Automatically Proving Termination Where Simplification Orderings Fail*

Thomas Arts[1] and Jürgen Giesl[2]

[1] Dept. of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands, E-mail: thomas@cs.ruu.nl

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

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 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) TRSs 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 [DH95, Ste95b], forward closures [LM78, DH95], semantic interpretations [Lan79, BL87, Ste94, Zan94, Gie95], transformation orderings [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. The formal definitions needed 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 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. For a collection

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

of examples see [AG96b]. In Sect. 6 we give some comments on related work followed by a short conclusion in Sect. 7.

2 Dependency Pairs

For *constructor systems* it is common to split the signature into two disjoint sets, the *defined symbols* and the *constructors*. The following definition extends these notions to arbitrary term rewriting systems $\mathcal{R}(\mathcal{F}, R)$ (with the rules R over a signature $\mathcal{F}$). Here, the *root* of a term $f(\ldots)$ is the leading function symbol f.

Definition 1 (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 $\{\mathrm{root}(l) \mid l \to r \in R\}$ 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 $\mathcal{R}(D, C, R)$. As an example consider the following TRS with the defined symbols app and sum and the constructors nil, '.', and +. Here, $x.l$ represents the insertion of a number x into a list l (where $x.y.l$ abbreviates $(x.(y.l))$), app computes the concatenation of lists, and $\mathsf{sum}(l)$ is used to compute the sum of all numbers in l (e.g. sum applied to the list $[1, 2, 3]$ returns $[1+2+3]$).

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

Unfortunately, most methods for automated termination proofs are restricted to *simplification orderings* [Der87, Ste95b]. These methods cannot prove termination of systems like the TRS above, because the left-hand side of the last sum-rule is homeomorphically embedded in its right-hand side.

Previous methods for proving termination usually tried to find a well-founded ordering such that left-hand sides of rules were 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. Hence, 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 $C[g(t_1, \ldots, t_m)]$ (where f and g are defined symbols and C denotes some context), then to prove termination we compare the argument tuples $s_1, \ldots, s_n$ and $t_1, \ldots, t_m$. 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 that the signature $\mathcal{F}$ consists of lower case function symbols only and denote the special symbols by the corresponding upper case symbols.

Definition 2 (Dependency Pairs). If $f(s_1, \ldots, s_n) \to C[g(t_1, \ldots, t_m)]$ is a rewrite rule of the TRS $\mathcal{R}(D, C, R)$, then $\langle F(s_1, \ldots, s_n), G(t_1, \ldots, t_m)\rangle$ is a *dependency pair* of $\mathcal{R}$.

In our example we obtain the following dependency pairs:

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

3 A Termination Criterion Using Dependency Pairs

Using the notion of dependency pairs we now 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 those subterms of the right-hand sides that have a defined root symbol. By regarding sequences of dependency pairs, the introduction of new defined symbols can be traced. This observation leads to the following definition.

Definition 3 ($\mathcal{R}$-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.

We always assume that two (occurrences of) dependency pairs have disjoint variables. Then 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'. If $\mathcal{R}$ is clear from the context, then we often write 'chain' instead of '$\mathcal{R}$-chain'. The following theorem proves that the absence of infinite chains is a sufficient and necessary criterion for termination.

Theorem 4 (Termination Criterion). *A TRS $\mathcal{R}$ is terminating if and only if no infinite $\mathcal{R}$-chain exists.*

Proof. **Sufficient Criterion**
We prove that any infinite reduction results in an infinite $\mathcal{R}$-chain.

Let t be a term that starts an infinite reduction. Any such term t contains a subterm[2] $f_1(\mathbf{u}_1)$ that starts an infinite reduction, but none of the terms $\mathbf{u}_1$ starts an infinite reduction, i.e. $\mathbf{u}_1$ are strongly normalising.

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

By assumption there exists no infinite reduction beginning with one of the terms $\mathbf{v}_1 = \mathbf{w}_1\sigma_1$. Hence, for all variables x occurring in $f_1(\mathbf{w}_1)$ the term $\sigma_1(x)$ is

[1] Throughout the paper we regard substitutions whose domain may be *infinite*.
[2] We denote tuples of terms $t_1,\ldots,t_n$ by $\mathbf{t}$.

strongly normalising. Thus, since $r_1\sigma_1$ starts an infinite reduction, there occurs a subterm $f_2(\mathbf{u}_2)$ in r_1, i.e. $r_1 = C[f_2(\mathbf{u}_2)]$ for some context C, such that $f_2(\mathbf{u}_2)\sigma_1$ starts an infinite reduction and $\mathbf{u}_2\sigma_1$ are strongly normalising terms.

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

Similar to the observations above, since $r_i\sigma_i$ starts an infinite reduction, there must be a subterm $f_{i+1}(\mathbf{u}_{i+1})$ in r_i such that $f_{i+1}(\mathbf{u}_{i+1})\sigma_i$ starts an infinite reduction and $\mathbf{u}_{i+1}\sigma_i$ are strongly normalising terms. This results in the i-th dependency pair of the $\mathcal{R}$-chain, viz. $\langle F_i(\mathbf{w}_i), F_{i+1}(\mathbf{u}_{i+1})\rangle$. In this way, one obtains the infinite sequence

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

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

Note that $F_i(\mathbf{u}_i\sigma_{i-1}) \to_{\mathcal{R}}^* F_i(\mathbf{v}_i)$ and $\mathbf{v}_i = \mathbf{w}_i\sigma_i$. Since we assume, without loss of generality, that the variables of consecutive dependency pairs are disjoint, we obtain one substitution $\sigma = \sigma_1 \circ \sigma_2 \circ \ldots$ such that $F_i(\mathbf{u}_i)\sigma \to_{\mathcal{R}}^* F_i(\mathbf{w}_i)\sigma$ for all i. Thus, we have in fact constructed an infinite $\mathcal{R}$-chain.

Necessary Criterion
We prove that any infinite $\mathcal{R}$-chain corresponds to an infinite reduction. Assume there exists an infinite $\mathcal{R}$-chain $\langle F_1(\mathbf{s}_1), F_2(\mathbf{t}_2)\rangle\ \langle F_2(\mathbf{s}_2), F_3(\mathbf{t}_3)\rangle\ \langle F_3(\mathbf{s}_3), F_4(\mathbf{t}_4)\rangle \ldots$ Hence, there must be a substitution σ such that

$$F_2(\mathbf{t}_2)\sigma \to_{\mathcal{R}}^* F_2(\mathbf{s}_2)\sigma,\ F_3(\mathbf{t}_3)\sigma \to_{\mathcal{R}}^* F_3(\mathbf{s}_3)\sigma, \ldots,$$

resp. $f_i(\mathbf{t}_i)\sigma \to_{\mathcal{R}}^* f_i(\mathbf{s}_i)\sigma$, as the upper case symbols F_i are not defined.

Every dependency pair $\langle F(\mathbf{s}), G(\mathbf{t})\rangle$ corresponds to a rewrite rule $f(\mathbf{s}) \to C[g(\mathbf{t})]$ for some context C. Therefore we obtain the following infinite reduction.

$$f_1(\mathbf{s}_1)\sigma \to_{\mathcal{R}} C_1[f_2(\mathbf{t}_2)]\sigma \to_{\mathcal{R}}^* C_1[f_2(\mathbf{s}_2)]\sigma \to_{\mathcal{R}} C_1[C_2[f_3(\mathbf{t}_3)]]\sigma \to_{\mathcal{R}}^* \cdots \qquad \square$$

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 $\mathsf{APP}(l, k)$.

4 Checking the Termination Criterion Automatically

In this section we present an approach to perform automated termination proofs using the criterion of Thm. 4, i.e. we develop 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 such that the existence of a well-founded ordering satisfying these inequalities is sufficient for termination of the TRS. A well-founded ordering satisfying the generated inequalities can often be synthesized by standard techniques, even if a *direct* termination proof is not possible with these techniques (i.e. even if a well-founded ordering orienting the rules of the TRS cannot be synthesized).

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 $\langle s_1, t_1\rangle\langle s_2, t_2\rangle\langle s_3, t_3\rangle \ldots$ 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 $\succ$, because 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 $\succsim$ is called *well-founded* if its strict part $\succ$ is well founded.)

Note that we cannot determine automatically for which substitutions σ we have $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ and moreover, it is practically impossible to examine infinite sequences of dependency pairs. Therefore, in the following we restrict ourselves to *weakly monotonic* quasi-orderings $\succsim$ where both $\succsim$ and its strict part $\succ$ are *closed under substitution*. (A quasi-ordering $\succsim$ is *weakly monotonic* if $s \succsim t$ implies $f(\ldots s \ldots) \succsim f(\ldots t \ldots)$.) Then, to guarantee $t_i\sigma \succsim s_{i+1}\sigma$ whenever $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ 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 demand $s \succ t$ for *all* dependency pairs $\langle s, t\rangle$.

Theorem 5 (Checking the Termination Criterion). *Let $\succsim$ be a well-founded, weakly monotonic quasi-ordering, where both $\succsim$ and $\succ$ are closed under substitution. A TRS $\mathcal{R}(D, C, R)$ is terminating, if*

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

Proof. As $l \succsim r$ holds for all rules $l \to r$ and as $\succsim$ is weakly monotonic and closed under substitution, we have $\to_{\mathcal{R}}^* \subseteq \succsim$, i.e. $t \to_{\mathcal{R}}^* s$ implies $t \succsim s$ (cf. e.g. [Der87]).

Suppose there is an infinite $\mathcal{R}$-chain $\langle s_1, t_1\rangle\langle s_2, t_2\rangle \ldots$, then there exists a substitution σ such that $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ 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 $s_1\sigma \succ t_1\sigma \succsim s_2\sigma \succ t_2\sigma \succsim \ldots$ which is a contradiction to the well-foundedness of $\succsim$ and therefore no infinite chain exists. Thus, by Thm. 4 $\mathcal{R}$ is terminating. □

The technique of Thm. 5 is very useful to apply standard methods like the recursive path ordering or polynomial interpretations to TRSs for which they are not directly applicable. For instance, in our example we have to find a quasi-ordering satisfying the following inequalities.

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

For example, these inequalities are satisfied by a 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, Gie95]. 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 $f(\ldots s \ldots) \gtrsim f(\ldots t \ldots)$. 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 we map $x.l$ to $l+1$.

Instead of polynomial orderings one can also use path orderings, which can easily be generated automatically. However, these path orderings are always strongly monotonic, whereas in our method we only need a weakly monotonic ordering. For that reason, before synthesizing a suitable path ordering some of the arguments of function symbols may be eliminated. For instance, one may eliminate the first arguments of the function symbols '.' and sum. Then every term $t.s$ in the inequalities is replaced by $.(s)$ and every term $\mathsf{sum}(t)$ is replaced by the constant sum. By comparing the terms resulting from this replacement (instead of the original terms) we can take advantage of the fact that '.' and sum do not have to be strongly monotonic in their first arguments. Now the resulting inequalities are satisfied by the recursive path ordering. Note that there exist only finitely many (and only few) possibilities to eliminate arguments of function symbols. Therefore all these possibilities can be checked automatically.

5 Dependency Graphs

To prove termination of a TRS according to Thm. 5 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, i.e. it is sufficient to demand $s \succ t$

for *some* dependency pairs only. For example, let us extend the TRS of Sect. 2 by the following rules for $+$.

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

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. 5 in addition to the inequalities in Sect. 4 we now obtain the following inequalities.

$$0 + y \gtrsim y \qquad \mathsf{SUM}(x.y.l) \succ \mathsf{PLUS}(x, y)$$
$$\mathsf{s}(x) + y \gtrsim \mathsf{s}(x + y) \qquad \mathsf{PLUS}(\mathsf{s}(x), y) \succ \mathsf{PLUS}(x, y)$$

Unfortunately, no polynomial ordering (and no path ordering which is amenable to automation) satisfies all resulting inequalities[3]. However, the constraint $\mathsf{SUM}(x.y.l) \succ \mathsf{PLUS}(x, y)$ 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 occur infinitely often in a chain we define a graph of dependency pairs where those dependency pairs that possibly occur consecutive in a chain are connected. In this way, any infinite chain corresponds to a cycle in the graph.

Definition 6 (Dependency Graph). The dependency graph of a TRS $\mathcal{R}$ is a directed graph whose nodes are labelled with the dependency pairs and there is an arc from $\langle s, t \rangle$ to $\langle u, v \rangle$ if there exists a substitution σ such that $t\sigma \to_{\mathcal{R}}^* u\sigma$.

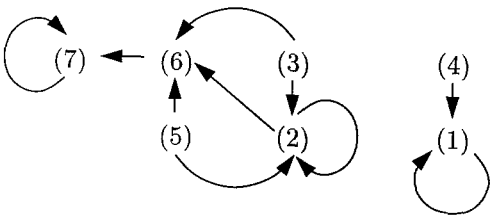

Fig. 1. The dependency graph of the example

[3] 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 $\mathsf{sum}(x.\mathsf{nil}) \gtrsim x.\mathsf{nil}$, 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.

Therefore, to prove termination of a TRS it is sufficient if $s \succ t$ holds for at least one dependency pair on each cycle of the dependency graph and if $s \succsim t$ holds for all other dependency pairs on cycles. 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 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.

Theorem 7 (Termination Proofs with Dependency Graphs). *Let $\succsim$ be a well-founded, weakly monotonic quasi-ordering, where both $\succsim$ and $\succ$ are closed under substitution. A TRS $\mathcal{R}(D, C, R)$ is terminating, if*

- *$l \succsim r$ 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.*

Proof. Suppose there is an infinite $\mathcal{R}$-chain, then this infinite chain corresponds to an infinite path in the dependency graph. This infinite path traverses at least one cycle infinitely many times, since there are only finitely many dependency pairs. 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. 5). Without loss of generality we may assume that the dependency pairs following $\langle s, t\rangle$ in the chain all occur on cycles of the graph. Hence, we obtain $s\sigma \succ t\sigma \succsim s\rho_1\sigma \succ t\rho_1\sigma \succsim s\rho_2\sigma \succ t\rho_2\sigma \succsim \ldots$ This is a contradiction to the well-foundedness of $\succ$. Hence, no infinite $\mathcal{R}$-chain exists and by Thm. 4 $\mathcal{R}$ is terminating. □

However, to perform termination proofs according to Thm. 7, 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. 7 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 the term t is connectable to u, we replace all subterms in t that have a defined root symbol by a new variable and check whether this modification of t unifies with 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 TRS from [Toy87].

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

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.

Definition 8 (Connectable Terms). For any term t, let $\mathrm{CAP}(t)$ result from replacing all subterms of t that have a defined root symbol by different new variables and let $\mathrm{REN}(t)$ 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 $\mathrm{REN}(\mathrm{CAP}(t))$ and u are unifiable.

For example, we have $\mathrm{CAP}(\mathsf{SUM}((x+y).l.l)) = \mathsf{SUM}(z.l.l)$ and by also replacing the variables by fresh ones, we have $\mathrm{REN}(\mathrm{CAP}(\mathsf{SUM}((x+y).l.l))) = \mathsf{SUM}(z.l_1.l_2)$. As $\mathrm{REN}(t)$ is always a linear term, to check whether two terms are connectable we can even use a unification algorithm without occur check.

To approximate the dependency graph, we draw an arc from a dependency pair $\langle s,t\rangle$ to $\langle u,v\rangle$ 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 computing connectable terms we in fact obtain a supergraph of the dependency graph. Using this supergraph, we can now prove termination according to Thm. 7.

Theorem 9 (Computing Dependency Graphs). *Let $\mathcal{R}$ be a TRS and t, u 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 $\text{REN}(\text{CAP}(t))$ matches v. Thus, in particular, if $t\sigma \to_{\mathcal{R}}^* u\sigma$, then $\text{REN}(\text{CAP}(t))$ matches $u\sigma$. As $\text{REN}(\text{CAP}(t))$ only contains new variables, this implies that $\text{REN}(\text{CAP}(t))$ 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, then $\text{REN}(\text{CAP}(t))$ is a variable, hence it matches v.

If $t = c(t_1, \ldots, t_k)$ for some constructor c, then $\text{REN}(\text{CAP}(t)) = c(\text{REN}(\text{CAP}(t_1)), \ldots, \text{REN}(\text{CAP}(t_k)))$. In this case, 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 i. By the induction hypothesis we obtain that $\text{REN}(\text{CAP}(t_i))$ matches v_i. Since the variables in $\text{REN}(\text{CAP}(t_i))$ are disjoint from the variables in $\text{REN}(\text{CAP}(t_j))$ for all $i \neq j$, $\text{REN}(\text{CAP}(t))$ also matches v. □

6 Related Work

The concept of *dependency pairs* was introduced in [Art96] and a first method for its automation was proposed in [AG96a]. However, these approaches were restricted to non-overlapping constructor systems without nested recursion, whereas in the present paper we extended the technique to arbitrary TRSs.

There is a relation between dependency pairs and *semantic labelling* [Zan95], because the dependency pairs correspond to the labels of a TRS labelled by the process of *self*-labelling. But in contrast to the approaches of [Art96, AG96a], our new termination criterion is no longer directly based on semantic labelling. Therefore this new criterion is better suited for automation (as one does not have to construct a suitable semantic interpretation any more) and its soundness can be proved in a much easier and shorter way. Moreover, by the introduction of dependency graphs we obtained a considerably more powerful automated technique than the method proposed in [AG96a].

Recently, we also developed a method for proving *innermost* normalisation using dependency pairs [AG97], which can be applied for termination proofs, too. However, 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 *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.

We have presented a sound and complete termination criterion. 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 complete criterion that has been used for automatic termination proofs (by J. Steinbach [Ste95a]) is the approach of *transformation orderings* [BL90]. It turns out that the termination of several examples where the automation of Steinbach failed can be proved by our technique automatically, cf. [AG96b].

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. 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 automatically. Second, the dependency graph is approximated by computing the 'connectable terms'. Third, well-known graph algorithms are used to determine those dependency pairs that are on a cycle of the dependency graph. Fourth, a set of inequalities is generated from the dependency pairs that occur on a cycle. Fifth, standard techniques, like polynomial interpretations or path orderings, are used to synthesize an ordering that satisfies the inequalities.

Our technique transforms a TRS 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. Moreover, if termination of 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 and 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 42 such examples, including arithmetical operations (e.g. mod, gcd, logarithm, average), sorting algorithms (such as selection sort, minimum sort, and quicksort), algorithms on graphs and trees, and several other well-known non-simply terminating TRSs (e.g. from [Der87, Ste95a, DH95]), can be found in [AG96b]. In 80 % of these examples, methods for the synthesis of path orderings could be applied to generate an ordering satisfying the inequalities resulting from our technique (whereas for the other examples we used polynomial orderings).

Acknowledgements. We thank Hans Zantema for helpful hints and comments.

References

[AG96a] T. Arts and J. Giesl. Termination of constructor systems. In *Proceedings of RTA-96*, LNCS 1103, pages 63–77, July 1996.

[AG96b] T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. Technical Report UU-CS-1996-44, Utrecht University, Utrecht, October 1996, `http://www.cs.ruu.nl`.

[AG97] T. Arts and J. Giesl. Proving innermost normalisation automatically. In *Proceedings of RTA'97*, June 2-4, 1997.

[Art96] T. Arts. Termination by absence of infinite chains of dependency pairs. In *Proceedings of CAAP'96*, LNCS 1059, pages 196–210, April 1996.

[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.

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

[Der81] N. Dershowitz. Termination of linear rewriting systems. In *Proceedings of ALP'81*, LNCS 115, pages 448–458, July 1981.

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

[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 *Handbook of Theoretical Computer Science*, volume B, pages 243–320, North-Holland, 1990.

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

[Gie95] J. Giesl. Generating polynomial orderings for termination proofs. In *Proceedings of RTA-95*, LNCS 914, pages 426–431, April 1995.

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

[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.

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

[Ste95a] J. Steinbach. Automatic termination proofs with transformation orderings. In *Proceedings of RTA-95*, LNCS 914, pages 11–25, April 1995. Long version appeared as Tech. Report SR-92-23, Univ. Kaiserslautern, Germany, 1992.

[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.

[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.