all* dependency pairs $\langle s,t\rangle$. However, similar to the refinement for termination proofs in Sect. 2.4, for certain rewrite systems this requirement can be weakened, i.e. it is sufficient to demand $s > t$ for *some* dependency pairs only.

For instance, in the quot example (Ex. 33) up to now we demanded that both dependency pairs (20) and (21) had to be decreasing. However, two occurrences of the dependency pair (21) can never follow each other in an innermost chain, because $\mathsf{Q}(x_1,\mathsf{s}(z_1),\mathsf{s}(z_1))\sigma$ can never reduce to any instantiation of

$\mathsf{Q}(x_2, \mathsf{0}, \mathsf{s}(z_2))$. The reason is that the second arguments $\mathsf{s}(z_1)$ resp. $\mathsf{0}$ of these two terms have different constructor root symbols. Hence, any possible infinite innermost chain would contain infinitely many occurrences of the other dependency pair (20). Therefore it is sufficient if (20) is decreasing and if (21) is just weakly decreasing. In this way, we obtain the following (weakened) constraints.

$$\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y), z) \quad > \quad \mathsf{Q}(x, y, z) \tag{22}$$

$$\mathsf{Q}(x, \mathsf{0}, \mathsf{s}(z)) \quad \geq \quad \mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z)) \tag{23}$$

In general, to determine those dependency pairs which may possibly follow each other in innermost chains, we define the following graph.

Definition 34 (Innermost dependency graph) *The* innermost dependency graph *of a TRS $\mathcal{R}$ is the directed graph whose nodes are the dependency pairs and there is an arc from $\langle s, t\rangle$ to $\langle v, w\rangle$ if $\langle s, t\rangle\langle v, w\rangle$ is an innermost $\mathcal{R}$-chain.*

For instance, in the innermost dependency graph for the quot example there are arcs from (20) to itself and to (21), and there is an arc from (21) to (20) (but *not* to itself).

$$\begin{tikzcd}
\langle \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y), z), \mathsf{Q}(x, y, z)\rangle \arrow[loop above] \arrow[d, shift left] \\
\langle \mathsf{Q}(x, \mathsf{0}, \mathsf{s}(z)), \mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z))\rangle \arrow[u, shift left]
\end{tikzcd}$$

Figure 2: The innermost dependency graph for the quot TRS (Ex. 33).

Of course, the fact that innermost chains are restricted chains cause innermost dependency graphs to be subgraphs of dependency graphs. Now any infinite innermost chain corresponds to a cycle in the innermost dependency graph. Hence, it is sufficient if $s > t$ holds for at least one dependency pair $\langle s, t\rangle$ on every cycle and if $s \geq t$ holds for the other dependency pairs on cycles. So, similar to Thm. 17 (for termination) we obtain the following refined theorem for automated innermost termination proofs.

Theorem 35 (Innermost dependency graph refinement) *Let $\mathcal{R}(D, C, R)$ be a TRS and let $\geq$ be a well-founded quasi-ordering where both $\geq$ and $>$ are closed under substitution. If $\geq$ is weakly monotonic on all symbols apart from the tuple symbols and if $\geq$ satisfies the following constraints for all dependency pairs $\langle s, t\rangle$ on a cycle of the innermost dependency graph*

- *$l \geq r$ for all usable rules $l \to r$ in $\mathbf{U}(R, t)$,*
- *$s \geq t$,*
- *$x_1 \geq y_1 \wedge \ldots \wedge x_n \geq y_n \Rightarrow C[x_1, \ldots, x_n] \geq C[y_1, \ldots, y_n]$,
  if $t = C[f_1(\vec{u}_1), \ldots, f_n(\vec{u}_n)]$, where C is a context without defined symbols and $f_1, \ldots, f_n$ are defined symbols,*

and if $s > t$ holds for at least one dependency pair $\langle s, t\rangle$ on each cycle of the innermost dependency graph, then $\mathcal{R}$ is innermost terminating.

Proof. The proof that Thm. 35 is a consequence of Thm. 31 is completely analogous to the proof that Thm. 17 is a consequence of Thm. 7. $\square$

Hence, in the quot example the constraints (22) and (23) are in fact sufficient for innermost termination. A suitable quasi-ordering satisfying these weakened constraints can easily be synthesized using the technique of Sect. 2.3. (For instance, one could use the polynomial interpretation where $\mathsf{0}$ and s are interpreted as usual and where $\mathsf{Q}(x, y, z)$ is mapped to x. If the constraints (22) and (23) are normalized w.r.t. an AFS which drops the second argument of Q, then they are also satisfied by the recursive path ordering.) This example demonstrates that the weakening of the constraints by using innermost dependency graphs often enables the application of much simpler orderings (e.g. now we can use the recursive path ordering or a linear weakly monotonic polynomial ordering whereas for the original constraints of Sect. 3.2 we needed a non-monotonic polynomial of degree 2).

However, for an automation of Thm. 35 we have to construct the innermost dependency graph. Again, this cannot be done automatically, since for two pairs $\langle s, t\rangle$ and $\langle v, w\rangle$ it is undecidable whether there exists a substitution σ such that $t\sigma$ reduces innermost to $v\sigma$ and such that $s\sigma$ and $v\sigma$ are normal forms. Hence, similar to the dependency graph, we can only approximate this graph by computing a supergraph containing the innermost dependency graph. Note that $t\sigma$ may only reduce to $v\sigma$ for some substitution σ, if either t has a defined root symbol or if both t and v have the same constructor root symbol. Recall that $\textsc{cap}(t)$ denotes the result of replacing all subterms in t with a defined root symbol by different fresh variables. Then $t\sigma$ can only reduce to $v\sigma$ if $\textsc{cap}(t)$ and v are unifiable.

However, this replacement of subterms of t must only be done for terms which are not equal to subterms of s. The reason is that such subterms are already in normal form when instantiated with σ. For example, if we modify the first rule of the TRS in Ex. 27 to $\mathsf{f}(\mathsf{g}(x), \mathsf{s}(\mathsf{0})) \to \mathsf{f}(\mathsf{g}(x), \mathsf{g}(x))$, then to determine whether there is an arc from the resulting dependency pair

$$\langle \mathsf{F}(\mathsf{g}(x), \mathsf{s}(\mathsf{0})),\ \mathsf{F}(\mathsf{g}(x), \mathsf{g}(x))\rangle$$

to itself, the subterms $\mathsf{g}(x)$ in the right-hand side do not have to be replaced by new variables. As both sides of this dependency pair do not unify after variable renaming, one can immediately see that this pair is not on a cycle of the innermost dependency graph (whereas $\text{CAP}(\mathsf{F}(\mathsf{g}(x), \mathsf{g}(x)) = \mathsf{F}(x_1, x_2)$ would unify with the left-hand side).

Let $\text{CAP}_s(t)$ only replace those subterms of t by different fresh variables which have a defined root symbol and which are not equal to subterms of s. Then to refine the approximation of innermost dependency graphs instead of $\text{CAP}(t)$ we check whether $\text{CAP}_s(t)$ unifies with v. Moreover, if μ is the most general unifier (mgu) of $\text{CAP}_s(t)$ and v, then there can only be an arc from $\langle s, t\rangle$ to $\langle v, w\rangle$ in the innermost dependency graph, if both $s\mu$ and $v\mu$ are in normal form.

So there are three differences between the approximation of the dependency graph and the approximation of the innermost dependency graph. First, for the innermost dependency graph we only replace subterms of t which do not occur in s, i.e. we use $\text{CAP}_s(t)$ instead of $\text{CAP}(t)$. Second, to approximate the dependency graph, multiple occurrences of the same variable in $\text{CAP}(t)$ are replaced by fresh variables (using the function REN), whereas the variables in $\text{CAP}_s(t)$ are left unchanged for the approximation of the innermost dependency graph. The reason is that any substitution used for instantiating the dependency pairs of an innermost chain is a normal substitution. Thus, variables are always instantiated by normal forms, and hence these instantiations are not reduced. Multiple occurrences of the same variable in a term result in multiple occurrences of the same subterm after reduction of the instantiated term. In contrast, for an arbitrary substitution, instantiated multiple occurrences of the same variable may result in different subterms after reduction of the instantiated term.

The third difference is that for innermost dependency graphs we only draw an arc from $\langle s, t\rangle$ to $\langle v, w\rangle$, if the mgu of $\text{CAP}_s(t)$ and v instantiates s and v to normal forms. This condition is due to the restriction to *innermost* chains.

Similar to the notion of *connectable terms* in Sect. 2.4, we call two dependency pairs *innermost connectable* if they should be connected by an arc in our approximation of the innermost dependency graph.

Definition 36 (Innermost connectable pairs) *For any dependency pairs $\langle s, t\rangle$ and $\langle v, w\rangle$, the pair $\langle s, t\rangle$ is* innermost connectable *to $\langle v, w\rangle$ if $\text{CAP}_s(t)$ and v are unifiable by some mgu μ such that $s\mu$ and $v\mu$ are in normal form.*

The following theorem proves the soundness of our approximation.

Theorem 37 (Computing innermost dependency graphs) *Let $\mathcal{R}$ be a TRS and let $\langle s, t\rangle$ and $\langle v, w\rangle$ be dependency pairs. If there is an arc from $\langle s, t\rangle$ to $\langle v, w\rangle$ in the innermost dependency graph, then $\langle s, t\rangle$ is innermost connectable to $\langle v, w\rangle$.*

Proof. Due to the additional conditions in the definition of innermost chains and the definition of *innermost* connectable pairs, the proof is slightly different from the proof of Thm. 19.

By induction on the structure of t we show that if there exists a substitution σ such that $s\sigma$ is a normal form and $t\sigma \to_{\mathcal{R}}^* u$ for some term u, then there exists a substitution τ (whose domain only includes variables that are introduced in the construction of $\textsc{cap}_s(t)$) with $\textsc{cap}_s(t)\sigma\tau = u$. Thus, in particular, if there exists a substitution σ such that $s\sigma$ and $v\sigma$ are normal forms and $t\sigma \to_{\mathcal{R}}^* v\sigma$, then $\textsc{cap}_s(t)\sigma\tau = v\sigma$ ($= v\sigma\tau$, since the variables of $v\sigma$ do not occur in the domain of τ). Hence, $\textsc{cap}_s(t)$ and v unify and the most general unifier μ is such that $s\mu$ and $v\mu$ are normal forms. (There exist instantiations of these two terms that are normal forms (viz. $s\sigma\tau = s\sigma$ and $v\sigma\tau = v\sigma$), hence the terms $s\mu$ and $v\mu$ are normal forms themselves.)

If t equals a subterm of s, then $t\sigma$ is in normal form, hence $t\sigma$ equals u. Moreover, we have $\textsc{cap}_s(t) = t$. So $\textsc{cap}_s(t)\sigma = u$, i.e. in this case τ is the empty substitution.

If t is not equal to a subterm of s and $root(t)$ is defined, then $\textsc{cap}_s(t)$ is a fresh variable. Let τ replace $\textsc{cap}_s(t)$ by u. Then we have $\textsc{cap}_s(t)\sigma\tau = \textsc{cap}_s(t)\tau = u$.

Otherwise, $t = c(t_1, \ldots, t_n)$ for some constructor c and we have

$$\textsc{cap}_s(t) = c(\textsc{cap}_s(t_1), \ldots, \textsc{cap}_s(t_n)).$$

In this case u is of the form $c(u_1, \ldots, u_n)$ and $t_j\sigma \to_{\mathcal{R}}^* u_j$ for all j. By the induction hypothesis there exist substitutions τ_j such that $\textsc{cap}_s(t_j)\sigma\tau_j = u_j$. Note that the variables newly introduced in $\textsc{cap}_s(t_j)$ are disjoint from those variables newly introduced in $\textsc{cap}_s(t_i)$ for $i \neq j$. Hence, if $\tau = \tau_1 \circ \ldots \circ \tau_n$, then for all j we have $\textsc{cap}_s(t_j)\sigma\tau = u_j$, and thus, $\textsc{cap}_s(t)\sigma\tau = c(u_1, \ldots, u_n)$. □

Using the approximation of Thm. 37, we can now compute the innermost dependency graph for the quot example in Fig. 2 automatically.

Example 38 *There are also examples where the innermost dependency graph does not contain any cycles.*

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

In this example, the dependency pair $\langle \mathsf{F}(x, \mathsf{g}(x)), \mathsf{F}(1, \mathsf{g}(x)) \rangle$ *is not on a cycle of the innermost dependency graph, although* $\textsc{cap}_{\mathsf{F}(x_1, \mathsf{g}(x_1))}(\mathsf{F}(1, \mathsf{g}(x_1))) = \mathsf{F}(1, \mathsf{g}(x_1))$ *unifies with* $\mathsf{F}(x_2, \mathsf{g}(x_2))$ *using a mgu that replaces* x_1 *and* x_2 *by* 1*. However, the instantiated left-hand side* $\mathsf{F}(1, \mathsf{g}(1))$ *is not a normal form, since it contains the redex* $\mathsf{g}(1)$*. The other dependency pairs* $\langle \mathsf{F}(x, \mathsf{g}(x)), \mathsf{G}(x) \rangle$ *and* $\langle \mathsf{G}(1), \mathsf{G}(0) \rangle$ *cannot occur on cycles either, since* $\mathsf{G}(\ldots)$ *does not unify with* $\mathsf{F}(\ldots)$ *and* $\mathsf{G}(0)$ *does not unify with* $\mathsf{G}(1)$*. Hence, using the refined techniques of Thm. 37 and 35 we obtain* no *constraint at all, i.e. innermost termination can be proved by only computing the (approximation of) the innermost dependency graph.*

3.4 Refined innermost termination proofs by narrowing dependency pairs

Similar to the termination technique of Sect. 2, the power of our technique can be increased if we consider narrowings of the dependency pairs.

Example 39 *For an illustration regard the following TRS.*

$$\begin{array}{rcl}
\mathsf{p}(\mathsf{0}) & \to & \mathsf{0} \\
\mathsf{p}(\mathsf{s}(x)) & \to & x \\
\mathsf{le}(\mathsf{0}, y) & \to & \mathsf{true} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{0}) & \to & \mathsf{false} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{le}(x, y) \\
\mathsf{minus}(x, y) & \to & \mathsf{if}(\mathsf{le}(x, y), x, y) \\
\mathsf{if}(\mathsf{true}, x, y) & \to & \mathsf{0} \\
\mathsf{if}(\mathsf{false}, x, y) & \to & \mathsf{s}(\mathsf{minus}(\mathsf{p}(x), y))
\end{array}$$

Here, a 'conditional' program for minus *has been encoded into an unconditional TRS. The dependency pairs on cycles of the innermost dependency graph are*

$$\langle \mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{LE}(x, y) \rangle \tag{24}$$

$$\langle \mathsf{M}(x, y), \mathsf{IF}(\mathsf{le}(x, y), x, y) \rangle \tag{25}$$

$$\langle \mathsf{IF}(\mathsf{false}, x, y), \mathsf{M}(\mathsf{p}(x), y) \rangle. \tag{26}$$

However, the constraints resulting from application of Thm. 35 would imply $\mathsf{M}(\mathsf{s}(x), \mathsf{0}) > \mathsf{M}(\mathsf{p}(\mathsf{s}(x)), \mathsf{0})$. *Therefore an automatic innermost termination proof using quasi-simplification orderings fails.*

The only dependency pair whose right-hand side does not unify with any left-hand side of a dependency pair is (25). Hence, in any innermost chain at least one rule of the TRS must be applied in order to reduce an instantiation of $\mathsf{IF}(\mathsf{le}(x, y), x, y)$ *to an instantiation of a left-hand side. So instead of examining the dependency pair (25) we may first perform all possible narrowing steps and replace (25) by*

$$\langle \mathsf{M}(\mathsf{0}, y), \mathsf{IF}(\mathsf{true}, \mathsf{0}, y) \rangle \tag{27}$$

$$\langle \mathsf{M}(\mathsf{s}(x), \mathsf{0}), \mathsf{IF}(\mathsf{false}, \mathsf{s}(x), \mathsf{0}) \rangle \tag{28}$$

$$\langle \mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{IF}(\mathsf{le}(x, y), \mathsf{s}(x), \mathsf{s}(y)) \rangle. \tag{29}$$

Note that while the right-hand side of (26) unified with the left-hand side of the original dependency pair (25), after this replacement the right-hand side of (26) does not unify with left-hand sides any more. Hence, the first narrowing of (25) now enables a subsequent narrowing of (26). So (26) is replaced by

$$\langle \mathsf{IF}(\mathsf{false}, \mathsf{0}, y), \mathsf{M}(\mathsf{0}, y) \rangle \tag{30}$$

$$\langle \mathsf{IF}(\mathsf{false}, \mathsf{s}(x), y), \mathsf{M}(x, y) \rangle. \tag{31}$$

In this way, the original set of dependency pairs (24) – (26) is transformed into (24) and (27) – (31). Note that the pairs (27) and (30) are not on cycles of the innermost dependency graph and can therefore be ignored in the innermost termination proof. In this way our method determined that instead of the original dependency pair (25) one only has to regard instantiations where x is instantiated with a term of the form $\mathsf{s}(\ldots)$*. But for those terms,* p *is decreasing and hence, the call of of* M *in the right-hand side of (31) is applied to smaller arguments than the call of* M *in the left-hand side of (28) or (29).*

Now innermost termination (and thereby termination) of the system can be proved by the technique of Thm. 35. This results in the following constraints.

$$\begin{array}{rcl}
\mathsf{le}(\mathsf{0},y) & \geq & \mathsf{true}\\
\mathsf{le}(\mathsf{s}(x),\mathsf{0}) & \geq & \mathsf{false}\\
\mathsf{le}(\mathsf{s}(x),\mathsf{s}(y)) & \geq & \mathsf{le}(x,y)\\
\mathsf{LE}(\mathsf{s}(x),\mathsf{s}(y)) & > & \mathsf{LE}(x,y)\\
\mathsf{M}(\mathsf{s}(x),\mathsf{0}) & \geq & \mathsf{IF}(\mathsf{false},\mathsf{s}(x),\mathsf{0})\\
\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) & \geq & \mathsf{IF}(\mathsf{le}(x,y),\mathsf{s}(x),\mathsf{s}(y))\\
\mathsf{IF}(\mathsf{false},\mathsf{s}(x),y) & > & \mathsf{M}(x,y)\\
x_1 \geq x_2 \;\Rightarrow\; \mathsf{IF}(x_1,\mathsf{s}(x),\mathsf{s}(y)) & \geq & \mathsf{IF}(x_2,\mathsf{s}(x),\mathsf{s}(y))
\end{array}$$

These constraints are satisfied by a polynomial interpretation where $\mathsf{0}$*,* true *and* false *are mapped to* 0*,* $\mathsf{s}(x)$ *is mapped to* $x+1$*,* $\mathsf{le}(x,y)$*,* $\mathsf{LE}(x,y)$*, and* $\mathsf{M}(x,y)$ *are mapped to* x*, and* $\mathsf{IF}(x,y,z)$ *is mapped to* y*. They are also satisfied by the recursive path ordering if an AFS is used to eliminate the first argument of* IF*.*

Narrowing pairs for the innermost termination technique has the side-effect that one may also drop some inequalities $l \geq r$ corresponding to the rules $l \to r$, since after narrowing the pairs some rules may not be usable any more. For example, for the original dependency pairs, the p-rules were usable, since (26) contains an occurrence of p in its right-hand side. But after narrowing this dependency pair, the occurrence of p is deleted and hence we do not have to demand that the p-rules are weakly decreasing.

So similar to the approach in Sect. 2.5 we may replace a dependency pair $\langle s,t\rangle$ by all its narrowings provided that the right-hand side t does not unify with any left-hand side of a dependency pair. In fact, due to the restriction to *innermost* chains we may even perform such a replacement if t unifies with the left-hand side v of a dependency pair, as long as their mgu does not instantiate both s and v to normal forms. Note that in contrast to the termination case, for innermost termination proofs we do not have to demand that t must be a linear term. Hence, we can indeed narrow the dependency pair (25) in the above example, although its right-hand side is not linear. However, this step would not have been possible with the method of Sect. 2. Therefore, for the TRS in Ex. 39 the constraints generated by the approach of Sect. 2 are not satisfied by any quasi-simplification ordering.

Theorem 40 (Narrowing refinement for innermost termination) *Let $\mathcal{R}$ be a TRS and let $\mathcal{P}$ be a set of pairs of terms. Let $\langle s,t\rangle \in \mathcal{P}$ such that all variables of t also occur in s and such that for all $\langle v,w\rangle \in \mathcal{P}$ where t and v are unifiable by some mgu μ (after renaming the variables), one of the terms $s\mu$ or $v\mu$ is no normal form. Let*

$$\mathcal{P}' = \mathcal{P} \setminus \{\langle s,t\rangle\} \cup \{\langle s',t'\rangle \mid \langle s',t'\rangle \textit{ is a narrowing of } \langle s,t\rangle\}.$$

If there exists no infinite innermost $\mathcal{R}$-chain of pairs from $\mathcal{P}'$, then there exists no infinite innermost $\mathcal{R}$-chain of pairs from $\mathcal{P}$ either.

Proof. The proof is analogous to the proof of Thm. 25. The only difference is that the right-hand side t of the dependency pair does not have to be linear any more. The reason is that in innermost chains we restrict ourselves to normal substitutions σ and therefore, reductions of $t\sigma$ can never take place 'in σ'. □

Note that unlike Thm. 25 for termination, the replacement of dependency pairs by their narrowings can destroy the necessity of our innermost termination criterion. The reason is that narrowing does not respect the *innermost* reduction strategy.

Example 41 *The TRS in Ex. 32 was innermost terminating. Hence, there does not exist an infinite innermost chain of dependency pairs. However, if we replace the dependency pair $\langle \mathsf{F}(\mathsf{s}(x)), \mathsf{F}(\mathsf{g}(\mathsf{h}(x)))\rangle$ by its narrowings*

$$\langle \mathsf{F}(\mathsf{s}(\mathsf{0})), \mathsf{F}(\mathsf{g}(\mathsf{a}))\rangle \tag{32}$$

$$\langle \mathsf{F}(\mathsf{s}(x)), \mathsf{F}(\mathsf{g}(x))\rangle, \tag{33}$$

then there exists an infinite innermost chain consisting of the new dependency pair (33), because $\mathsf{F}(\mathsf{g}(x_1))\sigma \overset{\mathrm{i}}{\to}_{\mathcal{R}} \mathsf{F}(\mathsf{s}(x_2))\sigma$ holds if σ instantiates x_1 and x_2 by $\mathsf{0}$. *(In particular, if (33) is again replaced by its narrowings, then we obtain the new pair $\langle \mathsf{F}(\mathsf{s}(\mathsf{0})), \mathsf{F}(\mathsf{s}(\mathsf{0}))\rangle$ which obviously forms an infinite innermost chain.) So although $\mathsf{g}(\mathsf{h}(x))$ has no redex as a proper subterm, narrowing this term leads to a failure of the innermost termination proof.*

So there are examples where narrowing transforms a set of dependency pairs without infinite innermost chains into a new set of pairs which form an infinite innermost chain. However, this can only happen for examples, where the *automation* of our method would have failed anyway, i.e. where the constraints generated without using narrowing would already have been unsatisfiable (as in Ex. 32). More precisely, if we use the approach of Thm. 35 and if we approximate innermost dependency graphs by computing the innermost connectable pairs (Thm. 37), then every ordering satisfying the constraints generated without narrowing also satisfies the constraints generated after narrowing dependency pairs. In fact, every constraint obtained when using narrowing is implied by the constraints that one would obtain without narrowing. (The reason is that if $\langle s', t' \rangle$ and $\langle v', w' \rangle$ are narrowings of $\langle s, t \rangle$ and $\langle v, w \rangle$ respectively, then $\langle s, t \rangle$ is innermost connectable to $\langle v, w \rangle$ whenever $\langle s', t' \rangle$ is innermost connectable to $\langle v', w' \rangle$.) Hence, the application of narrowing can only extend the number of systems where innermost termination can be proved automatically.

3.5 Summary

Combining all refinements, our technique to prove innermost termination automatically using the dependency pair approach works as follows:

- Determine the dependency pairs.
- Replace some (dependency) pairs by all their narrowings. Again, this step could be repeated several times.
- Approximate the innermost dependency graph by estimating for all (dependency) pairs whether an arc exists between two of them. For that purpose we introduced the function CAP_s.
- Compute the usable rules $\mathbf{U}$, i.e. (a superset of) those rules that can be used for the reductions between two (dependency) pairs.
- Transform the usable rules and the (dependency) pairs on cycles into inequalities.
- Find a well-founded quasi-ordering satisfying the inequalities after normalizing them with respect to one of the possible AFSs.

As for the termination approach, standard techniques like the recursive path ordering or polynomial interpretations can be used to find these orderings. However, since the ordering need not be weakly monotonic for tuple symbols, we may also search for different kinds of orderings, such as polynomial interpretations where some polynomials have negative coefficients.

Our approach is the first automatic method which can also prove innermost termination of TRSs that are not terminating. Moreover, for those classes of TRSs where innermost termination already implies termination, the technique described in this section can also be used for termination proofs. In particular, this holds for non-overlapping or at least locally confluent overlay systems. The difference to the termination technique is that we only need to prove absence of infinite *innermost* chains. For that reason several steps in the technique are different to the technique of Sect. 2:

- Right-hand sides of narrowed (dependency) pairs do not have to be linear and they may unify with left-hand sides as long as their mgu does not instantiate the left-hand sides to normal forms.

- For computing the innermost dependency graph instead of the functions REN and CAP we use the function CAP_s.
- We restrict ourselves to the usable rules when transforming the rules into inequalities.
- The quasi-ordering that has to be found in the end need not be weakly monotonic on tuple symbols (unless explicitly demanded).

As long as the system is non-overlapping it is always advantageous to prove innermost termination only (instead of termination). The reason is that every ordering satisfying the constraints of the termination technique in Sect. 2 also satisfies the constraints of our innermost termination technique, but not vice versa. For instance, termination of the systems in Ex. 33 and 39 can easily be proved with the technique introduced in this section, whereas the constraints generated by the method of Sect. 2 are not satisfied by any quasi-simplification ordering. A collection of examples demonstrating the power of our technique to prove innermost termination can be found in Sect. 5.

4 Conclusion and related work

We have introduced techniques to prove termination and innermost termination of term rewriting systems automatically. For that purpose we have developed sufficient and necessary criteria for both termination and innermost termination. To automate the checking of these criteria, a set of constraints is synthesized for each TRS and standard techniques developed for termination proofs can be used to generate a well-founded ordering satisfying these constraints. If such an ordering can be found, then termination resp. innermost termination of the system is proved.

Most other methods for automated termination proofs are restricted to *simplification orderings*. Compared to proving termination directly, our approach has the advantage that the constraints generated by our method are often satisfied by standard (simplification) orderings, even if termination of the original TRS cannot be proved with these orderings. Moreover, for all those TRSs where termination can be proved with a simplification ordering directly, this simplification ordering also satisfies the inequalities resulting from our technique. Therefore, instead of using simplification orderings for direct termination proofs, it is always advantageous to combine them with the technique presented in this paper.

We implemented our technique for the generation of constraints and in this way termination could be proved automatically for many challenge problems from literature as well as for practically relevant TRSs from different areas of computer science. See Sect. 5 for a collection of numerous such examples, including arithmetical operations (e.g. mod, gcd, logarithm, average), sorting algorithms (such as selection sort, minimum sort, and quicksort), algorithms on graphs and trees, and several other well-known non-simply terminating TRSs (e.g. from [18, 20, 54]).

Our termination criteria are based on the notion of *dependency pairs*. The concept of dependency pairs was introduced in [5] and a first method for its automation was proposed in [1]. For that purpose, we transferred the *estimation technique* [33, 34], which was originally developed for termination proofs of functional programs, to rewrite systems. However, this first method was restricted to non-overlapping constructor systems without nested recursion. In this approach, the dependency pair technique was based on a special form of *semantic labelling* (cf. [60]), called self-labelling (similar to the notion of self-labelling in [49]). Self-labelling determines unique labels for the terms and a dependency pair can be regarded as a combination of the label for the left-hand side with the labels for the right-hand side of a rule.

In [2] we developed a refined framework for dependency pairs which is independent from semantic labelling. Therefore this framework is better suited for automation (as one does not have to construct an appropriate semantic interpretation any more) and its soundness can be proved in a much easier and shorter way. Moreover, in this framework we could show that our technique is applicable to arbitrary TRSs and we proved that the formulated criterion (Thm. 6) is not only sufficient, but also necessary for termination.

The present paper extends the approach of [2] by the introduction of argument filtering TRSs, the addition of narrowing dependency pairs, and by proving that the whole approach up to the search for suitable quasi-orderings is sound and *complete*, i.e. the inequalities for which an ordering should be found by standard techniques are satisfiable if and only if the TRS is terminating. This result suggests that the transformation described in this paper should always be applied before using any of the standard techniques for termination proofs.

In [3] we presented a modification of the framework, in which the notion of chains was restricted to innermost chains and we showed that a TRS is innermost terminating if and only if no infinite innermost chains exist for the TRS. This approach is the first automatic method which can also prove innermost termination of systems that are not terminating. Moreover, our technique can very successfully be used for termination proofs of non-overlapping systems, because for those systems innermost termination is already sufficient for termination.

In the present paper we extended the technique described in [3] by a refined definition of innermost dependency graphs, a method to compute better approximations of these graphs, and a more powerful approach for narrowing dependency pairs. In Sect. 5 we give a collection of several examples which can now be proved terminating resp. innermost terminating automatically, but where automatic proofs using the techniques in [2, 3] failed.

We have presented a sound and complete termination criterion. In contrast to most other complete approaches (semantic path ordering [40], general path ordering [20], semantic labelling [60] etc.) our method is particularly well suited for automation as has been demonstrated in this paper. The only other complete criterion that has been used for automatic termination proofs (by Steinbach [54]) is the approach of *transformation orderings* [8, 11]. It turns out that the termination of several examples where the automation of Steinbach failed can be proved by our technique automatically, cf. Sect. 5.

At first sight there seem to be some similarities between our method and *forward closures* [20, 47]. The idea of forward closures is to restrict the application of rules to that part of a term created by previous rewrites. Similar to our notion of chains, this notion also results in a sequence of terms, but the semantics of these sequences are completely different. For example, forward closures are reductions whereas in general the terms in a chain do not form a reduction. The reason is that in the dependency pair approach we do not restrict the *application of rules*, but we restrict the examination of *terms* to those subterms that can possibly be reduced further. Compared to the forward closure approach, the dependency pair technique has the advantage that it can be used for *arbitrary* TRSs, whereas the absence of infinite forward closures only implies termination for right-linear [16] or non-overlapping [30] TRSs. Moreover, in contrast to the dependency pair method, we do not know of any attempt to automate the forward closure approach.

The framework of dependency pairs, as introduced in this paper, is very general and is therefore well suited to be used for more general rewriting problems, too. For example, the framework of dependency pairs can easily be extended for termination modulo associativity and commutativity [48]. Moreover, several well-known and new modularity results can be derived in this framework [4, 6].

5 Examples

This collection of examples demonstrates the power of the described method. The majority of them occurred as challenge problems in the literature, whereas the other examples are added to point out specific failures of existing techniques.

Sect. 5.1 contains a collection of TRSs where termination can be proved by the technique of Sect. 2 automatically. Several of these examples are not simply terminating. Thus, all methods based on simplification orderings fail in proving termination of these systems. For those examples which are overlay systems with joinable critical pairs, termination can also be verified by proving innermost termination using the technique of Sect. 3.

In Sect. 5.2, we consider TRSs which either are not terminating or cannot be proved terminating by the technique of Sect. 2. For these systems innermost termination can be proved automatically by the technique of Sect. 3.

In most of the examples, only the inequalities resulting from dependency pairs on cycles in the (innermost) dependency graph are mentioned. We refer to these inequalities as the *relevant* inequalities. But of course, the inequalities $l \geq r$ are also synthesized for each (usable) rewrite rule $l \to r$ in the term rewriting system (and when proving innermost termination, we also obtain certain monotonicity constraints).

After generating the inequalities, an argument filtering TRS may be applied to normalize the resulting constraints. We give at most one AFS for each example, such that the normalized inequalities are satisfied by an appropriate quasi-ordering. In the following collection of examples three different techniques are used to find this quasi-ordering, viz. the recursive path ordering, the lexicographic path ordering, and polynomial interpretations.

5.1 Examples for proving termination

In this section we give a collection of examples where the technique of Sect. 2 can be used to prove termination automatically. Note that for the examples 5.1.39 – 5.1.46 we use the refinement of narrowing dependency pairs, i.e. these proofs were not possible with the method of [2].

5.1.1 Division, version 1

The TRS of Ex. 2

$$\begin{array}{rcl}
\mathsf{minus}(x,\mathsf{0}) & \to & x\\
\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{minus}(x,y)\\
\mathsf{quot}(\mathsf{0},\mathsf{s}(y)) & \to & \mathsf{0}\\
\mathsf{quot}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x,y),\mathsf{s}(y)))
\end{array}$$

is not simply terminating. In this example, apart from the four inequalities corresponding to the rewrite rules, two relevant inequalities are obtained.

$$\begin{array}{rcl}
\mathsf{M}(\mathsf{s}(x),\mathsf{s}(y)) & > & \mathsf{M}(x,y)\\
\mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)) & > & \mathsf{Q}(\mathsf{minus}(x,y),\mathsf{s}(y))
\end{array}$$

By the AFS $\mathsf{minus}(x,y) \to x$, the recursive path ordering satisfies the demands on the ordering.

5.1.2 Division, version 2

This TRS for division uses different minus-rules. Again, it is not simply terminating.

$$\begin{array}{rcl}
\mathsf{pred}(\mathsf{s}(x)) & \to & x\\
\mathsf{minus}(x,\mathsf{0}) & \to & x\\
\mathsf{minus}(x,\mathsf{s}(y)) & \to & \mathsf{pred}(\mathsf{minus}(x,y))\\
\mathsf{quot}(\mathsf{0},\mathsf{s}(y)) & \to & \mathsf{0}\\
\mathsf{quot}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x,y),\mathsf{s}(y)))
\end{array}$$

The inequalities obtained from the dependency pairs on cycles in the dependency graph are given by:

$$\begin{aligned}
\mathsf{M}(x, \mathsf{s}(y)) &> \mathsf{M}(x, y) \\
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{s}(y)).
\end{aligned}$$

Finding a suitable ordering is as easy as it was for the previous example, by choosing the AFS $\mathsf{minus}(x, y) \to x$, $\mathsf{pred}(x) \to x$. The demands on the ordering are then satisfied by the recursive path ordering.

5.1.3 Division, version 3

This TRS for division uses again different minus-rules. Similar to the preceding examples it is not simply terminating. In the examples of this collection, we often use functions like $\mathsf{if}_{\mathsf{minus}}$ to encode conditions. This ensures that conditions are evaluated first (to true or to false) and that the corresponding result is evaluated afterwards. Hence, the first argument of $\mathsf{if}_{\mathsf{minus}}$ is the condition that has to be tested and the other arguments are the original arguments of minus. Further evaluation is only possible after the condition has been reduced to true or to false.

$$\begin{aligned}
\mathsf{le}(\mathsf{0}, y) &\to \mathsf{true} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{0}) &\to \mathsf{false} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{le}(x, y) \\
\mathsf{minus}(\mathsf{0}, y) &\to \mathsf{0} \\
\mathsf{minus}(\mathsf{s}(x), y) &\to \mathsf{if}_{\mathsf{minus}}(\mathsf{le}(\mathsf{s}(x), y), \mathsf{s}(x), y) \\
\mathsf{if}_{\mathsf{minus}}(\mathsf{true}, \mathsf{s}(x), y) &\to \mathsf{0} \\
\mathsf{if}_{\mathsf{minus}}(\mathsf{false}, \mathsf{s}(x), y) &\to \mathsf{s}(\mathsf{minus}(x, y)) \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) &\to \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y)))
\end{aligned}$$

The relevant inequalities are given by

$$\begin{aligned}
\mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{LE}(x, y) \\
\mathsf{M}(\mathsf{s}(x), y) &\geq \mathsf{IF}_{\mathsf{minus}}(\mathsf{le}(\mathsf{s}(x), y), \mathsf{s}(x), y) \\
\mathsf{IF}_{\mathsf{minus}}(\mathsf{false}, \mathsf{s}(x), y) &> \mathsf{M}(x, y) \\
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{s}(y)).
\end{aligned}$$

Note that only one of the dependency pairs on a cycle in the dependency graph should result in a strict inequality, therefore the inequality

$$\mathsf{M}(\mathsf{s}(x), y) \geq \mathsf{IF}_{\mathsf{minus}}(\mathsf{le}(\mathsf{s}(x), y), \mathsf{s}(x), y)$$

need not be strict. By normalizing the inequalities with respect to the following AFS,

$$\begin{aligned}
\mathsf{minus}(x, y) &\rightarrow x \\
\mathsf{if}_{\mathsf{minus}}(b, x, y) &\rightarrow x \\
\mathsf{IF}_{\mathsf{minus}}(b, x, y) &\rightarrow x \\
\mathsf{M}(x, y) &\rightarrow x,
\end{aligned}$$

the inequalities are satisfied by the recursive path ordering.

5.1.4 Plus and minus

The following example demonstrates the use of the dependency graph. For that purpose we extend the TRS of Ex. 5.1.1 by three additional rules (Ex. 13) and write infix operators for the defined symbols minus and plus to ease readability.

$$\begin{aligned}
x - \mathsf{0} &\rightarrow x \\
\mathsf{s}(x) - \mathsf{s}(y) &\rightarrow x - y \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) &\rightarrow \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) &\rightarrow \mathsf{s}(\mathsf{quot}(x - y, \mathsf{s}(y))) \\
\mathsf{0} + y &\rightarrow y \\
\mathsf{s}(x) + y &\rightarrow \mathsf{s}(x + y) \\
(x - y) - z &\rightarrow x - (y + z)
\end{aligned}$$

In this example, termination cannot be proved with our method using a simplification ordering, unless we use the dependency graph to determine that the dependency pair $\langle \mathsf{M}(\ldots), \mathsf{P}(\ldots) \rangle$ does not occur on any cycle. Then, the only relevant inequalities are

$$\begin{aligned}
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{M}(x, y) \\
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{Q}(x - y, \mathsf{s}(y)) \\
\mathsf{P}(\mathsf{s}(x), y) &> \mathsf{P}(x, y) \\
\mathsf{M}(x - y, z) &> \mathsf{M}(x, y + z).
\end{aligned}$$

After normalizing the resulting constraints w.r.t. the AFS $x - y \rightarrow \mathsf{m}(x)$, $\mathsf{M}(x, y) \rightarrow x$, they are satisfied by the recursive path ordering.

5.1.5 Remainder, version 1 – 3

Similar to the TRSs for division, three versions of the following TRS are obtained, which again are not simply terminating. Only one of them is presented.

$$
\begin{array}{rcl}
\mathsf{le}(0, y) & \to & \mathsf{true} \\
\mathsf{le}(\mathsf{s}(x), 0) & \to & \mathsf{false} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{le}(x, y) \\
\mathsf{minus}(x, 0) & \to & x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\
\mathsf{mod}(0, y) & \to & 0 \\
\mathsf{mod}(\mathsf{s}(x), 0) & \to & 0 \\
\mathsf{mod}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{if}_{\mathsf{mod}}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{if}_{\mathsf{mod}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{mod}(\mathsf{minus}(x, y), \mathsf{s}(y)) \\
\mathsf{if}_{\mathsf{mod}}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(x)
\end{array}
$$

The relevant inequalities of this TRS are given by

$$
\begin{array}{rcl}
\mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{LE}(x, y) \\
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{M}(x, y) \\
\mathsf{MOD}(\mathsf{s}(x), \mathsf{s}(y)) & \geq & \mathsf{IF}_{\mathsf{mod}}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{IF}_{\mathsf{mod}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{MOD}(\mathsf{minus}(x, y), \mathsf{s}(y)).
\end{array}
$$

By normalizing the inequalities with respect to the following AFS,

$$
\begin{array}{rcl}
\mathsf{minus}(x, y) & \to & x \\
\mathsf{mod}(x, y) & \to & x \\
\mathsf{if}_{\mathsf{mod}}(b, x, y) & \to & x \\
\mathsf{MOD}(x, y) & \to & x \\
\mathsf{IF}_{\mathsf{mod}}(b, x, y) & \to & x,
\end{array}
$$

the interpreted inequalities are satisfied by the recursive path ordering.

5.1.6 Greatest common divisor, version 1 – 3

There are also three versions of the following TRS for the computation of the greatest common divisor, which are not simply terminating. Again, only one of them is presented.

$$
\begin{array}{rcl}
\mathsf{le}(0, y) & \to & \mathsf{true} \\
\mathsf{le}(\mathsf{s}(x), 0) & \to & \mathsf{false} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{le}(x, y) \\
\mathsf{pred}(\mathsf{s}(x)) & \to & x \\
\mathsf{minus}(x, 0) & \to & x
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{minus}(x, \mathsf{s}(y)) & \rightarrow & \mathsf{pred}(\mathsf{minus}(x, y)) \\
\mathsf{gcd}(0, y) & \rightarrow & y \\
\mathsf{gcd}(\mathsf{s}(x), 0) & \rightarrow & \mathsf{s}(x) \\
\mathsf{gcd}(\mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{if}_{\mathsf{gcd}}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{if}_{\mathsf{gcd}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{gcd}(\mathsf{minus}(x, y), \mathsf{s}(y)) \\
\mathsf{if}_{\mathsf{gcd}}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) & \rightarrow & \mathsf{gcd}(\mathsf{minus}(y, x), \mathsf{s}(x))
\end{array}
$$

(Of course the ordering of the arguments in the right-hand side of the last rule could have been switched. But this version here is even more difficult: Termination of the corresponding algorithm cannot be proved by the method of Walther [58], because this method cannot deal with permutations of arguments.) The relevant inequalities of this TRS are

$$
\begin{array}{rcl}
\mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{LE}(x, y) \\
\mathsf{M}(x, \mathsf{s}(y)) & > & \mathsf{M}(x, y) \\
\mathsf{GCD}(\mathsf{s}(x), \mathsf{s}(y)) & \geq & \mathsf{IF}_{\mathsf{gcd}}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{IF}_{\mathsf{gcd}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{GCD}(\mathsf{minus}(x, y), \mathsf{s}(y)) \\
\mathsf{IF}_{\mathsf{gcd}}(\mathsf{false}, \mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{GCD}(\mathsf{minus}(y, x), \mathsf{s}(x)).
\end{array}
$$

A suitable AFS is given by

$$
\begin{array}{rcl}
\mathsf{pred}(x) & \rightarrow & x \\
\mathsf{minus}(x, y) & \rightarrow & x \\
\mathsf{if}_{\mathsf{gcd}}(b, x, y) & \rightarrow & \mathsf{i}_{\mathsf{gcd}}(x, y) \\
\mathsf{IF}_{\mathsf{gcd}}(b, x, y) & \rightarrow & \mathsf{I}_{\mathsf{gcd}}(x, y).
\end{array}
$$

The normalized inequalities are satisfied by the recursive path ordering.

This example was taken from Boyer and Moore [14] and Walther [57]. A variant of this example could be proved terminating using Steinbach's method for the automated generation of transformation orderings [54], but there the rules for le and minus were missing.

5.1.7 Logarithm, version 1

The following TRS computes the dual logarithm.

$$
\begin{array}{rcl}
\mathsf{half}(0) & \rightarrow & 0 \\
\mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \rightarrow & \mathsf{s}(\mathsf{half}(x)) \\
\mathsf{log}(\mathsf{s}(0)) & \rightarrow & 0 \\
\mathsf{log}(\mathsf{s}(\mathsf{s}(x))) & \rightarrow & \mathsf{s}(\mathsf{log}(\mathsf{s}(\mathsf{half}(x))))
\end{array}
$$

The relevant inequalities of this TRS are

$$
\begin{array}{rcl}
\mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) & > & \mathsf{HALF}(x) \\
\mathsf{LOG}(\mathsf{s}(\mathsf{s}(x))) & > & \mathsf{LOG}(\mathsf{s}(\mathsf{half}(x))).
\end{array}
$$

No AFS is needed since the inequalities are satisfied by the recursive path ordering. (Termination of the original system can also be proved using the recursive path ordering with precedence $\mathsf{log} \triangleright \mathsf{s} \triangleright \mathsf{half}$.)

5.1.8 Logarithm, version 2 – 4

The following TRS again computes the dual logarithm, but instead of half now the function quot is used. Depending on which version of quot is used, three different versions of the TRS are obtained (all of which are not simply terminating, since the quot TRS already was not simply terminating).

$$
\begin{array}{rcl}
\mathsf{minus}(x, \mathsf{0}) & \to & x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \to & \mathsf{0} \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y))) \\
\mathsf{log}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{0} \\
\mathsf{log}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{s}(\mathsf{log}(\mathsf{s}(\mathsf{quot}(x, \mathsf{s}(\mathsf{s}(\mathsf{0}))))))
\end{array}
$$

There are three inequalities obtained from the dependency pairs on cycles in the dependency graph:

$$
\begin{array}{rcl}
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{M}(x, y) \\
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{s}(y)) \\
\mathsf{LOG}(\mathsf{s}(\mathsf{s}(x))) & > & \mathsf{LOG}(\mathsf{s}(\mathsf{quot}(x, \mathsf{s}(\mathsf{s}(\mathsf{0}))))).
\end{array}
$$

The inequalities normalized by the AFS $\mathsf{quot}(x, y) \to x$, $\mathsf{minus}(x, y) \to x$ are satisfied by the recursive path ordering.

5.1.9 Eliminating duplicates

The following TRS eliminates duplicates from a list. To represent lists the constructors nil and add are used, where nil represents the empty list and $\mathsf{add}(n, x)$ represents the insertion of n into the list x.

$$
\begin{array}{rcl}
\mathsf{eq}(\mathsf{0}, \mathsf{0}) & \to & \mathsf{true} \\
\mathsf{eq}(\mathsf{0}, \mathsf{s}(x)) & \to & \mathsf{false} \\
\mathsf{eq}(\mathsf{s}(x), \mathsf{0}) & \to & \mathsf{false} \\
\mathsf{eq}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{eq}(x, y)
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{rm}(n, \mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{rm}(n, \mathsf{add}(m, x)) & \to & \mathsf{if}_{\mathsf{rm}}(\mathsf{eq}(n, m), n, \mathsf{add}(m, x)) \\
\mathsf{if}_{\mathsf{rm}}(\mathsf{true}, n, \mathsf{add}(m, x)) & \to & \mathsf{rm}(n, x) \\
\mathsf{if}_{\mathsf{rm}}(\mathsf{false}, n, \mathsf{add}(m, x)) & \to & \mathsf{add}(m, \mathsf{rm}(n, x)) \\
\mathsf{purge}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{purge}(\mathsf{add}(n, x)) & \to & \mathsf{add}(n, \mathsf{purge}(\mathsf{rm}(n, x)))
\end{array}
$$

The relevant inequalities are

$$
\begin{array}{rcl}
\mathsf{EQ}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{EQ}(x, y) \\
\mathsf{RM}(n, \mathsf{add}(m, x)) & \geq & \mathsf{IF}_{\mathsf{rm}}(\mathsf{eq}(n, m), n, \mathsf{add}(m, x)) \\
\mathsf{IF}_{\mathsf{rm}}(\mathsf{true}, n, \mathsf{add}(m, x)) & > & \mathsf{RM}(n, x) \\
\mathsf{IF}_{\mathsf{rm}}(\mathsf{false}, n, \mathsf{add}(m, x)) & > & \mathsf{RM}(n, x) \\
\mathsf{PURGE}(\mathsf{add}(n, x)) & > & \mathsf{PURGE}(\mathsf{rm}(n, x)).
\end{array}
$$

By normalizing the inequalities with respect to the AFS

$$
\begin{array}{rcl}
\mathsf{rm}(n, x) & \to & x \\
\mathsf{if}_{\mathsf{rm}}(b, x, y) & \to & y \\
\mathsf{RM}(n, x) & \to & x \\
\mathsf{IF}_{\mathsf{rm}}(b, x, y) & \to & y,
\end{array}
$$

the inequalities are satisfied by the recursive path ordering.

This example comes from Walther [57] and a similar example was mentioned by Steinbach [54], but in Steinbach's version the rules for eq and $\mathsf{if}_{\mathsf{rm}}$ were missing.

If in the right-hand side of the last rule, $\mathsf{add}(n, \mathsf{purge}(\mathsf{rm}(\mathbf{n}, x)))$, the $\mathbf{n}$ is replaced by a term containing $\mathsf{add}(n, x)$ then a non-simply terminating TRS is obtained, but termination can still be proved in the same way.

5.1.10 Minimum sort

This TRS can be used to sort a list x by repeatedly removing its minimum. For that purpose elements of x are shifted into the second argument of $\mathsf{minsort}$, until the minimum of the list is reached. Then the function rm is used to eliminate *all* occurrences of the minimum and finally $\mathsf{minsort}$ is called recursively on the remaining list. Hence, $\mathsf{minsort}$ does not only sort a list but it also eliminates duplicates. (The corresponding version of $\mathsf{minsort}$ where duplicates are not eliminated could also be proved terminating with our technique.)

$$
\begin{array}{rcl}
\mathsf{eq}(0, 0) & \to & \mathsf{true} \\
\mathsf{eq}(0, \mathsf{s}(x)) & \to & \mathsf{false}
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{eq}(\mathsf{s}(x),\mathsf{0}) & \to & \mathsf{false}\\
\mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}(x,y)\\
\mathsf{le}(\mathsf{0},y) & \to & \mathsf{true}\\
\mathsf{le}(\mathsf{s}(x),\mathsf{0}) & \to & \mathsf{false}\\
\mathsf{le}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{le}(x,y)\\
\mathsf{app}(\mathsf{nil},y) & \to & y\\
\mathsf{app}(\mathsf{add}(n,x),y) & \to & \mathsf{add}(n,\mathsf{app}(x,y))\\
\mathsf{min}(\mathsf{add}(n,\mathsf{nil})) & \to & n\\
\mathsf{min}(\mathsf{add}(n,\mathsf{add}(m,x))) & \to & \mathsf{if}_{\mathsf{min}}(\mathsf{le}(n,m),\mathsf{add}(n,\mathsf{add}(m,x)))\\
\mathsf{if}_{\mathsf{min}}(\mathsf{true},\mathsf{add}(n,\mathsf{add}(m,x))) & \to & \mathsf{min}(\mathsf{add}(n,x))\\
\mathsf{if}_{\mathsf{min}}(\mathsf{false},\mathsf{add}(n,\mathsf{add}(m,x))) & \to & \mathsf{min}(\mathsf{add}(m,x))\\
\mathsf{rm}(n,\mathsf{nil}) & \to & \mathsf{nil}\\
\mathsf{rm}(n,\mathsf{add}(m,x)) & \to & \mathsf{if}_{\mathsf{rm}}(\mathsf{eq}(n,m),n,\mathsf{add}(m,x))\\
\mathsf{if}_{\mathsf{rm}}(\mathsf{true},n,\mathsf{add}(m,x)) & \to & \mathsf{rm}(n,x)\\
\mathsf{if}_{\mathsf{rm}}(\mathsf{false},n,\mathsf{add}(m,x)) & \to & \mathsf{add}(m,\mathsf{rm}(n,x))\\
\mathsf{minsort}(\mathsf{nil},\mathsf{nil}) & \to & \mathsf{nil}\\
\mathsf{minsort}(\mathsf{add}(n,x),y) & \to & \mathsf{if}_{\mathsf{minsort}}(\mathsf{eq}(n,\mathsf{min}(\mathsf{add}(n,x))),\mathsf{add}(n,x),y)\\
\mathsf{if}_{\mathsf{minsort}}(\mathsf{true},\mathsf{add}(n,x),y) & \to & \mathsf{add}(n,\mathsf{minsort}(\mathsf{app}(\mathsf{rm}(n,x),y),\mathsf{nil}))\\
\mathsf{if}_{\mathsf{minsort}}(\mathsf{false},\mathsf{add}(n,x),y) & \to & \mathsf{minsort}(x,\mathsf{add}(n,y))
\end{array}
$$

The relevant inequalities of this TRS are given by

$$
\begin{array}{rcl}
\mathsf{EQ}(\mathsf{s}(x),\mathsf{s}(y)) & > & \mathsf{EQ}(x,y)\\
\mathsf{LE}(\mathsf{s}(x),\mathsf{s}(y)) & > & \mathsf{LE}(x,y)\\
\mathsf{APP}(\mathsf{add}(n,x),y) & > & \mathsf{APP}(x,y)\\
\mathsf{MIN}(\mathsf{add}(n,\mathsf{add}(m,x))) & \geq & \mathsf{IF}_{\mathsf{min}}(\mathsf{le}(n,m),\mathsf{add}(n,\mathsf{add}(m,x)))\\
\mathsf{IF}_{\mathsf{min}}(\mathsf{true},\mathsf{add}(n,\mathsf{add}(m,x))) & > & \mathsf{MIN}(\mathsf{add}(n,x))\\
\mathsf{IF}_{\mathsf{min}}(\mathsf{false},\mathsf{add}(n,\mathsf{add}(m,x))) & > & \mathsf{MIN}(\mathsf{add}(m,x))\\
\mathsf{RM}(n,\mathsf{add}(m,x)) & \geq & \mathsf{IF}_{\mathsf{rm}}(\mathsf{eq}(n,m),n,\mathsf{add}(m,x))\\
\mathsf{IF}_{\mathsf{rm}}(\mathsf{true},n,\mathsf{add}(m,x)) & > & \mathsf{RM}(n,x)\\
\mathsf{IF}_{\mathsf{rm}}(\mathsf{false},n,\mathsf{add}(m,x)) & > & \mathsf{RM}(n,x)\\
\mathsf{MINSORT}(\mathsf{add}(n,x),y) & > & \mathsf{IF}_{\mathsf{minsort}}(\mathsf{eq}(n,\mathsf{min}(\mathsf{add}(n,x))),\mathsf{add}(n,x),y)\\
\mathsf{IF}_{\mathsf{minsort}}(\mathsf{true},\mathsf{add}(n,x),y) & \geq & \mathsf{MINSORT}(\mathsf{app}(\mathsf{rm}(n,x),y),\mathsf{nil})\\
\mathsf{IF}_{\mathsf{minsort}}(\mathsf{false},\mathsf{add}(n,x),y) & \geq & \mathsf{MINSORT}(x,\mathsf{add}(n,y)).
\end{array}
$$

These constraints are satisfied by a polynomial ordering where false, true, $\mathsf{0}$, nil, eq and le are mapped to 0, $\mathsf{s}(x)$ is mapped to $x+1$, $\mathsf{min}(x)$, $\mathsf{if}_{\mathsf{min}}(b,x)$,

$\mathsf{EQ}(x,y)$, $\mathsf{LE}(x,y)$, $\mathsf{MIN}(x)$, and $\mathsf{IF_{min}}(b,x)$ are mapped to x, $\mathsf{add}(n,x)$ is mapped to $n+x+1$, $\mathsf{app}(x,y)$ and $\mathsf{APP}(x,y)$ are mapped to $x+y$, $\mathsf{rm}(n,x)$, $\mathsf{if_{rm}}(b,n,x)$, $\mathsf{RM}(n,x)$, and $\mathsf{IF_{rm}}(b,n,x)$ are mapped to x, $\mathsf{minsort}(x,y)$ and $\mathsf{if_{minsort}}(b,x,y)$ are mapped to $x+y$, $\mathsf{MINSORT}(x,y)$ is mapped to $(x+y)^2+2x+y+1$, and $\mathsf{IF_{minsort}}(b,x,y)$ is mapped to $(x+y)^2+2x+y$.

This example is inspired by an algorithm from Boyer and Moore [14] and Walther [58]. In the corresponding example from Steinbach [54] the rules for eq, le, $\mathsf{if_{rm}}$, and $\mathsf{if_{min}}$ were missing.

5.1.11 Quicksort

The following TRS is used to sort a list by the well-known quicksort algorithm. It uses the functions $\mathsf{low}(n,x)$ (resp. $\mathsf{high}(n,x)$) which return the sublist of x containing only the elements smaller than or equal to (resp. greater than) n.

$$
\begin{array}{rcl}
\mathsf{le}(\mathsf{0},y) & \to & \mathsf{true}\\
\mathsf{le}(\mathsf{s}(x),\mathsf{0}) & \to & \mathsf{false}\\
\mathsf{le}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{le}(x,y)\\
\mathsf{app}(\mathsf{nil},y) & \to & y\\
\mathsf{app}(\mathsf{add}(n,x),y) & \to & \mathsf{add}(n,\mathsf{app}(x,y))\\
\mathsf{low}(n,\mathsf{nil}) & \to & \mathsf{nil}\\
\mathsf{low}(n,\mathsf{add}(m,x)) & \to & \mathsf{if_{low}}(\mathsf{le}(m,n),n,\mathsf{add}(m,x))\\
\mathsf{if_{low}}(\mathsf{true},n,\mathsf{add}(m,x)) & \to & \mathsf{add}(m,\mathsf{low}(n,x))\\
\mathsf{if_{low}}(\mathsf{false},n,\mathsf{add}(m,x)) & \to & \mathsf{low}(n,x)\\
\mathsf{high}(n,\mathsf{nil}) & \to & \mathsf{nil}\\
\mathsf{high}(n,\mathsf{add}(m,x)) & \to & \mathsf{if_{high}}(\mathsf{le}(m,n),n,\mathsf{add}(m,x))\\
\mathsf{if_{high}}(\mathsf{true},n,\mathsf{add}(m,x)) & \to & \mathsf{high}(n,x)\\
\mathsf{if_{high}}(\mathsf{false},n,\mathsf{add}(m,x)) & \to & \mathsf{add}(m,\mathsf{high}(n,x))\\
\mathsf{quicksort}(\mathsf{nil}) & \to & \mathsf{nil}\\
\mathsf{quicksort}(\mathsf{add}(n,x)) & \to & \mathsf{app}(\mathsf{quicksort}(\mathsf{low}(n,x)),\\
 & & \qquad \mathsf{add}(n,\mathsf{quicksort}(\mathsf{high}(n,x))))
\end{array}
$$

The relevant inequalities are

$$
\begin{array}{rcl}
\mathsf{LE}(\mathsf{s}(x),\mathsf{s}(y)) & > & \mathsf{LE}(x,y)\\
\mathsf{APP}(\mathsf{add}(n,x),y) & > & \mathsf{APP}(x,y)\\
\mathsf{LOW}(n,\mathsf{add}(m,x)) & \geq & \mathsf{IF_{low}}(\mathsf{le}(m,n),n,\mathsf{add}(m,x))\\
\mathsf{IF_{low}}(\mathsf{true},n,\mathsf{add}(m,x)) & > & \mathsf{LOW}(n,x)\\
\mathsf{IF_{low}}(\mathsf{false},n,\mathsf{add}(m,x)) & > & \mathsf{LOW}(n,x)\\
\mathsf{HIGH}(n,\mathsf{add}(m,x)) & \geq & \mathsf{IF_{high}}(\mathsf{le}(m,n),n,\mathsf{add}(m,x))
\end{array}
$$

$$\begin{array}{rcl}
\mathsf{IF_{high}}(\mathsf{true}, n, \mathsf{add}(m, x)) & > & \mathsf{HIGH}(n, x) \\
\mathsf{IF_{high}}(\mathsf{false}, n, \mathsf{add}(m, x)) & > & \mathsf{HIGH}(n, x) \\
\mathsf{QUICKSORT}(\mathsf{add}(n, x)) & > & \mathsf{QUICKSORT}(\mathsf{low}(n, x)) \\
\mathsf{QUICKSORT}(\mathsf{add}(n, x)) & > & \mathsf{QUICKSORT}(\mathsf{high}(n, x)).
\end{array}$$

By normalizing the inequalities by the AFS

$$\begin{array}{rcl}
\mathsf{low}(n, x) & \to & x \\
\mathsf{high}(n, x) & \to & x \\
\mathsf{if_{low}}(b, n, x) & \to & x \\
\mathsf{if_{high}}(b, n, x) & \to & x \\
\mathsf{IF_{low}}(b, n, x) & \to & \mathsf{I_{low}}(n, x) \\
\mathsf{IF_{high}}(b, n, x) & \to & \mathsf{I_{high}}(n, x),
\end{array}$$

the recursive path ordering satisfies the demands on the ordering.

Steinbach could prove termination of a corresponding example with transformation orderings [54], but in his example the rules for le, $\mathsf{if_{low}}$, $\mathsf{if_{high}}$, and app were omitted.

If in the right-hand side of the last rule,

$$\mathsf{app}(\mathsf{quicksort}(\mathsf{low}(\mathbf{n}, x)), \mathsf{add}(n, \mathsf{quicksort}(\mathsf{high}(\mathbf{n}, x)))),$$

one of the $\mathbf{n}$'s is replaced by a term containing $\mathsf{add}(n, x)$ then a non-simply terminating TRS is obtained. With our technique, termination can still be proved in the same way.

5.1.12 Permutation of lists

This example is a TRS from Walther [58] to compute a permutation of a list. For instance, $\mathsf{shuffle}([1, 2, 3, 4, 5])$ reduces to $[1, 5, 2, 4, 3]$.

$$\begin{array}{rcl}
\mathsf{app}(\mathsf{nil}, y) & \to & y \\
\mathsf{app}(\mathsf{add}(n, x), y) & \to & \mathsf{add}(n, \mathsf{app}(x, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{reverse}(\mathsf{add}(n, x)) & \to & \mathsf{app}(\mathsf{reverse}(x), \mathsf{add}(n, \mathsf{nil})) \\
\mathsf{shuffle}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{shuffle}(\mathsf{add}(n, x)) & \to & \mathsf{add}(n, \mathsf{shuffle}(\mathsf{reverse}(x)))
\end{array}$$

The inequalities obtained from the dependency pairs on cycles in the dependency graph are

$$\begin{array}{rcl}
\mathsf{APP}(\mathsf{add}(n, x), y) & > & \mathsf{APP}(x, y) \\
\mathsf{REVERSE}(\mathsf{add}(n, x)) & > & \mathsf{REVERSE}(x) \\
\mathsf{SHUFFLE}(\mathsf{add}(n, x)) & > & \mathsf{SHUFFLE}(\mathsf{reverse}(x)).
\end{array}$$

A suitable polynomial interpretation of the function symbols is: nil is mapped to 0, $\mathsf{add}(n, x)$ is mapped to $x + 1$, $\mathsf{shuffle}(x)$, $\mathsf{SHUFFLE}(x)$, $\mathsf{reverse}(x)$, and $\mathsf{REVERSE}(x)$ are mapped to x, and $\mathsf{app}(x, y)$ and $\mathsf{APP}(x, y)$ are mapped to $x + y$.

5.1.13 Reachability on directed graphs

To check whether there is a path from the node x to the node y in a directed graph g, the term $\mathsf{reach}(x, y, g, \epsilon)$ must be reducible to true with the rules of the following TRS from Giesl [32]. The fourth argument of reach is used to store edges that have already been examined but that are not included in the actual solution path. If an edge from u to v (with $x \neq u$) is found, then it is rejected at first. If an edge from x to v (with $v \neq y$) is found then one either searches for further edges beginning in x (then one will never need the edge from x to v again) or one tries to find a path from v to y and now all edges that were rejected before have to be considered again.

The function union is used to unite two graphs. The constructor ϵ denotes the empty graph and $\mathsf{edge}(x, y, g)$ represents the graph g extended by an edge from x to y. Nodes are labelled with natural numbers.

$$
\begin{array}{rcl}
\mathsf{eq}(\mathsf{0}, \mathsf{0}) & \to & \mathsf{true} \\
\mathsf{eq}(\mathsf{0}, \mathsf{s}(x)) & \to & \mathsf{false} \\
\mathsf{eq}(\mathsf{s}(x), \mathsf{0}) & \to & \mathsf{false} \\
\mathsf{eq}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{eq}(x, y) \\
\mathsf{or}(\mathsf{true}, y) & \to & \mathsf{true} \\
\mathsf{or}(\mathsf{false}, y) & \to & y \\
\mathsf{union}(\epsilon, h) & \to & h \\
\mathsf{union}(\mathsf{edge}(x, y, i), h) & \to & \mathsf{edge}(x, y, \mathsf{union}(i, h)) \\
\mathsf{reach}(x, y, \epsilon, h) & \to & \mathsf{false} \\
\mathsf{reach}(x, y, \mathsf{edge}(u, v, i), h) & \to & \mathsf{if}_{\mathsf{reach\_1}}(\mathsf{eq}(x, u), x, y, \mathsf{edge}(u, v, i), h) \\
\mathsf{if}_{\mathsf{reach\_1}}(\mathsf{true}, x, y, \mathsf{edge}(u, v, i), h) & \to & \mathsf{if}_{\mathsf{reach\_2}}(\mathsf{eq}(y, v), x, y, \mathsf{edge}(u, v, i), h) \\
\mathsf{if}_{\mathsf{reach\_2}}(\mathsf{true}, x, y, \mathsf{edge}(u, v, i), h) & \to & \mathsf{true} \\
\mathsf{if}_{\mathsf{reach\_2}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) & \to & \mathsf{or}(\mathsf{reach}(x, y, i, h), \\
& & \quad \mathsf{reach}(v, y, \mathsf{union}(i, h), \epsilon)) \\
\mathsf{if}_{\mathsf{reach\_1}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) & \to & \mathsf{reach}(x, y, i, \mathsf{edge}(u, v, h))
\end{array}
$$

The inequalities obtained from dependency pairs on cycles in the dependency graph are given by

$$
\begin{array}{rcl}
\mathsf{EQ}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{EQ}(x, y) \\
\mathsf{UNION}(\mathsf{edge}(x, y, i), h) & > & \mathsf{UNION}(i, h)
\end{array}
$$

$$
\begin{array}{rcl}
\mathsf{REACH}(x, y, \mathsf{edge}(u, v, i), h) & \geq & \mathsf{IF}_{\mathsf{reach\_1}}(\mathsf{eq}(x, u), x, y, \mathsf{edge}(u, v, i), h) \\
\mathsf{IF}_{\mathsf{reach\_1}}(\mathsf{true}, x, y, \mathsf{edge}(u, v, i), h) & \geq & \mathsf{IF}_{\mathsf{reach\_2}}(\mathsf{eq}(y, v), x, y, \mathsf{edge}(u, v, i), h) \\
\mathsf{IF}_{\mathsf{reach\_2}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) & > & \mathsf{REACH}(x, y, i, h) \\
\mathsf{IF}_{\mathsf{reach\_2}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) & > & \mathsf{REACH}(v, y, \mathsf{union}(i, h), \epsilon) \\
\mathsf{IF}_{\mathsf{reach\_1}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) & > & \mathsf{REACH}(x, y, i, \mathsf{edge}(u, v, h)).
\end{array}
$$

A mapping to polynomials results in a suitable ordering. The interpretation is: $\mathsf{eq}(x, y)$, true, false, ϵ, and $\mathsf{0}$ are mapped to 0, $\mathsf{or}(x, y)$ is mapped to $x + y$, $\mathsf{s}(x)$ is mapped to $x + 1$, $\mathsf{EQ}(x, y)$ is mapped to x, $\mathsf{edge}(x, y, g)$ is mapped to $g + 2$, $\mathsf{union}(g, h)$ and $\mathsf{UNION}(g, h)$ are mapped to $g + h$, $\mathsf{reach}(x, y, g, h)$, $\mathsf{if}_{\mathsf{reach\_1}}(b, x, y, g, h)$, and $\mathsf{if}_{\mathsf{reach\_2}}(b, x, y, g, h)$ are mapped to 0, $\mathsf{REACH}(x, y, g, h)$ is mapped to $(g + h)^2 + 2g + h + 2$, $\mathsf{IF}_{\mathsf{reach\_1}}(b, x, y, g, h)$ is mapped to $(g + h)^2 + 2g + h + 1$, and $\mathsf{IF}_{\mathsf{reach\_2}}(b, x, y, g, h)$ is mapped to $(g + h)^2 + 2g + h$.

5.1.14 Comparison of binary trees

This TRS is used to find out if one binary tree has less leaves than another one. It uses a function $\mathsf{concat}(x, y)$ to replace the rightmost leaf of x by y. Here, $\mathsf{cons}(u, v)$ is used to built a tree with the two direct subtrees u and v.

$$
\begin{array}{rcl}
\mathsf{concat}(\mathsf{leaf}, y) & \to & y \\
\mathsf{concat}(\mathsf{cons}(u, v), y) & \to & \mathsf{cons}(u, \mathsf{concat}(v, y)) \\
\mathsf{less\_leaves}(x, \mathsf{leaf}) & \to & \mathsf{false} \\
\mathsf{less\_leaves}(\mathsf{leaf}, \mathsf{cons}(w, z)) & \to & \mathsf{true} \\
\mathsf{less\_leaves}(\mathsf{cons}(u, v), \mathsf{cons}(w, z)) & \to & \mathsf{less\_leaves}(\mathsf{concat}(u, v), \mathsf{concat}(w, z))
\end{array}
$$

The inequalities corresponding to the dependency pairs on cycles in the dependency graph are:

$$
\begin{array}{rcl}
\mathsf{CONCAT}(\mathsf{cons}(u, v), y) & > & \mathsf{CONCAT}(v, y) \\
\mathsf{LESS\_LEAVES}(\mathsf{cons}(u, v), \mathsf{cons}(w, z)) & > & \mathsf{LESS\_LEAVES}(\mathsf{concat}(u, v), \mathsf{concat}(w, z)).
\end{array}
$$

A suitable (polynomial) interpretation is: leaf, false, and true are mapped to 0, $\mathsf{cons}(u, v)$ is mapped to $1 + u + v$, $\mathsf{concat}(u, v)$ and $\mathsf{CONCAT}(u, v)$ are mapped to $u + v$, and $\mathsf{less\_leaves}(x, y)$ and $\mathsf{LESS\_LEAVES}(x, y)$ are mapped to x.

If $\mathsf{concat}(w, z)$ in the second argument of $\mathsf{less\_leaves}$ (in the right-hand side of the last rule) would be replaced by an appropriate argument, we would obtain a non-simply terminating TRS whose termination could be proved in the same way.

5.1.15 Average of naturals

The following locally confluent overlay system computes the average of two numbers [20].

$$\begin{array}{rcl}
\mathsf{average}(\mathsf{s}(x), y) & \to & \mathsf{average}(x, \mathsf{s}(y)) \\
\mathsf{average}(x, \mathsf{s}(\mathsf{s}(\mathsf{s}(y)))) & \to & \mathsf{s}(\mathsf{average}(\mathsf{s}(x), y)) \\
\mathsf{average}(\mathsf{0}, \mathsf{0}) & \to & \mathsf{0} \\
\mathsf{average}(\mathsf{0}, \mathsf{s}(\mathsf{0})) & \to & \mathsf{0} \\
\mathsf{average}(\mathsf{0}, \mathsf{s}(\mathsf{s}(\mathsf{0}))) & \to & \mathsf{s}(\mathsf{0})
\end{array}$$

The relevant inequalities are

$$\begin{array}{rcl}
\mathsf{AVERAGE}(\mathsf{s}(x), y) & > & \mathsf{AVERAGE}(x, \mathsf{s}(y)) \\
\mathsf{AVERAGE}(x, \mathsf{s}(\mathsf{s}(\mathsf{s}(y)))) & > & \mathsf{AVERAGE}(\mathsf{s}(x), y)).
\end{array}$$

By the following polynomial interpretation, termination of this TRS is easily proved: $\mathsf{0}$ is mapped to 0, $\mathsf{s}(x)$ is mapped to $x + 1$, $\mathsf{average}(x, y)$ is mapped to $x + y$, and $\mathsf{AVERAGE}(x, y)$ is mapped to $2x + y$.

5.1.16 Plus and times

The following TRS [20] is again a locally confluent overlay system. To ease readability we use an infix notation for $+$ and $\times$.

$$\begin{array}{rcl}
x \times \mathsf{0} & \to & \mathsf{0} \\
x \times \mathsf{s}(y) & \to & (x \times y) + x \\
x + \mathsf{0} & \to & x \\
\mathsf{0} + x & \to & x \\
x + \mathsf{s}(y) & \to & \mathsf{s}(x + y) \\
\mathsf{s}(x) + y & \to & \mathsf{s}(x + y)
\end{array}$$

Applying the technique results in a set of inequalities which is satisfied by the polynomial interpretation where $\mathsf{0}$ is mapped to 0, $\mathsf{s}(x)$ is mapped to $x + 1$, $x + y$ is mapped to the sum of x and y, $x \times y$ is mapped to the product of x and y, $\mathsf{TIMES}(x, y)$ is mapped to y, and $\mathsf{P}(x, y)$ is mapped to the sum of x and y (where P denotes the tuple symbol for '+').

5.1.17 Summing elements of lists

This TRS, which has overlapping rules, can be used to compute the sum of all elements of a list [2]. Here, $x.\,l$ represents the insertion of a number x into a list l (where $x.\,y.\,l$ abbreviates $(x.\,(y.\,l))$), app computes the concatenation of lists, and $\mathsf{sum}(l)$ is used to compute the sum of all numbers in l (e.g. sum applied to the list $[1,2,3]$ returns $[1+2+3]$).

$$\begin{array}{rcl}
\mathsf{app}(\mathsf{nil}, k) & \to & k \\
\mathsf{app}(l, \mathsf{nil}) & \to & l \\
\mathsf{app}(x.\, l, k) & \to & x.\, \mathsf{app}(l, k) \\
\mathsf{sum}(x.\, \mathsf{nil}) & \to & x.\, \mathsf{nil} \\
\mathsf{sum}(x.\, y.\, l) & \to & \mathsf{sum}((x+y).\, l) \\
\mathsf{sum}(\mathsf{app}(l, x.\, y.\, k)) & \to & \mathsf{sum}(\mathsf{app}(l, \mathsf{sum}(x.\, y.\, k))) \\
\mathsf{0} + y & \to & y \\
\mathsf{s}(x) + y & \to & \mathsf{s}(x+y)
\end{array}$$

While this system is not simply terminating, the inequalities generated by the technique are satisfied by the polynomial ordering where nil is mapped to the constant 0, $x.\, l$ is mapped to $l+1$, $x+y$ is mapped to the sum of x and y, $\mathsf{app}(l,k)$ is mapped to $l+k+1$, $\mathsf{sum}(l)$ is mapped to the constant 1, $\mathsf{APP}(l,k)$ and $\mathsf{SUM}(l)$ are both mapped to l, and $\mathsf{P}(x,y)$ is mapped to x.
If the above TRS is extended by the rules

$$\begin{array}{rcl}
\mathsf{sum}(\mathsf{0}.\, x+y.\, l) & \to & \mathsf{pred}(\mathsf{sum}(\mathsf{s}(x).\, y.\, l)) \\
\mathsf{pred}(\mathsf{s}(x).\, \mathsf{nil}) & \to & x.\, \mathsf{nil},
\end{array}$$

then termination can still be proved in the same way (where the polynomial interpretation should map $\mathsf{pred}(l)$ to the constant 1).

5.1.18 Addition and subtraction

The following system is again overlapping and not simply terminating.

$$\begin{array}{rcl}
\mathsf{minus}(x, \mathsf{0}) & \to & x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\
\mathsf{double}(\mathsf{0}) & \to & \mathsf{0} \\
\mathsf{double}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(\mathsf{double}(x))) \\
\mathsf{plus}(\mathsf{0}, y) & \to & y \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(x, y)) \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{plus}(x, \mathsf{s}(y)) \\
\mathsf{plus}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{plus}(\mathsf{minus}(x, y), \mathsf{double}(y)))
\end{array}$$

The inequalities generated by the technique of Sect. 2 are satisfied by the lexicographic path ordering, after normalizing the inequalities by the AFS $\mathsf{minus}(x,y) \to x$.

5.1.19 Addition with nested recursion, version 1

If the following additional rule is added to the above system, then it is turned into a TRS that is not an overlay system any more and which furthermore introduces nested recursion.

$$\mathsf{plus}(\mathsf{s}(\mathsf{plus}(x,y)),z) \;\to\; \mathsf{s}(\mathsf{plus}(\mathsf{plus}(x,y),z))$$

Still, the resulting inequalities are satisfied using the same AFS and the lexicographic path ordering.

5.1.20 Addition with nested recursion, version 2

The following alternative TRS for addition from Steinbach [54] has nested recursion, too.

$$\begin{aligned} \mathsf{0} + y &\to y \\ \mathsf{s}(x) + \mathsf{0} &\to \mathsf{s}(x) \\ \mathsf{s}(x) + \mathsf{s}(y) &\to \mathsf{s}(\mathsf{s}(x) + (y + \mathsf{0})) \end{aligned}$$

The 'natural' polynomial interpretation (where $+$ is mapped to the addition) maps left and right-hand sides of the rules to the same numbers. Therefore this polynomial ordering cannot be used for a direct termination proof, but it nevertheless satisfies the inequalities generated by the technique of Sect. 2. In this way, termination can easily be proved.

5.1.21 Multiplication and addition

The following example is taken from Dershowitz [18].

$$\begin{aligned} x \times (y + \mathsf{1}) &\to (x \times (y + (\mathsf{1} \times \mathsf{0}))) + x \\ x \times \mathsf{1} &\to x \\ x + \mathsf{0} &\to x \\ x \times \mathsf{0} &\to \mathsf{0} \end{aligned}$$

The only inequality resulting from a dependency pair on a cycle in the dependency graph is $\mathsf{TIMES}(x, y + \mathsf{1}) > \mathsf{TIMES}(x, y + (\mathsf{1} \times \mathsf{0}))$.

This system is not simply terminating (and Dershowitz illustrates the use of the semantic path ordering with it). However, termination of this example can be proved automatically. The inequalities obtained are satisfied by the natural polynomial ordering, where $\mathsf{TIMES}(x, y)$ is mapped to y.

5.1.22 Extended multiplication and addition

Similarly, termination of the following 'extended' version of the above system can be proved. In this system, the full rules for $+$ and $\times$ are added. Again, this system is not an overlay system.

$$\begin{array}{rcl} x \times (y + \mathsf{s}(z)) & \to & (x \times (y + (\mathsf{s}(z) \times \mathsf{0}))) + (x \times \mathsf{s}(z)) \\ x \times \mathsf{0} & \to & \mathsf{0} \\ x \times \mathsf{s}(y) & \to & (x \times y) + x \\ x + \mathsf{0} & \to & x \\ x + \mathsf{s}(y) & \to & \mathsf{s}(x + y) \end{array}$$

The generated inequalities for this extended example, i.e. the inequalities corresponding to the rewrite rules and

$$\begin{array}{rcl} \mathsf{TIMES}(x, y + \mathsf{s}(z)) & \geq & \mathsf{TIMES}(x, \mathsf{s}(z)) \\ \mathsf{TIMES}(x, y + \mathsf{s}(z)) & > & \mathsf{TIMES}(x, y + (\mathsf{s}(z) \times \mathsf{0})) \\ \mathsf{TIMES}(x, \mathsf{s}(y)) & > & \mathsf{TIMES}(x, y) \\ \mathsf{P}(x, \mathsf{s}(y)) & > & \mathsf{P}(x, y) \end{array}$$

are satisfied by the same polynomial ordering that has been used above (where $\mathsf{P}(x, y)$ and $\mathsf{TIMES}(x, y)$ are both mapped to y).

5.1.23 Nested recursion, version 1

The following system was introduced by Giesl [34, '$\mathsf{nest2}$'] as an example for a small TRS with nested recursion where all simplification orderings fail.

$$\begin{array}{rcl} \mathsf{f}(\mathsf{0}, y) & \to & \mathsf{0} \\ \mathsf{f}(\mathsf{s}(x), y) & \to & \mathsf{f}(\mathsf{f}(x, y), y) \end{array}$$

For this example, a polynomial ordering can be used where $\mathsf{0}$ and s are interpreted as usual and both $\mathsf{f}(x, y)$ and $\mathsf{F}(x, y)$ are mapped to x.

5.1.24 Nested recursion, version 2

This system by Walther, which is similar to the preceding one, has been examined in [54].

$$\begin{array}{rcl} \mathsf{f}(\mathsf{0}) & \to & \mathsf{s}(\mathsf{0}) \\ \mathsf{f}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{s}(\mathsf{0}) \\ \mathsf{f}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{f}(\mathsf{f}(\mathsf{s}(x))) \end{array}$$

The inequalities resulting from our transformation are satisfied by the polynomial ordering, where $\mathsf{f}(x)$ is mapped to the constant 1, $\mathsf{F}(x)$ is mapped to x, and where $\mathsf{0}$ and s are interpreted as usual.

5.1.25 Nested recursion, version 3

The following TRS by Ferreira and Zantema [24] is a string rewriting system with minimal ordinal ω^ω associated to it.

$$\begin{array}{rcl} \mathsf{f}(\mathsf{g}(x)) & \to & \mathsf{g}(\mathsf{f}(\mathsf{f}(x))) \\ \mathsf{f}(\mathsf{h}(x)) & \to & \mathsf{h}(\mathsf{g}(x)) \end{array}$$

The relevant inequalities corresponding to this system are

$$\begin{array}{rcl} \mathsf{F}(\mathsf{g}(x)) & > & \mathsf{F}(x) \\ \mathsf{F}(\mathsf{g}(x)) & > & \mathsf{F}(\mathsf{f}(x)). \end{array}$$

After normalizing the inequalities by the AFS

$$\begin{array}{rcl} \mathsf{h}(x) & \to & \mathsf{h}' \\ \mathsf{f}(x) & \to & x, \end{array}$$

all inequalities are satisfied by the recursive path ordering.

5.1.26 Nested recursion, version 4

The following TRS is again an example of a TRS for which all kind of path orderings cannot show termination directly, but these path orderings can be used for solving the inequalities resulting from our technique.

$$\begin{array}{rcl} \mathsf{f}(x) & \to & \mathsf{s}(x) \\ \mathsf{f}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{s}(\mathsf{f}(\mathsf{f}(x))) \end{array}$$

The inequalities to satisfy are

$$\begin{array}{rcl} \mathsf{f}(x) & \geq & \mathsf{s}(x) \\ \mathsf{f}(\mathsf{s}(\mathsf{s}(x))) & \geq & \mathsf{s}(\mathsf{f}(\mathsf{f}(x))) \\ \mathsf{F}(\mathsf{s}(\mathsf{s}(x))) & > & \mathsf{F}(x) \\ \mathsf{F}(\mathsf{s}(\mathsf{s}(x))) & > & \mathsf{F}(\mathsf{f}(x)). \end{array}$$

An appropriate path ordering is found by choosing f and s to be equal in the precedence.

5.1.27 Nested symbols on left-hand sides

The following example is from Dershowitz [19]. It has been proved terminating by a lexicographic combination of two orderings.

$$\begin{array}{rcl} \mathsf{f}(\mathsf{f}(x)) & \to & \mathsf{g}(\mathsf{f}(x)) \\ \mathsf{g}(\mathsf{g}(x)) & \to & \mathsf{f}(x) \end{array}$$

The inequalities corresponding to dependency pairs on cycles in the dependency graph are

$$\begin{aligned}
\mathsf{F}(\mathsf{f}(x)) &> \mathsf{F}(x)\\
\mathsf{F}(\mathsf{f}(x)) &\geq \mathsf{G}(\mathsf{f}(x))\\
\mathsf{G}(\mathsf{g}(x)) &> \mathsf{F}(x).
\end{aligned}$$

By choosing f and g as well as F and G equal in the precedence, the inequalities are satisfied by the recursive path ordering.

5.1.28 Nested symbols on both sides of rules

Termination of the following TRS cannot be proved by the lexicographic path ordering and therefore this is one of the systems for which the semantic path ordering has been used in literature [19]. However, the system can be shown to terminate using the lexicographic path ordering after applying our technique, since the demanded ordering may now be a *weakly* monotonic ordering instead of a monotonic ordering. Therefore, after mapping some function symbols to some of their arguments or to a constant the lexicographic path ordering can nevertheless be used to prove termination of the TRS.

$$\begin{aligned}
(x \times y) \times z &\to x \times (y \times z)\\
(x + y) \times z &\to (x \times z) + (y \times z)\\
z \times (x + \mathsf{f}(y)) &\to \mathsf{g}(z, y) \times (x + \mathsf{a})
\end{aligned}$$

Apart from the three inequalities corresponding to the rewrite rules, four other inequalities are obtained from the cycles in the dependency graph.

$$\begin{aligned}
\mathsf{TIMES}(x \times y, z) &> \mathsf{TIMES}(y, z)\\
\mathsf{TIMES}(x \times y, z) &> \mathsf{TIMES}(x, y \times z)\\
\mathsf{TIMES}(x + y, z) &> \mathsf{TIMES}(x, z)\\
\mathsf{TIMES}(x + y, z) &> \mathsf{TIMES}(y, z)
\end{aligned}$$

The seven inequalities are satisfied by the lexicographic path ordering if the inequalities are normalized by the AFS $\mathsf{g}(z, y) \to z$.

5.1.29 A TRS that is not left-linear

The following TRS, originally from Geerling [28], cannot be proved terminating by the recursive path ordering (but one needs a generalization of the recursive path ordering as defined by Ferreira [27]). It is also very easily proved terminating by the automatic technique described in this paper.

$$\mathsf{f}(\mathsf{s}(x), y, y) \to \mathsf{f}(y, x, \mathsf{s}(x))$$

The only two generated inequalities are

$$\begin{array}{rcl} \mathsf{f}(\mathsf{s}(x), y, y) & \geq & \mathsf{f}(y, x, \mathsf{s}(x)) \\ \mathsf{F}(\mathsf{s}(x), y, y) & > & \mathsf{F}(y, x, \mathsf{s}(x)) \end{array}$$

which are satisfied by mapping $\mathsf{f}(x, y, z)$ to 0, mapping $\mathsf{s}(x)$ to $x+1$, and mapping $\mathsf{F}(x, y, z)$ to $x + y$.

5.1.30 Advantage of the dependency graph, version 1

The following system is from [54].

$$\begin{array}{rcl} \mathsf{f}(\mathsf{a}, \mathsf{b}) & \to & \mathsf{f}(\mathsf{a}, \mathsf{c}) \\ \mathsf{f}(\mathsf{c}, \mathsf{d}) & \to & \mathsf{f}(\mathsf{b}, \mathsf{d}) \end{array}$$

With our method, the termination proof for this system is trivial, because its dependency graph does not contain any cycles. This can easily be determined automatically, as $\mathsf{F}(\mathsf{a}, \mathsf{c})$ is not connectable to $\mathsf{F}(\mathsf{a}, \mathsf{b})$ or $\mathsf{F}(\mathsf{c}, \mathsf{d})$, neither is $\mathsf{F}(\mathsf{c}, \mathsf{d})$ connectable to $\mathsf{F}(\mathsf{a}, \mathsf{b})$ or $\mathsf{F}(\mathsf{c}, \mathsf{d})$.

5.1.31 Advantage of the dependency graph, version 2

Another example where the dependency graph plays an important role is a TRS introduced by Ferreira and Zantema [26] to demonstrate the technique of 'dummy elimination'.

$$\mathsf{f}(\mathsf{g}(x)) \quad \to \quad \mathsf{f}(\mathsf{a}(\mathsf{g}(\mathsf{g}(\mathsf{f}(x))), \mathsf{g}(\mathsf{f}(x))))$$

Since $\mathsf{F}(\mathsf{a}(y, z))$ does not unify with $\mathsf{F}(\mathsf{g}(x))$, the only two inequalities to satisfy are

$$\begin{array}{rcl} \mathsf{f}(\mathsf{g}(x)) & \geq & \mathsf{f}(\mathsf{a}(\mathsf{g}(\mathsf{g}(\mathsf{f}(x))), \mathsf{g}(\mathsf{f}(x)))) \\ \mathsf{F}(\mathsf{g}(x)) & > & \mathsf{F}(x). \end{array}$$

The recursive path ordering satisfies the inequalities when normalizing them by the AFS $\mathsf{a}(x, y) \to \mathsf{a}'$.

5.1.32 A TRS that is not totally terminating, version 1

The most famous example of a TRS that is terminating, but not *totally* terminating is the following [18].

$$\begin{array}{rcl} \mathsf{f}(\mathsf{a}) & \to & \mathsf{f}(\mathsf{b}) \\ \mathsf{g}(\mathsf{b}) & \to & \mathsf{g}(\mathsf{a}) \end{array}$$

With our approach, termination of this system is obvious, because the dependency graph does not contain any cycles.

5.1.33 A TRS that is not totally terminating, version 2

A TRS introduced by Ferreira [27] as an example of a TRS that is not totally terminating and in particular for which the recursive path ordering and the Knuth-Bendix ordering cannot be used to prove termination, is given by:

$$\begin{aligned} \mathsf{p}(\mathsf{f}(\mathsf{f}(x))) &\rightarrow \mathsf{q}(\mathsf{f}(\mathsf{g}(x))) \\ \mathsf{p}(\mathsf{g}(\mathsf{g}(x))) &\rightarrow \mathsf{q}(\mathsf{g}(\mathsf{f}(x))) \\ \mathsf{q}(\mathsf{f}(\mathsf{f}(x))) &\rightarrow \mathsf{p}(\mathsf{f}(\mathsf{g}(x))) \\ \mathsf{q}(\mathsf{g}(\mathsf{g}(x))) &\rightarrow \mathsf{p}(\mathsf{g}(\mathsf{f}(x))). \end{aligned}$$

Termination is trivially concluded from the fact that there are no cycles in the dependency graph.

5.1.34 Systems with 'undefined' function symbols

The following well-known system from Dershowitz [18] is one of the smallest non-simply terminating TRSs.

$$\mathsf{f}(\mathsf{f}(x)) \rightarrow \mathsf{f}(\mathsf{g}(\mathsf{f}(x)))$$

As $\mathsf{F}(\mathsf{g}(\mathsf{f}(x)))$ is not connectable to $\mathsf{F}(\mathsf{f}(x))$, the only dependency pair on a cycle of the dependency graph is $\langle \mathsf{F}(\mathsf{f}(x)), \mathsf{F}(x)\rangle$. The resulting inequalities are for instance satisfied by a polynomial ordering where $\mathsf{f}(x)$ is mapped to $x+1$ and g is mapped to the identity. In a completely analogous way, termination of the one rule TRS $\mathsf{f}(\mathsf{g}(x)) \rightarrow \mathsf{f}(\mathsf{h}(\mathsf{g}(x)))$ from Bellegarde and Lescanne [10] and of the one rule system $\mathsf{f}(\mathsf{g}(x,y),y) \rightarrow \mathsf{f}(\mathsf{h}(\mathsf{g}(x,y)),\mathsf{a})$ from Steinbach [54] can also be proved.

5.1.35 Mutual recursion, version 1

The following system is from Steinbach [54] again.

$$\begin{aligned} \mathsf{g}(\mathsf{s}(x)) &\rightarrow \mathsf{f}(x) \\ \mathsf{f}(0) &\rightarrow \mathsf{s}(0) \\ \mathsf{f}(\mathsf{s}(x)) &\rightarrow \mathsf{s}(\mathsf{s}(\mathsf{g}(x))) \\ \mathsf{g}(0) &\rightarrow 0 \end{aligned}$$

The relevant inequalities are

$$\begin{aligned} \mathsf{G}(\mathsf{s}(x)) &\geq \mathsf{F}(x) \\ \mathsf{F}(\mathsf{s}(x)) &> \mathsf{G}(x). \end{aligned}$$

After normalizing the resulting inequalities w.r.t. the AFS $\mathsf{g}(x) \rightarrow x$, the constraints are satisfied by the recursive path ordering.

5.1.36 Mutual recursion, version 2

The following system was given to us by Kühler.

$$
\begin{array}{rcl}
\mathsf{minus}(x, \mathsf{0}) & \to & x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{minus}(x, y) \\
\mathsf{f}(\mathsf{0}) & \to & \mathsf{s}(\mathsf{0}) \\
\mathsf{f}(\mathsf{s}(x)) & \to & \mathsf{minus}(\mathsf{s}(x), \mathsf{g}(\mathsf{f}(x))) \\
\mathsf{g}(\mathsf{0}) & \to & \mathsf{0} \\
\mathsf{g}(\mathsf{s}(x)) & \to & \mathsf{minus}(\mathsf{s}(x), \mathsf{f}(\mathsf{g}(x)))
\end{array}
$$

The inequalities resulting from dependency pairs on cycles of the innermost dependency graph are

$$
\begin{array}{rcl}
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{M}(x, y) \\
\mathsf{F}(\mathsf{s}(x)) & > & \mathsf{F}(x) \\
\mathsf{F}(\mathsf{s}(x)) & \geq & \mathsf{G}(\mathsf{f}(x)) \\
\mathsf{G}(\mathsf{s}(x)) & > & \mathsf{G}(x) \\
\mathsf{G}(\mathsf{s}(x)) & > & \mathsf{F}(\mathsf{g}(x)).
\end{array}
$$

After normalizing the resulting constraints with the AFS $\mathsf{minus}(x, y) \to x$, the resulting inequalities are satisfied by the recursive path ordering (using a precedence where f and s are equal and greater than g.)

5.1.37 Even and Odd

The following (non-simply terminating) TRS can be used to compute whether a natural number is even resp. odd. More precisely, $\mathsf{evenodd}(t, \mathsf{0})$ reduces to true if t is even and $\mathsf{evenodd}(t, \mathsf{s}(\mathsf{0}))$ reduces to true if t is odd. (In other words, the second argument of $\mathsf{evenodd}$ determines whether $\mathsf{evenodd}$ computes the 'even' or the 'odd' function. Such rewrite systems are often obtained when transforming mutually recursive functions into one function without mutual recursion, cf. [34].)

$$
\begin{array}{rcl}
\mathsf{not}(\mathsf{true}) & \to & \mathsf{false} \\
\mathsf{not}(\mathsf{false}) & \to & \mathsf{true} \\
\mathsf{evenodd}(x, \mathsf{0}) & \to & \mathsf{not}(\mathsf{evenodd}(x, \mathsf{s}(\mathsf{0}))) \\
\mathsf{evenodd}(\mathsf{0}, \mathsf{s}(\mathsf{0})) & \to & \mathsf{false} \\
\mathsf{evenodd}(\mathsf{s}(x), \mathsf{s}(\mathsf{0})) & \to & \mathsf{evenodd}(x, \mathsf{0})
\end{array}
$$

We obtain the following relevant inequalities.

$$
\begin{array}{rcl}
\mathsf{EVENODD}(x, \mathsf{0}) & \geq & \mathsf{EVENODD}(x, \mathsf{s}(\mathsf{0})) \\
\mathsf{EVENODD}(\mathsf{s}(x), \mathsf{s}(\mathsf{0})) & > & \mathsf{EVENODD}(x, \mathsf{0})
\end{array}
$$

After application of the AFS $\mathsf{not}(x) \to \mathsf{n}$, $\mathsf{EVENODD}(x, y) \to x$, the recursive path ordering satisfies the resulting constraints.

5.1.38 Reversing Lists

The following system is a slight variant of a TRS proposed by Huet and Hullot [38, 'brev']. Given a list $x.\,l$, the function rev calls two other functions $\mathsf{rev1}$ and $\mathsf{rev2}$, where $\mathsf{rev1}(x, l)$ returns the last element of $x.\,l$ and $\mathsf{rev2}(x, l)$ returns the reversed list $\mathsf{rev}(x.\,l)$ *without its first element.* Hence, $\mathsf{rev}(\mathsf{rev2}(y, l))$ returns the list $y.\,l$ without its last element. Note that this system is mutually recursive and that mutually recursive functions also occur nested.

$$
\begin{aligned}
\mathsf{rev}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{rev}(x.\,l) &\to \mathsf{rev1}(x, l).\,\mathsf{rev2}(x, l)\\
\mathsf{rev1}(\mathsf{0}, \mathsf{nil}) &\to \mathsf{0}\\
\mathsf{rev1}(\mathsf{s}(x), \mathsf{nil}) &\to \mathsf{s}(x)\\
\mathsf{rev1}(x, y.\,l) &\to \mathsf{rev1}(y, l)\\
\mathsf{rev2}(x, \mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{rev2}(x, y.\,l) &\to \mathsf{rev}(x.\,\mathsf{rev}(\mathsf{rev2}(y, l)))
\end{aligned}
$$

The relevant inequalities are

$$
\begin{aligned}
\mathsf{REV}(x.\,l) &> \mathsf{REV2}(x, l)\\
\mathsf{REV1}(x, y.\,l) &> \mathsf{REV1}(y, l)\\
\mathsf{REV2}(x, y.\,l) &> \mathsf{REV2}(y, l)\\
\mathsf{REV2}(x, y.\,l) &\geq \mathsf{REV}(\mathsf{rev2}(y, l))\\
\mathsf{REV2}(x, y.\,l) &\geq \mathsf{REV}(x.\,\mathsf{rev}(\mathsf{rev2}(y, l))).
\end{aligned}
$$

We use the following AFS:

$$
\begin{aligned}
x.\,y &\to \mathsf{f}(y)\\
\mathsf{s}(x) &\to \mathsf{s}'\\
\mathsf{rev}(x) &\to x\\
\mathsf{rev1}(x, y) &\to y\\
\mathsf{rev2}(x, y) &\to y\\
\mathsf{REV}(x) &\to x\\
\mathsf{REV1}(x, y) &\to y\\
\mathsf{REV2}(x, y) &\to y.
\end{aligned}
$$

Then the resulting constraints are satisfied by the recursive path ordering.

5.1.39 Narrowing of dependency pairs

The following example (Ex. 20) demonstrates the need for narrowing dependency pairs. We replace the last rule of the TRS in Ex. 5.1.4 by a 'commutativity' rule (here, $\mathsf{s0}$ abbreviates $\mathsf{s}(0)$ etc.):

$$\begin{array}{rcl}
x - 0 & \to & x \\
\mathsf{s}(x) - \mathsf{s}(y) & \to & x - y \\
\mathsf{quot}(0, \mathsf{s}(y)) & \to & 0 \\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(x - y, \mathsf{s}(y))) \\
0 + y & \to & y \\
\mathsf{s}(x) + y & \to & \mathsf{s}(x + y) \\
(x - \mathsf{s0}) + (y - \mathsf{ss}z) & \to & (y - \mathsf{ss}z) + (x - \mathsf{s0}).
\end{array}$$

Without the use of narrowing, we would obtain the constraint

$$\mathsf{P}(x - \mathsf{s0}, y - \mathsf{ss}z) > \mathsf{P}(y - \mathsf{ss}z, x - \mathsf{s0}),$$

because the dependency pair $\langle \mathsf{P}(x - \mathsf{s0}, y - \mathsf{ss}z), \mathsf{P}(y - \mathsf{ss}z, x - \mathsf{s0})\rangle$ forms a cycle of the dependency graph. In order to use a simplification ordering we have to normalize the inequalities w.r.t. an AFS which rewrites $x - y$ to $\mathsf{m}(x)$ (or to x). However, then this constraint is not satisfied by any well-founded ordering closed under substitution. Therefore we replace this dependency pair by its narrowings

$$\begin{array}{l}
\langle \mathsf{P}(x - \mathsf{s0}, \mathsf{s}y - \mathsf{ss}z), \mathsf{P}(y - \mathsf{s}z, x - \mathsf{s0})\rangle \\
\langle \mathsf{P}(\mathsf{s}x - \mathsf{s0}, y - \mathsf{ss}z), \mathsf{P}(y - \mathsf{ss}z, x - 0)\rangle.
\end{array}$$

Now the resulting constraints are again satisfied by the recursive path ordering if we use the AFS $x - y \to x$, $\mathsf{M}(x, y) \to x$.

5.1.40 Narrowing to approximate the dependency graph

Narrowing of dependency pairs may also be helpful in examples where the failure of the automation is due to our approximation of dependency graphs. For example, let us add the following second 'commutation' rule to the TRS from Ex. 5.1.39

$$(x + \mathsf{s}(0)) \ + \ (y + \mathsf{s}(\mathsf{s}(z))) \to (y + \mathsf{s}(\mathsf{s}(z))) \ + \ (x + \mathsf{s}(0)).$$

Now we obtain three additional dependency pairs.

$$\langle \mathsf{P}(x + \mathsf{s0}, y + \mathsf{ss}z), \mathsf{P}(y, \mathsf{ss}z)\rangle \tag{34}$$

$$\langle \mathsf{P}(x + \mathsf{s0}, y + \mathsf{ss}z), \mathsf{P}(x, \mathsf{s0})\rangle \tag{35}$$

$$\langle \mathsf{P}(x + \mathsf{s0}, y + \mathsf{ss}z), \mathsf{P}(y + \mathsf{ss}z, x + \mathsf{s0})\rangle \tag{36}$$

We have to compute a graph containing the dependency graph. For that purpose, we draw an arc from a dependency pair $\langle s,t\rangle$ to $\langle v,w\rangle$ whenever $\textsc{ren}(\textsc{cap}(t))$ and v are unifiable. However, for some examples this approximation is too rough.

Note that in our approximation of the dependency graph there would be an arc from (36) to itself, because after replacing $y + \mathsf{ss}z$ and $x + \mathsf{s0}$ by new variables, the right- and the left-hand side of (36) obviously unify. Hence, we have to demand that the dependency pair (36) is strictly decreasing, i.e.

$$\mathsf{P}(x + \mathsf{s0}, y + \mathsf{ss}z) \;>\; \mathsf{P}(y + \mathsf{ss}z, x + \mathsf{s0}).$$

But this constraint is not satisfied by any polynomial or any path ordering amenable to automation[9].

However, in the *real* dependency graph, there is no arc from (36) to itself, because there is no substitution σ such that $y + \mathsf{ss}z\sigma$ reduces to $x + \mathsf{s0}\sigma$. Hence, there is no cycle consisting of (36) only and therefore it is sufficient if (36) is just *weakly* decreasing. In this way, the constraints resulting from this example would again be satisfied by the recursive path ordering (after normalisation w.r.t. the AFS mentioned in Ex. 5.1.39).

Note that the narrowing refinement introduced in Sect. 2.5 also serves to compute a better approximation of the dependency graph. The right-hand side of (36) is linear and it does not unify with the left-hand side of any dependency pair. Hence, we may replace (36) by its narrowings:

$$\langle \mathsf{P}(x + \mathsf{s0}, \mathsf{0} + \mathsf{ss}z),\ \mathsf{P}(\mathsf{ss}z, x + \mathsf{s0})\rangle \tag{37}$$
$$\langle \mathsf{P}(x + \mathsf{s0}, \mathsf{s}y + \mathsf{ss}z),\ \mathsf{P}(\mathsf{s}(y + \mathsf{s}z), x + \mathsf{s0})\rangle \tag{38}$$
$$\langle \mathsf{P}(\mathsf{0} + \mathsf{s0}, y + \mathsf{ss}z),\ \mathsf{P}(y + \mathsf{ss}z, \mathsf{s0})\rangle \tag{39}$$
$$\langle \mathsf{P}(\mathsf{s}x + \mathsf{s0}, y + \mathsf{ss}z),\ \mathsf{P}(y + \mathsf{ss}z, \mathsf{s}(x + \mathsf{0}))\rangle. \tag{40}$$

Now the technique of connectable terms presented in Thm. 19 immediately proves that (37) - (40) are not on a cycle of the dependency graph, because application of $\textsc{ren}$ and $\textsc{cap}$ to their right-hand sides yields terms of the form $\mathsf{P}(\mathsf{s}(\ldots),\ldots)$ or $\mathsf{P}(\ldots,\mathsf{s}(\ldots))$ which do not unify with

$$\mathsf{P}(\ldots + \ldots\,,\,\ldots + \ldots).$$

[9]This inequality is not satisfied by any path ordering (that can be generated automatically), because neither a lexicographic comparison nor a comparison as multisets makes $(x + \mathsf{s0}, y + \mathsf{ss}z)$ greater than $(y + \mathsf{ss}z, x + \mathsf{s0})$. When using polynomial orderings, P is mapped to some polynomial p. Then we either have $\lim_{y\to\infty}(\,p(y,x) - p(x,y)\,) = \infty$ or $\lim_{y\to\infty}(\,p(y,x) - p(x,y)\,) = -\infty$. In the first case, $\mathsf{P}(y + \mathsf{ss}z, x + \mathsf{s0}) \;>\; \mathsf{P}(x + \mathsf{s0}, y + \mathsf{ss}z)$ holds for large enough y and in the second case $\mathsf{P}(y + \mathsf{ss}z, x + \mathsf{s0}) \;>\; \mathsf{P}(x + \mathsf{s0}, y + \mathsf{ss}z)$ holds for large enough x.

5.1.41 Factorial

The following non-simply terminating TRS for computing the factorial of a natural number (cf. [54, 60])

$$\begin{aligned} \mathsf{p}(\mathsf{s}(x)) &\to x \\ \mathsf{fac}(0) &\to \mathsf{s}(0) \\ \mathsf{fac}(\mathsf{s}(x)) &\to \mathsf{s}(x) \times \mathsf{fac}(\mathsf{p}(\mathsf{s}(x))) \end{aligned}$$

cannot be proved terminating automatically by the technique described in [2], since there narrowing dependency pairs was not considered. By using narrowing, the dependency pair

$$\langle \mathsf{FAC}(\mathsf{s}(x)), \mathsf{FAC}(\mathsf{p}(\mathsf{s}(x)))\rangle$$

is replaced by the dependency pair

$$\langle \mathsf{FAC}(\mathsf{s}(x)), \mathsf{FAC}(x)\rangle$$

resulting in inequalities which can easily be satisfied.

5.1.42 Binary numbers

The following non-simply terminating example is due to Geser [11, 54].

$$\begin{aligned} \mathsf{half}(0) &\to 0 \\ \mathsf{half}(\mathsf{s}(0)) &\to 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) &\to \mathsf{s}(\mathsf{half}(x)) \\ \mathsf{lastbit}(0) &\to 0 \\ \mathsf{lastbit}(\mathsf{s}(0)) &\to \mathsf{s}(0) \\ \mathsf{lastbit}(\mathsf{s}(\mathsf{s}(x))) &\to \mathsf{lastbit}(x) \\ \mathsf{conv}(0) &\to \mathsf{nil}.\, 0 \\ \mathsf{conv}(\mathsf{s}(x)) &\to \mathsf{conv}(\mathsf{half}(\mathsf{s}(x))).\, \mathsf{lastbit}(\mathsf{s}(x)) \end{aligned}$$

Narrowing the dependency pair $\langle \mathsf{CONV}(\mathsf{s}(x)), \mathsf{CONV}(\mathsf{half}(\mathsf{s}(x)))\rangle$ results in $\langle \mathsf{CONV}(\mathsf{s}(0)), \mathsf{CONV}(0)\rangle$ and $\langle \mathsf{CONV}(\mathsf{s}(\mathsf{s}(x))), \mathsf{CONV}(\mathsf{s}(\mathsf{half}(x)))\rangle$. After this replacement, the relevant inequalities are

$$\begin{aligned} \mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) &> \mathsf{HALF}(x) \\ \mathsf{LASTBIT}(\mathsf{s}(\mathsf{s}(x))) &> \mathsf{LASTBIT}(x) \\ \mathsf{CONV}(\mathsf{s}(\mathsf{s}(x))) &> \mathsf{CONV}(\mathsf{s}(\mathsf{half}(x))). \end{aligned}$$

After normalizing the inequalities w.r.t. the AFS $\mathsf{half}(x) \to x$, $x.\, y \to x$, the constraints are satisfied by the recursive path ordering.

5.1.43 Termination by narrowing, version 1

The following TRS by Plaisted [52, 54]

$$\begin{aligned} \mathsf{f}(\mathsf{c}) &\to \mathsf{g}(\mathsf{h}(\mathsf{c})) \\ \mathsf{h}(\mathsf{g}(x)) &\to \mathsf{g}(\mathsf{h}(\mathsf{f}(x))) \\ \mathsf{k}(x, \mathsf{h}(x), \mathsf{c}) &\to \mathsf{h}(x) \\ \mathsf{k}(\mathsf{f}(x), y, x) &\to \mathsf{f}(x) \end{aligned}$$

can automatically be proved terminating by only replacing the dependency pair $\langle \mathsf{H}(\mathsf{g}(x)), \mathsf{H}(\mathsf{f}(x)) \rangle$ by its narrowing $\langle \mathsf{H}(\mathsf{g}(\mathsf{c})), \mathsf{H}(\mathsf{g}(\mathsf{h}(\mathsf{c}))) \rangle$ and computing the dependency graph. As there is no cycle consisting of the resulting pairs, the TRS is terminating.

5.1.44 Termination by narrowing, version 2

To prove termination of the following TRS from Bachmair [9, 54]

$$\begin{aligned} \mathsf{f}(\mathsf{h}(x)) &\to \mathsf{f}(\mathsf{i}(x)) \\ \mathsf{g}(\mathsf{i}(x)) &\to \mathsf{g}(\mathsf{h}(x)) \\ \mathsf{h}(\mathsf{a}) &\to \mathsf{b} \\ \mathsf{i}(\mathsf{a}) &\to \mathsf{b} \end{aligned}$$

the dependency pairs

$$\begin{gathered} \langle \mathsf{F}(\mathsf{h}(x)), \mathsf{F}(\mathsf{i}(x)) \rangle \\ \langle \mathsf{G}(\mathsf{i}(x)), \mathsf{G}(\mathsf{h}(x)) \rangle \end{gathered}$$

are replaced by their narrowings

$$\begin{gathered} \langle \mathsf{F}(\mathsf{h}(\mathsf{a})), \mathsf{F}(\mathsf{b}) \rangle \\ \langle \mathsf{G}(\mathsf{i}(\mathsf{a})), \mathsf{G}(\mathsf{b}) \rangle. \end{gathered}$$

Then termination is automatically proved by the fact that the dependency graph has no cycles.

5.1.45 Termination by narrowing, version 3

For the following TRS we also need narrowing in order to prove its termination using a quasi-simplification ordering.

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

Narrowing the dependency pair $\langle \mathsf{H}(x.\,y), \mathsf{H}(\mathsf{g}(x.\,y)) \rangle$ results in

$$\begin{array}{l} \langle \mathsf{H}(0.\,y), \mathsf{H}(\mathsf{g}(y)) \rangle \\ \langle \mathsf{H}(\mathsf{s}(x).\,y), \mathsf{H}(\mathsf{s}(x)) \rangle. \end{array}$$

Now the relevant inequalities are

$$\begin{array}{rcl} \mathsf{F}(\mathsf{s}(x)) & > & \mathsf{F}(x) \\ \mathsf{G}(0.\,y) & > & \mathsf{G}(y) \\ \mathsf{H}(0.\,y) & > & \mathsf{H}(\mathsf{g}(y)). \end{array}$$

The resulting constraints are satisfied by the recursive path ordering, if they are normalized using the AFS $\mathsf{h}(x) \to \mathsf{h}'$.

5.1.46 A non-totally terminating TRS

The following example is from Steinbach [54].

$$\begin{array}{rcl} \mathsf{f}(x,x) & \to & \mathsf{f}(\mathsf{a},\mathsf{b}) \\ \mathsf{b} & \to & \mathsf{c} \end{array}$$

This TRS is not totally terminating and without using narrowing, the inequalities generated by our technique are not satisfied by any total well-founded weakly monotonic quasi-ordering. However, after applying one narrowing step to $\langle \mathsf{F}(x,x), \mathsf{F}(\mathsf{a},\mathsf{b}) \rangle$, the pair $\langle \mathsf{F}(x,x), \mathsf{F}(\mathsf{a},\mathsf{c}) \rangle$ is obtained, whose right-hand side is not unifiable with $\mathsf{F}(x,x)$. Hence, there is no cycle in the dependency graph. Thus, the TRS is terminating.

5.2 Examples for proving innermost termination

This section contains a collection of examples to demonstrate the use of the innermost termination technique presented in Sect. 3. The examples 5.2.1 – 5.2.18 are term rewriting systems that are innermost terminating, but not terminating. The remainder of the examples (5.2.19 – 5.2.29) are non-overlapping term rewriting systems for which innermost termination suffices to guarantee termination. Note that for the examples 5.2.14 – 5.2.18 and 5.2.24 – 5.2.29 we had to use refinements which were not included in the method of [3].

5.2.1 Toyama example

A famous example of a TRS that is innermost terminating, but not terminating, is the following system from Toyama [56].

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

This TRS has only one dependency pair, viz. $\langle \mathsf{F}(0,1,x), \mathsf{F}(x,x,x)\rangle$. This dependency pair does not occur on a cycle in the innermost dependency graph, since $\mathsf{F}(x_1,x_1,x_1)$ does not unify with $\mathsf{F}(0,1,x_2)$. Thus, no inequalities are generated and therefore the TRS is innermost terminating.

5.2.2 Variations on the Toyama example, version 1

The following example (Ex. 27) is a non-terminating TRS

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

with only one dependency pair on a cycle in the innermost dependency graph, viz. $\langle \mathsf{G}(\mathsf{s}(x)), \mathsf{G}(x)\rangle$. Since no defined symbols occur in $\mathsf{G}(x)$, there are no usable rules. Therefore, the only constraint on the ordering is given by

$$\mathsf{G}(\mathsf{s}(x)) \quad > \quad \mathsf{G}(x)$$

which is easily satisfied by the recursive path ordering. Hence, the TRS is innermost terminating.

5.2.3 Variations on the Toyama example, version 2

Similar to the preceding example, the following modification of the Toyama example

$$\begin{aligned} \mathsf{f}(\mathsf{g}(x,y), x, z) &\rightarrow \mathsf{f}(z,z,z) \\ \mathsf{g}(x,y) &\rightarrow x \\ \mathsf{g}(x,y) &\rightarrow y \end{aligned}$$

is not a constructor system, since the subterm $\mathsf{g}(x,y)$ occurs in the left-hand side of the first rule. Again the innermost dependency graph does not contain any cycles and hence, this TRS is innermost terminating. This TRS is, however, not terminating.

5.2.4 Variations on the Toyama example, version 3

The non-terminating TRS

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

is no constructor system either. The dependency pair $\langle \mathsf{F}(\mathsf{g}(x), x, y), \mathsf{F}(y, y, \mathsf{g}(y))\rangle$ cannot occur in an infinite innermost chain, since $\text{CAP}_{\mathsf{F}(\mathsf{g}(x_1), x_1, y_1)}(\mathsf{F}(y_1, y_1,$

$\mathsf{g}(y_1)))$ does not unify with $\mathsf{F}(\mathsf{g}(x_2), x_2, y_2)$. The dependency pair $\langle \mathsf{G}(\mathsf{g}(x)), \mathsf{G}(x)\rangle$ cannot occur in an infinite innermost chain either, since by unifying the right projection of this dependency pair with a renaming of it, the left projection is instantiated in such a way that it is not a normal form. Hence, there are no cycles in the innermost dependency graph and therefore the TRS is innermost terminating.

5.2.5 Redex in left-hand side

The following system (from Ex. 24)

$$\begin{array}{rcl} \mathsf{f}(\mathsf{0}) & \to & \mathsf{f}(\mathsf{0}) \\ \mathsf{0} & \to & \mathsf{1} \end{array}$$

is innermost terminating, because there is no cycle in the innermost dependency graph. The reason is that the left-hand side $\mathsf{F}(\mathsf{0})$ of the (only) dependency pair is not a normal form.

5.2.6 Narrowing required, version 1

In the following, again non-terminating, variant of the Toyama example

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

one narrowing step is needed to determine that there are no cycles in the innermost dependency graph (because $\langle \mathsf{F}(\mathsf{0}, \mathsf{1}, x), \mathsf{F}(\mathsf{g}(x, x), x, x)\rangle$ narrows to $\langle \mathsf{F}(\mathsf{0}, \mathsf{1}, x), \mathsf{F}(x, x, x)\rangle$). Thus, this TRS is also innermost terminating.

5.2.7 Narrowing required, version 2

The following example (from Ex. 24) can be solved in a similar way:

$$\begin{array}{rcl} \mathsf{f}(\mathsf{s}(x)) & \to & \mathsf{f}(\mathsf{g}(x, x)) \\ \mathsf{g}(\mathsf{0}, \mathsf{1}) & \to & \mathsf{s}(\mathsf{0}) \\ \mathsf{0} & \to & \mathsf{1}. \end{array}$$

The dependency pair $\langle \mathsf{F}(\mathsf{s}(x)), \mathsf{F}(\mathsf{g}(x, x))\rangle$ may be deleted as it cannot be narrowed. Hence, there is no dependency pair left and therefore, innermost termination is proved.

5.2.8 Narrowing required, version 3

Consider the following TRS

$$\begin{array}{rcl} x + \mathsf{0} & \to & x \\ x + \mathsf{s}(y) & \to & \mathsf{s}(x + y) \\ \mathsf{f}(\mathsf{0}, \mathsf{s}(\mathsf{0}), x) & \to & \mathsf{f}(x, x + x, x) \\ \mathsf{g}(x, y) & \to & x \\ \mathsf{g}(x, y) & \to & y \end{array}$$

which is not terminating as can be seen by the infinite reduction

$$\begin{array}{rcl} \mathsf{f}(\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0}))) & \to & \mathsf{f}(\mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})), \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})) + \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})), \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0}))) \\ & \to & \mathsf{f}(\mathsf{0}, \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})) + \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})), \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0}))) \\ & \to & \mathsf{f}(\mathsf{0}, \mathsf{s}(\mathsf{0}) + \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0})), \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0}))) \\ & \to & \mathsf{f}(\mathsf{0}, \mathsf{s}(\mathsf{0}) + \mathsf{0}, \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0}))) \\ & \to & \mathsf{f}(\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{g}(\mathsf{0}, \mathsf{s}(\mathsf{0}))) \\ & \to & \ldots \end{array}$$

Innermost termination of this TRS can be proved if the dependency pair $\langle \mathsf{F}(\mathsf{0}, \mathsf{s}(\mathsf{0}), x), \mathsf{F}(x, x + x, x) \rangle$ is replaced by its narrowings

$$\begin{array}{l} \langle \mathsf{F}(\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{0}), \mathsf{F}(\mathsf{0}, \mathsf{0}, \mathsf{0}) \rangle \\ \langle \mathsf{F}(\mathsf{0}, \mathsf{s}(\mathsf{0}), \mathsf{s}(y)), \mathsf{F}(\mathsf{s}(y), \mathsf{s}(\mathsf{s}(y) + y), \mathsf{s}(y))) \rangle. \end{array}$$

Now our approximation determines that these dependency pairs are not on cycles in the innermost dependency graph. Therefore, the only inequality generated for this TRS is

$$\mathsf{P}(x, \mathsf{s}(y)) > \mathsf{P}(x, y)$$

which is satisfied by the recursive path ordering. Hence, this TRS is proved innermost terminating.

5.2.9 Narrowing required, version 4

The following modification of the above TRS

$$\begin{array}{rcl} x + \mathsf{0} & \to & x \\ x + \mathsf{s}(y) & \to & \mathsf{s}(x + y) \\ \mathsf{double}(x) & \to & x + x \\ \mathsf{f}(\mathsf{0}, \mathsf{s}(\mathsf{0}), x) & \to & \mathsf{f}(x, \mathsf{double}(x), x) \\ \mathsf{g}(x, y) & \to & x \\ \mathsf{g}(x, y) & \to & y \end{array}$$

is also non-terminating. Similar to the example above, we now need two narrowing steps to derive that the narrowings of the dependency pair

$$\langle \mathsf{F}(0,\mathsf{s}(0),x), \mathsf{F}(x,\mathsf{double}(x),x)\rangle$$

do not occur on cycles in the innermost dependency graph. The generated inequality is therefore the same as for the above example, which is satisfied by the recursive path ordering. Hence, this TRS is proved innermost terminating.

5.2.10 Non-normal most general unifier

The following TRS (Ex. 38)

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

is obviously not terminating as $\mathsf{f}(1,\mathsf{g}(1))$ can be reduced to itself. The dependency pair

$$\langle \mathsf{F}(x,\mathsf{g}(x)), \mathsf{F}(1,\mathsf{g}(x))\rangle$$

does not occur on a cycle in the innermost dependency graph, because

$$\text{CAP}_{\mathsf{F}(x_1,\mathsf{g}(x_1))}(\mathsf{F}(1,\mathsf{g}(x_1))) = \mathsf{F}(1,\mathsf{g}(x_1))$$

and the most general unifier of $\mathsf{F}(1,\mathsf{g}(x_1))$ and $\mathsf{F}(x_2,\mathsf{g}(x_2))$ replaces x_1 and x_2 by 1. Hence, the instantiation of the left projection is not a normal form. Obviously, the other dependency pairs $\langle \mathsf{F}(x,\mathsf{g}(x)), \mathsf{G}(x)\rangle$ and $\langle \mathsf{G}(1), \mathsf{G}(0)\rangle$ cannot occur on cycles either. Thus, there are no cycles in the innermost dependency graph. Hence, the TRS is innermost terminating.

5.2.11 Innermost chains of arbitrary finite length

The following non-terminating TRS has an innermost chain of any finite length, but it has no infinite innermost chain, hence it is innermost terminating.

$$\begin{array}{rcl} \mathsf{h}(x,z) & \to & \mathsf{f}(x,\mathsf{s}(x),z) \\ \mathsf{f}(x,y,\mathsf{g}(x,y)) & \to & \mathsf{h}(0,\mathsf{g}(x,y)) \\ \mathsf{g}(0,y) & \to & 0 \\ \mathsf{g}(x,\mathsf{s}(y)) & \to & \mathsf{g}(x,y) \end{array}$$

An infinite reduction is given by

$$\mathsf{h}(0,\mathsf{g}(0,\mathsf{s}(0)) \to \mathsf{f}(0,\mathsf{s}(0),\mathsf{g}(0,\mathsf{s}(0))) \to \mathsf{h}(0,\mathsf{g}(0,\mathsf{s}(0)) \to \dots$$

So the TRS is not terminating.

The inequality resulting from the dependency pair on the only cycle in the innermost dependency graph is

$$\mathsf{G}(x, \mathsf{s}(y)) \quad > \quad \mathsf{G}(x, y).$$

(The reason is that the most general unifier of $\text{CAP}_{\mathsf{H}(x_1, z_1)}(\mathsf{F}(x_1, \mathsf{s}(x_1), z_1))$ and $\mathsf{F}(x_2, y_2, \mathsf{g}(x_2, y_2))$ does not instantiate the latter term to a normal form.)

There are no usable rules. Thus, innermost termination is easily proved by the recursive path ordering.

5.2.12 Negative coefficients

The following non-terminating TRS has two dependency pairs on a cycle in the innermost dependency graph, but it has no infinite innermost chain. Hence, it is innermost terminating.

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

An infinite reduction is given by

$$\begin{aligned}
\mathsf{f}(\mathsf{0}, \mathsf{1}, \mathsf{g}(\mathsf{0}, \mathsf{1})) &\;\to\; \mathsf{h}(\mathsf{g}(\mathsf{0}, \mathsf{1}), \mathsf{g}(\mathsf{0}, \mathsf{1})) \\
&\;\to\; \mathsf{h}(\mathsf{0}, \mathsf{g}(\mathsf{0}, \mathsf{1})) \\
&\;\to\; \mathsf{f}(\mathsf{0}, \mathsf{g}(\mathsf{0}, \mathsf{1}), \mathsf{g}(\mathsf{0}, \mathsf{1})) \\
&\;\to\; \mathsf{f}(\mathsf{0}, \mathsf{1}, \mathsf{g}(\mathsf{0}, \mathsf{1})) \to \ldots
\end{aligned}$$

The inequalities resulting from the dependency pairs on a cycle in the innermost dependency graph are

$$\begin{aligned}
\mathsf{H}(\mathsf{0}, x) &\;\geq\; \mathsf{F}(\mathsf{0}, x, x) \\
\mathsf{F}(\mathsf{0}, \mathsf{1}, x) &\;>\; \mathsf{H}(x, x)
\end{aligned}$$

and there are no usable rules. These inequalities are satisfied by the polynomial interpretation where $\mathsf{0}$ and $\mathsf{1}$ are interpreted as usual and where $\mathsf{H}(x, y)$ and $\mathsf{F}(x, y, z)$ are both mapped to $(x - y)^2$.

Note that the inequalities obtained in this example are not satisfied by any weakly monotonic total well-founded quasi-ordering. For that reason a polynomial ordering with negative coefficients has been used.

In a similar way one can also prove innermost termination of the system where the first rule has been changed to

$$\mathsf{h}(x, y) \to \mathsf{f}(x, y, x).$$

5.2.13 Drosten example

A confluent and innermost terminating TRS that is not terminating was given by Drosten [23].

$$
\begin{array}{rcl}
\mathsf{f}(\mathsf{0},\mathsf{1},x) & \to & \mathsf{f}(x,x,x) \\
\mathsf{f}(x,y,z) & \to & \mathsf{2} \\
\mathsf{0} & \to & \mathsf{2} \\
\mathsf{1} & \to & \mathsf{2} \\
\mathsf{g}(x,x,y) & \to & y \\
\mathsf{g}(x,y,y) & \to & x
\end{array}
$$

As there exists no cycle in the innermost dependency graph, the TRS is innermost terminating.

5.2.14 Better approximations of the innermost dependency graph, version 1

For the approximation of innermost dependency graphs we use the function $\textsc{cap}_s$ (instead of just the function $\textsc{cap}$). An example where this refinement is needed can be obtained from Ex. 5.2.2 by modification of the first rule.

$$
\begin{array}{rcl}
\mathsf{f}(\mathsf{g}(x),\mathsf{s}(\mathsf{0})) & \to & \mathsf{f}(\mathsf{g}(x),\mathsf{g}(x)) \\
\mathsf{g}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{g}(x)) \\
\mathsf{g}(\mathsf{0}) & \to & \mathsf{0}
\end{array}
$$

If we would approximate the innermost dependency graph by just using $\textsc{cap}$ then in our approximation we would draw an arc from the dependency pair

$$\langle \mathsf{F}(\mathsf{g}(x),\mathsf{s}(\mathsf{0}))\, \mathsf{F}(\mathsf{g}(x),\mathsf{g}(x))\rangle$$

to itself, because $\textsc{cap}(\mathsf{F}(\mathsf{g}(x),\mathsf{g}(x))) = \mathsf{F}(x_1,x_2)$ unifies with its left-hand side. But then we would have to demand that this dependency pair is strictly decreasing, i.e. $\mathsf{F}(\mathsf{g}(x),\mathsf{s}(\mathsf{0})) > \mathsf{F}(\mathsf{g}(x),\mathsf{g}(x))$. However, then the resulting constraints would imply

$$\mathsf{F}(\mathsf{gs0},\mathsf{s0}) > \mathsf{F}(\mathsf{gs0},\mathsf{gs0}) \geq \mathsf{F}(\mathsf{gs0},\mathsf{sg0}) \geq \mathsf{F}(\mathsf{gs0},\mathsf{s0}).$$

Hence, they would not be satisfied by any well-founded ordering closed under substitution. Therefore the approach of [3] would fail with this example.

However, by the refined approximation of using $\textsc{cap}_s$ we can immediately determine that this dependency pair is not on a cycle of the innermost dependency graph. The reason is that $\textsc{cap}_{\mathsf{F}(\mathsf{g}(x_1),\mathsf{s}(\mathsf{0}))}(\mathsf{F}(\mathsf{g}(x_1),\mathsf{g}(x_1)) = \mathsf{F}(\mathsf{g}(x_1),\mathsf{g}(x_1))$ does not unify with $\mathsf{F}(\mathsf{g}(x_2),\mathsf{s}(\mathsf{0}))$. (This example could also be solved by narrowing the dependency pair. But there are also examples where the innermost termination proof using $\textsc{cap}_s$ succeeds whereas it would not succeed when using narrowing and $\textsc{cap}$, cf. the next example, Ex. 5.2.15.) Now the only remaining constraint is

$$\mathsf{G}(\mathsf{s}(x)) > \mathsf{G}(x)$$

from the second rule of the TRS. For example, this constraint is satisfied by the recursive path ordering.

In a similar way we can also handle the following modification of Ex. 5.2.4:

$$\begin{array}{rcl} \mathsf{f}(\mathsf{g}(x), x) & \to & \mathsf{f}(\mathsf{g}(x), \mathsf{g}(x)) \\ \mathsf{g}(\mathsf{g}(x)) & \to & \mathsf{g}(x). \end{array}$$

5.2.15 Better approximations of the innermost dependency graph, version 2

This is a variation of the Toyama example, where the approximation using $\textsc{cap}_s$ is necessary to perform the innermost termination proof. In contrast to the preceding example, here narrowing the dependency pairs (and just using $\textsc{cap}$ instead of $\textsc{cap}_s$) would not help.

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

The dependency pair

$$\langle \mathsf{F}(0, 1, \mathsf{g}(x, y), z), \mathsf{F}(\mathsf{g}(x, y), \mathsf{g}(x, y), \mathsf{g}(x, y), \mathsf{h}(x)) \rangle$$

is not on a cycle of the innermost dependency graph. This can also be determined by our approximation, because $\textsc{cap}_{\mathsf{F}(0,1,\mathsf{g}(x,y),z)}(\mathsf{F}(\mathsf{g}(x, y), \mathsf{g}(x, y), \mathsf{g}(x, y), \mathsf{h}(x))) = \mathsf{F}(\mathsf{g}(x, y), \mathsf{g}(x, y), \mathsf{g}(x, y), \mathsf{h}(x))$ does not unify with $\mathsf{F}(0, 1, \ldots)$.

However, if we use just the approximation with $\textsc{cap}$, then we would have an arc from this dependency pair to itself. Now the resulting constraints would imply

$$\mathsf{F}(0, 1, \mathsf{g}(0, 1), \mathsf{h}(0)) > \mathsf{F}(\mathsf{g}(0, 1), \mathsf{g}(0, 1), \mathsf{g}(0, 1), \mathsf{h}(0)) \geq \mathsf{F}(0, 1, \mathsf{g}(0, 1), \mathsf{h}(0)).$$

Hence, they would not be satisfied by any well-founded ordering closed under substitution.

Note that in this example narrowing the dependency pair would not help, because the narrowings would include the pair

$$\langle \mathsf{F}(0, 1, \mathsf{g}(\mathsf{g}(x', y'), y), z),\ \mathsf{F}(\mathsf{g}(\mathsf{g}(x', y'), y), \mathsf{g}(\mathsf{g}(x', y'), y), \mathsf{g}(\mathsf{g}(x', y'), y), \mathsf{h}(x')) \rangle$$

which would lead to the same problem. (The same statement holds for repeated applications of narrowing.) Hence, this example demonstrates that we really need the refinement of $\textsc{cap}_s$ to approximate innermost dependency graphs.

5.2.16 Instantiation with Normal Form

The following TRS

$$\begin{array}{rcl} \mathsf{f}(\mathsf{s}(\mathsf{0}),\mathsf{g}(x)) & \to & \mathsf{f}(x,\mathsf{g}(x)) \\ \mathsf{g}(\mathsf{s}(x)) & \to & \mathsf{g}(x) \end{array}$$

is obviously not terminating as can be seen by the following infinite reduction

$$\mathsf{f}(\mathsf{s}(\mathsf{0}),\mathsf{g}(\mathsf{s}(\mathsf{0}))) \to \mathsf{f}(\mathsf{s}(\mathsf{0}),\mathsf{g}(\mathsf{s}(\mathsf{0}))) \to \ldots$$

The dependency pair

$$\langle \mathsf{F}(\mathsf{s}(\mathsf{0}),\mathsf{g}(x)), \mathsf{F}(x,\mathsf{g}(x)) \rangle$$

does not occur on a cycle of the innermost dependency graph, because $\textsc{cap}_{\mathsf{F}(\mathsf{s}(\mathsf{0}),\mathsf{g}(x_1))}(\mathsf{F}(x_1,\mathsf{g}(x_1)))$ and $\mathsf{F}(\mathsf{s}(\mathsf{0}),\mathsf{g}(x_2))$ unify using a most general unifier that instantiates $\mathsf{F}(\mathsf{s}(\mathsf{0}),\mathsf{g}(x_2))$ in such a way that it is not a normal form. (However, this would not have been determined by the approximation of innermost dependency graphs as presented in [3].) The only dependency pair that occurs on a cycle in the innermost dependency graph is $\langle \mathsf{G}(\mathsf{s}(x)), \mathsf{G}(x) \rangle$, resulting in the inequality

$$\mathsf{G}(\mathsf{s}(x)) \quad > \quad \mathsf{G}(x)$$

which is easily satisfied by the recursive path ordering.

5.2.17 Narrowing of pairs where right-hand sides unify with left-hand sides

In the following example we have to narrow a pair whose right-hand side unifies with a left-hand side of a dependency pair. When proving innermost termination, we may indeed perform this narrowing as long as the mgu does not instantiate the left-hand sides of the dependency pairs under consideration to normal forms.

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

The dependency pair

$$\langle \mathsf{F}(\mathsf{g}(x),\mathsf{s}(\mathsf{0}),y), \mathsf{F}(\mathsf{g}(\mathsf{s}(\mathsf{0})),y,\mathsf{g}(x)) \rangle$$

does not form a cycle in the innermost dependency graph, because an instantiation of its right-hand side can only reduce to an instantiation of its left-hand side where x is instantiated by $\mathsf{s}(\mathsf{0})$. But then this instantiated left-hand side would contain the redex $\mathsf{g}(\mathsf{s}(\mathsf{0}))$.

However, in our *approximation* there would be an arc from this dependency pair to itself, because $\text{CAP}_{\mathsf{F}(\mathsf{g}(x_1),\,\mathsf{s}(0),\,y_1)}(\mathsf{F}(\mathsf{g}(\mathsf{s}(0)), y_1, \mathsf{g}(x_1))) = \mathsf{F}(z, y_1, \mathsf{g}(x_1))$ unifies with $\mathsf{F}(\mathsf{g}(x_2), \mathsf{s}(0), y_2)$ (and the mgu instantiates the left-hand sides to normal forms). So one would have to demand that this dependency pair should be strictly decreasing, i.e. one would obtain the constraint $\mathsf{F}(\mathsf{g}(x), \mathsf{s}(0), y) > \mathsf{F}(\mathsf{g}(\mathsf{s}(0)), y, \mathsf{g}(x))$. However, together with the remaining constraints, this inequality is not satisfied by any well-founded ordering closed under substitution, because we would have

$$\begin{array}{rcl} \mathsf{F}(\mathsf{g}(\mathsf{s}(0)), \mathsf{s}(0), \mathsf{s}(0)) & > & \mathsf{F}(\mathsf{g}(\mathsf{s}(0)), \mathsf{s}(0), \mathsf{g}(\mathsf{s}(0))) \\ & \geq & \mathsf{F}(\mathsf{g}(\mathsf{s}(0)), \mathsf{s}(0), \mathsf{s}(\mathsf{g}(0))) \\ & \geq & \mathsf{F}(\mathsf{g}(\mathsf{s}(0)), \mathsf{s}(0), \mathsf{s}(0)). \end{array}$$

So we have to narrow this dependency pair. Note that the right-hand side unifies with the left-hand side of this dependency pair. However, the mgu instantiates the left-hand side to a term containing the redex $\mathsf{g}(\mathsf{s}(0))$. Hence, by Thm. 40 we may indeed replace this dependency pair by its narrowings.

$$\begin{array}{l} \langle \mathsf{F}(\mathsf{g}(x), \mathsf{s}(0), y), \mathsf{F}(\mathsf{s}(\mathsf{g}(0)), y, \mathsf{g}(x)) \rangle \\ \langle \mathsf{F}(\mathsf{g}(\mathsf{s}(x)), \mathsf{s}(0), y), \mathsf{F}(\mathsf{g}(\mathsf{s}(0)), y, \mathsf{s}(\mathsf{g}(x))) \rangle \\ \langle \mathsf{F}(\mathsf{g}(0), \mathsf{s}(0), y), \mathsf{F}(\mathsf{g}(\mathsf{s}(0)), y, 0) \rangle \end{array}$$

None of these new pairs is on a cycle of our approximated innermost dependency graph. Hence, the only constraint in this example is

$$\mathsf{G}(\mathsf{s}(x)) > \mathsf{G}(x)$$

from the second rule of the TRS. A well-founded ordering satisfying this constraint can of course be synthesized easily (e.g. the recursive path ordering).

5.2.18 Smallest normalizing non-terminating one-rule string rewriting system

The following example from Geser [29] is the smallest normalizing non-terminating one-rule string rewriting system.

$$\mathsf{a}(\mathsf{b}(\mathsf{a}(\mathsf{b}(x)))) \to \mathsf{b}(\mathsf{a}(\mathsf{b}(\mathsf{a}(\mathsf{a}(\mathsf{b}(x))))))$$

The dependency pairs in this example are

$$\begin{array}{l} \langle \mathsf{A}(\mathsf{b}(\mathsf{a}(\mathsf{b}(x)))), \mathsf{A}(\mathsf{b}(x)) \rangle \\ \langle \mathsf{A}(\mathsf{b}(\mathsf{a}(\mathsf{b}(x)))), \mathsf{A}(\mathsf{a}(\mathsf{b}(x))) \rangle \\ \langle \mathsf{A}(\mathsf{b}(\mathsf{a}(\mathsf{b}(x)))), \mathsf{A}(\mathsf{b}(\mathsf{a}(\mathsf{a}(\mathsf{b}(x))))) \rangle. \end{array}$$

The second and the third dependency pair can be narrowed to

$$\langle \mathsf{A}(\mathsf{b}(\mathsf{a}(\mathsf{b}(\mathsf{a}(\mathsf{b}(x)))))), \mathsf{A}(\mathsf{b}(\mathsf{a}(\mathsf{b}(\mathsf{a}(\mathsf{a}(\mathsf{b}(x)))))))\rangle$$
$$\langle \mathsf{A}(\mathsf{b}(\mathsf{a}(\mathsf{b}(\mathsf{a}(\mathsf{b}(x)))))), \mathsf{A}(\mathsf{b}(\mathsf{a}(\mathsf{b}(\mathsf{a}(\mathsf{b}(\mathsf{a}(\mathsf{a}(\mathsf{b}(x)))))))))\rangle.$$

These dependency pairs are not on cycles of the innermost dependency graph, because their left-hand sides contain redeces. Hence, the only constraint in this example is

$$\mathsf{A}(\mathsf{b}(\mathsf{a}(\mathsf{b}(x)))) > \mathsf{A}(\mathsf{b}(x))$$

which is satisfied by the recursive path ordering.

5.2.19 Another division example, version 1

The TRS of Ex. 33

$$\begin{aligned}
\mathsf{quot}(\mathsf{0}, \mathsf{s}(y), \mathsf{s}(z)) &\rightarrow \mathsf{0}\\
\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y), z) &\rightarrow \mathsf{quot}(x, y, z)\\
\mathsf{quot}(x, \mathsf{0}, \mathsf{s}(z)) &\rightarrow \mathsf{s}(\mathsf{quot}(x, \mathsf{s}(z), \mathsf{s}(z)))
\end{aligned}$$

is a non-simply terminating system. As explained in Sect. 3.2 this TRS cannot be proved terminating automatically by the technique of Sect. 2. The only two generated inequalities are

$$\begin{aligned}
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y), z) &> \mathsf{Q}(x, y, z)\\
\mathsf{Q}(x, \mathsf{0}, \mathsf{s}(z)) &\geq \mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z)),
\end{aligned}$$

since there are no usable rules. By normalizing the inequalities with respect to the AFS $\mathsf{Q}(x, y, z) \rightarrow x$, the obtained inequalities are satisfied by the recursive path ordering. Thus, the TRS is innermost terminating. Termination of the TRS can now be concluded from the fact that it is non-overlapping.

5.2.20 Narrowing to approximate the innermost dependency graph

Similar to Ex. 5.1.40, narrowing of pairs also helps to obtain a better approximation of the *innermost* dependency graph. To illustrate this, let us replace the last rule of the TRS in Ex. 5.2.19 by the following three rules.

$$\begin{aligned}
\mathsf{0} + y &\rightarrow y\\
\mathsf{s}(x) + y &\rightarrow \mathsf{s}(x + y)\\
\mathsf{quot}(x, \mathsf{0}, \mathsf{s}(z)) &\rightarrow \mathsf{s}(\mathsf{quot}(x, z + \mathsf{s}(\mathsf{0}), \mathsf{s}(z)))
\end{aligned}$$

Now instead of dependency pair

$$\langle \mathsf{Q}(x, \mathsf{0}, \mathsf{s}(z)), \mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z))\rangle \qquad (41)$$

we obtain the dependency pair

$$\langle \mathsf{Q}(x, \mathsf{0}, \mathsf{s}(z)), \mathsf{Q}(x, z + \mathsf{s}(\mathsf{0}), \mathsf{s}(z))\rangle. \tag{42}$$

Note that in the approximation of the innermost dependency graph there would be an arc from (42) to itself, because after replacing $z + \mathsf{s}(\mathsf{0})$ by a new variable, the right- and the left-hand side of (42) obviously unify (and an instantiation with the mgu is a normal form). Hence, due to Thm. 35 we would have to find an ordering such that (42) is strictly decreasing. But then no linear or weakly monotonic polynomial ordering satisfies all resulting inequalities in this example (and the recursive path ordering does not succeed either).

However, in the *real* innermost dependency graph, there is no arc from (42) to itself, because, similar to the original dependency pair (41), there is no substitution σ such that $(z + \mathsf{s}(\mathsf{0}))\sigma$ reduces to $\mathsf{0}$. Hence, there is no cycle consisting of (42) only and therefore it is sufficient if (42) is just *weakly* decreasing. For this reason we replace the dependency pair (42) by its narrowings, viz.

$$\langle \mathsf{Q}(x, \mathsf{0}, \mathsf{s}(\mathsf{0})), \mathsf{Q}(x, \mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{0}))\rangle \tag{43}$$

$$\langle \mathsf{Q}(x, \mathsf{0}, \mathsf{s}(\mathsf{s}(z))), \mathsf{Q}(x, \mathsf{s}(z + \mathsf{s}(\mathsf{0})), \mathsf{s}(\mathsf{0}))\rangle, \tag{44}$$

and compute the innermost dependency graph afterwards. Now neither (43) nor (44) are innermost connectable to themselves. Hence, if in our example we perform at least one narrowing step, then we can determine that the dependency pair (42) does not form a cycle in the innermost dependency graph and then termination can again be verified using the recursive path ordering.

5.2.21 Selection sort

This TRS from Walther [58] is obviously not simply terminating. The TRS can be used to sort a list by repeatedly replacing the minimum of the list by the head of the list. It uses $\mathsf{replace}(n, m, x)$ to replace the leftmost occurrence of n in the list x by m.

$$\begin{array}{rcl}
\mathsf{eq}(\mathsf{0}, \mathsf{0}) & \to & \mathsf{true} \\
\mathsf{eq}(\mathsf{0}, \mathsf{s}(x)) & \to & \mathsf{false} \\
\mathsf{eq}(\mathsf{s}(x), \mathsf{0}) & \to & \mathsf{false} \\
\mathsf{eq}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{eq}(x, y) \\
\mathsf{le}(\mathsf{0}, y) & \to & \mathsf{true} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{0}) & \to & \mathsf{false} \\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{le}(x, y) \\
\mathsf{min}(\mathsf{add}(n, \mathsf{nil})) & \to & n \\
\mathsf{min}(\mathsf{add}(n, \mathsf{add}(m, x))) & \to & \mathsf{if}_{\mathsf{min}}(\mathsf{le}(n, m), \mathsf{add}(n, \mathsf{add}(m, x))) \\
\mathsf{if}_{\mathsf{min}}(\mathsf{true}, \mathsf{add}(n, \mathsf{add}(m, x))) & \to & \mathsf{min}(\mathsf{add}(n, x))
\end{array}$$

$$
\begin{aligned}
\mathsf{if}_{\mathsf{min}}(\mathsf{false}, \mathsf{add}(n, \mathsf{add}(m, x))) &\to \mathsf{min}(\mathsf{add}(m, x)) \\
\mathsf{replace}(n, m, \mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{replace}(n, m, \mathsf{add}(k, x)) &\to \mathsf{if}_{\mathsf{replace}}(\mathsf{eq}(n, k), n, m, \mathsf{add}(k, x)) \\
\mathsf{if}_{\mathsf{replace}}(\mathsf{true}, n, m, \mathsf{add}(k, x)) &\to \mathsf{add}(m, x) \\
\mathsf{if}_{\mathsf{replace}}(\mathsf{false}, n, m, \mathsf{add}(k, x)) &\to \mathsf{add}(k, \mathsf{replace}(n, m, x)) \\
\mathsf{selsort}(\mathsf{nil}) &\to \mathsf{nil} \\
\mathsf{selsort}(\mathsf{add}(n, x)) &\to \mathsf{if}_{\mathsf{selsort}}(\mathsf{eq}(n, \mathsf{min}(\mathsf{add}(n, x))), \mathsf{add}(n, x)) \\
\mathsf{if}_{\mathsf{selsort}}(\mathsf{true}, \mathsf{add}(n, x)) &\to \mathsf{add}(n, \mathsf{selsort}(x)) \\
\mathsf{if}_{\mathsf{selsort}}(\mathsf{false}, \mathsf{add}(n, x)) &\to \mathsf{add}(\mathsf{min}(\mathsf{add}(n, x)), \\
&\qquad \mathsf{selsort}(\mathsf{replace}(\mathsf{min}(\mathsf{add}(n, x)), n, x)))
\end{aligned}
$$

The relevant inequalities are

$$
\begin{aligned}
\mathsf{EQ}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{EQ}(x, y) \\
\mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{LE}(x, y) \\
\mathsf{MIN}(\mathsf{add}(n, \mathsf{add}(m, x))) &\geq \mathsf{IF}_{\mathsf{min}}(\mathsf{le}(n, m), \mathsf{add}(n, \mathsf{add}(m, x))) \\
\mathsf{IF}_{\mathsf{min}}(\mathsf{true}, \mathsf{add}(n, \mathsf{add}(m, x))) &> \mathsf{MIN}(\mathsf{add}(n, x)) \\
\mathsf{IF}_{\mathsf{min}}(\mathsf{false}, \mathsf{add}(n, \mathsf{add}(m, x))) &> \mathsf{MIN}(\mathsf{add}(m, x)) \\
\mathsf{REPLACE}(n, m, \mathsf{add}(k, x)) &\geq \mathsf{IF}_{\mathsf{replace}}(\mathsf{eq}(n, k), n, m, \mathsf{add}(k, x)) \\
\mathsf{IF}_{\mathsf{replace}}(\mathsf{false}, n, m, \mathsf{add}(k, x)) &> \mathsf{REPLACE}(n, m, x) \\
\mathsf{SELSORT}(\mathsf{add}(n, x)) &\geq \mathsf{IF}_{\mathsf{selsort}}(\mathsf{eq}(n, \mathsf{min}(\mathsf{add}(n, x))), \mathsf{add}(n, x)) \\
\mathsf{IF}_{\mathsf{selsort}}(\mathsf{true}, \mathsf{add}(n, x)) &> \mathsf{SELSORT}(x) \\
\mathsf{IF}_{\mathsf{selsort}}(\mathsf{false}, \mathsf{add}(n, x)) &> \mathsf{SELSORT}(\mathsf{replace}(\mathsf{min}(\mathsf{add}(n, x)), n, x)).
\end{aligned}
$$

Moreover, all rules except the four last ones are usable. The resulting constraints are satisfied by the polynomial ordering, where $\mathsf{eq}(x, y)$, $\mathsf{0}$, true, false, $\mathsf{le}(x, y)$, and nil are mapped to 0, $\mathsf{s}(x)$ is mapped to $x+1$, $\mathsf{add}(n, x)$ is mapped to $n+x+1$, $\mathsf{min}(x)$ and $\mathsf{if}_{\mathsf{min}}(b, x)$ are mapped to x, $\mathsf{replace}(n, m, x)$ and $\mathsf{if}_{\mathsf{replace}}(b, n, m, x)$ are mapped to $m+x$, $\mathsf{EQ}(x, y)$, $\mathsf{LE}(x, y)$, $\mathsf{MIN}(x)$, $\mathsf{IF}_{\mathsf{min}}(b, x)$, $\mathsf{SELSORT}(x)$, and $\mathsf{IF}_{\mathsf{selsort}}(b, x)$ are mapped to x, and $\mathsf{REPLACE}(n, m, x)$ and $\mathsf{IF}_{\mathsf{replace}}(b, n, m, x)$ are mapped to $m + x$. Hence, as the TRS is non-overlapping, in this way its termination is also proved. (If the first min rule would be replaced by $\mathsf{min}(\mathsf{add}(n, \mathsf{nil})) \to \mathsf{element}(n)$, then termination could also be proved by the termination technique of Sect. 2 using an appropriate AFS and the recursive path ordering to satisfy the constraints obtained.)

5.2.22 Intervals of Natural Numbers

The following TRS from Steinbach [54]

$$
\begin{aligned}
\mathsf{intlist}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{intlist}(x.\,y) &\to \mathsf{s}(x).\,\mathsf{intlist}(y)\\
\mathsf{int}(\mathsf{0},\mathsf{0}) &\to \mathsf{0}.\,\mathsf{nil}\\
\mathsf{int}(\mathsf{0},\mathsf{s}(y)) &\to \mathsf{0}.\,\mathsf{int}(\mathsf{s}(\mathsf{0}),\mathsf{s}(y))\\
\mathsf{int}(\mathsf{s}(x),\mathsf{0}) &\to \mathsf{nil}\\
\mathsf{int}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{intlist}(\mathsf{int}(x,y))
\end{aligned}
$$

is non-overlapping, too. The set of usable rules is empty and the generated inequalities are

$$
\begin{aligned}
\mathsf{INTLIST}(x.\,y) &> \mathsf{INTLIST}(y)\\
\mathsf{INT}(\mathsf{0},\mathsf{s}(y)) &\geq \mathsf{INT}(\mathsf{s}(\mathsf{0}),\mathsf{s}(y))\\
\mathsf{INT}(\mathsf{s}(x),\mathsf{s}(y)) &> \mathsf{INT}(x,y).
\end{aligned}
$$

By using the AFS $\mathsf{INT}(x,y) \to y$ these inequalities are satisfied by the recursive path ordering. Thus, the TRS is terminating. Again, termination of this system cannot be proved automatically using the method of Sect. 2.

5.2.23 Another non-totally terminating TRS

To prove termination of the system

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

we apply narrowing on the dependency pair $\langle \mathsf{F}(x,x), \mathsf{F}(\mathsf{g}(x),x)\rangle$. In this way we can directly determine that the innermost dependency graph does not contain any cycles.

5.2.24 Narrowing of dependency pairs for innermost termination

In the following example (Ex. 39) we have to apply narrowing of dependency pairs.

$$
\begin{aligned}
\mathsf{p}(\mathsf{0}) &\to \mathsf{0}\\
\mathsf{p}(\mathsf{s}(x)) &\to x\\
\mathsf{le}(\mathsf{0},y) &\to \mathsf{true}\\
\mathsf{le}(\mathsf{s}(x),\mathsf{0}) &\to \mathsf{false}\\
\mathsf{le}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{le}(x,y)\\
\mathsf{minus}(x,y) &\to \mathsf{if}(\mathsf{le}(x,y),x,y)\\
\mathsf{if}(\mathsf{true},x,y) &\to \mathsf{0}\\
\mathsf{if}(\mathsf{false},x,y) &\to \mathsf{s}(\mathsf{minus}(\mathsf{p}(x),y))
\end{aligned}
$$

Note that without narrowing, the resulting constraints would imply $\mathsf{M}(\mathsf{s}(x), \mathsf{0}) > \mathsf{M}(\mathsf{p}(\mathsf{s}(x)), \mathsf{0})$. Therefore an automatic innermost termination proof using quasi-simplification orderings fails.

However, if we replace the dependency pair $\langle \mathsf{M}(x, y), \mathsf{IF}(\mathsf{le}(x, y), x, y)\rangle$ by its narrowings

$$\begin{aligned}
&\langle \mathsf{M}(\mathsf{0}, y), \mathsf{IF}(\mathsf{true}, \mathsf{0}, y)\rangle,\\
&\langle \mathsf{M}(\mathsf{s}(x), \mathsf{0}), \mathsf{IF}(\mathsf{false}, \mathsf{s}(x), \mathsf{0})\rangle,\\
&\langle \mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{IF}(\mathsf{le}(x, y), \mathsf{s}(x), \mathsf{s}(y))\rangle
\end{aligned}$$

then this also enables a narrowing of the dependency pair $\langle \mathsf{IF}(\mathsf{false}, x, y), \mathsf{M}(\mathsf{p}(x), y)\rangle$ (whose right-hand side unified with a left-hand side before). Hence, now this dependency pair can be replaced by

$$\begin{aligned}
&\langle \mathsf{IF}(\mathsf{false}, \mathsf{0}, y), \mathsf{M}(\mathsf{0}, y)\rangle,\\
&\langle \mathsf{IF}(\mathsf{false}, \mathsf{s}(x), y), \mathsf{M}(x, y)\rangle.
\end{aligned}$$

Note that the first narrowing step would not have been possible with the method of Sect. 2, because the right-hand side is not linear. The relevant inequalities are

$$\begin{aligned}
\mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) &> \mathsf{LE}(x, y)\\
\mathsf{M}(\mathsf{s}(x), \mathsf{0}) &\geq \mathsf{IF}(\mathsf{false}, \mathsf{s}(x), \mathsf{0})\\
\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) &\geq \mathsf{IF}(\mathsf{le}(x, y), \mathsf{s}(x), \mathsf{s}(y))\\
\mathsf{IF}(\mathsf{false}, \mathsf{s}(x), y) &> \mathsf{M}(x, y).
\end{aligned}$$

Using the AFS $\mathsf{IF}(b, x, y) \to \mathsf{I}(x, y)$, the resulting constraints are satisfied by the recursive path ordering. As the TRS is non-overlapping, in this way we have also proved its termination.

5.2.25 Subtraction and predecessor

The following system is an alternative way to define subtraction using the predecessor function. Again this TRS is terminating, but not simply terminating.

$$\begin{aligned}
\mathsf{p}(\mathsf{0}) &\to \mathsf{0}\\
\mathsf{p}(\mathsf{s}(x)) &\to x\\
\mathsf{le}(\mathsf{0}, y) &\to \mathsf{true}\\
\mathsf{le}(\mathsf{s}(x), \mathsf{0}) &\to \mathsf{false}\\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{le}(x, y)\\
\mathsf{minus}(x, \mathsf{0}) &\to x\\
\mathsf{minus}(x, \mathsf{s}(y)) &\to \mathsf{if}(\mathsf{le}(x, \mathsf{s}(y)), \mathsf{0}, \mathsf{p}(\mathsf{minus}(x, \mathsf{p}(\mathsf{s}(y)))))\\
\mathsf{if}(\mathsf{true}, x, y) &\to x\\
\mathsf{if}(\mathsf{false}, x, y) &\to y
\end{aligned}$$

If we narrow the dependency pair $\langle \mathsf{M}(x, \mathsf{s}(y)), \mathsf{M}(x, \mathsf{p}(\mathsf{s}(y)))\rangle$, then we obtain the new pair $\langle \mathsf{M}(x, \mathsf{s}(y)), \mathsf{M}(x, y)\rangle$. Now (as there are no usable rules any more) the only constraints are

$$\begin{array}{rcl} \mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{LE}(x, y) \\ \mathsf{M}(x, \mathsf{s}(y)) & > & \mathsf{M}(x, y), \end{array}$$

which are satisfied by the recursive path ordering. Hence, innermost termination (and thereby, termination) has been proved, as the TRS is non-overlapping.

A similar example was mentioned by Steinbach [54], but there the rules for le and if were missing.

5.2.26 Length of bit representation

The following non-simply terminating TRS corresponds to the logarithm example (Ex. 5.1.7). Here, $\mathsf{bits}(x)$ computes the number of bits that are necessary to represent all numbers smaller than or equal to x.

$$\begin{array}{rcl} \mathsf{half}(0) & \to & 0 \\ \mathsf{half}(\mathsf{s}(0)) & \to & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{s}(\mathsf{half}(x)) \\ \mathsf{bits}(0)) & \to & 0 \\ \mathsf{bits}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \end{array}$$

After narrowing the dependency pair $\langle \mathsf{BITS}(\mathsf{s}(x)), \mathsf{BITS}(\mathsf{half}(\mathsf{s}(x)))\rangle$ to $\langle \mathsf{BITS}(\mathsf{s}(0)), \mathsf{BITS}(0)\rangle$ and $\langle \mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))), \mathsf{BITS}(\mathsf{s}(\mathsf{half}(x))\rangle$ we obtain the relevant inequalities

$$\begin{array}{rcl} \mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) & > & \mathsf{HALF}(x) \\ \mathsf{BITS}(\mathsf{s}(\mathsf{s}(x))) & > & \mathsf{BITS}(\mathsf{s}(\mathsf{half}(x)). \end{array}$$

The resulting constraints are satisfied by the recursive path ordering.

5.2.27 Multiplication for even and odd numbers

The following non-simply terminating example is inspired by Walther [57].

$$\begin{array}{rcl} \mathsf{even}(0) & \to & \mathsf{true} \\ \mathsf{even}(\mathsf{s}(0)) & \to & \mathsf{false} \\ \mathsf{even}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{even}(x) \\ \mathsf{half}(0) & \to & 0 \\ \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{s}(\mathsf{half}(x)) \\ \mathsf{plus}(0, y) & \to & y \end{array}$$

$$\begin{aligned}
\mathsf{plus}(\mathsf{s}(x), y) &\rightarrow \mathsf{s}(\mathsf{plus}(x, y))\\
\mathsf{times}(\mathsf{0}, y) &\rightarrow \mathsf{0}\\
\mathsf{times}(\mathsf{s}(x), y) &\rightarrow \mathsf{if}_{\mathsf{times}}(\mathsf{even}(\mathsf{s}(x)), \mathsf{s}(x), y)\\
\mathsf{if}_{\mathsf{times}}(\mathsf{true}, \mathsf{s}(x), y) &\rightarrow \mathsf{plus}(\mathsf{times}(\mathsf{half}(\mathsf{s}(x)), y), \mathsf{times}(\mathsf{half}(\mathsf{s}(x)), y))\\
\mathsf{if}_{\mathsf{times}}(\mathsf{false}, \mathsf{s}(x), y) &\rightarrow \mathsf{plus}(y, \mathsf{times}(x, y))
\end{aligned}$$

To prove termination using a quasi-simplification ordering, we have to narrow the dependency pair $\langle \mathsf{IF}_{\mathsf{times}}(\mathsf{true}, \mathsf{s}(x), y), \mathsf{TIMES}(\mathsf{half}(\mathsf{s}(x)), y)\rangle$ to

$$\langle \mathsf{IF}_{\mathsf{times}}(\mathsf{true}, \mathsf{s}(\mathsf{s}(x)), y), \mathsf{TIMES}(\mathsf{s}(\mathsf{half}(x)), y)\rangle.$$

Now the relevant inequalities are the following.

$$\begin{aligned}
\mathsf{EVEN}(\mathsf{s}(\mathsf{s}(x))) &> \mathsf{EVEN}(x)\\
\mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) &> \mathsf{HALF}(x)\\
\mathsf{P}(\mathsf{s}(x), y) &> \mathsf{P}(x, y)\\
\mathsf{TIMES}(\mathsf{s}(x), y) &\geq \mathsf{IF}_{\mathsf{times}}(\mathsf{even}(\mathsf{s}(x)), \mathsf{s}(x), y)\\
\mathsf{IF}_{\mathsf{times}}(\mathsf{true}, \mathsf{s}(\mathsf{s}(x)), y) &> \mathsf{TIMES}(\mathsf{s}(\mathsf{half}(x)), y)\\
\mathsf{IF}_{\mathsf{times}}(\mathsf{false}, \mathsf{s}(x), y) &> \mathsf{TIMES}(x, y)
\end{aligned}$$

If an AFS $\mathsf{IF}_{\mathsf{times}}(b, x, y) \rightarrow \mathsf{I}_{\mathsf{times}}(x, y)$ is used, then the resulting constraints are satisfied by the recursive path ordering.

5.2.28 Narrowing for division, remainder, and gcd

The TRSs for division (Ex. 5.1.1–5.1.3) can also be transformed into systems where we need narrowing for the (innermost) termination proof. We only present one of them.

$$\begin{aligned}
\mathsf{minus}(x, \mathsf{0}) &\rightarrow x\\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) &\rightarrow \mathsf{minus}(x, y)\\
\mathsf{le}(\mathsf{0}, y) &\rightarrow \mathsf{true}\\
\mathsf{le}(\mathsf{s}(x), \mathsf{0}) &\rightarrow \mathsf{false}\\
\mathsf{le}(\mathsf{s}(x), \mathsf{s}(y)) &\rightarrow \mathsf{le}(x, y)\\
\mathsf{quot}(x, \mathsf{s}(y)) &\rightarrow \mathsf{if}_{\mathsf{quot}}(\mathsf{le}(\mathsf{s}(y), x), x, \mathsf{s}(y))\\
\mathsf{if}_{\mathsf{quot}}(\mathsf{true}, x, y) &\rightarrow \mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), y))\\
\mathsf{if}_{\mathsf{quot}}(\mathsf{false}, x, y) &\rightarrow \mathsf{0}
\end{aligned}$$

Again this system is not simply terminating. After narrowing the dependency pair $\langle \mathsf{Q}(x, \mathsf{s}(y)), \mathsf{IF}_{\mathsf{quot}}(\mathsf{le}(\mathsf{s}(y), x), x, \mathsf{s}(y))\rangle$ to

$$\begin{gathered}
\langle \mathsf{Q}(\mathsf{0}, \mathsf{s}(y)), \mathsf{IF}_{\mathsf{quot}}(\mathsf{false}, \mathsf{0}, \mathsf{s}(y))\rangle\\
\langle \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{IF}_{\mathsf{quot}}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y))\rangle
\end{gathered}$$

we can narrow $\langle \mathsf{IF}_{\mathsf{quot}}(\mathsf{true}, x, y), \mathsf{Q}(\mathsf{minus}(x, y), y)\rangle$ to

$$\begin{array}{l}\langle \mathsf{IF}_{\mathsf{quot}}(\mathsf{true}, x, \mathsf{0}), \mathsf{Q}(x, y)\rangle \\ \langle \mathsf{IF}_{\mathsf{quot}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)), \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{s}(y))\rangle.\end{array}$$

Now the relevant inequalities are

$$\begin{array}{rcl}\mathsf{M}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{M}(x, y) \\ \mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{LE}(x, y) \\ \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) & \geq & \mathsf{IF}_{\mathsf{quot}}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \\ \mathsf{IF}_{\mathsf{quot}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) & > & \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{s}(y)).\end{array}$$

Using the AFS $\mathsf{minus}(x, y) \to x, \mathsf{IF}(b, x, y) \to \mathsf{I}(x, y)$ the constraints are satisfied by the recursive path ordering. Hence, in this way (innermost) termination of this TRS is proved.

A simpler modification of the quotient TRS where one should also use narrowing is obtained if instead of the last three rules the following rules are used.

$$\begin{array}{rcl}\mathsf{quot}(\mathsf{0}, \mathsf{s}(y)) & \to & \mathsf{0} \\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y)) & \to & \mathsf{s}(\mathsf{quot}(\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y)))\end{array}$$

A similar modification is also possible for the remainder TRSs (Ex. 5.1.5), i.e. the rule $\mathsf{if}_{\mathsf{mod}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{mod}(\mathsf{minus}(x, y), \mathsf{s}(y))$ may be replaced by

$$\mathsf{if}_{\mathsf{mod}}(\mathsf{true}, x, y) \to \mathsf{mod}(\mathsf{minus}(x, y), y).$$

In an analogous way, in the greatest common divisor TRSs (Ex. 5.1.6) one could also replace the last two rules by

$$\begin{array}{rcl}\mathsf{if}_{\mathsf{gcd}}(\mathsf{true}, x, y) & \to & \mathsf{gcd}(\mathsf{minus}(x, y), y) \\ \mathsf{if}_{\mathsf{gcd}}(\mathsf{false}, x, y) & \to & \mathsf{gcd}(\mathsf{minus}(y, x), x).\end{array}$$

All these modified TRSs could again be proved (innermost) terminating by using narrowing first.

5.2.29 Braid problem

The following string rewriting system (which encodes a braid problem from topology) was given by Zantema as a challenge during the 3rd International Termination Workshop. As shown by Geser, it is not simply terminating.

$$\begin{array}{rcl}\mathsf{a}(\mathsf{d}(x)) & \to & \mathsf{d}(\mathsf{c}(\mathsf{b}(\mathsf{a}(x)))) \\ \mathsf{b}(\mathsf{c}(x)) & \to & \mathsf{c}(\mathsf{d}(\mathsf{a}(\mathsf{b}(x)))) \\ \mathsf{a}(\mathsf{c}(x)) & \to & x \\ \mathsf{b}(\mathsf{d}(x)) & \to & x\end{array}$$

The dependency pairs in this example are

$$\langle \mathsf{A}(\mathsf{d}(x)), \mathsf{A}(x)\rangle \tag{45}$$
$$\langle \mathsf{A}(\mathsf{d}(x)), \mathsf{B}(\mathsf{a}(x))\rangle \tag{46}$$
$$\langle \mathsf{B}(\mathsf{c}(x)), \mathsf{B}(x)\rangle \tag{47}$$
$$\langle \mathsf{B}(\mathsf{c}(x)), \mathsf{A}(\mathsf{b}(x))\rangle. \tag{48}$$

Dependency pair (46) can be replaced by its narrowings

$$\langle \mathsf{A}(\mathsf{d}(\mathsf{d}(x))), \mathsf{B}(\mathsf{d}(\mathsf{c}(\mathsf{b}(\mathsf{a}(x)))))\rangle$$
$$\langle \mathsf{A}(\mathsf{d}(\mathsf{c}(x))), \mathsf{B}(x)\rangle$$

and dependency pair (48) can be narrowed to

$$\langle \mathsf{B}(\mathsf{c}(\mathsf{c}(x))), \mathsf{A}(\mathsf{c}(\mathsf{d}(\mathsf{a}(\mathsf{b}(x)))))\rangle$$
$$\langle \mathsf{B}(\mathsf{c}(\mathsf{d}(x))), \mathsf{A}(x)\rangle.$$

As there are no usable rules, the resulting constraints are

$$\begin{aligned} \mathsf{A}(\mathsf{d}(x)) &> \mathsf{A}(x) \\ \mathsf{A}(\mathsf{d}(\mathsf{c}(x))) &\geq \mathsf{B}(x) \\ \mathsf{B}(\mathsf{c}(x)) &> \mathsf{B}(x) \\ \mathsf{B}(\mathsf{c}(\mathsf{d}(x))) &\geq \mathsf{A}(x), \end{aligned}$$

which are satisfied by the recursive path ordering. Hence, as the TRS is non-overlapping, its termination is proved.

Acknowledgement

We would like to thank Hans Zantema, Aart Middeldorp, Thomas Kolbe, and Bernhard Gramlich for constructive criticism and many helpful comments. This work was partially supported by the Deutsche Forschungsgemeinschaft under grants no. Wa 652/7-1,2 as part of the focus program 'Deduktion'.

References

[1] T. Arts and J. Giesl, Termination of constructor systems, in: *Proc. RTA-96*, Lecture Notes in Computer Science, Vol. 1103 (Springer, Berlin, 1996) 63–77.

[2] T. Arts and J. Giesl, Automatically proving termination where simplification orderings fail, in: *Proc. TAPSOFT '97*, Lecture Notes in Computer Science, Vol. 1214 (Springer, Berlin, 1997) 261–272.

[3] T. Arts and J. Giesl, Proving innermost normalisation automatically, in: *Proc. RTA-97*, Lecture Notes in Computer Science, Vol. 1232 (Springer, Berlin, 1997) 157–171.

[4] T. Arts and J. Giesl, Modularity of termination using dependency pairs, Technical Report IBN 97/45, TH Darmstadt, Germany, 1997.

[5] T. Arts, Termination by absence of infinite chains of dependency pairs, in: *Proc. CAAP '96*, Lecture Notes in Computer Science, Vol. 1059 (Springer, Berlin, 1996) 196–210.

[6] T. Arts, Automatically proving termination and innermost normalisation of term rewriting systems, Ph.D. Thesis, Utrecht University, The Netherlands, 1997.

[7] T. Arts and H. Zantema, Termination of logic programs using semantic unification, in: *Proc. LoPSTr '95*, Lecture Notes in Computer Science, Vol. 1048 (Springer, Berlin, 1995) 219–233.

[8] L. Bachmair and N. Dershowitz, Commutation, transformation and termination, in: *Proc. CADE-8*, Lecture Notes in Computer Science, Vol. 230 (Springer, Berlin, 1986) 5–20.

[9] L. Bachmair, Proof methods for equational theories, Ph.D. Thesis, University of Illinois, Urbana, IL, 1987.

[10] F. Bellegarde and P. Lescanne, Termination proofs based on transformation techniques, Technical Report, Centre de Recherche en Informatique de Nancy, France, 1988.

[11] F. Bellegarde and P. Lescanne, Termination by completion, *Applicable Algebra in Engineering, Communication and Computing* **1** (1990) 79–96.

[12] A. Ben Cherifa and P. Lescanne, Termination of rewriting systems by polynomial interpretations and its implementation, *Sci. Comput. Programming* **9** (1987) 137–159.

[13] E. Bevers and J. Lewi, Proving termination of (conditional) rewrite systems, *Acta Informatica* **30** (1993) 537–568.

[14] R. S. Boyer and J S. Moore, *A Computational Logic* (Academic Press, 1979).

[15] B. Courcelle, Recursive applicative program schemes, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science. Vol. B* (North-Holland, 1990) 459–492.

[16] N. Dershowitz, Termination of linear rewriting systems, in: *Proc. ICALP '81*, Lecture Notes in Computer Science, Vol. 115 (Springer, Berlin, 1981) 448–458.

[17] N. Dershowitz, Orderings for term-rewriting systems, *Theoret. Comp. Sci.* **17** (1982) 279–301.

[18] N. Dershowitz, Termination of rewriting, *J. Symb. Comp.* **3** (1987) 69–116.

[19] N. Dershowitz, 33 examples of termination, in: *Proc. Term Rewriting, French Spring School of Theoretical Computer Science*, Lecture Notes in Computer Science, Vol. 909 (Springer, Berlin, 1993) 16–27.

[20] N. Dershowitz and C. Hoot, Natural termination, *Theoret. Comp. Sci.* **142** (1995) 179–207.

[21] N. Dershowitz and J.-P. Jouannaud, Rewrite systems, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science. Vol. B* (North-Holland, 1990) 243–320.

[22] J. Dick, J. Kalmus, and U. Martin, Automating the Knuth Bendix ordering, *Acta Informatica* **28** (1990) 95–119.

[23] K. Drosten, *Termersetzungssysteme: Grundlagen der Prototyp-Generierung algebraischer Spezifikationen* (Springer, Berlin, 1989).

[24] M. Ferreira and H. Zantema, Total termination of term rewriting, in: *Proc. RTA-93*, Lecture Notes in Computer Science, Vol. 690 (Springer, Berlin, 1993) 213–227.

[25] M. Ferreira and H. Zantema, Syntactical analysis of total termination, in: *Proc. ALP '94*, Lecture Notes in Computer Science, Vol. 850 (Springer, Berlin, 1994) 204–222.

[26] M. Ferreira and H. Zantema, Dummy elimination: making termination easier, in: *Proc. FCT '95*, Lecture Notes in Computer Science, Vol. 965 (Springer, Berlin, 1995) 243–252.

[27] M. Ferreira, Termination of Term Rewriting – Well-foundedness, Totality and Transformations, Ph.D. Thesis, Utrecht University, The Netherlands, 1995.

[28] M. Geerling, Termination of term rewriting systems, Master's thesis, Utrecht University, The Netherlands, 1991.

[29] A. Geser, On normalizing, non-terminating one-rule string rewriting systems, Technical Report, WSI 96-34, Universität Tübingen, 1996.

[30] O. Geupel, Overlap closures and termination of term rewriting systems, Technical Report MIP-8922 283, Universität Passau, Germany, 1989.

[31] J. Giesl, Generating polynomial orderings for termination proofs, in: *Proc. RTA-95*, Lecture Notes in Computer Science, Vol. 914 (Springer, Berlin, 1995) 426–431.

[32] J. Giesl, *Automatisierung von Terminierungsbeweisen für rekursiv definierte Algorithmen*, Ph.D. Thesis (Infix, St. Augustin, 1995).

[33] J. Giesl, Termination analysis for functional programs using term orderings, in: *Proc. SAS '95*, Lecture Notes in Computer Science, Vol. 983 (Springer, Berlin, 1995) 154–171.

[34] J. Giesl, Termination of nested and mutually recursive algorithms, *Journal of Automated Reasoning* **19** (1997) 1–29.

[35] B. Gramlich, Abstract relations between restricted termination and confluence properties of rewrite systems, *Fundamenta Informaticae* **24** (1995) 3–23.

[36] B. Gramlich, On proving termination by innermost termination, in: *Proc. RTA-96*, Lecture Notes in Computer Science, Vol. 1103 (Springer, Berlin, 1996) 93–107.

[37] G. Huet and D. Lankford, On the uniform halting problem for term rewriting systems, Technical Report 283, INRIA, Le Chesnay, France, 1978.

[38] G. Huet and J. M. Hullot, Proofs by induction in equational theories with constructors, *Journal of Computer and System Sciences* **25** (1982) 239–299.

[39] J. M. Hullot, Canonical forms and unification, in: *Proc. CADE-5*, Lecture Notes in Computer Science, Vol. 87 (Springer, Berlin, 1980) 318–334.

[40] S. Kamin and J.-J. Lévy, Two generalizations of the recursive path ordering, Unpublished Note, Dept. of Computer Science, University of Illinois, Urbana, IL, 1980.

[41] J. W. Klop, Term rewriting systems, in: S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, eds., *Handbook of Logic in Computer Science, Vol. 2* (Oxford University Press, New York, 1992) 1–116.

[42] D. E. Knuth and P. B. Bendix, Simple word problems in universal algebras, in: J. Leech, ed., *Computational problems in abstract algebra* (Pergamon Press, 1970) 263–297.

[43] T. Kolbe, Challenge problems for automated termination proofs of term rewriting systems, Technical Report IBN 96/42, TU Darmstadt, Germany, 1996.

[44] M. R. K. Krishna Rao, Modular proofs for completeness of hierarchical term rewriting systems, *Theoret. Comput. Sci.* **151** (1995) 487–512.

[45] M. R. K. Krishna Rao, Some characteristics of strong innermost normalization, in: *Proc. AMAST '96*, Lecture Notes in Computer Science, Vol. 1101 (Springer, Berlin, 1996) 406–420.

[46] D. S. Lankford, On proving term rewriting systems are Noetherian, Memo MTP-3, Dept. of Mathematics, Louisiana Technical University, Ruston, LA, 1979.

[47] D. S. Lankford and D. R. Musser, A finite termination criterion, 1978.

[48] C. Marché and X. Urbain, Personal communication, 1997.

[49] A. Middeldorp, H. Ohsaki, and H. Zantema, Transforming termination by self-labelling, in: *Proc. CADE-13*, Lecture Notes in Computer Science, Vol. 1104 (Springer, Berlin, 1996) 373–387.

[50] A. Middeldorp and H. Zantema, Simple termination of rewrite systems, *Theoret. Comput. Sci.* **175** (1997) 127–158.

[51] D. A. Plaisted, A recursively defined ordering for proving termination of term rewriting systems, Report R-78-943, Dept. of Computer Science, University of Illinois, Urbana, IL, 1978.

[52] D. A. Plaisted, A simple non-termination test for the Knuth-Bendix method, in: *Proc. CADE-8*, Lecture Notes in Computer Science, Vol. 230 (Springer, Berlin, 1986) 79–88.

[53] J. Steinbach, Generating polynomial orderings, *Inform. Processing Lett.* **49** (1994) 85–93.

[54] J. Steinbach, Automatic termination proofs with transformation orderings, in: *Proc. RTA-95*, Lecture Notes in Computer Science, Vol. 914 (Springer, Berlin, 1995) 11–25. Full version appeared as Technical Report SR-92-93, Universität Kaiserslautern, Germany, 1992.

[55] J. Steinbach, Simplification orderings: history of results, *Fundamenta Informaticae* **24** (1995) 47–87.

[56] Y. Toyama, Counterexamples to the termination for the direct sum of term rewriting systems, *Inform. Processing Lett.* **25** (1987) 141–143.

[57] C. Walther, *Automatisierung von Terminierungsbeweisen* (Vieweg, Braunschweig, 1991).

[58] C. Walther, On Proving the Termination of Algorithms by Machine, *Artif. Intell.* **71** (1994) 101–157.

[59] H. Zantema, Termination of term rewriting: interpretation and type elimination, *J. Symb. Comp.* **17** (1994) 23–50.

[60] H. Zantema, Termination of Term Rewriting by Semantic Labelling, *Fundamenta Informaticae* **24** (1995) 89–105.