Automatically Proving Termination Where Simplification Orderings Fail[†]

Thomas Arts[‡] Jürgen Giesl[§]

Abstract

To prove termination of term rewriting systems (TRSs), several methods have been developed to synthesize suitable well-founded orderings automatically. However, virtually all orderings that are amenable to automation are so-called simplification orderings. Unfortunately, there exist numerous interesting and relevant TRSs that cannot be oriented by orderings of this restricted class and therefore their termination cannot be proved automatically with the existing techniques.

In this paper we present a new automatic approach which allows to apply the standard techniques for automated termination proofs to those TRSs where these techniques failed up to now. For that purpose we have developed a procedure which, given a TRS, generates a set of inequalities (constraints) automatically. If there exists a well-founded ordering satisfying these constraints, then the TRS is terminating. It turns out that for many TRSs where a *direct* application of standard techniques fails, these standard techniques can nevertheless synthesize a well-founded ordering satisfying the generated constraints. In this way, termination of numerous (also non-simply terminating) term rewriting systems can be proved fully automatically.

1. Introduction

Termination is one of the most fundamental properties of a term rewriting system, cf. e.g. [DJ90]. While in general this problem is undecidable [HL78], several methods for proving termination have been developed (e.g. path orderings [Pla78, Der82, DH95, Ste95b], Knuth-Bendix orderings [KB70, DKM90], forward closures [LM78, DH95], semantic interpretations [Lan79, BL87, BL93, Ste94, Zan94, Gie95b], transformation orderings [BD86, BL90, Ste95a], semantic labelling [Zan95] etc. — for surveys see e.g. [Der87, Ste95b]).

In this paper we present a new approach for the *automation* of termination proofs. Previous methods for proving termination usually tried to find a well-founded ordering (with certain additional features) such that left-hand sides of rules are greater than right-hand sides. However, 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 formal definitions needed for this approach are introduced in Sect. 2 and in Sect. 3 we present a new termination criterion and prove its soundness and completeness.

The main advantage of our termination criterion is that it is especially well suited for automation. Therefore, in Sect. 4 we show how this criterion can be checked

[†]This work was partially supported by the Deutsche Forschungsgemeinschaft under grant no. Wa 652/7-1 as part of the focus program 'Deduktion'.

[‡]Utrecht University, E-mail: thomas@cs.ruu.nl

[§]FB Informatik, TH Darmstadt, Alexanderstr. 10, 64283 Darmstadt, Germany, E-mail: giesl@inferenzsysteme.informatik.th-darmstadt.de

automatically. To increase the power of our method we introduce a refined approach for its automation in Sect. 5. In this way we obtain a very powerful technique which enables automated termination proofs for many TRSs where termination could not be proved automatically before. In Sect. 6 we give some comments on related work followed by a short conclusion in Sect. 7. The use and power of the technique is demonstrated in Sect. 8 by a collection of examples.

2. Dependency Pairs

In this section we introduce the definitions needed for our termination criterion. For TRSs of a certain kind, viz. the so-called *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 $\mathcal{F}$). Here, the *root* of a term $f(\ldots)$ denotes the leading function symbol f.

2.1 Definition (Defined Symbols and Constructors, cf. [Kri95]). The set $D_\mathcal{R}$ of defined symbols of a TRS $\mathcal{R}(\mathcal{F}, R)$ is defined as $\{\mathrm{root}(l) | l \to r \in R\}$ and the set $C_\mathcal{R}$ of constructor symbols of $\mathcal{R}(\mathcal{F}, R)$ 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 this omission does not cause any confusion.
As an example consider the following TRS. 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 $[1+2+3]$).

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

The defined symbols of this TRS are app and sum, whereas $+$, nil, and $.$ are constructors.
A TRS is terminating if there does not exist a term that starts an infinite reduction. Unfortunately, most methods for automated termination proofs are restricted to *simplification orderings* [Der79, Ste95b]. These methods cannot be used to prove termination of systems like the TRS above, because the left-hand side of the last rule is homeomorphically embedded in its right-hand side.
To prove termination of such a system, instead of comparing s with every term t it can be reduced to (i.e. instead of demanding $s \succ t$ whenever $s \to_\mathcal{R} t$), the central idea of our approach is to compare s only with those subterms of t that are instantiations of left-hand sides of rules. For that purpose we only regard terms with defined root symbols, because rewrite rules can only be applied to such subterms. Therefore, instead of comparing left- and right-hand sides of rules, we only concentrate on those subterms of the right-hand sides whose root is a defined symbol.
More precisely, if a term $f(s_1, \ldots, s_n)$ rewrites to another term $C[g(t_1, \ldots, t_m)]$ (where f and g are defined symbols and C denotes some context), then to prove termination we compare the argument tuple $s_1, \ldots, s_n$ with the tuple $t_1, \ldots, t_m$. In

order to avoid the handling of *tuples*, for a formal definition we introduce a special symbol F, not occurring in the signature of the TRS, for every defined symbol f in D and compare the *terms* $F(s_1,\ldots,s_n)$ and $G(t_1,\ldots,t_m)$ instead. To ease readability we assume in this paper that the signature $\mathcal{F}$ consists of lower case function symbols only and denote the special symbols by the corresponding upper case symbols.

2.2 Definition (Dependency Pairs). 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 $f,g \in D$, then $\langle F(s_1,\ldots,s_n), G(t_1,\ldots,t_m)\rangle$ is called a *dependency pair* of $\mathcal{R}$.

In our example we obtain the following dependency pairs:

$$\langle \mathsf{APP}(x.\,l,k), \mathsf{APP}(l,k)\rangle, \tag{1}$$
$$\langle \mathsf{SUM}(x.\,y.\,l), \mathsf{SUM}((x+y).\,l)\rangle, \tag{2}$$
$$\langle \mathsf{SUM}(\mathsf{app}(l,x.\,y.\,k)), \mathsf{SUM}(x.\,y.\,k)\rangle, \tag{3}$$
$$\langle \mathsf{SUM}(\mathsf{app}(l,x.\,y.\,k)), \mathsf{APP}(l,\mathsf{sum}(x.\,y.\,k))\rangle, \tag{4}$$
$$\langle \mathsf{SUM}(\mathsf{app}(l,x.\,y.\,k)), \mathsf{SUM}(\mathsf{app}(l,\mathsf{sum}(x.\,y.\,k)))\rangle. \tag{5}$$

Two dependency pairs $\langle s_1,t_1\rangle$ and $\langle s_2,t_2\rangle$ are equivalent, if there exists a renaming substitution ρ such that $s_1\rho = s_2$ and $t_1\rho = t_2$. We are interested in dependency pairs up to equivalence and when useful, we may assume, without loss of generality, that two (occurrences of) dependency pairs have disjoint sets of variables.

3. A Termination Criterion Using Dependency Pairs

Using the notion of dependency pairs, in this section we introduce a criterion for termination of TRSs. Recall that a left-hand side of a rewrite rule only matches subterms with defined root symbols. Thus, there occurs a defined symbol in any term in an infinite reduction. In a reduction, new defined symbols are introduced by the right-hand sides of the applied rewrite rules. Therefore, the dependency pairs focus on the subterms of the right-hand sides that have a defined symbol as root symbol. By regarding a sequence of these dependency pairs, the introduction of new defined symbols can be traced. This observation is the motivation for the following definition.

3.1 Definition ($\mathcal{R}$-chains). Let $\mathcal{R}(D,C,R)$ be a TRS. A sequence of dependency pairs is called an *$\mathcal{R}$-chain* if there exists a substitution[1] σ, such that $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ holds for all consecutive pairs $\langle s_i,t_i\rangle$ and $\langle s_{i+1},t_{i+1}\rangle$ in the sequence.

For example, $\langle \mathsf{APP}(x.\,l,k), \mathsf{APP}(l,k)\rangle\langle \mathsf{APP}(x'.\,l',k'), \mathsf{APP}(l',k')\rangle$ is an $\mathcal{R}$-chain, because $\mathsf{APP}(l,k)\sigma \to_{\mathcal{R}}^* \mathsf{APP}(x'.\,l',k')\sigma$ holds for the substitution σ that replaces l by $x'.\,l'$ and k by k'.
With the concept of dependency pairs we can now use the following criterion for termination: A TRS $\mathcal{R}$ is terminating iff no infinite $\mathcal{R}$-chain exists. If it is clear from the context which TRS is involved, we omit this information and write 'chain' instead of '$\mathcal{R}$-chain'. In this section we prove that the absence of infinite chains is a sufficient (Thm. 3.2) and necessary (Thm. 3.3) condition for termination.

3.2 Theorem (Soundness). *Let $\mathcal{R}(D,C,R)$ be a TRS. If no infinite $\mathcal{R}$-chain exists, then $\mathcal{R}$ is terminating.*

[1] Throughout the paper we regard substitutions whose domain may be *infinite*.

PROOF. We prove that for any infinite reduction we can construct an infinite $\mathcal{R}$-chain.
Let t be a term that starts an infinite reduction. Then the term t contains a subterm[2] $f_1(\vec{u}_1)$ that starts an infinite reduction, but none of the terms $\vec{u}_1$ starts an infinite reduction, i.e. $\vec{u}_1$ are strongly normalising.
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 $f_1(\vec{w}_1) \to r$ is applied to $f_1(\vec{v}_1)$, i.e. a substitution σ exists such that $f_1(\vec{v}_1) = f_1(\vec{w}_1)\sigma \to_{\mathcal{R}} r\sigma$. Now the infinite reduction continues with $r\sigma$, i.e. the term $r\sigma$ starts an infinite reduction, too.
Note that by assumption there exists no infinite reduction beginning with one of the terms $\vec{v}_1 = \vec{w}_1\sigma$. Hence, for all variables x occurring in $f_1(\vec{w}_1)$ the terms $\sigma(x)$ are strongly normalising. Thus, since $r\sigma$ starts an infinite reduction, there occurs a subterm $f_2(\vec{u}_2)$ in r, i.e. $r = C[f_2(\vec{u}_2)]$ for some context C, such that

- $f_2(\vec{u}_2)\sigma$ starts an infinite reduction and
- $\vec{u}_2\sigma$ are strongly normalising terms.

The first dependency pair of the infinite 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: $f_2(\vec{u}_2)\sigma$ starts an infinite reduction and the terms $\vec{u}_2\sigma$ are strongly normalising. Again, in zero or more steps $f_2(\vec{u}_2)\sigma$ reduces to $f_2(\vec{v}_2)$ to which a rewrite rule $f_2(\vec{w}_2) \to r_2$ can be applied such that $r_2\tau$ starts an infinite reduction for some substitution τ with $\vec{v}_2 = \vec{w}_2\tau$.
Similar to the observations above, since $r_2\tau$ starts an infinite reduction, there must be a subterm $f_3(\vec{u}_3)$ in r_2 such that

- $f_3(\vec{u}_3)\tau$ starts an infinite reduction and
- $\vec{u}_3\tau$ are strongly normalising terms.

This results in the second dependency pair of the $\mathcal{R}$-chain, viz. $\langle F_2(\vec{w}_2), F_3(\vec{u}_3)\rangle$. By repetition of this process[3] 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 $F_2(\vec{u}_2\sigma) \to^*_{\mathcal{R}} F_2(\vec{v}_2)$ and $\vec{v}_2 = \vec{w}_2\tau$. Since we assume, without loss of generality, that the variables of consecutive dependency pairs are disjoint, we obtain one substitution $\rho = \sigma \circ \tau \circ \ldots$ such that

$$F_2(\vec{u}_2)\rho \to^*_{\mathcal{R}} F_2(\vec{w}_2)\rho, F_3(\vec{u}_3)\rho \to^*_{\mathcal{R}} F_3(\vec{w}_3)\rho, \ldots$$

Thus, we have in fact constructed an infinite $\mathcal{R}$-chain. □

This criterion can now be used to prove termination of TRSs. For instance, in our example there cannot be an infinite chain of the form

$$\langle \mathsf{APP}(x.l,k), \mathsf{APP}(l,k)\rangle\langle \mathsf{APP}(x'.l',k'), \mathsf{APP}(l',k')\rangle\ldots,$$

because for every substitution σ, the term $\mathsf{APP}(x.l,k)$ contains one more occurrence of the symbol '.' than $\mathsf{APP}(l,k)$. In Sect. 4 we will show how the absence of infinite chains can be proved automatically.
After having shown that the absence of infinite chains is sufficient for termination, now we prove that this criterion is even necessary for termination.

[2] We denote tuples of terms $t_1, \ldots, t_n$ by $\vec{t}$.
[3] Formally, this sequence has to be defined by an inductive definition.

3.3 THEOREM (Completeness). *Let $\mathcal{R}(D, C, R)$ be a TRS. If $\mathcal{R}$ is terminating, then no infinite $\mathcal{R}$-chain exists.*

PROOF. We prove that any infinite $\mathcal{R}$-chain corresponds to an infinite reduction. Assume there exists an infinite $\mathcal{R}$-chain.

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

Hence, there must be a substitution σ such that

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

resp. $f_i(\vec{t}_i)\sigma \to^*_{\mathcal{R}} f_i(\vec{s}_i)\sigma$, as the upper case symbols F_i are not defined.
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 following infinite reduction

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

□

We derived that a TRS $\mathcal{R}$ is terminating if and only if no infinite $\mathcal{R}$-chain exists. Since it is undecidable whether a TRS is terminating, it is also undecidable whether an infinite $\mathcal{R}$-chain exists. However, for certain TRSs $\mathcal{R}$ we obtain a set of dependency pairs for which we can automatically derive that these dependency pairs can never form an infinite $\mathcal{R}$-chain and therefore we can prove termination of these TRSs automatically.

4. Checking the Termination Criterion Automatically

Our termination criterion states that a TRS $\mathcal{R}$ is terminating iff no infinite $\mathcal{R}$-chain exists. In this section we present an approach to perform automated termination proofs using this criterion, i.e. we introduce a method to prove the absence of infinite chains automatically.
For that purpose, we introduce a procedure which, given a TRS, generates a set of inequalities (which represent constraints). This transformation of a TRS into a set of inequalities is such that if there exists a well-founded ordering satisfying these inequalities, then termination of the TRS has been proved. 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. a well-founded ordering satisfying the generated inequalities can be synthesized, whereas a well-founded ordering compatible with the reduction ordering cannot be synthesized.
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 will synthesize a well-founded ordering $\succ$ such that all dependency pairs are decreasing w.r.t. this ordering. More precisely, if for any sequence of dependency pairs

$$\langle s_1, t_1\rangle\langle s_2, t_2\rangle\langle s_3, t_3\rangle \dots$$

and for any substitution σ with $t_i\sigma \to^*_{\mathcal{R}} s_{i+1}\sigma$ we have $s_1\sigma \succ t_1\sigma$, $s_2\sigma \succ t_2\sigma$, $s_3\sigma \succ t_3\sigma$, ... and $t_1\sigma \succ s_2\sigma$, $t_2\sigma \succ s_3\sigma$, ..., then no infinite chain exists.

However, for most TRSs, the above inequalities are not satisfied by any well-founded ordering $\succ$. The reason is that the terms $t_i\sigma$ and $s_{i+1}\sigma$ of consecutive dependency pairs in chains are often identical and therefore $t_i\sigma \succ s_{i+1}\sigma$ does not hold.
But obviously not *all* of the inequalities $s_i\sigma \succ t_i\sigma$ and $t_i\sigma \succ s_{i+1}\sigma$ have to be *strict*. For instance, to guarantee the absence of infinite chains it is sufficient if there exists a well-founded *quasi*-ordering $\succsim$ such that the strict inequality $s_i\sigma \succ t_i\sigma$ and the *non-strict* inequality $t_i\sigma \succsim s_{i+1}\sigma$ hold for each sequence of dependency pairs as above. (A quasi-ordering $\succsim$ is a reflexive and transitive relation and $\succ$ is called *well-founded* if its strict part $\succ$ is well founded.)
Note that we cannot determine automatically for which substitutions σ we have $t_i\sigma \to^* s_{i+1}\sigma$ and moreover, it is practically impossible to examine infinite sequences of dependency pairs. In the following we will restrict ourselves to *weakly monotonic* quasi-orderings $\succsim$ where both $\succsim$ and its strict part $\succ$ are *closed under substitution*. (A quasi-ordering $\succsim$ is *weakly monotonic* if $s \succsim t$ implies $f(\dots s \dots) \succsim f(\dots t \dots)$.) Then, to guarantee $t_i\sigma \succsim s_{i+1}\sigma$ whenever $t_i\sigma \to^* s_{i+1}\sigma$ holds, it is sufficient to demand $l \succsim r$ for all rewrite rules $l \to r$ of the TRS. To ensure $s_i\sigma \succ t_i\sigma$ for those dependency pairs occurring in possibly infinite chains, we will demand $s \succ t$ for *all* dependency pairs $\langle s, t\rangle$.

4.1 Theorem (Checking the Termination Criterion). *Let $\mathcal{R}(D, C, R)$ be a TRS. If there exists a well-founded, weakly monotonic quasi-ordering $\succsim$, where both $\succsim$ and $\succ$ are closed under substitution, such that*

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

then $\mathcal{R}$ is terminating.

Proof. Note that as $l \succsim r$ holds for all rules $l \to r$ in R and as $\succsim$ is weakly monotonic and closed under substitution, we have $\to^*_{\mathcal{R}} \subseteq \succsim$, i.e. if $t \to^*_{\mathcal{R}} s$ then $t \succsim s$.
Suppose there is an infinite $\mathcal{R}$-chain

$$\langle s_1, t_1\rangle\langle s_2, t_2\rangle \dots,$$

then there exists a substitution σ such that $t_i\sigma \to^*_{\mathcal{R}} s_{i+1}\sigma$ holds for all i. As $\to^*_{\mathcal{R}} \subseteq \succsim$, this implies $t_i\sigma \succsim s_{i+1}\sigma$. Hence, we obtain the infinite sequence

$$s_1\sigma \succ t_1\sigma \succsim s_2\sigma \succ t_2\sigma \succsim \dots$$

which is a contradiction to the well-foundedness of $\succsim$ and therefore no infinite chain exists. Thus, by Thm. 3.2 $\mathcal{R}$ is terminating. □

The technique of Thm. 4.1 is very useful to make standard methods like the recursive path ordering or polynomial interpretations applicable to TRSs for which they are not directly applicable. For instance, in our example we have to find a quasi-ordering satisfying the inequalities

$$\begin{aligned}
\mathsf{app}(\mathsf{nil}, k) &\succsim k\\
\mathsf{app}(l, \mathsf{nil}) &\succsim l\\
\mathsf{app}(x.l, k) &\succsim x.\mathsf{app}(l, k)\\
\mathsf{sum}(x.\mathsf{nil}) &\succsim x.\mathsf{nil}\\
\mathsf{sum}(x.y.l) &\succsim \mathsf{sum}((x+y).l)\\
\mathsf{sum}(\mathsf{app}(l, x.y.k)) &\succsim \mathsf{sum}(\mathsf{app}(l, \mathsf{sum}(x.y.k)))\\
\mathsf{APP}(x.l, k) &\succ \mathsf{APP}(l, k)
\end{aligned}$$

$$\begin{aligned}
\mathsf{SUM}(x.y.l) &\succ \mathsf{SUM}((x+y).l)\\
\mathsf{SUM}(\mathsf{app}(l,x.y.k)) &\succ \mathsf{SUM}(x.y.k)\\
\mathsf{SUM}(\mathsf{app}(l,x.y.k)) &\succ \mathsf{APP}(l,\mathsf{sum}(x.y.k))\\
\mathsf{SUM}(\mathsf{app}(l,x.y.k)) &\succ \mathsf{SUM}(\mathsf{app}(l,\mathsf{sum}(x.y.k)))
\end{aligned}$$

For example, these inequalities are satisfied by the polynomial ordering [Lan79] where nil is mapped to the constant 0, $x.l$ is mapped to $l+1$, $(x+y)$ is mapped to $x+y$, $\mathsf{app}(l,k)$ is mapped to $l+k+1$, $\mathsf{sum}(l)$ is mapped to the constant 1, and $\mathsf{APP}(l,k)$ and $\mathsf{SUM}(l)$ are both mapped to l. Methods for the automated generation of polynomial orderings have for instance been developed in [Ste94, Gie95b]. In this way, termination of this TRS can be proved fully automatically, although a direct termination proof with simplification orderings was not possible.
Note that when using polynomial orderings for *direct* termination proofs of TRSs, then the polynomials have to be (strongly) monotonic in all their arguments, i.e. $s \succ t$ implies $f(\ldots s \ldots) \succ f(\ldots t \ldots)$. However, for the approach of this paper, we only need a *weakly* monotonic quasi-ordering satisfying the inequalities. Thus, $s \succ t$ only implies $f(\ldots s \ldots) \succsim f(\ldots t \ldots)$. 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 map $\mathsf{sum}(l)$ to the constant 1 and map $x.l$ to $l+1$.
We can also use for example the recursive path ordering, which is (strongly) monotonic, by first eliminating some of the arguments of several function symbols. Thus, in our example, we eliminate all arguments of sum and the first argument of the function symbol '.'. The resulting inequalities are then satisfied by the recursive path ordering.

5. Dependency Graphs

To prove termination of a TRS according to Thm. 4.1 we have to find an ordering such that $s \succ t$ holds for *all* dependency pairs $\langle s,t\rangle$. However, for certain rewrite systems this requirement can be weakened easily, in this section we show that we have to demand $s \succ t$ for *some* dependency pairs only.
For example, let us extend the TRS for sum and app by the following rules for $+$.

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

Now $+$ is no longer a constructor, but a defined symbol. This results in two new dependency pairs

$$\langle \mathsf{SUM}(x.y.l), \mathsf{PLUS}(x,y)\rangle, \tag{6}$$
$$\langle \mathsf{PLUS}(\mathsf{s}(x),y), \mathsf{PLUS}(x,y)\rangle \tag{7}$$

and to prove termination according to Thm. 4.1 in addition to the inequalities in Sect. 4 we now obtain the following inequalities.

$$\begin{aligned}
\mathsf{0}+y &\succsim y,\\
\mathsf{s}(x)+y &\succsim \mathsf{s}(x+y),\\
\mathsf{SUM}(x.y.l) &\succ \mathsf{PLUS}(x,y),\\
\mathsf{PLUS}(\mathsf{s}(x),y) &\succ \mathsf{PLUS}(x,y).
\end{aligned}$$

Unfortunately, no polynomial ordering (and no simplification ordering either) satisfies all resulting inequalities[4]. However, in our example demanding $\mathsf{SUM}(x.\,y.\,l) \succ \mathsf{PLUS}(x,y)$ 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*. Recall that a dependency pair $\langle u, v\rangle$ may only follow a pair $\langle s, t\rangle$ in a chain, if there exists a substitution σ such that $t\sigma \to_{\mathcal{R}}^* u\sigma$. As the upper case symbol PLUS is not a defined symbol, $\mathsf{PLUS}(x,y)\sigma$ can only be reduced to terms with the same root symbol PLUS. Hence, the only dependency pair following $\langle \mathsf{SUM}(\ldots), \mathsf{PLUS}(\ldots)\rangle$ can be $\langle \mathsf{PLUS}(\mathsf{s}(x), y), \mathsf{PLUS}(x,y)\rangle$, i.e. (6) can never occur twice in a chain.
To determine those dependency pairs which may possibly occur infinitely often in a chain we define a graph whose nodes are the dependency pairs. Those dependency pairs that possibly occur consecutive in a chain are connected in this graph. In this way, any infinite chain corresponds to a cycle in the graph and therefore it suffices to consider cycles in a graph instead of sequences of dependency pairs.

5.1 Definition (Dependency Graph). Let $\mathcal{R}$ be a TRS. The dependency graph of $\mathcal{R}$ is a directed graph whose nodes are labelled with the dependency pairs of $\mathcal{R}$ and there is an arc from $\langle s, t\rangle$ to $\langle u, v\rangle$ if there exists a substitution σ such that $t\sigma \to_{\mathcal{R}}^* u\sigma$.

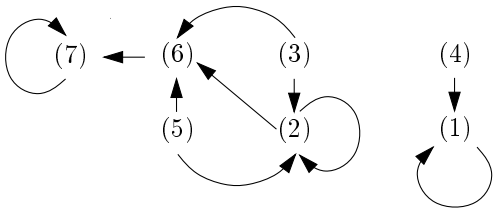

Figure 1: The dependency graph of the example

Therefore, to prove termination of the TRS it is sufficient if $s \succ t$ holds for at least one dependency pair on each cycle and $s \succsim t$ for all dependency pairs on a cycle. Dependency pairs that do not occur on a cycle can be ignored. So we only have to demand that the dependency pairs (1), (2), and (7) are strictly decreasing. Now a polynomial ordering satisfying the resulting inequalities is obtained by extending the polynomial ordering we used in Sect. 4 as follows: The symbol $\mathsf{0}$ is mapped to the number 0, $\mathsf{s}(x)$ is mapped to $x+1$, and $\mathsf{PLUS}(x,y)$ is mapped to x. In general, we obtain the following refined theorem to check our termination criterion automatically.

5.2 Theorem (Termination Proofs with Dependency Graphs). *Let $\mathcal{R}(D, C, R)$ be a TRS. If there exists a well-founded, weakly monotonic quasi-ordering $\succsim$, where both $\succsim$ and $\succ$ are closed under substitution, such that*

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

[4]The reason is that to satisfy $\mathsf{SUM}(x.\,y.\,l) \succ \mathsf{PLUS}(x,y)$, the polynomial for '.' has to depend on its first argument. But then to satisfy $\mathsf{sum}(x.\,\mathsf{nil}) \succsim x.\,\mathsf{nil}$, sum can no longer be mapped to a constant. Hence, for large enough arguments, the subterm $x.\,y.\,k$ of the left-hand side of $\mathsf{sum}(\mathsf{app}(l, x.\,y.\,k)) \to \mathsf{sum}(\mathsf{app}(l, \mathsf{sum}(x.\,y.\,k)))$ will be mapped to a smaller number than the subterm $\mathsf{sum}(x.\,y.\,k)$ of its right-hand side.

then $\mathcal{R}$ is terminating.

PROOF. Suppose there is an infinite $\mathcal{R}$-chain, then this infinite chain must correspond 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. Every cycle has at least one dependency pair $\langle s, t\rangle$ with $s \succ t$ and therefore one such dependency pair occurs (up to renaming of the variables) infinitely many times in an infinite $\mathcal{R}$-chain. Thus the infinite chain must have the form

$$\ldots \langle s, t\rangle \ldots \langle s\rho_1, t\rho_1\rangle \ldots \langle s\rho_2, t\rho_2\rangle \ldots$$

where $\rho_1, \rho_2, \ldots$ are renamings. There exists a substitution σ such that for all consecutive dependency pairs $\langle s_i, t_i\rangle$ and $\langle s_{i+1}, t_{i+1}\rangle$ we have $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$. This implies $t_i\sigma \succsim s_{i+1}\sigma$, because $\to_{\mathcal{R}}^* \subseteq \succsim$ (as in Thm. 4.1). Without loss of generality we may assume that the dependency pairs following $\langle s, t\rangle$ in the chain all occur on some cycle of the graph. Hence, we obtain

$$s\sigma \succ t\sigma \succsim s\rho_1\sigma \succ t\rho_1\sigma \succsim s\rho_2\sigma \succ t\rho_2\sigma \succsim \ldots$$

and thus $s\sigma \succ s\rho_1\sigma \succ s\rho_2\sigma \ldots$. This is a contradiction to the well-foundedness of $\succ$, hence no infinite $\mathcal{R}$-chain exists and by Thm. 3.2 $\mathcal{R}$ is terminating. □

However, to perform termination proofs according to Thm. 5.2, we would have to construct the dependency graph automatically. Unfortunately, in general this is not possible, since for given terms t, u it is undecidable whether there exists a substitution σ such that $t\sigma \to_{\mathcal{R}}^* u\sigma$.
Therefore, we introduce a technique to approximate the dependency graph, i.e. the technique computes a *superset* of those pairs t, u where $t\sigma \to_{\mathcal{R}}^* u\sigma$ holds for some substitution σ. We call terms t, u 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. 5.2 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 we unify a modification of these terms instead. 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 $t\sigma$ can be reduced to $u\sigma$, we replace all subterms in t that have a defined root symbol by a new variable.
Subsequently, we use syntactic unification to determine whether the terms are connectable, i.e. we check whether the modified term t unifies with u. For example, $\mathsf{SUM}(\ldots)$ is not connectable to $\mathsf{PLUS}(x, y)$. On the other hand, $\mathsf{SUM}(\mathsf{sum}(\ldots))$ would be connectable to $\mathsf{SUM}(x.\,y.\,l)$ (because before unification, $\mathsf{sum}(\ldots)$ would be replaced by a new variable).
In order to ensure that t is connectable to u whenever there exists a substitution σ such that $t\sigma \to_{\mathcal{R}}^* u\sigma$, before unification we also have to *rename* multiple occurrences of the same variable. As an example consider the following well-known TRS from [Toy87].

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

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.

5.3 DEFINITION (Connectable Terms). For any term t, let CAP(t) result from replacing all subterms of t that have a defined root symbol by different new variables and let REN(t) result 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. The term t is *connectable* to the term u iff REN(CAP(t)) and u are unifiable.

For example, we have CAP($\mathsf{SUM}((x+y).l.l)) = \mathsf{SUM}(z.l.l)$ and REN(CAP($\mathsf{SUM}((x+y).l.l))) = \mathsf{SUM}(z.l_1.l_2)$. As 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 determine whether there should be an arc between two dependency pairs in the dependency graph we unify the terms of the dependency pairs after modifying them by REN and CAP, i.e. we draw an arc from a dependency pair $\langle s,t\rangle$ to $\langle u,v\rangle$ whenever t is *connectable* to u. In this way, for our example a graph containing the dependency graph of Fig. 1 is constructed automatically (where there are additional arcs from (5) to (3), (4) and itself). In this way, termination of the TRS can be proved automatically (because (5) is also decreasing w.r.t. the mentioned polynomial ordering).
The following theorem proves the soundness of this approach: by the computation of connectable terms we in fact obtain a supergraph of the dependency graph, i.e. a graph containing all cycles of the dependency graph. Using this supergraph, we can now prove termination according to Thm. 5.2.

5.4 THEOREM (Computing Dependency Graphs). *Let $\mathcal{R}(D,C,R)$ be a TRS and let t,u be terms. If a substitution σ exists such that $t\sigma \to_{\mathcal{R}}^* u\sigma$, then t is connectable to u.*

PROOF. By induction on the structure of t we prove that if $t\sigma \to_{\mathcal{R}}^* v$ for some term v, then REN(CAP(t)) matches v. Thus, in particular, if $t\sigma \to_{\mathcal{R}}^* u\sigma$, then REN(CAP($t$)) matches $u\sigma$. As REN(CAP(t)) only contains new variables, this implies that REN(CAP(t)) and u are unifiable.
Assume that $t\sigma \to_{\mathcal{R}}^* v$ for some term v. If t is a variable or if $t = f(t_1,\ldots,t_k)$ for some defined symbol $f \in D$, then REN(CAP(t)) is a variable x. Hence, x matches v. If $t = c(t_1,\ldots,t_k)$ for some constructor $c \in C$, then we have

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

In this case, v has to be of the form $c(v_1,\ldots,v_k)$ and $t_i\sigma \to_{\mathcal{R}}^* v_i$ holds for all $1 \le i \le k$. By the induction hypothesis we obtain that REN(CAP(t_i)) matches v_i for all $1 \le i \le k$. Since by the definition of REN, the variables in REN(CAP(t_i)) are disjoint from the variables in REN(CAP(t_j)) for all $i \neq j$, REN(CAP(t)) also matches v. □

6. Related Work

The concept of *dependency pairs* was introduced in [Art96] and a first method for the automation of the dependency pair approach was proposed in [AG96b]. In the present paper we formulated an alternative version of the termination criterion using

dependency pairs which is better suited for automation than the original criterion of [Art96]. Moreover, in this way we could prove the soundness of the criterion in a very easy and short way (while the corresponding proof in [Art96] used semantic labelling [Zan95]) and we could also prove its completeness. By having the theory based on semantic labelling, in earlier approaches we were forced to construct a semantic interpretation for the TRS, which is undecidable in general. Our technique is now proved to be sound and complete without using semantic labelling and as an advantage over the earlier approaches, we do not need such an interpretation any more.

By the introduction of dependency graphs we obtained a considerably more powerful automated technique than the method proposed in [AG96b]. Most significant, while in [Art96, AG96b] dependency pairs were only used for termination proofs of non-overlapping constructor systems without nested recursion, we extended the technique to arbitrary term rewriting systems.

Recently, we also developed a method to use dependency pairs for proving *innermost* normalisation, which is applicable to arbitrary TRSs [AG96c]. By restricting the notion of chains, considering normal substitutions and innermost reductions only, we obtain a soundness and completeness result for innermost normalisation. Adapting the restrictions to the notion of dependency graphs as well results in a powerful technique to prove innermost normalisation of TRSs automatically. Although the latter method can also be used for proving termination, this can only be done for non-overlapping TRSs (where innermost normalisation is sufficient for termination), whereas the technique described in the present paper can be used for arbitrary rewrite systems.

Most other methods for automated termination proofs are restricted to the generation of *simplification orderings*. Instead of using these methods to prove termination directly, it is always advantageous to combine them with the technique presented in this paper. The reason is that for all those TRSs where termination could be proved with a simplification ordering directly, this simplification ordering also satisfies the inequalities resulting from our technique.

In this paper we presented a sound and complete criterion for termination. In contrast to most other complete approaches (semantic path ordering [KL80], general path ordering [DH95], semantic labelling [Zan95] etc.) our method is particularly well suited for automation as has been demonstrated in this paper. The only other sound and complete criterion that has been used for automatic termination proofs (by J. Steinbach [Ste92, Ste95a]) is the approach of *transformation orderings* [BD86, BL90]. It turns out that the termination of several examples where the automation of Steinbach failed [Ste92] can be proved by our technique automatically (Sect. 8).

There is a relation between the dependency pair approach and *semantic labelling* [Zan95], because the dependency pairs correspond to the labels of a TRS labelled by the process of *self*-labelling. However, the semantic labelling method presupposes a semantic interpretation, which in general cannot be found automatically, whereas the dependency pair approach does not rely on any semantical interpretation.

At first sight there seem to be some similarities between our method and *forward closures* [LM78, DH95]. 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. This construction is motivated by the fact that some terms are not essential for the generation of infinite reductions. However, these terms nevertheless have to be considered in the forward closure approach.

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 forwards closures only implies termination for right-linear [Der81] or non-overlapping [Geu89] TRSs. Moreover, in contrast to the dependency pair method, we do not know of any attempt to automate the forward closure approach.

7. Conclusion

We have developed a method for automated termination proofs of term rewriting systems. Based on the concept of dependency pairs we developed a termination criterion and we showed how the checking of this criterion can be automated. First, the dependency pairs are determined, which is trivially automated. Second, an approximation of the dependency graph is computed. For this purpose the notion of 'connectable terms' is automated, which can be done by a unification algorithm (without occur check). Third, the dependency pairs that are on a cycle have to be computed, for which several algorithms exist. Dependency pairs not on a cycle of the dependency graph can be ignored. Fourth, a set of inequalities is generated from the dependency pairs that occur on a cycle. Since only some dependency pairs should correspond to strict inequalities, either all possible sets of inequalities should be generated or heuristics should find a set of inequalities that is most suitable. Fifth, a standard technique, like polynomial interpretations or path orderings, is used to synthesize an ordering that satisfies the inequalities.
Methods for proving termination are often based on finding well-founded reduction orderings, i.e. the left-hand side of any rule has to be greater than the right-hand side of that rule for some well-founded ordering closed under context and substitution. Thus, a set of inequalities obtained by replacing every arrow in the TRS by an ordering symbol should be satisfied by a well-founded monotonic ordering closed under substitution. Our technique can be considered as a transformation that transforms these inequalities into a set of inequalities that only has to be satisfied by a well-founded *weakly* monotonic quasi-ordering closed under substitution. Compared to proving termination directly, our approach has the advantage that these inequalities are often satisfied by standard (simplification) orderings, even if termination of the original TRS cannot be proved with these orderings. In this way, these standard techniques can now be applied to prove termination of TRSs whose termination could not be proved automatically before. Moreover, if the TRS could be proved by synthesizing a simplification ordering directly, then the inequalities obtained by our technique are also satisfied by this ordering.
We implemented our procedure for the synthesis of inequalities. For the generation of a well-founded quasi-ordering satisfying these inequalities, we used well-known automatic techniques, as recursive path ordering, lexicographic path ordering, and polynomial interpretation. In this way, termination could be proved automatically for many challenge problems from literature as well as for numerous practically relevant TRSs from different areas of computer science. A collection of such examples, including arithmetical operations, sorting algorithms, and several well-known non-simply terminating TRSs, can be found in the next section.

8. Examples

This collection of examples demonstrates the power of the described method. Several of these examples are not simply terminating. Thus all methods based on simplification orderings fail in proving termination of these (non-simply terminating) term rewriting systems.

For proving termination of the examples, our technique first transforms the TRS into a set of inequalities. Two kinds of such inequalities can be distinguished: For each rewrite rule $l \to r$ we obtain an inequality $l \gtrsim r$ and for each dependency pair $\langle s, t \rangle$ on a cycle of the dependency graph we obtain the inequality $s \gtrsim t$. Furthermore, for each cycle one of these inequalities must be strict, i.e. $s \succ t$.
In most of the examples, we will only mention the inequalities resulting from dependency pairs on cycles of the dependency graph. We will refer to these inequalities as the *relevant* inequalities. But of course, the inequalities $l \gtrsim r$ are also synthesized for each rewrite rule $l \to r$.
After having obtained the inequalities, a well-founded weakly monotonic quasi-ordering satisfying these inequalities is generated. In the following collection of examples we use two different methods for that purpose.
The first approach is the well-known approach of synthesizing polynomial orderings [Lan79]. Several techniques exist to derive polynomial interpretations automatically [Ste94, Gie95b]. In contrast to the use of polynomial orderings for direct termination proofs, we can use polynomial interpretations with *weakly* monotonic polynomials. For instance, we can map a binary function symbol $f(x, y)$ to the polynomial $x + 1$ which is not strictly monotonic in its second argument. Moreover, we can map any function symbol to a constant.
The second approach is based on path orderings (e.g. recursive or lexicographic path orderings) [Pla78, Der82, DH95, Ste95b]. The path orderings are simplification orderings that are easily generated automatically. Note that path orderings are always strictly 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. More precisely, any function symbol f can be replaced by a function symbol f of smaller arity. For instance, the second argument of a binary function f may be eliminated. In that case every term $f(t, s)$ in the inequalities is replaced by $f(t)$. By comparing terms resulting from this replacement (instead of the original terms) we can take advantage of the fact that f does not have to be strictly monotonic in its second argument.
Moreover, we also allow the possibility that a function symbol may be mapped to one of its arguments. So a binary symbol f could also be mapped to its first argument. Thus, any term $f(t, s)$ in the inequalities would be replaced by t.
Note that there exist only finitely many (and only few) different possibilities to eliminate arguments of function symbols. Therefore, all these possibilities can be checked automatically.
First, the described technique is used to prove termination of all examples from [AG96a] (Ex. 8.1 – 8.14). While the method of [AG96b, AG96a] is restricted to non-overlapping constructor systems without nested recursion, the approach used in this paper can handle arbitrary term rewriting systems. Therefore, subsequently several examples are listed where the technique has been successfully applied to TRSs that do not meet the above restrictions.

8.1. Division, Version 1

This is the running example of the article [AG96b], which is not simply terminating.

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

In this example, apart from the four inequalities corresponding to the rewrite rules, two relevant inequalities are obtained

$$\begin{aligned} \mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{MINUS}(x,y) \\ \mathsf{QUOT}(\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{QUOT}(\mathsf{minus}(x,y),\mathsf{s}(y)). \end{aligned}$$

By mapping $\mathsf{minus}(x,y)$ to x, the recursive path ordering satisfies the demands on the ordering.
With the other approach, of polynomials, a suitable quasi-ordering is also found automatically. The normal ordering on the natural numbers together with the following interpretation of the function symbols satisfies the inequalities: the function symbol $\mathsf{0}$ is mapped to the number 0, $\mathsf{s}(x)$ is mapped to $x+1$ and $\mathsf{quot}(x,y)$, $\mathsf{QUOT}(x,y)$, $\mathsf{MINUS}(x,y)$ and $\mathsf{minus}(x,y)$ are mapped to x.

8.2. Division, Version 2

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

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

The inequalities obtained from the dependency pairs on a cycle of the dependency graph are given by:

$$\begin{aligned} \mathsf{MINUS}(x,\mathsf{s}(y)) &\succ \mathsf{MINUS}(x,y) \\ \mathsf{QUOT}(\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{QUOT}(\mathsf{minus}(x,y),\mathsf{s}(y)) \end{aligned}$$

Synthesizing a suitable ordering is as easy as it was for the previous example, since we just have to map $\mathsf{minus}(x,y)$ to x and $\mathsf{pred}(x)$ to x, too. The demands on the ordering are then satisfied by the recursive path ordering.

8.3. Division, Version 3

This TRS for division uses again different minus-rules. Similar to the preceding examples it is not simply terminating. We always use functions like $\mathsf{if}_{\mathsf{minus}}$ to encode conditions and to ensure 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.

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

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

The relevant inequalities are given by

$$\mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) \succ \mathsf{LE}(x, y)$$

$$\begin{aligned}
\mathsf{MINUS}(\mathsf{s}(x), y) &\succsim \mathsf{IF}_{\mathsf{minus}}(\mathsf{le}(\mathsf{s}(x), y), \mathsf{s}(x), y) \\
\mathsf{IF}_{\mathsf{minus}}(\mathsf{false}, \mathsf{s}(x), y) &\succ \mathsf{MINUS}(x, y)
\end{aligned}$$

$$\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) \succ \mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y))$$

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

$$\mathsf{MINUS}(\mathsf{s}(x), y) \succsim \mathsf{IF}_{\mathsf{minus}}(\mathsf{le}(\mathsf{s}(x), y), \mathsf{s}(x), y)$$

need not be strict.
By the following mapping

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

the inequalities are satisfied by the recursive path ordering.

8.4. Remainder, Version 1 - 3

Similar to the TRSs for division, we also obtain three versions of the following TRS which again are not simply terminating. We only present one of them.

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

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

The relevant inequalities of this TRS are given by

$$\begin{aligned}
\mathsf{LE}(\mathsf{s}(x), \mathsf{s}(y)) &\succ \mathsf{LE}(x, y) \\
\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) &\succ \mathsf{MINUS}(x, y)
\end{aligned}$$

$$\begin{aligned}
\mathsf{MOD}(\mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{IF}_{\mathsf{mod}}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{IF}_{\mathsf{mod}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) &\succ \mathsf{MOD}(\mathsf{minus}(x, y), \mathsf{s}(y))
\end{aligned}$$

By mapping $\mathsf{minus}(x,y)$, $\mathsf{mod}(x,y)$, $\mathsf{if_{mod}}(b,x,y)$, $\mathsf{MOD}(x,y)$, and $\mathsf{IF_{mod}}(b,x,y)$ to x, the interpreted inequalities are satisfied by the recursive path ordering.

8.5. Greatest Common Divisor, Version 1 - 3

There are also three versions of the following TRS for the computation of the gcd, which are not simply terminating. Again, we only present one of them.

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

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

(Of course we also could have switched the ordering of the arguments in the right-hand side of the last rule. But this version here is even more difficult: Termination of the corresponding algorithm cannot be proved by the method of [Wal94], because this method cannot deal with permutations of arguments.)
The relevant inequalities of this TRS are

$$\begin{aligned}
\mathsf{LE}(\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{LE}(x,y)\\
\mathsf{MINUS}(x,\mathsf{s}(y)) &\succ \mathsf{MINUS}(x,y)
\end{aligned}$$

$$\begin{aligned}
\mathsf{GCD}(\mathsf{s}(x),\mathsf{s}(y)) &\succsim \mathsf{IF_{gcd}}(\mathsf{le}(y,x),\mathsf{s}(x),\mathsf{s}(y))\\
\mathsf{IF_{gcd}}(\mathsf{true},\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{GCD}(\mathsf{minus}(x,y),\mathsf{s}(y))\\
\mathsf{IF_{gcd}}(\mathsf{false},\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{GCD}(\mathsf{minus}(y,x),\mathsf{s}(x))
\end{aligned}$$

A suitable mapping is given by

$$\begin{aligned}
\mathsf{pred}(x) &\mapsto x\\
\mathsf{minus}(x,y) &\mapsto x\\
\mathsf{if_{gcd}}(b,x,y) &\mapsto \mathsf{if_{gcd}}(x,y)\\
\mathsf{IF_{gcd}}(b,x,y) &\mapsto \mathsf{IF_{gcd}}(x,y)
\end{aligned}$$

The interpreted inequalities are satisfied by the recursive path ordering.
This example was taken from [BM79] resp. [Wal91]. A variant of this example could be proved terminating using Steinbach's method for the automated generation of transformation orderings [Ste95a], but there the rules for le and minus were missing.

8.6. Logarithm, Version 1

The following TRS computes the dual logarithm.

$$\mathsf{half}(\mathsf{0}) \to \mathsf{0}$$

$$\begin{aligned} \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) &\to \mathsf{s}(\mathsf{half}(x)) \\ \\ \mathsf{log}(\mathsf{0}) &\to \mathsf{0} \\ \mathsf{log}(\mathsf{s}(\mathsf{s}(x))) &\to \mathsf{s}(\mathsf{log}(\mathsf{s}(\mathsf{half}(x)))) \end{aligned}$$

The relevant inequalities of this TRS are

$$\begin{aligned} \mathsf{HALF}(\mathsf{s}(\mathsf{s}(x))) &\succ \mathsf{HALF}(x) \\ \mathsf{LOG}(\mathsf{s}(\mathsf{s}(x))) &\succ \mathsf{LOG}(\mathsf{s}(\mathsf{half}(x))) \end{aligned}$$

A mapping for the function symbols is not 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.)

8.7. Logarithm, Version 2 - 4

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

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

There are three inequalities obtained from the dependency pairs on a cycle of the dependency graph:

$$\begin{aligned} \mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) &\succ \mathsf{MINUS}(x, y) \\ \mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) &\succ \mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y)) \\ \mathsf{LOG}(\mathsf{s}(\mathsf{s}(x))) &\succ \mathsf{LOG}(\mathsf{s}(\mathsf{quot}(x, \mathsf{s}(\mathsf{s}(\mathsf{0})))))\text{.} \end{aligned}$$

The interpretation to derive a quasi-ordering that satisfies all inequalities is given by: $\mathsf{quot}(x, y)$ and $\mathsf{minus}(x, y)$ are mapped to x.

8.8. Eliminating Duplicates

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

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

$$\begin{aligned}
\mathsf{purge}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{purge}(\mathsf{add}(n,x)) &\to \mathsf{add}(n,\mathsf{purge}(\mathsf{rm}(n,x)))
\end{aligned}$$

The relevant inequalities are

$$\mathsf{EQ}(\mathsf{s}(x),\mathsf{s}(y)) \succ \mathsf{EQ}(x,y)$$

$$\begin{aligned}
\mathsf{RM}(n,\mathsf{add}(m,x)) &\succsim \mathsf{IF}_{\mathsf{rm}}(\mathsf{eq}(n,m),n,\mathsf{add}(m,x))\\
\mathsf{IF}_{\mathsf{rm}}(\mathsf{true},n,\mathsf{add}(m,x)) &\succ \mathsf{RM}(n,x)\\
\mathsf{IF}_{\mathsf{rm}}(\mathsf{false},n,\mathsf{add}(m,x)) &\succ \mathsf{RM}(n,x)
\end{aligned}$$

$$\mathsf{PURGE}(\mathsf{add}(n,x)) \succ \mathsf{PURGE}(\mathsf{rm}(n,x))$$

A suitable mapping is

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

With this interpretation the inequalities are satisfied by the recursive path ordering. This example comes from [Wal91] and a similar example was mentioned in [Ste95a], 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 $\mathbf{n}$ would be replaced by a term containing $\mathsf{add}(n,x)$ then we would obtain a non-simply terminating TRS, but termination could still be proved with our technique in the same way.

8.9. Selection Sort

This TRS from [Wal94], which is a slight modification of the corresponding TRS in [AG96a], 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.

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

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

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

The relevant inequalities are

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

A suitable mapping is given by

$$
\begin{aligned}
\mathsf{element}(x) &\mapsto \mathsf{element}\\
\mathsf{add}(n, x) &\mapsto \mathsf{add}(x)\\
\mathsf{if}_{\mathsf{min}}(b, x) &\mapsto \mathsf{if}_{\mathsf{min}}(x)\\
\mathsf{replace}(x, y, z) &\mapsto z\\
\mathsf{if}_{\mathsf{replace}}(b, x, y, z) &\mapsto z\\
\mathsf{if}_{\mathsf{selsort}}(b, x) &\mapsto \mathsf{if}_{\mathsf{selsort}}(x)\\
\mathsf{MIN}(x) &\mapsto x\\
\mathsf{IF}_{\mathsf{min}}(b, x) &\mapsto x\\
\mathsf{REPLACE}(x, y, z) &\mapsto z\\
\mathsf{IF}_{\mathsf{replace}}(b, x, y, z) &\mapsto z\\
\mathsf{SELSORT}(x) &\mapsto x\\
\mathsf{IF}_{\mathsf{selsort}}(b, x) &\mapsto x
\end{aligned}
$$

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

8.10. Minimum Sort

This TRS can be used to sort a list x by repeatedly removing the minimum of it. For that purpose elements of x are shifted into the second argument of minsort, until the minimum of the list is reached. Then the function rm is used to eliminate *all* occurrences of the minimum and finally minsort is called recursively on the remaining list. Hence, minsort does not only sort a list but it also eliminates duplicates. (Of

course, the corresponding version of minsort where duplicates are not eliminated could also be proved terminating with our method.)

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

$$
\begin{aligned}
\mathsf{minsort}(\mathsf{nil},\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{minsort}(\mathsf{add}(n,x),y) &\to \mathsf{if}_{\mathsf{minsort}}(\mathsf{eq}(n,\mathsf{min}(\mathsf{add}(n,x))),\mathsf{add}(n,x),y)\\
\mathsf{if}_{\mathsf{minsort}}(\mathsf{true},\mathsf{add}(n,x),y) &\to \mathsf{add}(n,\mathsf{minsort}(\mathsf{app}(\mathsf{rm}(n,x),y),\mathsf{nil}))\\
\mathsf{if}_{\mathsf{minsort}}(\mathsf{false},\mathsf{add}(n,x),y) &\to \mathsf{minsort}(x,\mathsf{add}(n,y))
\end{aligned}
$$

The relevant inequalities of this TRS are given by

$$
\begin{aligned}
\mathsf{EQ}(\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{EQ}(x,y)\\
\mathsf{LE}(\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{LE}(x,y)\\
\mathsf{APP}(\mathsf{add}(n,x),y) &\succ \mathsf{APP}(x,y)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{MIN}(\mathsf{add}(n,\mathsf{add}(m,x))) &\succsim \mathsf{IF}_{\mathsf{min}}(\mathsf{le}(n,m),\mathsf{add}(n,\mathsf{add}(m,x)))\\
\mathsf{IF}_{\mathsf{min}}(\mathsf{true},\mathsf{add}(n,\mathsf{add}(m,x))) &\succ \mathsf{MIN}(\mathsf{add}(n,x))\\
\mathsf{IF}_{\mathsf{min}}(\mathsf{false},\mathsf{add}(n,\mathsf{add}(m,x))) &\succ \mathsf{MIN}(\mathsf{add}(m,x))
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{RM}(n,\mathsf{add}(m,x)) &\succsim \mathsf{IF}_{\mathsf{rm}}(\mathsf{eq}(n,m),n,\mathsf{add}(m,x))\\
\mathsf{IF}_{\mathsf{rm}}(\mathsf{true},n,\mathsf{add}(m,x)) &\succ \mathsf{RM}(n,x)\\
\mathsf{IF}_{\mathsf{rm}}(\mathsf{false},n,\mathsf{add}(m,x)) &\succ \mathsf{RM}(n,x)
\end{aligned}
$$

$$
\begin{aligned}
\mathsf{MINSORT}(\mathsf{add}(n,x),y) &\succ \mathsf{IF}_{\mathsf{minsort}}(\mathsf{eq}(n,\mathsf{min}(\mathsf{add}(n,x))),\mathsf{add}(n,x),y)\\
\mathsf{IF}_{\mathsf{minsort}}(\mathsf{true},\mathsf{add}(n,x),y) &\succsim \mathsf{MINSORT}(\mathsf{app}(\mathsf{rm}(n,x),y),\mathsf{nil})\\
\mathsf{IF}_{\mathsf{minsort}}(\mathsf{false},\mathsf{add}(n,x),y) &\succsim \mathsf{MINSORT}(x,\mathsf{add}(n,y))
\end{aligned}
$$

The synthesized weakly monotonic ordering is a polynomial ordering with interpretations where false, true, 0, nil, eq and le are mapped to 0, $\mathsf{s}(x)$ is mapped to $x+1$, $\mathsf{min}(x)$, $\mathsf{if_{min}}(b,x)$, $\mathsf{EQ}(x,y)$, $\mathsf{LE}(x,y)$, $\mathsf{MIN}(x)$ 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, $\mathsf{minsort}(x,y)$, $\mathsf{if_{minsort}}(b,x,y)$ 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 [BM79] and [Wal94]. In the corresponding example from [Ste92] the rules for le, eq, $\mathsf{if_{rm}}$ and $\mathsf{if_{min}}$ were missing.

8.11. Quicksort

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

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

$$
\begin{aligned}
\mathsf{quicksort}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{quicksort}(\mathsf{add}(n,x)) &\to \mathsf{app}(\mathsf{quicksort}(\mathsf{low}(n,x)),\\
&\qquad \mathsf{add}(n,\mathsf{quicksort}(\mathsf{high}(n,x))))
\end{aligned}
$$

The relevant inequalities are

$$
\begin{aligned}
\mathsf{LE}(\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{LE}(x,y)\\
\mathsf{APP}(\mathsf{add}(n,x),y) &\succ \mathsf{APP}(x,y)\\
\mathsf{LOW}(n,\mathsf{add}(m,x)) &\succsim \mathsf{IF_{low}}(\mathsf{le}(m,n),n,\mathsf{add}(m,x))\\
\mathsf{IF_{low}}(\mathsf{true},n,\mathsf{add}(m,x)) &\succ \mathsf{LOW}(n,x)\\
\mathsf{IF_{low}}(\mathsf{false},n,\mathsf{add}(m,x)) &\succ \mathsf{LOW}(n,x)\\
\mathsf{HIGH}(n,\mathsf{add}(m,x)) &\succsim \mathsf{IF_{high}}(\mathsf{le}(m,n),n,\mathsf{add}(m,x))\\
\mathsf{IF_{high}}(\mathsf{true},n,\mathsf{add}(m,x)) &\succ \mathsf{HIGH}(n,x)\\
\mathsf{IF_{high}}(\mathsf{false},n,\mathsf{add}(m,x)) &\succ \mathsf{HIGH}(n,x)\\
\mathsf{QUICKSORT}(\mathsf{add}(n,x)) &\succ \mathsf{QUICKSORT}(\mathsf{low}(n,x))\\
\mathsf{QUICKSORT}(\mathsf{add}(n,x)) &\succ \mathsf{QUICKSORT}(\mathsf{high}(n,x))
\end{aligned}
$$

A suitable mapping is

$$\begin{aligned}
\mathsf{low}(n,x) &\mapsto x\\
\mathsf{high}(n,x) &\mapsto x\\
\mathsf{if}_{\mathsf{low}}(b,n,x) &\mapsto x\\
\mathsf{if}_{\mathsf{high}}(b,n,x) &\mapsto x\\
\mathsf{IF}_{\mathsf{low}}(b,n,x) &\mapsto \mathsf{IF}_{\mathsf{low}}(n,x)\\
\mathsf{IF}_{\mathsf{high}}(b,n,x) &\mapsto \mathsf{IF}_{\mathsf{high}}(n,x)
\end{aligned}$$

This interpretation and the recursive path ordering satisfy the demands on the ordering.
Steinbach could prove termination of a corresponding example with transformation orderings [Ste95a], but in his example the rules for le, $\mathsf{if}_{\mathsf{low}}$ $\mathsf{if}_{\mathsf{high}}$ 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 was replaced by a term containing $\mathsf{add}(n,x)$ then we would obtain a non-simply terminating TRS. With our method, termination could still be proved in the same way.

8.12. Permutation of Lists

This example is a TRS from [Wal94] to compute a permutation of a list, for instance, $\mathsf{shuffle}([1,2,3,4,5])$ reduces to $[1,5,2,4,3]$.

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

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

$$\begin{aligned}
\mathsf{APP}(\mathsf{add}(n,x),y) &\succ \mathsf{APP}(x,y)\\
\mathsf{REVERSE}(\mathsf{add}(n,x)) &\succ \mathsf{REVERSE}(x)\\
\mathsf{SHUFFLE}(\mathsf{add}(n,x)) &\succ \mathsf{SHUFFLE}(\mathsf{reverse}(x))
\end{aligned}$$

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 $x+y$.

8.13. 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 TRS of this example from [Gie95a]. 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.

$$
\begin{aligned}
\mathsf{eq}(0,0) &\to \mathsf{true}\\
\mathsf{eq}(0,\mathsf{s}(x)) &\to \mathsf{false}\\
\mathsf{eq}(\mathsf{s}(x),0) &\to \mathsf{false}\\
\mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) &\to \mathsf{eq}(x,y)\\
\mathsf{or}(\mathsf{true},x) &\to \mathsf{true}\\
\mathsf{or}(\mathsf{false},\mathsf{true}) &\to \mathsf{true}\\
\mathsf{or}(\mathsf{false},\mathsf{false}) &\to \mathsf{false}\\
\mathsf{union}(\epsilon,h) &\to h\\
\mathsf{union}(\mathsf{edge}(x,y,i),h) &\to \mathsf{edge}(x,y,\mathsf{union}(i,h))
\end{aligned}
$$

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

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

$$
\begin{aligned}
\mathsf{EQ}(\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{EQ}(x,y)\\
\mathsf{UNION}(\mathsf{edge}(x,y,i),h) &\succ \mathsf{UNION}(i,h)\\
\mathsf{REACH}(x,y,\mathsf{edge}(u,v,i),h) &\succsim \mathsf{IF}_{\mathsf{reach\_1}}(\mathsf{eq}(x,u),x,y,\mathsf{edge}(u,v,i),h)\\
\mathsf{IF}_{\mathsf{reach\_1}}(\mathsf{true},x,y,\mathsf{edge}(u,v,i),h) &\succsim \mathsf{IF}_{\mathsf{reach\_2}}(\mathsf{eq}(y,v),x,y,\mathsf{edge}(u,v,i),h)\\
\mathsf{IF}_{\mathsf{reach\_2}}(\mathsf{false},x,y,\mathsf{edge}(u,v,i),h) &\succ \mathsf{REACH}(x,y,i,h)\\
\mathsf{IF}_{\mathsf{reach\_2}}(\mathsf{false},x,y,\mathsf{edge}(u,v,i),h) &\succ \mathsf{REACH}(v,y,\mathsf{union}(i,h),\epsilon)\\
\mathsf{IF}_{\mathsf{reach\_1}}(\mathsf{false},x,y,\mathsf{edge}(u,v,i),h) &\succ \mathsf{REACH}(x,y,i,\mathsf{edge}(u,v,h))
\end{aligned}
$$

A mapping to polynomials results in a suitable ordering. The interpretation is: $\mathsf{eq}(x,y)$, $\mathsf{or}(x,y)$, true, false, ϵ and 0 are mapped to 0, $\mathsf{s}(x)$ 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 $(g+h)^2+2g+h$.

8.14. 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 new tree with the two direct subtrees u and v.

$$\begin{aligned}
\mathsf{concat}(\mathsf{leaf}, y) &\to y \\
\mathsf{concat}(\mathsf{cons}(u, v), y) &\to \mathsf{cons}(u, \mathsf{concat}(v, y))
\end{aligned}$$

$$\begin{aligned}
\mathsf{less_leaves}(x, \mathsf{leaf}) &\to \mathsf{false} \\
\mathsf{less_leaves}(\mathsf{leaf}, \mathsf{cons}(w, z)) &\to \mathsf{true} \\
\mathsf{less_leaves}(\mathsf{cons}(u, v), \mathsf{cons}(w, z)) &\to \mathsf{less_leaves}(\mathsf{concat}(u, v), \mathsf{concat}(w, z))
\end{aligned}$$

The inequalities corresponding to the dependency pairs that are on a cycle of the dependency graph are:

$$\begin{aligned}
\mathsf{CONCAT}(\mathsf{cons}(u, v), y) &\succ \mathsf{CONCAT}(v, y) \\
\mathsf{LESS_LEAVES}(\mathsf{cons}(u, v), \mathsf{cons}(w, z)) &\succ \mathsf{LESS_LEAVES}(\mathsf{concat}(u, v), \mathsf{concat}(w, z))
\end{aligned}$$

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 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.

8.15. Average of Naturals

The following overlay system, which computes the average of two numbers [DH95], is locally confluent and therefore innermost termination suffices for proving termination. However, the technique presented in this paper can prove termination directly.

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

For proving termination of this TRS the inequalities corresponding to the rewrite rules and the two strict inequalities corresponding to the dependency pairs on a cycle

$$\begin{aligned}
\mathsf{AVERAGE}(\mathsf{s}(x), y) &\succ \mathsf{AVERAGE}(x, \mathsf{s}(y)) \\
\mathsf{AVERAGE}(x, \mathsf{s}(\mathsf{s}(\mathsf{s}(y)))) &\succ \mathsf{AVERAGE}(\mathsf{s}(x), y))
\end{aligned}$$

should be satisfied by a well-founded weakly monotonic quasi-ordering.
In this way, termination of this TRS is easily proved by mapping 0 to 0, $\mathsf{s}(x)$ to $x + 1$, $\mathsf{average}(x, y)$ to $x + y$, and $\mathsf{AVERAGE}(x, y)$ to $2x + y$.

8.16. Plus and Times

The following TRS [DH95] is again a locally confluent overlay system. To ease readability we use an infix notation for $+$ and $\times$.

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

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

Applying the technique results in a set of inequalities satisfied by the automatically synthesized 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, and $x \times y$ is mapped to the product of x and y.

8.17. Summing Elements of Lists

This TRS, which has overlapping rules, is the leading example of the paper.

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

While this system is not simply terminating, the inequalities generated by our method 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 $\mathsf{PLUS}(x, y)$ is mapped to x.

8.18. Sum and Predecessor

Note that termination of the preceding TRS can also be proved without using dependency graphs, i.e. one can also verify its termination by using the method of Sect. 4 only. However, then the resulting inequalities are not satisfied by any polynomial (or simplification) ordering. Instead, one has to use a *lexicographic* combination of a polynomial ordering where $x.\, l$ is mapped to $x + l$ and the mentioned polynomial ordering.
However, if we add the following rules to the system, then this lexicographic ordering does no longer satisfy the inequalities resulting from the method of Sect. 4. Here, pred is a predecessor function on one-element lists.

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

With the concept of connectable terms we can easily determine that the dependency pair

$$\langle \mathsf{SUM}(\mathsf{0}.\, x + y.\, l), \mathsf{SUM}(\mathsf{s}(x).\, y.\, l) \rangle$$

is not on a cycle of the dependency graph. The reason is that $\mathsf{s}(x)$ does not unify with $\mathsf{0}$. Hence, $\mathsf{SUM}(\mathsf{s}(x).\, y.\, l)$ is not connectable to $\mathsf{SUM}(\mathsf{0}.\, x' + y'.\, l')$. To satisfy the resulting inequalities, we have to extend the above polynomial interpretation by mapping pred to the constant 1.

8.19. Addition and Subtraction

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

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

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

Again, this system is overlapping and not simply terminating. However, the inequalities obtained by our approach are satisfied by the lexicographic path ordering, if $\mathsf{minus}(x, y)$ is replaced by x. For that purpose we have to use a precedence where double and plus are greater than s.

8.20. Addition with Nested Recursion – Version 1

We may also add an additional rule to the above system which turns it 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 by the interpretation and the lexicographic path ordering mentioned above.

8.21. Addition with Nested Recursion – Version 2

The following alternative TRS for addition from [Ste92] 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 our method. In this way, termination can easily be proved.

8.22. Multiplication and Addition

The following example is taken from [Der87, p. 101].

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

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

$$\mathsf{TIMES}(x, y+1) \succ \mathsf{TIMES}(x, y + (1 \times 0))$$

This system is not simply terminating (and in [Der87] it is used to illustrate the use of the semantic path ordering). However, with our method 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.

8.23. Extended Multiplication and Addition

Similarly we can also prove termination of the following 'extended' version of the above system where the full rules for $+$ and $\times$ are added. Again, this system is not an overlay system any more.

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

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

$$\begin{array}{rcl} \mathsf{PLUS}(x, \mathsf{s}(y)) & \succ & \mathsf{PLUS}(x, y) \\ \mathsf{TIMES}(x, y + \mathsf{s}(z)) & \succ & \mathsf{TIMES}(x, y + (\mathsf{s}(z) \times 0)) \\ \mathsf{TIMES}(x, \mathsf{s}(y)) & \succ & \mathsf{TIMES}(x, y) \\ \mathsf{TIMES}(x, y + \mathsf{s}(z)) & \succsim & \mathsf{TIMES}(x, \mathsf{s}(z)) \end{array}$$

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

8.24. Nested Recursion 1

The following system was introduced in [Gie96, 'nest2'] as an example for a small TRS with nested recursion where all simplification orderings fail.

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

With our approach, however, an automated termination proof is directly possible. For instance, we may use a polynomial ordering where 0 and s are interpreted as usual and both $\mathsf{f}(x, y)$ and $\mathsf{F}(x, y)$ are mapped to x.

8.25. Nested Recursion 2

This system (by *Christoph Walther*), which is similar system to the preceding one, has been examined in [Ste92].

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

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.

8.26. Nested Recursion 3

As an example of a string rewrite system with minimal ordinal ω^ω associated to it, *Hans Zantema* and *Maria Ferreira* presented the following TRS [FZ93].

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

The inequalities corresponding to this system, except for the inequalities corresponding to the two rules, are

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

All inequalities are satisfied by the polynomial interpretation mapping $\mathsf{f}(x)$ and $\mathsf{F}(x)$ to x, $\mathsf{h}(x)$ to 0 and $\mathsf{g}(x)$ to $x+1$.

8.27. Nested Recursion 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.

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

The inequalities to satisfy are

$$\begin{aligned} \mathsf{f}(x) &\succsim \mathsf{s}(x) \\ \mathsf{f}(\mathsf{s}(\mathsf{s}(x))) &\succsim \mathsf{s}(\mathsf{f}(\mathsf{f}(x))) \\ \mathsf{F}(\mathsf{s}(\mathsf{s}(x))) &\succ \mathsf{F}(x) \\ \mathsf{F}(\mathsf{s}(\mathsf{s}(x))) &\succ \mathsf{F}(\mathsf{f}(x)) \end{aligned}$$

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

8.28. Nested Symbols on Left-hand Sides

The following example is from [Der93]. It has been proved terminating by a lexicographic combination of two orderings.

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

By using the technique of dependency pairs, the resulting inequalities are satisfied by the mapping of $\mathsf{f}(x)$ and $\mathsf{g}(x)$ to $x+1$ and mapping $\mathsf{F}(x)$ and $\mathsf{G}(x)$ to x. In this way, the dependency pair $\langle \mathsf{F}(\mathsf{f}(x)), \mathsf{G}(\mathsf{f}(x)) \rangle$ does not need to be strictly decreasing, but $\langle \mathsf{G}(\mathsf{g}(x)), \mathsf{F}(x) \rangle$ is strictly decreasing.

8.29. Nested Symbols on Both Sides of Rules

The following TRS cannot be shown terminating by the lexicographic path ordering and is therefore one of the systems for which the semantic path ordering has been used in literature [Der93]. However, the system can be shown to terminate using the lexicographic path ordering after applying the described 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.

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

Apart from the three inequalities corresponding to the rewrite rules, by using the technique four other inequalities are obtained from the cycles of the dependency graph.

$$\begin{array}{rcl} \mathsf{TIMES}(x \times y, z) & \succ & \mathsf{TIMES}(y, z) \\ \mathsf{TIMES}(x \times y, z) & \succ & \mathsf{TIMES}(x, y \times z) \\ \mathsf{TIMES}(x + y, z) & \succ & \mathsf{TIMES}(x, z) \\ \mathsf{TIMES}(x + y, z) & \succ & \mathsf{TIMES}(y, z) \end{array}$$

The seven inequalities are satisfied by the lexicographic path ordering if we map $g(z, y)$ to z.

8.30. A System which is not left-linear

The following TRS, originally from Geerling [Gee91], cannot be proved terminating by the recursive path ordering (but one needs a generalisation of the recursive path ordering as defined in [Fer95]). 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 mapping of $\mathsf{f}(x, y, z)$ to $x + y$ and the mapping of $\mathsf{F}(x, y, z)$ to $x + y$ satisfies the two inequalities obtained by the technique.

8.31. Systems without Cycles in Dependency Graphs 1

The following system is from [Ste92].

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

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 $\mathsf{f}(\mathsf{c}, \mathsf{d})$.

8.32. Systems without Cycles in Dependency Graphs 2

Another example in which the dependency graph plays an important role is a TRS introduced in [FZ95] 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 $\mathsf{a}(x, y)$ does not unify with $\mathsf{g}(x)$, the only two inequalities to satisfy are

$$\begin{array}{lcl} \mathsf{f}(\mathsf{g}(x)) & \succsim & \mathsf{f}(\mathsf{a}(\mathsf{g}(\mathsf{g}(\mathsf{f}(x))), \mathsf{g}(\mathsf{f}(x)))) \\ \mathsf{F}(\mathsf{g}(x)) & \succ & \mathsf{F}(x) \end{array}$$

which are easily solved by mapping $\mathsf{a}(x, y)$ to 0, $\mathsf{F}(x)$ to x, and $\mathsf{g}(x)$ to $x + 1$.

8.33. A TRS which is not totally terminating 1

The most famous example of a TRS that is terminating, but not *totally* terminating is the following [Der87].

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

With our approach, termination of this system is again obvious, because the dependency graph does not contain any cycles (as $\mathsf{F}(\mathsf{b})$ is not connectable to $\mathsf{F}(\mathsf{a})$ and $\mathsf{G}(\mathsf{a})$ is not connectable to $\mathsf{G}(\mathsf{b})$). Hence, the set of constraints generated by our method are the two inequalities

$$\begin{array}{lcl} \mathsf{f}(\mathsf{a}) & \succsim & \mathsf{f}(\mathsf{b}) \\ \mathsf{g}(\mathsf{b}) & \succsim & \mathsf{g}(\mathsf{a}) \end{array}$$

corresponding to the two rewrite rules. The mapping of $\mathsf{f}(x)$ and $\mathsf{g}(x)$ to 0 satisfies these two inequalities.

8.34. A TRS which is not totally terminating 2

A TRS introduced in [Fer95] 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:

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

Termination is trivially concluded from the fact that there are no cycles in the dependency graph together with the mapping of $\mathsf{p}(x)$ and $\mathsf{q}(x)$ to 0.

8.35. Systems with 'Undefined' Function Symbols

The following well-known system from [Der87] is one of the smallest non-simply terminating TRSs.

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

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, we can also prove termination of the system

$$\mathsf{f}(\mathsf{g}(x)) \;\to\; \mathsf{f}(\mathsf{h}(\mathsf{g}(x))).$$

from [BL88].

8.36. Reversing Lists

The following system is a slight variant of a TRS proposed in [HH82, 'brev']. Given a list $x.l$, the function rev calls two other functions rev1 and 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.

$$\begin{aligned}
\mathsf{rev}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{rev}(x.l) &\to \mathsf{rev1}(x,l).\mathsf{rev2}(x,l)\\[1ex]
\mathsf{rev1}(0,\mathsf{nil}) &\to 0\\
\mathsf{rev1}(\mathsf{s}(x),\mathsf{nil}) &\to \mathsf{s}(x)\\
\mathsf{rev1}(x,y.l) &\to \mathsf{rev1}(y,l)\\[1ex]
\mathsf{rev2}(x,\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{rev2}(x,y.l) &\to \mathsf{rev}(x.\mathsf{rev}(\mathsf{rev2}(y,l)))
\end{aligned}$$

The resulting inequalities are satisfied by a polynomial ordering, where nil is mapped to 0, $x.l$ is mapped to $l+1$, $\mathsf{rev}(l)$ is mapped to l, the symbols $\mathsf{rev1}(x,l)$, 0 and $\mathsf{s}(x)$ are all mapped to the constant 0, and $\mathsf{rev2}(x,l)$ is mapped to l. The tuple symbol $\mathsf{REV}(l)$ is mapped to the identity and both $\mathsf{REV1}(x,l)$ and $\mathsf{REV2}(x,l)$ are mapped to l. In this way the following two dependency pairs

$$\begin{gathered}
\langle \mathsf{REV}(x.l), \mathsf{REV2}(x,l)\rangle\\
\langle \mathsf{REV2}(x,y.l), \mathsf{REV}(x.\mathsf{rev}(\mathsf{rev2}(y,l)))\rangle
\end{gathered}$$

which form a cycle of the dependency graph are both decreasing, but only the first one is strictly decreasing.

References

[AG96a] T. Arts and J. Giesl. Termination of constructor systems. Technical Report UU-CS-1996-07, Utrecht University, PO box 80.089, 3508 TB Utrecht, February 1996.

[AG96b] T. Arts and J. Giesl. Termination of constructor systems. In Harald Ganzinger, editor, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications, RTA-96*, volume 1103 of *Lecture Notes in Computer Science*, pages 63–77, New Brunswick, NJ, USA, July 1996. Springer Verlag, Berlin.

[AG96c] T. Arts and J. Giesl. Proving innermost normalisation automatically. Technical Report IBN 96/39, Technische Hochschule Darmstadt, Germany, October 1996.

[Art96] T. Arts. Termination by absence of infinite chains of dependency pairs. In Hélène Kirchner, editor, *Proceedings of the 21st International Colloquium on Trees in Algebra and Programming, CAAP'96*, volume 1059 of *Lecture Notes in Computer Science*, pages 196–210, Linköping, Sweden, April 1996. Springer Verlag, Berlin.

[BD86] L. Bachmair and N. Dershowitz. Commutation, transformation and termination. In Jörg H. Siekmann, editor, *Proceedings of the 8th International Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20, Oxford, England, July 1986. Springer Verlag, Berlin.

[BL87] A. Ben Cherifa and P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9:137–159, 1987.

[BL88] F. Bellegarde and P. Lescanne. Termination proofs based on transformation techniques. Technical report, Centre de Recherche en Informatique de Nancy, Nancy (France), March 1988.

[BL90] F. Bellegarde and P. Lescanne. Termination by completion. *Applicable Algebra in Engineering, Communication and Computing*, 1:79–96, 1990.

[BL93] E. Bevers and J. Lewi. Proving termination of (conditional) rewrite systems. *Acta Informatica*, 30:537–568, 1993.

[BM79] R.S. Boyer and J S. Moore. *A Computational Logic*. Academic Press, 1979.

[Der79] N. Dershowitz. A note on simplification orderings. *Information Processing Letters*, 9(5):212–215, 1979.

[Der81] N. Dershowitz. Termination of linear rewriting systems. In S. Even and O. Kariv, editors, *Proceedings of the 8th International Colloquium on Automata, Languages and Programming (ALP)*, volume 115 of *Lecture Notes in Computer Science*, pages 448–458, Acre, Israel, July 1981. Springer Verlag, Berlin.

[Der82] N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.

[Der87] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1 and 2):69–116, 1987.

[Der93] N. Dershowitz. 33 examples of termination. In Hubert Comon and Jean-Pierre Jouannaud, editors, *Term Rewriting, Proceedings Spring School of Theoretical Computer Science*, volume 909 of *Lecture Notes in Computer Science*, pages 16–27, Font Romeux, France, May 1993. Springer Verlag, Berlin.

[DH95] N. Dershowitz and C. Hoot. Natural termination. *Theoretical Computer Science*, 142(2):179–207, 1995.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 243–320. North-Holland, 1990.

[DKM90] J. Dick, J. Kalmus, and U. Martin. Automating the Knuth Bendix ordering. *Acta Informatica*, 28:95–119, 1990.

[Fer95] M. Ferreira. *Termination of Term Rewriting,Well-foundedness, Totality and Transformations.* PhD thesis, Utrecht University, PO Box 80.089, 3508 TB Utrecht, The Netherlands, 1995.

[FZ93] M. Ferreira and H. Zantema. Total termination of term rewriting. In Claude Kirchner, editor, *Proceedings of the 5th Conference on Rewrite Techniques and Applications, RTA-93*, volume 690 of *Lecture Notes in Computer Science*, pages 213–227, Montreal, Canada, June 1993. Springer Verlag, Berlin.

[FZ95] M. Ferreira and H. Zantema. Dummy elimination: making termination easier. In Horst Reichel, editor, *Proceedings of the 10th International Conference on Fundamentals of Computation Theory, FCT'95*, volume 965 of *Lecture Notes in Computer Science*, pages 243–252, Dresden, Germany, August 1995. Springer Verlag, Berlin.

[Gee91] M. Geerling. Termination of term rewriting systems. Master's thesis, Utrecht University, PO Box 80.089, 3508 TB Utrecht, The Netherlands, 1991.

[Geu89] O. Geupel. Overlap closures and termination of term rewriting systems. Technical Report MIP-8922 283, Universität Passau, Passau, Germany, 1989.

[Gie95a] J. Giesl. *Automatisierung von Terminierungsbeweisen für rekursiv definierte Algorithmen.* PhD thesis, Technische Hochschule Darmstadt, Germany, 1995. In German.

[Gie95b] J. Giesl. Generating polynomial orderings for termination proofs. In Jieh Hsiang, editor, *Proceedings of the 6th International Conference on Rewriting Techniques and Applications, RTA-95*, volume 914 of *Lecture Notes in Computer Science*, pages 426–431, Kaiserslautern, Germany, April 1995. Springer Verlag, Berlin.

[Gie96] J. Giesl. Termination of nested and mutually recursive algorithms. *Journal of Automated Reasoning*, 1996. To appear.

[HH82] G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25:239–299, 1982.

[HL78] G. Huet and D. Lankford. On the uniform halting problem for term rewriting systems. Technical Report 283, INRIA, Le Chesnay, France, 1978.

[KB70] D. E. Knuth and P. B. Bendix. Simple word problems in universal algebras. *Computational problems in abstract algebra*, pages 263–297, 1970.

[KL80] S. Kamin and J.-J. Levy. Two generalizations of the recursive path ordering. Department of Computer Science, University of Illinois, IL, 1980.

[Kri95] M.R.K. Krishna Rao. Modular proofs for completeness of hierarchical term rewriting systems. *Theoretical Computer Science*, 151:487–512, 1995.

[Lan79] D. S. Lankford. On proving term rewriting systems are noetherian. Technical Report Memo MTP-3, Louisiana Tech. University, Ruston, LA, 1979.

[LM78] D. S. Lankford and D. R. Musser. A finite termination criterion, 1978.

[Pla78] D. A. Plaisted. A recursively defined ordering for proving termination of term rewriting systems. Technical Report R-78-943, Dept. of Computer Science, University of Illinois, Urbana, IL, 1978.

[Ste92] J. Steinbach. Notes on transformation orderings. Technical Report SR-92-23, Universität Kaiserslautern, Kaiserslautern, Germany, 1992.

[Ste94] J. Steinbach. Generating polynomial orderings. *Information Processing Letters*, 49:85–93, 1994.

[Ste95a] J. Steinbach. Automatic termination proofs with transformation orderings. In Jieh Hsiang, editor, *Proceedings of the 6th International Conference on Rewriting Techniques and Applications, RTA-95*, volume 914 of *Lecture Notes in Computer Science*, pages 11–25, Kaiserslautern, Germany, April 1995. Springer Verlag, Berlin.

[Ste95b] J. Steinbach. Simplification orderings: history of results. *Fundamenta Informaticae*, 24:47–87, 1995.

[Toy87] Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.

[Wal91] C. Walther. *Automatisierung von Terminierungsbeweisen*. Vieweg Verlag, Braunschweig, 1991.

[Wal94] C. Walther. On proving the termination of algorithms by machine. *Artificial Intelligence*, 71(1):101–157, 1994.

[Zan94] H. Zantema. Termination of term rewriting: interpretation and type elimination. *Journal of Symbolic Computation*, 17:23–50, 1994.

[Zan95] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.