Proving Innermost Normalisation Automatically*

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. We present a technique to prove innermost normalisation of term rewriting systems (TRSs) automatically. In contrast to previous methods, our technique is able to prove innermost normalisation of TRSs that are not terminating.

Our technique can also be used for termination proofs of all TRSs where innermost normalisation implies termination, such as non-overlapping TRSs or locally confluent overlay systems. In this way, termination of many (also non-simply terminating) TRSs can be verified automatically.

1 Introduction

Innermost rewriting, i.e. rewriting where only innermost redeces are contracted, can be used to model call-by-value computation semantics. For that reason, there has been an increasing interest in *innermost normalisation* (also called innermost termination), i.e. in proving that the length of every innermost reduction is finite. Techniques for proving innermost normalisation can for example be utilized for termination proofs of functional programs (modelled by TRSs) or of logic programs. (When transforming logic programs into TRSs, innermost normalisation of the TRS implies termination of the logic program [AZ95].)

While both termination and innermost normalisation are undecidable properties [HL78], several techniques have been developed for proving termination of TRSs automatically (e.g. path orderings [Pla78, Der82, DH95, Ste95b], Knuth-Bendix orderings [KB70, DKM90], semantic interpretations [Lan79, BL87, BL93, Ste94, Zan94, Gie95b], transformation orderings [BD86, BL90, Ste95a] etc. — for surveys see e.g. [Der87, Ste95b]). However, there has not been any specific method for innermost normalisation, i.e. the only way to prove innermost normalisation *automatically* was by showing termination of the TRS. Therefore, none of the techniques could prove innermost normalisation of non-terminating systems.

---

* Technical Report IBN 96/39, Technische Hochschule Darmstadt. This is an extended version of an article with the same title presented at RTA-97 [AG97b].

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

In the following we present a technique for innermost normalisation proofs. For that purpose, in Sect. 2 we introduce a criterion for innermost normalisation. Subsequently, in Sect. 3 we develop a technique to check the requirements of this criterion automatically. For every TRS, our technique generates a set of constraints such that the existence of a well-founded ordering satisfying these constraints is sufficient for innermost normalisation. Now standard techniques developed for automated termination proofs of TRSs can be applied for the generation of appropriate well-founded orderings. In this way, innermost normalisation can be proved automatically. In Sect. 4 and 5 our technique is refined further and in Sect. 6 we give a summary and comment on connections and possible combinations with related approaches.

For several classes of TRSs, innermost normalisation already suffices for termination [Gra95, Gra96]. Moreover, several modularity results exist for innermost normalisation [Kri95, Art96], which do not hold for termination. Therefore, for those classes of TRSs *termination* can be proved by splitting the TRS and proving *innermost normalisation* of the subsystems separately. The advantage of this approach is that there are several interesting TRSs where a direct termination proof is not possible with the existing automatic techniques. However in many of these examples, a suitable ordering satisfying the constraints generated by our method can nevertheless be synthesized automatically. The reason is that for many TRSs proving innermost normalisation automatically is essentially easier than proving termination. In this way, innermost normalisation (and thereby, termination) of many also non-simply terminating systems can now be verified automatically. A collection of numerous examples where our technique proved successful can be found in Sect. 7 and Sect. 8.

2 A Criterion for Innermost Normalisation

In this section we introduce a new criterion for innermost normalisation. For that purpose the notions of *constructors* and *defined symbols* (that are well-known for the subclass of *constructor systems*) are extended to arbitrary TRSs. In the following, the *root* of a term $f(\ldots)$ is the leading function symbol f.

Definition 1 (Defined Symbols and Constructors). Let $\mathcal{R}(\mathcal{F},R)$ be a TRS (with the rules R over a signature $\mathcal{F}$). Then $D_{\mathcal{R}} = \{\text{root}(l) | l \to r \in R\}$ is the set of the *defined symbols* of $\mathcal{R}$ and $C_{\mathcal{R}} = \mathcal{F} \setminus D_{\mathcal{R}}$ is the set of *constructors* of $\mathcal{R}$. To stress the splitting of the signature we denote a TRS by $\mathcal{R}(D, C, R)$.

For example consider the following TRS, with the defined symbols f and g and the constructors $\mathsf{0}$ and s.

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

In contrast to the existing approaches for termination proofs, which compare left and right-hand sides of rules, in the following we only examine those subterms that are responsible for starting new reductions. For that purpose we concentrate on the subterms in the right-hand sides of rules that have a defined root symbol (because these are the only terms a rewrite rule can ever be applied to).

More precisely, for every rule $f(s_1, \ldots, s_n) \to C[g(t_1, \ldots, t_m)]$ (where f and g are defined symbols and C denotes some context), 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 extend the signature of the TRS by a new special *tuple symbol* F for every defined symbol f in D. Now instead of the tuples $s_1, \ldots, s_n$ and $t_1, \ldots, t_m$ we compare the *terms* $F(s_1, \ldots, s_n)$ and $G(t_1, \ldots, t_m)$. In this paper we assume that the signature $\mathcal{F}$ consists of lower case function symbols only and we denote the tuple symbols by the corresponding upper case symbols.

Definition 2 (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 a *dependency pair* of $\mathcal{R}$.

In the above example we obtain the following dependency pairs:

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

$$\langle \mathsf{F}(\mathsf{g}(x), \mathsf{s}(0), y), \mathsf{G}(x)\rangle \tag{2}$$

$$\langle \mathsf{G}(\mathsf{s}(x)), \mathsf{G}(x)\rangle \tag{3}$$

Using the concept of dependency pairs we can now develop a criterion for innermost normalisation. Note that in our example, we have the following infinite (cycling) reduction. (Here, s0 abbreviates s(0) etc.)

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

However, this reduction is not an innermost reduction, because in the first reduction step the subterm gs0 is a redex and would have to be reduced first. It turns out that although this TRS is not terminating, it is nevertheless innermost normalising. In the following, innermost reductions are denoted by "$\xrightarrow{\mathrm{i}}$".

Every infinite reduction corresponds to an infinite introduction of new redeces. To trace these newly introduced redeces we consider special sequences of dependency pairs, so-called *chains*. A sequence of dependency pairs is a chain if there exists a substitution σ such that for all consecutive pairs $\langle s_j, t_j\rangle$ and $\langle s_{j+1}, t_{j+1}\rangle$ in the sequence we have $t_j\sigma \to_{\mathcal{R}}^* s_{j+1}\sigma$ (cf. [AG97a]). In this way, the right-hand side of every dependency pair can be seen as the newly introduced redex that should be traced and the reductions $t_j\sigma \to_{\mathcal{R}}^* s_{j+1}\sigma$ are necessary to normalize the arguments of the redex that is traced. When regarding innermost reductions, arguments of a redex should be in normal form before the redex is contracted. Moreover, when concentrating on innermost reductions, the reductions of the arguments to normal form should also be innermost reductions. This results in the following restricted notion of a chain.

Definition 3 (Innermost $\mathcal{R}$-chains). Let $\mathcal{R}(D, C, R)$ be a TRS. A sequence of dependency pairs $\langle s_1, t_1\rangle \langle s_2, t_2\rangle \dots$ is called an *innermost $\mathcal{R}$-chain* if there exists a substitution σ, such that all $s_j\sigma$ are in normal form and $t_j\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} s_{j+1}\sigma$ holds for every two consecutive pairs $\langle s_j, t_j\rangle$ and $\langle s_{j+1}, t_{j+1}\rangle$ in the sequence.

We always assume that different (occurrences of) dependency pairs have disjoint sets of variables and we always regard substitutions whose domain may be infinite. Hence, in our example we have the innermost chain

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

because $\mathsf{G}(x_1)\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} \mathsf{G}(\mathsf{s}(x_2))\sigma$ and $\mathsf{G}(x_2)\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} \mathsf{G}(\mathsf{s}(x_3))\sigma$ holds for the substitution σ that replaces x_1 by $\mathsf{s}(\mathsf{s}(x_3))$ and x_2 by $\mathsf{s}(x_3)$. In fact any finite sequence of the dependency pair $\langle \mathsf{G}(\mathsf{s}(x)), \mathsf{G}(x)\rangle$ is an innermost chain. In the next section we will demonstrate that the above TRS actually has no infinite innermost chain. The following theorem shows that the absence of infinite innermost chains is a (sufficient and necessary) criterion for innermost normalisation.

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

Proof. **Sufficient Criterion**
Let t be a term that starts an infinite innermost reduction. Then the term t contains a subterm[1] $f_1(\mathbf{u}_1)$ that starts an infinite innermost reduction, but none of the terms $\mathbf{u}_1$ starts an infinite innermost reduction, i.e. the terms $\mathbf{u}_1$ are innermost normalising.
Let us consider an infinite innermost reduction starting with $f_1(\mathbf{u}_1)$. The arguments $\mathbf{u}_1$ are reduced innermost to normal form, say $\mathbf{v}_1$, 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 $f_1(\mathbf{v}_1) = f_1(\mathbf{w}_1)\sigma_1 \xrightarrow{\mathsf{i}}_{\mathcal{R}} r_1\sigma_1$. Hence, we have $\mathbf{u}_1 \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} \mathbf{w}_1\sigma_1$ and the terms $\mathbf{w}_1\sigma_1$ are in normal form.
Now the infinite innermost reduction continues with $r_1\sigma_1$, i.e. the term $r_1\sigma_1$ starts an infinite innermost reduction, too. Thus, r_1 contains a subterm $f_2(\mathbf{u}_2)$, 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 innermost reduction and $\mathbf{u}_2\sigma_1$ are innermost normalising terms. The first dependency pair of the infinite innermost 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 innermost 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 innermost reduction and the terms $\mathbf{u}_i\sigma_{i-1}$ are innermost normalising. Again, in zero or more steps $f_i(\mathbf{u}_i)\sigma_{i-1}$ reduces innermost to $f_i(\mathbf{v}_i)$ with $\mathbf{v}_i$ normal forms. A rewrite rule $f_i(\mathbf{w}_i) \to r_i$ can be applied to $f_i(\mathbf{v}_i)$ such that $r_i\sigma_i$ starts an infinite innermost reduction for some substitution σ_i with $\mathbf{v}_i = \mathbf{w}_i\sigma_i$.

[1] We denote tuples of terms $t_1, \dots, t_n$ by $\mathbf{t}$.

Similar to the observations above, since $r_i\sigma_i$ starts an infinite innermost 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 innermost reduction and $\mathbf{u}_{i+1}\sigma_i$ are innermost normalising terms. This results in the i-th dependency pair $\langle F_i(\mathbf{w}_i), F_{i+1}(\mathbf{u}_{i+1})\rangle$ in the innermost chain. In this way, one obtains the infinite sequence

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

It remains to prove that this sequence is really an innermost $\mathcal{R}$-chain.
Note that $F_i(\mathbf{u}_i\sigma_{i-1}) \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} F_i(\mathbf{v}_i)$ where $\mathbf{v}_i = \mathbf{w}_i\sigma_i$ and all terms $\mathbf{w}_i\sigma_i$ and thus all terms $F_i(\mathbf{w}_i)\sigma_i$ are normal forms. Since we assume that the variables of consecutive dependency pairs are disjoint, we obtain one substitution $\sigma = \sigma_1\circ\sigma_2\circ\sigma_3\circ\dots$ such that $F_i(\mathbf{u}_i)\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} F_i(\mathbf{w}_i)\sigma$ for all i. Thus, this sequence is indeed an infinite innermost $\mathcal{R}$-chain.

Necessary Criterion
We prove that any infinite innermost $\mathcal{R}$-chain can be transformed into an infinite innermost reduction. Assume there exists an infinite innermost chain.

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

Hence, there must be a substitution σ such that all $F_j(\mathbf{s}_j)\sigma$ are in normal form and such that

$$F_2(\mathbf{t}_2)\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} F_2(\mathbf{s}_2)\sigma,\ F_3(\mathbf{t}_3)\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} F_3(\mathbf{s}_3)\sigma, \dots,$$

resp. $f_j(\mathbf{t}_j)\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} f_j(\mathbf{s}_j)\sigma$, as $\mathcal{R}$ contains no F_j-rules for upper case symbols F_j. Note that 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. Since no redex occurs in $\mathbf{s}\sigma$, this reduction also follows the innermost strategy, i.e. $f(\mathbf{s})\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} C[g(\mathbf{t})]\sigma$. Therefore, we obtain the following infinite innermost reduction.

$$f_1(\mathbf{s}_1)\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} C_1[f_2(\mathbf{t}_2)]\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} C_1[f_2(\mathbf{s}_2)]\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} C_1[C_2[f_3(\mathbf{t}_3)]]\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} \dots$$

□

3 Automation of Innermost Normalisation Proofs

The advantage of our innermost normalisation criterion is that it is particularly well suited for automation. In this section we present a method for proving the absence of infinite innermost chains automatically. For this automation we assume the TRSs to be finite, such that only finitely many dependency pairs need to be considered.

Assume that there is a sequence $\langle s_1, t_1\rangle\langle s_2, t_2\rangle\langle s_3, t_3\rangle \dots$ of dependency pairs and a substitution σ such that all terms $s_j\sigma$ are in normal form and such that

$t_j\sigma$ reduces innermost to $s_{j+1}\sigma$ for all j. Then to prove that this sequence is finite, it suffices to find a well-founded[2] quasi-ordering $\succsim$ such that

$$s_1\sigma \succ t_1\sigma \succsim s_2\sigma \succ t_2\sigma \succsim s_3\sigma \succ t_3\sigma \ldots \tag{4}$$

In other words, we search for a quasi-ordering such that terms *in* dependency pairs are decreasing and terms *in between* dependency pairs are weakly decreasing. The reason for only demanding the weak inequalities $t_j\sigma \succsim s_{j+1}\sigma$ is that the terms $t_j\sigma$ and $s_{j+1}\sigma$ are often identical.

To automate this search for a suitable ordering we now present a procedure which, given a TRS, generates a set of *constraints* which are sufficient for (4). Then standard techniques developed for termination proofs of TRSs can be used to synthesize a well-founded quasi-ordering satisfying these constraints.

In the following we restrict ourselves to quasi-orderings where both $\succsim$ and $\succ$ are closed under substitution. To ensure that all dependency pairs are decreasing, we demand $s \succ t$ for all dependency pairs $\langle s, t\rangle$. In our example this results in the following constraints, cf. (1), (2), (3):

$$\mathsf{F}(\mathsf{g}(x), \mathsf{s}(0), y) \succ \mathsf{F}(y, y, \mathsf{g}(x)), \quad \mathsf{F}(\mathsf{g}(x), \mathsf{s}(0), y) \succ \mathsf{G}(x), \quad \mathsf{G}(\mathsf{s}(x)) \succ \mathsf{G}(x). \tag{5}$$

Moreover, we have to ensure $t_j\sigma \succsim s_{j+1}\sigma$ whenever $t_j\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} s_{j+1}\sigma$ holds. For that purpose we demand the constraints $l \succsim r$ for all those rules $l \to r$ that can be used in an innermost reduction of $t_j\sigma$. Note that as all terms $s_j\sigma$ are normal, σ is a *normal substitution* (i.e. it instantiates all variables with normal forms). Hence, for the dependency pairs (2) and (3) we directly obtain that *no* rule can ever be used to reduce a normal instantiation of $\mathsf{G}(x)$ (because G is no defined symbol). The only dependency pair whose right-hand side can be reduced if its variables are instantiated with normal forms is (1), because this is a dependency pair with *defined symbols* in the right-hand side. As the only defined symbol in $\mathsf{F}(y, y, \mathsf{g}(x))$ is g, the only rules that may be applied on normal instantiations of this term are the two g-rules of the TRS. Since these g-rules can never introduce a new redex with root symbol f, these two g-rules are the only rules that can be used to reduce any normal instantiation of $\mathsf{F}(y, y, \mathsf{g}(x))$. Hence, in this example we only have to demand that these rules should be weakly decreasing.

$$\mathsf{g}(\mathsf{s}(x)) \succsim \mathsf{s}(\mathsf{g}(x)), \quad \mathsf{g}(0) \succsim 0 \tag{6}$$

In general, to determine the *usable rules*, i.e. (a superset of) those rules that may possibly be used in a reduction of a normal instantiation of t, we proceed as follows. If t contains a defined symbol f, then all f-rules are *usable* and furthermore, all rules that are *usable* for right-hand sides of f-rules are also *usable* for t.

Definition 5 (Usable Rules). Let $\mathcal{R}(D, C, R)$ be a TRS. For any $f \in D$ let $Rls(f) = \{f(\mathbf{s}) \to r \mid f(\mathbf{s}) \to r \text{ in } R\}$. For any term t, $\mathcal{U}(t)$ is the smallest subset of R such that

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

- $\mathcal{U}(x) = \emptyset$,
- $\mathcal{U}(f(t_1, \ldots, t_n)) = \begin{cases} Rls(f) \cup \bigcup_{l \to r \in Rls(f)} \mathcal{U}(r) \cup \mathcal{U}(t_1) \cup \ldots \cup \mathcal{U}(t_n) & \\ & \text{if } f \in D \\ \mathcal{U}(t_1) \cup \ldots \cup \mathcal{U}(t_n) & \text{if } f \notin D \end{cases}$

Hence, we have $\mathcal{U}(\mathsf{F}(y, y, \mathsf{g}(x))) = Rls(\mathsf{g}) = \{\mathsf{g}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{g}(x)), \mathsf{g}(0) \to 0\}$.

So the constraints (6) ensure that whenever $\mathsf{F}(y, y, \mathsf{g}(x))$ is instantiated by a normal substitution σ, then reductions can only decrease the value of the *subterm* $\mathsf{g}(x)\sigma$. However, we have to guarantee that the value of the *whole* term $\mathsf{F}(y, y, \mathsf{g}(x))$ is weakly decreasing if an instantiation of $\mathsf{g}(x)$ is replaced by a smaller term. For that purpose, we demand that $\mathsf{F}(y, y, \mathsf{g}(x))$ must be *weakly monotonic* on the position of its subterm $\mathsf{g}(x)$, i.e. we also have to demand the following constraint:

$$x_1 \succsim x_2 \;\Rightarrow\; \mathsf{F}(y, y, x_1) \succsim \mathsf{F}(y, y, x_2). \tag{7}$$

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

Theorem 6 (Proving Innermost Normalisation). *Let $\mathcal{R}$ be a TRS and let $\succsim$ be a well-founded quasi-ordering where both $\succsim$ and $\succ$ are closed under substitution. If $\succsim$ is weakly monotonic on all symbols apart from the tuple symbols and if $\succsim$ satisfies the following constraints for all dependency pairs $\langle s, t\rangle$*

(a) $s \succ t$,
(b) $l \succsim r$ *for all usable rules* $l \to r$ *in* $\mathcal{U}(t)$,
(c) $x_1 \succsim y_1 \wedge \ldots \wedge x_n \succsim y_n \Rightarrow C[x_1, \ldots, x_n] \succsim C[y_1, \ldots, y_n]$, *where C is a context without defined symbols and $f_1, \ldots, f_n$ are defined symbols such that* $t = C[f_1(\mathbf{u}_1), \ldots, f_n(\mathbf{u}_n)]$,

then $\mathcal{R}$ is innermost normalising.

Proof. Suppose $\langle s_1, t_1\rangle\langle s_2, t_2\rangle \ldots$ is an infinite innermost $\mathcal{R}$-chain. Then there exists a substitution σ such that $s_j\sigma$ is in normal form and $t_j\sigma$ reduces innermost to $s_{j+1}\sigma$ for all j. Hence, σ replaces all variables by normal forms and therefore, the only rules that can be applied in this reduction are the usable rules $\mathcal{U}(t_j)$. All usable rules are weakly decreasing and the terms t_j are weakly monotonic on those positions where they are applied. (This also holds for reductions in $\mathbf{u}_i$, because all lower case symbols are weakly monotonic.) Hence, we have $t_j\sigma \succsim s_{j+1}\sigma$. This results in an infinite decreasing 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 $\succsim$. Thus, no infinite innermost $\mathcal{R}$-chain exists and by Thm. 4, the TRS is innermost normalising. □

Hence, in our example to prove innermost normalisation it is sufficient to find a well-founded quasi-ordering satisfying the constraints in (5), (6), and (7). For that purpose one may for instance use the well-known technique of synthesizing *polynomial orderings* [Lan79]. For example, these constraints are fulfilled by the polynomial ordering where the constant 0 is mapped to the number 0, $\mathsf{s}(x)$ is mapped to $x+1$, $\mathsf{g}(x)$ is mapped to $x+2$, $\mathsf{F}(x,y,z)$ is mapped to $(x-y)^2+1$, and $\mathsf{G}(x)$ is mapped to x. Methods to synthesize polynomial orderings automatically have for instance been developed in [Ste94, Gie95b]. Note that for our technique we do not require the quasi-ordering to be weakly monotonic on tuple symbols. The only monotonicity constraint in our example is (7), which is obviously satisfied as $\mathsf{F}(x,y,z)$ is mapped to a polynomial which is weakly monotonic[3] in its third argument z. However, this polynomial is not weakly monotonic in x or y.

In this way, innermost normalisation of our example can be proved automatically, i.e. this technique allows the application of standard techniques for innermost normalisation proofs, even if the TRS is not terminating. Moreover, using the results of [Gra95], Thm. 6 can also be applied for proving termination of TRSs that are non-overlapping (or for locally confluent overlay systems).

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

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

A problem with virtually all automatic approaches for termination proofs is that they are restricted to *simplification orderings* [Der79, Ste95b] and therefore can only prove termination of TRS that are simply terminating. However, there are numerous relevant and important terminating TRSs where simplification orderings fail. For instance, the above system is not simply terminating (the left-hand side of the last rule is embedded in the right-hand side if z is instantiated with 0).

Nevertheless, with our technique we can prove innermost normalisation and therefore termination of this system automatically. As quot is the only defined symbol of this system, we obtain the following dependency pairs (where Q denotes the tuple symbol for quot).

$$\langle \mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y),z), \mathsf{Q}(x,y,z)\rangle \tag{8}$$

$$\langle \mathsf{Q}(x,0,\mathsf{s}(z)), \mathsf{Q}(x,\mathsf{s}(z),\mathsf{s}(z))\rangle \tag{9}$$

Note that in this example there are no usable rules, as in the right-hand sides of the dependency pairs no defined symbols occur. Hence, due to Thm. 6 we only have to find a well-founded quasi-ordering such that both dependency pairs

[3] When using polynomial interpretations, the monotonicity constraint (c) of Thm. 6 can also be represented as an inequality. For instance, if F is mapped to some polynomial $[\mathsf{F}]$, then instead of (7) one could demand that the *partial derivative* of $[\mathsf{F}](y,y,x)$ with respect to x should be non-negative, i.e. $\frac{\partial[\mathsf{F}](y,y,x)}{\partial x} \geq 0$, cf. [Gie95b]. If one uses other techniques (e.g. path orderings) which can only generate monotonic orderings, then of course one may drop the monotonicity constraint (c).

are decreasing. These constraints are for instance satisfied by the polynomial ordering where 0 is mapped to the number 0, $\mathsf{s}(x)$ is mapped to $x + 1$, and $\mathsf{Q}(x, y, z)$ is mapped to $x + (x - y + z)^2$. Hence, innermost normalisation and thereby also termination of this TRS is proved (as it is non-overlapping). Note that again we benefit from the fact that the tuple symbol Q need not be weakly monotonic in its arguments. Therefore an interpretation like the polynomial $x + (x - y + z)^2$ may be used, which is not weakly monotonic in any of its arguments. In fact, if the set of usable rules is empty, the quasi-ordering need not even be weakly monotonic for any symbol.

4 A Refinement using Innermost Dependency Graphs

While the method of Thm. 6 can be very successfully used for both innermost normalisation and termination proofs, in this section we introduce a refinement of this approach, i.e. we show how the constraints obtained can be weakened. By this weakening, the (automatic) search for a suitable quasi-ordering satisfying these constraints can be eased significantly.

In order to ensure that every possible infinite innermost chain would result in an infinite decreasing sequence of terms, in the preceding section we demanded $s \succ t$ for *all* dependency pairs $\langle s, t\rangle$. However, in many examples it is sufficient if just *some* of the dependency pairs are decreasing.

For instance, in the quot-example up to now we demanded that both dependency pairs (8) and (9) had to be decreasing. However, two occurrences of the dependency pair (9) can never follow each other in a chain, because $\mathsf{Q}(x_1, \mathsf{s}(z_1), \mathsf{s}(z_1))\sigma$ can never reduce to any instantiation of $\mathsf{Q}(x_2, 0, \mathsf{s}(z_2))$. The reason is that the second arguments $\mathsf{s}(z_1)$ resp. 0 of these two terms have different constructor root symbols. Hence, any possible infinite chain would contain infinitely many occurrences of the other dependency pair (8). Therefore it is sufficient if (8) is decreasing and if (9) is just weakly decreasing. In this way, we obtain the following (weakened) constraints.

$$\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y), z) \succ \mathsf{Q}(x, y, z) \tag{10}$$

$$\mathsf{Q}(x, 0, \mathsf{s}(z)) \succsim \mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z)) \tag{11}$$

In general, to determine those dependency pairs which may possibly follow each other in innermost chains, we define the following graph[4].

Definition 7 (Innermost Dependency Graph). The *innermost dependency graph* of a TRS $\mathcal{R}$ is a directed graph whose nodes are the dependency pairs and

[4] Note that the conditions in Def. 7 are weaker than the conditions in the definition of innermost chains (Def. 3): Instead of using one "global" substitution σ for all dependency pairs, now one may use different "local" substitutions σ. Moreover, we only demand that these σ should be normal substitutions and that $v\sigma$ must be normal (but $s\sigma$ does not have to be in normal form any more). The reason for this weakening is that the conditions of Def. 7 are more suitable for automation.

there is an arc from $\langle s, t\rangle$ to $\langle v, w\rangle$ if there exists a normal substitution σ such that $t\sigma \xrightarrow{\mathrm{i}}{}^{*}_{\mathcal{R}} v\sigma$ and $v\sigma$ is a normal form.

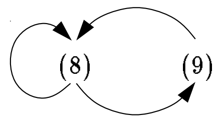

Fig. 1. Innermost Dependency graph of the quot TRS

For instance, in the innermost dependency graph for the quot example there are arcs from (8) to itself and to (9), and there is an arc from (9) to (8) (but *not* to itself).

Now any infinite innermost chain corresponds to a cycle in the innermost dependency graph. Hence, it is sufficient that $s \succ t$ holds for at least *one* dependency pair on every cycle and that $s \succsim t$ holds for the other dependency pairs on the cycles.

Theorem 8 (Proving IN with Innermost Dependency Graphs). *Let $\mathcal{R}$ be a TRS and let $\succsim$ be a well-founded quasi-ordering where both $\succsim$ and $\succ$ are closed under substitution. If $\succsim$ is weakly monotonic on all symbols apart from the tuple symbols, if $\succsim$ satisfies the following constraints for all dependency pairs $\langle s, t\rangle$ on a cycle in the innermost dependency graph*

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

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

Proof. Every possible infinite innermost $\mathcal{R}$-chain corresponds to an infinite path in the innermost dependency graph. This infinite path traverses at least one cycle infinitely many times. Note that $s \succ t$ holds for one dependency pair $\langle s, t\rangle$ on this cycle and that this dependency pair must occur infinitely often in the infinite innermost chain. As we may assume, without loss of generality, that all other dependency pairs in an infinite innermost chain are also on cycles in the innermost dependency graph, similar to the proof of Thm. 6 we again obtain an infinite sequence of inequalities of which infinitely many inequalities are strict. This is a contradiction to the well-foundedness of $\succsim$. Thus, no infinite innermost $\mathcal{R}$-chain exists and by Thm. 4, the TRS is innermost normalising. □

Hence, in the quot example the constraints (10) and (11) are in fact sufficient for innermost normalisation. A suitable quasi-ordering satisfying these weakened constraints can easily be synthesized (for instance, one could use the polynomial interpretation where 0 and s are interpreted as usual and where $\mathsf{Q}(x, y, z)$ is mapped to x). This example demonstrates that this weakening of the constraints often enables the use of much simpler orderings (e.g. now we can use a linear, weakly monotonic polynomial ordering whereas for the original constraints of Sect. 3 we needed a non-weakly monotonic polynomial of degree 2).

However, for an automation of Thm. 8 we have to construct the innermost dependency graph. Unfortunately, this cannot be done automatically, since for two terms t and v it is undecidable whether there exists a normal substitution σ such that $t\sigma$ reduces innermost to a normal form $v\sigma$. Hence, we can only approximate this graph by computing a supergraph containing the innermost dependency graph. Note that $t\sigma$ may only reduce to $v\sigma$ for some normal substitution σ, if either t has a defined root symbol or if both t and v have the same constructor root symbol. Let $\textsc{cap}(t)$ denote the result of replacing all subterms in t with a defined root symbol by different fresh variables. Then $t\sigma$ may only reduce to $v\sigma$ if $\textsc{cap}(t)$ and v are unifiable. Moreover, the most general unifier (mgu) of $\textsc{cap}(t)$ and v must be a normal substitution.

Theorem 9 (Computing Innermost Dependency Graphs). *Let $\mathcal{R}$ be a TRS. If $t\sigma \to_{\mathcal{R}}^* v\sigma$ holds for some normal substitution σ such that $v\sigma$ is a normal form, then $\textsc{cap}(t)$ and v unify and their mgu is a normal substitution.*

Proof. By induction on the structure of t we show that if a normal substitution σ and a normal term u exists such that $t\sigma \to_{\mathcal{R}}^* u$, then there exists a normal substitution τ (whose domain only includes variables that were newly introduced in the construction of $\textsc{cap}(t)$) such that[5] $\textsc{cap}(t)\sigma\tau = u$. Thus in particular, if $t\sigma \to_{\mathcal{R}}^* v\sigma$, we have $\textsc{cap}(t)\sigma\tau = v\sigma$ ($= v\sigma\tau$, because the variables of $v\sigma$ do not occur in the domain of τ). Hence, $\textsc{cap}(t)$ and v unify. Moreover, for the mgu μ of $\textsc{cap}(t)$ and v, there exists a substitution ρ with $\mu\rho = \sigma\tau$. As both σ and τ are normal, μ must be a normal substitution, too.

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

If the root symbol of t is defined, then $\textsc{cap}(t) = x$ for some fresh variable x. Let τ replace x by u. Then we have $\textsc{cap}(t)\sigma\tau = \textsc{cap}(t)\tau = u$ and τ is normal.

If $t = c(t_1, \ldots, t_k)$ for some constructor $c \in C$, then u has to be of the form $c(u_1, \ldots, u_k)$ and $t_i\sigma \to_{\mathcal{R}}^* u_i$ holds for all i. By the induction hypothesis we obtain that normal substitutions τ_i exist such that $\textsc{cap}(t_i)\sigma\tau_i = u_i$ for all i. Note that those variables in $\textsc{cap}(t_i)$ that were introduced by $\textsc{cap}$ are disjoint from the newly introduced variables in $\textsc{cap}(t_j)$ (for $i \neq j$). Hence, if $\tau = \tau_1 \circ \ldots \circ \tau_k$, then for all i we have $\textsc{cap}(t_i)\sigma\tau = u_i$ resp. $\textsc{cap}(t)\sigma\tau = c(\textsc{cap}(t_1), \ldots, \textsc{cap}(t_k))\sigma\tau = c(u_1, \ldots, u_k) = u$ and again, τ is normal. □

[5] Here, "$t\sigma\tau$" is defined as "$(t\sigma)\tau$", i.e. σ is applied first.

Now an approximation of the innermost dependency graph is computed by drawing an arc from $\langle s,t\rangle$ to $\langle v,w\rangle$ if $\textsc{cap}(t)$ and v unify (using a *normal* mgu μ). In this way we can compute the innermost dependency graph in the quot example automatically. There are also examples where the innermost dependency graph does not contain any cycles.

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

In this example, the first dependency pair $\langle \mathsf{F}(x,\mathsf{g}(x)),\mathsf{F}(1,\mathsf{g}(x))\rangle$ does not occur on a cycle in the innermost dependency graph, although $\textsc{cap}(\mathsf{F}(1,\mathsf{g}(x))) = \mathsf{F}(1,y)$ unifies with $\mathsf{F}(x,\mathsf{g}(x))$ using a mgu that replaces x by 1 and y by $\mathsf{g}(1)$. However, $\mathsf{g}(1)$ is not a normal form and hence, this mgu is not a normal substitution. The second dependency pair $\langle \mathsf{G}(1),\mathsf{G}(0)\rangle$ cannot occur on a cycle either, since $\mathsf{G}(0)$ does not unify with $\mathsf{G}(1)$. Hence, using the refined technique of Thm. 8 we obtain *no* constraint at all, i.e. innermost normalisation can be proved by only computing the (approximation of) the innermost dependency graph.

5 Computing Dependency Graphs by Narrowing

To perform innermost normalisation proofs according to the method of Thm. 8 we have to compute a graph containing the innermost dependency graph. However, for some examples the approximation presented in the preceding section is too rough. For instance, let us replace the last rule of the quot system by the following three rules.

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

Now instead of dependency pair (9) we obtain

$$\langle \mathsf{Q}(x,0,\mathsf{s}(z)),\mathsf{Q}(x,z+\mathsf{s}(0),\mathsf{s}(z))\rangle. \tag{12}$$

Note that in our approximation of the innermost dependency graph there would be an arc from (12) to itself, because after replacing $z+\mathsf{s}(0)$ by a new variable, the right- and the left-hand side of (12) obviously unify (and the mgu is normal). Hence, due to Thm. 8 we would have to find an ordering such that (12) is strictly decreasing. But then no linear or weakly monotonic polynomial ordering satisfies all resulting inequalities in this example.

However, in the *real* innermost dependency graph, there is no arc from (12) to itself, because, similar to the original dependency pair (9), there is no substitution σ such that $(z+\mathsf{s}(0))\sigma$ reduces to 0. Hence, there is no cycle consisting of (12) only and therefore it is sufficient if (12) is just *weakly* decreasing. In this way, the simple (linear) polynomial ordering of the last section would also satisfy the constraints resulting from this example (if the tuple symbol $\mathsf{PLUS}(x,y)$ is mapped to x). Therefore to ease the innermost normalisation (resp. termination) proof of this example we need a method to compute a better approximation of the innermost dependency graph.

Hence, we present a better technique to determine whether for two terms t and v there exists a normal substitution σ such that $t\sigma$ reduces innermost to the normal form $v\sigma$. For this purpose we use narrowing (cf. e.g. [Hul80]).

Definition 10 (Narrowing). Let $\mathcal{R}$ be a TRS. A term t *narrows* to a term q (denoted by $t \leadsto_{\mathcal{R}} q$), if there exists a nonvariable position p in t, μ is the most general unifier of $t|_p$ and l for some rewrite rule $l \to r$ of $\mathcal{R}$, and $q = t\mu[r\mu]_p$. (Here, the variables of $l \to r$ must have been renamed to fresh variables.)

To find out whether $t\sigma \xrightarrow{\text{i}}{}^*_{\mathcal{R}} v\sigma$ holds for some normal substitution σ, up to now we checked whether $\textsc{cap}(t)$ is unifiable with v. However, in those cases where t itself is not already unifiable with v (i.e. in those cases where at least one rule of $\mathcal{R}$ is needed to reduce $t\sigma$ to $v\sigma$), instead of examining t and v we may first perform all possible narrowing steps on t (resulting in new terms $t_1, \ldots, t_n$). Now it suffices to check whether $t_k\sigma$ reduces innermost to $v\sigma$ for one $k \in \{1, \ldots, n\}$.

For example, to find out whether $\mathsf{Q}(x, z + \mathsf{s}(0), \mathsf{s}(z))\sigma \xrightarrow{\text{i}}{}^*_{\mathcal{R}} \mathsf{Q}(x_2, 0, \mathsf{s}(z_2))\sigma$ holds for some normal substitution σ we first compute all terms that $\mathsf{Q}(\ldots z + \mathsf{s}(0)\ldots)$ narrows to. Here, $z + \mathsf{s}(0)$ is the only nonvariable subterm which is unifiable with a left-hand side of a rule. Hence, we only have

$\mathsf{Q}(\ldots z + \mathsf{s}(0)\ldots) \leadsto_{\mathcal{R}} \mathsf{Q}(\ldots \mathsf{s}(0)\ldots)$ by the first $+$ rule, and
$\mathsf{Q}(\ldots z + \mathsf{s}(0)\ldots) \leadsto_{\mathcal{R}} \mathsf{Q}(\ldots \mathsf{s}(x + \mathsf{s}(0))\ldots)$ by the second $+$ rule.

Note that any term t can only be narrowed in one step to *finitely many* terms $t_1, \ldots, t_n$ (up to variable renaming) and these terms $t_1, \ldots, t_n$ can easily be computed automatically.

In our example, now it suffices to check whether a normal substitution σ exists such that $\mathsf{Q}(\ldots \mathsf{s}(0)\ldots)\sigma$ or $\mathsf{Q}(\ldots \mathsf{s}(x + \mathsf{s}(0))\ldots)\sigma$ reduces innermost to a normal form $\mathsf{Q}(\ldots 0 \ldots)\sigma$. For that purpose we can use the technique presented in Thm. 9. This technique immediately proves that such a substitution cannot exist because neither $\mathsf{s}(0)$ nor $\textsc{cap}(\mathsf{s}(x + \mathsf{s}(0)))$ unify with the subterm 0.

Of course instead of using the technique of Thm. 9 on the obtained terms, we could also apply narrowing again and replace $\mathsf{Q}(\ldots \mathsf{s}(x + \mathsf{s}(0))\ldots)$ by those terms it narrows to. In general, to determine whether $t\sigma \xrightarrow{\text{i}}{}^*_{\mathcal{R}} v\sigma$ holds for some normal substitution σ one can apply an *arbitrary* number of narrowing steps to t. Subsequently, the method of Thm. 9 is applied to test whether after application of $\textsc{cap}$ one of the resulting terms is unifiable with v (using a normal mgu).

By the use of narrowing we obtain a method to compute much better approximations of innermost dependency graphs. For instance, if in our example we perform at least one narrowing step, then we can determine that the dependency pair (12) does not form a cycle in the innermost dependency graph and then termination can again be verified using a linear, weakly monotonic polynomial ordering. The following theorem proves the soundness of this approach.

Theorem 11 (Computing Dependency Graphs by Narrowing). *Let $\mathcal{R}$ be a TRS and let t, v be terms with disjoint sets of variables. If there exists a normal substitution σ such that $t\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} v\sigma$ and $v\sigma$ is a normal form, then*

- *t and v are unifiable, or*
- *there exists a term q and a normal substitution τ such that $t \leadsto_{\mathcal{R}} q$, $q\tau \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} v\tau$ and $v\tau$ is a normal form.*

Proof. The proof is done by induction on the length of the reduction $t\sigma \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} v\sigma$. If the length is zero, then t and v unify. Otherwise we have $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} t' \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} v\sigma$ for some term t'. As σ is a normal substitution, the reduction $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} t'$ cannot take place "in σ". Hence, t contains some subterm $f(\mathbf{u})$ such that a rule $l \to r$ has been applied to $f(\mathbf{u})\sigma$. In other words, l matches $f(\mathbf{u})\sigma$ (i.e. $l\rho = f(\mathbf{u})\sigma$, where ρ is a *normal* substitution, because for innermost reductions the terms $\mathbf{u}$ must be in normal form). Hence, the reduction has the following form: $t\sigma = t\sigma[f(\mathbf{u})\sigma]_p = t\sigma[l\rho]_p \xrightarrow{\mathsf{i}}_{\mathcal{R}} t\sigma[r\rho]_p = t'$. Similar to Def. 10 we assume that the variables of $l \to r$ have been renamed to fresh ones. Then $\sigma\rho$ is a unifier of l and $f(\mathbf{u})$ and hence, there also exists a mgu μ. By the definition of most general unifiers there must also be a substitution τ such that $\sigma\rho = \mu\tau$. Here, τ is a normal substitution because both σ and ρ are normal. As the variables of t and v are disjoint, we can assume that μ never introduces any variables from v. Thus, we may define τ to be like σ for the variables of v, i.e. $v\sigma = v\tau$ is a normal form.

Let q be the term $t\mu[r\mu]_p$. Then $t \leadsto_{\mathcal{R}} q$ holds by the definition of narrowing. Moreover we have $q\tau = t\mu\tau[r\mu\tau]_p = t\sigma\rho[r\sigma\rho]_p = t\sigma[r\rho]_p = t' \xrightarrow{\mathsf{i}}{}^{*}_{\mathcal{R}} v\sigma = v\tau$. $\square$

6 Conclusion and Related Work

We have introduced a technique to automate innermost normalisation proofs of term rewriting systems. For that purpose we have developed a new criterion for innermost normalisation which is based on the concept of dependency pairs. To automate the checking of this criterion, a set of constraints is synthesized for each TRS and standard techniques developed for termination proofs can be used to generate a well-founded ordering satisfying these constraints. If such an ordering can be found, then innermost normalisation of the system is proved.

Our approach is the first automatic method which can also prove innermost normalisation of systems that are not terminating. Moreover, our technique can also very successfully be used for termination proofs of non-overlapping systems, because for those systems innermost normalisation is already sufficient for termination. We implemented our technique for the generation of constraints and a large collection of TRSs of which innermost normalisation resp. termination has been proved automatically can be found in Sect. 7 and Sect. 8. These examples include well-known non-simply terminating challenge problems from literature as well as many practically relevant TRSs from different areas of computer science (such as arithmetical operations, several sorting algorithms, a reachability algorithm on graphs, a TRS for substitutions in the lambda calculus etc.).

The concept of dependency pairs has been introduced in [Art96] and a first automation of this concept can be found in [AG96b]. However, these approaches were restricted to non-overlapping constructor systems without nested recursion, whereas in the present paper we dealt with arbitrary rewrite systems. Moreover, in contrast to these first approaches, in this paper we developed a *complete* criterion for innermost normalisation and proved its soundness in a short and easy way (while the corresponding proof in [Art96] was based on semantic labelling [Zan95]). Finally, the introduction of innermost dependency graphs led to a considerably more powerful technique than the method proposed in [AG96b].

Dependency pairs have a connection to *semantic labelling* [Zan95] (resp. to *self*-labelling [MOZ96]). However, compared to semantic labelling the dependency pair approach is better suited for automation, because here one does not have to find an appropriate semantic interpretation. At first sight, there also seems to be a similarity between innermost chains and innermost *forward closures* [LM78, DH95], but it turns out that these approaches are fundamentally different. While forward closures restrict the *application of rules* (to that part of a term created by previous rewrites), the dependency pair approach restricts the *examination of terms* (to those subterms that may possibly be reduced further). So in contrast to innermost chains, innermost forward closures are reductions. Moreover, while the dependency pair approach is very well suited for automation, we do not know of any approach to automate the forward closure approach.

As our technique can only be applied for *termination* proofs if the TRS is non-overlapping (or at least an overlay system with joinable critical pairs), in [AG97a] we also showed how dependency pairs can be used for termination proofs of arbitrary TRSs. However, as long as the system is non-overlapping, it is always advantageous to prove innermost normalisation only (instead of termination). For instance, termination of the quot system can easily be proved with the technique introduced in the present paper, whereas the constraints generated by the method of [AG97a] are not satisfied by any quasi-ordering which is amenable to automation (i.e. by any total or quasi-simplification ordering).

Most previous methods developed for automatic termination proofs are based on *simplification orderings*. For non-overlapping systems, these methods should always be combined with our technique, because there are many examples where *direct* termination proofs using the standard methods fail, but these methods can nevertheless synthesize an ordering satisfying the constraints resulting from our technique. Moreover, whenever a direct termination proof is possible with a simplification ordering, then this simplification ordering also satisfies the constraints resulting from our technique. The only other approach for automated termination proofs of non-simply terminating systems is a technique for generating *transformation orderings* [BL90] by *Steinbach* [Ste95a]. Several examples which can automatically be proved terminating by our technique, but where Steinbach's approach fails, can be found in Sect. 8.

7 Examples of Innermost Normalisation Proofs

In this section and the next section a collection of examples is listed that demonstrates the power of the described method.

The examples in this section are term rewriting systems that are not terminating. Thus all methods based on proving termination fail in proving innermost normalisation of these term rewriting systems. It is shown how our method can automatically derive innermost normalisation of these term rewriting systems.

The examples in the next section are term rewriting systems for which innermost normalisation suffices to guarantee termination by the results of Gramlich [Gra95, Gra96]. Many of these examples are term rewriting systems that are not simply terminating. Therefore, their termination cannot be shown by most other automatic methods. However, by our approach they can be proved terminating.

For proving termination of the examples, our technique first transforms the TRS into a set of constraints. Three kinds of such constraints can be distinguished: For each usable rewrite rule $l \to r$ we obtain an inequality $l \succsim r$ and for each dependency pair $\langle s, t \rangle$ on a cycle of the innermost dependency graph we obtain the inequality $s \succsim t$. Furthermore, for each cycle one of these inequalities must be strict, i.e. $s \succ t$. Third, for each such dependency pair $\langle s, t \rangle$ we must demand that t must be weakly monotonic on all positions p where the root of $t|_p$ is defined. We perform narrowing to obtain a better approximation of the innermost dependency graph, therefore we also mention the number of narrowing steps required for each example under consideration (unless narrowing is not needed).

After having obtained the constraints, a well-founded quasi-ordering is generated, which is weakly monotonic for all symbols apart from the tuple symbols and which satisfies these constraints. 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, e.g. [Ste94, Gie95b]. In contrast to the use of polynomial orderings for direct termination proofs, we can use polynomial interpretations with *weakly* monotonic polynomials (and tuple symbols may be mapped to polynomials that are not even weakly monotonic on all arguments). For instance, we may 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]. 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.

7.1 First Running Example

For the first example of this paper

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

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

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

which is easily satisfied by the recursive path ordering.

7.2 Toyama Example

The most famous example of a TRS that is innermost normalising, but not terminating, is the following system from [Toy87].

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

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

7.3 Variations on the Toyama Example, Version 1

The following modification of the Toyama example

$$\begin{aligned} &\mathsf{f}(\mathsf{g}(x, y), x, z) \to \mathsf{f}(z, z, z) \\ &\mathsf{g}(x, y) \qquad\quad\;\; \to x \\ &\mathsf{g}(x, y) \qquad\quad\;\; \to y \end{aligned}$$

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

7.4 Variations on the Toyama Example, Version 2

The TRS

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

is no constructor system either. The dependency pair $\langle \mathsf{F}(\mathsf{g}(x), x, y), \mathsf{F}(y, y, \mathsf{g}(y))\rangle$ cannot occur in an infinite innermost chain, since $\textsc{cap}(\mathsf{F}(y_1, y_1, \mathsf{g}(y_1)))$ does not unify with $\mathsf{F}(\mathsf{g}(x_2), x_2, y_2)$. Hence, we only obtain the constraint

$$\mathsf{G}(\mathsf{g}(x)) \succ \mathsf{G}(x)$$

as there are no usable rules. As this constraint is satisfied by the recursive path ordering, the TRS is innermost normalising.

7.5 Narrowing, Version 1

In the following variant of the Toyama example

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

we need one narrowing step to determine that there are no cycles in the innermost dependency graph (because $\mathsf{f}(\mathsf{g}(x, x), x, x)$ narrows to $\mathsf{f}(x, x, x)$). Hence, this TRS is also innermost normalising.

7.6 Narrowing, Version 2

Consider the following TRS

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

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

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

Innermost normalisation of this TRS can be proved if the innermost dependency graph is computed using narrowing. The right projection of the dependency pair $\langle \mathsf{F}(0, \mathsf{s}(0), x), \mathsf{F}(x, x + x, x)\rangle$ narrows to both $\mathsf{F}(0, 0, 0)$ and $\mathsf{F}(\mathsf{s}(y), \mathsf{s}(\mathsf{s}(y) +$

$y), \mathsf{s}(y))$, which are not unifiable with the left projection of this dependency pair. Therefore, the only generated inequality for this TRS is

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

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

7.7 Narrowing, Version 3

The following modification of the above TRS

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

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

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

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

7.8 Non-Normal Most General Unifier

The following TRS

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

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

The dependency pair

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

does not occur on a cycle of the innermost dependency graph, because $\textsc{cap}(\mathsf{F}(1, \mathsf{g}(x_1))) = \mathsf{F}(1, y)$ and the mgu of $\mathsf{F}(1, y)$ and $\mathsf{F}(x_2, \mathsf{g}(x_2))$ is not a normal substitution. (It replaces y by $\mathsf{g}(1)$.) Obviously, the other dependency pair $\langle \mathsf{G}(1), \mathsf{G}(0) \rangle$ cannot occur on a cycle either. Thus, there are no cycles in the innermost dependency graph. Hence, the TRS is innermost normalising.

7.9 Innermost Chains of Arbitrary Finite Length

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

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

An infinite reduction is given by

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

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

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

(The reason is that the most general unifier of $\textsc{cap}(\mathsf{F}(x_1,\mathsf{s}(x_1),z_1))$ and $\mathsf{F}(x_2,y_2,\mathsf{g}(x_2,y_2))$ is not normal.)

There are no usable rules. Thus, innermost normalisation is easily proved by the polynomial interpretation that maps $\mathsf{s}(y)$ to $y+1$ and $\mathsf{G}(x,y)$ to y.

7.10 Negative Coefficients

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

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

An infinite reduction is given by

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

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

$$\begin{array}{rl} \mathsf{H}(0,x) & \succsim \mathsf{F}(0,x,x) \\ \mathsf{F}(0,1,x) & \succ \mathsf{H}(x,x) \end{array}$$

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

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

7.11 Drosten example

A confluent and innermost normalising TRS that is not terminating was given by *Drosten* [Dro89].

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

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

8 Examples of Termination Proofs

In this section a collection of non-overlapping resp. of locally confluent overlay systems is proved terminating by our technique. For these TRSs innermost normalisation implies termination. Therefore, applying our technique to prove innermost normalisation to these TRSs results in an automatic approach for termination proofs. In particular, this collection also includes several systems that are not simply terminating, cf. [Der79, Ste95b].

As mentioned in Sect. 1, in contrast to termination there exist several modularity results for innermost normalisation, e.g. [Kri95, Art96]. In particular, we can use the following result for *hierarchical combinations*, cf. [AG96b]. A TRS is a hierarchical combination of two subsystems if defined symbols of the first system occur as constructors in the second system, but not vice versa. If $\mathcal{R}$ is such a hierarchical combination of $\mathcal{R}_0$ with $\mathcal{R}_1$ and if the subsystem $\mathcal{R}_0$ is innermost normalising, then one does not have to consider all dependency pairs of $\mathcal{R}$, but it suffices to examine only those dependency pairs $\langle F(\ldots), G(\ldots)\rangle$ where f and g are defined symbols of $\mathcal{R}_1$. In this way it is possible to prove innermost normalisation of hierarchical combinations by successively proving innermost normalisation of each subsystem and by defining $\mathcal{R}_0$ to consist of those subsystems whose innermost normalisation has already been proved before. (In other words, one only has to prove that there exists no infinite innermost chain consisting of dependency pairs of $\mathcal{R}_1$.) The justification for this approach is the following theorem.

Theorem 12 (IN of Hierarchical Combinations). *Let $\mathcal{R}(D_0 \cup D_1, C, R)$ be a hierarchical combination of $\mathcal{R}_0(D_0, C, R_0)$ and $\mathcal{R}_1(D_1, C \cup D_0, R_1)$. If $\mathcal{R}_0$ is innermost normalising, then any infinite innermost $\mathcal{R}$-chain consists of dependency pairs of $\mathcal{R}_1$ only.*

Proof. Assume there exists an infinite innermost $\mathcal{R}$-chain in which an $\mathcal{R}_0$ dependency pair occurs. By the definition of a hierarchical combination, then all dependency pairs in this chain are dependency pairs of $\mathcal{R}_0$. For any innermost chain, there is a substitution σ such that all $s_j\sigma$ are in normal form and such that $t_j\sigma \xrightarrow{\mathrm{i}}{}^* s_{j+1}\sigma$ holds for all consecutive pairs $\langle s_j, t_j\rangle$, $\langle s_{j+1}, t_{j+1}\rangle$ in the sequence.

Since no defined symbols of $\mathcal{R}_1$ occur in t_j for any j and since all defined symbols of $\mathcal{R}_1$ that occur in σ occur in normal forms, any reduction $t_j\sigma \xrightarrow{\mathrm{i}}{}^* s_{j+1}\sigma$ can only use rules from $\mathcal{R}_0$. Therefore this sequence of dependency pairs is also an (infinite) innermost $\mathcal{R}_0$-chain. But due to Thm. 4 this is a contradiction to the fact that $\mathcal{R}_0$ is innermost normalising. □

This modularity result will be used in several of the following examples. However, in most of the examples innermost normalisation can also be proved without application of the modularity result. In those of the following examples where the modularity result is applied, the TRSs are presented as two sets of rewrite rules. The upper system always denotes $\mathcal{R}_0$, whereas the bottom rules denote $\mathcal{R}_1$. In the examples, termination of $\mathcal{R}_0$ is always easy to show (either by applying our method again or by standard techniques, i.e. $\mathcal{R}_0$ is usually simply terminating).

First, our technique is used to prove termination of all examples of [AG96a] (Ex. 8.1 - 8.14). While the method of [AG96a, AG96b] was only applicable to a restricted class of *constructor* systems (without *nested* recursion), the present technique can be used for termination proofs of arbitrary locally confluent overlay systems. Therefore, subsequently we mention several examples that are no constructor systems or have nested recursion, but where our technique can nevertheless prove termination.

In this paper we have presented a method for innermost normalisation which can also be used for termination proofs if the TRS is non-overlapping or at least a locally confluent overlay system. In [AG97a] however, we have also developed a method for termination proofs of *arbitrary* TRSs (i.e. there, they do not have to be locally confluent overlay systems). Termination of the examples 8.15 - 8.27 can also be proved by the method of [AG96c, AG97a]. However, as innermost normalisation is essentially easier to prove than termination, when using the method of the present paper, we often obtain considerably less constraints than when using the technique of [AG97a]. For that reason, termination of the Examples 8.31 - 8.35 cannot be shown automatically by the method of [AG97a] whereas with the technique of the present paper we can prove innermost normalisation (and thereby termination) automatically. On the other hand, there are also (overlapping) TRSs, where the method of [AG97a] can prove termination, but the method of the present paper cannot be used for that purpose, because for these systems innermost normalisation is not sufficient for termination.

8.1 Division, Version 1

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

$$\mathsf{minus}(x, 0) \rightarrow x$$

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

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

In the termination proof of this example one can apply the modularity result of Thm. 12. Then termination of $\mathcal{R}_0$ (the first two minus rules) is easily proved (either by our approach or directly by the recursive path ordering, for example).

For innermost normalisation of $\mathcal{R}$ it now suffices to show that there is no infinite innermost chain of dependency pairs of $\mathcal{R}_1$. For these dependency pairs of $\mathcal{R}_1$, the subtraction rules are the usable rules and there is one dependency pair on a cycle of the innermost dependency graph. This results in the constraints

$$\begin{aligned}\mathsf{Q}(\mathsf{s}(x),\mathsf{s}(y)) &\succ \mathsf{Q}(\mathsf{minus}(x,y),\mathsf{s}(y))\\ x_1 \succsim x_2 &\Rightarrow \mathsf{Q}(x_1,\mathsf{s}(y)) \succsim \mathsf{Q}(x_2,\mathsf{s}(y))\\ \mathsf{minus}(x,0) &\succsim x\\ \mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) &\succsim \mathsf{minus}(x,y).\end{aligned}$$

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 0 is mapped to the number 0, $\mathsf{s}(x)$ is mapped to $x+1$, and $\mathsf{Q}(x,y)$ and $\mathsf{minus}(x,y)$ are mapped to x.

These orderings could also be used for the innermost normalisation proof if the modularity result would not be applied. Then one would obtain the additional constraint $\mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) \succ \mathsf{MINUS}(x,y)$.

8.2 Division, Version 2

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

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

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

Again the TRS $\mathcal{R}_0$ is terminating (which can either be proved by our method or by the recursive path ordering). The inequality obtained from the dependency pair of $\mathcal{R}_1$ is

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

The first three rules of the TRS are the usable rules, resulting in the three inequalities

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

and the demand that $\mathsf{Q}(x, \mathsf{s}(y))$ should be weakly monotonic in x.

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. The demands on the ordering are then satisfied by the recursive path ordering. (As for most of the following examples, innermost normalisation could also be proved without using the modularity result.)

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.

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

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

The subsystem $\mathcal{R}_0$ is terminating (this can be proved by our technique again). The constraints generated for the dependency pairs of $\mathcal{R}_1$ are

$$\begin{aligned} \mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y)) &\succ \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{s}(y)) \\ x_1 \succsim x_2 &\Rightarrow \mathsf{Q}(x_1, \mathsf{s}(y)) \succsim \mathsf{Q}(x_2, \mathsf{s}(y)) \end{aligned}$$

plus $l \succsim r$ for all rules of $\mathcal{R}_0$ (as all of these rules are usable).

By the following mapping

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

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.

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

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

The TRS $\mathcal{R}_0$ is terminating. This can be proved by the recursive path ordering or by our technique. The constraints generated for $\mathcal{R}_1$ are

$$
\begin{aligned}
\mathsf{MOD}(\mathsf{s}(x), \mathsf{s}(y)) &\succsim \mathsf{IF}_{\mathsf{mod}}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \\
\mathsf{IF}_{\mathsf{mod}}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) &\succ \mathsf{MOD}(\mathsf{minus}(x, y), \mathsf{s}(y)) \\
x_1 \succsim x_2 &\Rightarrow \mathsf{IF}_{\mathsf{mod}}(x_1, \mathsf{s}(x), \mathsf{s}(y)) \succsim \mathsf{IF}_{\mathsf{mod}}(x_2, \mathsf{s}(x), \mathsf{s}(y)) \\
x_1 \succsim x_2 &\Rightarrow \mathsf{MOD}(x_1, \mathsf{s}(y)) \succsim \mathsf{MOD}(x_2, \mathsf{s}(y))
\end{aligned}
$$

plus $l \succsim r$ for all rules of $\mathcal{R}_0$.

By mapping $\mathsf{minus}(x, y)$, $\mathsf{MOD}(x, y)$, and $\mathsf{IF}_{\mathsf{mod}}(b, x, y)$ 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.

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

$$
\mathsf{gcd}(0, y) \to 0
$$

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

(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.)

Termination of $\mathcal{R}_0$ can be proved by our approach. The constraints for innermost normalisation (from the dependency pairs of $\mathcal{R}_1$) are

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

plus some monotonicity demands and $l \succsim r$ for all rules of $\mathcal{R}_0$.

A suitable mapping is given by

$$\begin{aligned}
\mathsf{pred}(x) &\mapsto x\\
\mathsf{minus}(x,y) &\mapsto x\\
\mathsf{IF}_{\mathsf{gcd}}(b,x,y) &\mapsto \mathsf{IF}_{\mathsf{gcd}}(x,y).
\end{aligned}$$

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.

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

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

The TRS $\mathcal{R}_0$ is terminating (as proved by the recursive path ordering or by our approach). To prove innermost normalisation of the whole system we obtain the inequality

$$\mathsf{LOG}(\mathsf{s}(\mathsf{s}(x))) \succ \mathsf{LOG}(\mathsf{s}(\mathsf{half}(x)))$$

from the dependency pair of $\mathcal{R}_1$ as well as a monotonicity condition and $l \succsim r$ for the (usable) half-rules.

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).

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

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

For innermost normalisation we obtain the constraints

$$
\begin{aligned}
\mathsf{LOG}(\mathsf{s}(\mathsf{s}(x))) &\succ \mathsf{LOG}(\mathsf{s}(\mathsf{quot}(x, \mathsf{s}(\mathsf{s}(0))))) \\
x_1 \succsim x_2 \Rightarrow \mathsf{LOG}(\mathsf{s}(x_1)) &\succsim \mathsf{LOG}(\mathsf{s}(x_2))
\end{aligned}
$$

from the dependency pair of $\mathcal{R}_1$ and $l \succsim r$ for all rules of $\mathcal{R}_0$ (varying with the different versions of $\mathcal{R}_0$ we use).

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. The function rm is used to eliminate all occurrences of an element from a list.

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

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

Termination of $\mathcal{R}_0$ can be proved with our approach by considering this subsystem as a combination of the eq rules and the other rules. For $\mathcal{R}_1$ we obtain the constraint

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

Moreover, PURGE must be (weakly) monotonic on its argument and $l \succsim r$ must holds for all rules of $\mathcal{R}_0$.

A suitable mapping is

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

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] 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.

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

$$\mathsf{if}_{\mathsf{replace}}(\mathsf{false}, n, m, \mathsf{add}(k, x)) \to \mathsf{add}(k, \mathsf{replace}(n, m, x))$$

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

The TRS $\mathcal{R}_0$ is innermost normalising (resp. terminating) as can be proved by application of our technique. To complete the innermost normalisation proof we obtain the following inequalities for $\mathcal{R}_1$

$$\begin{aligned}
\mathsf{SELSORT}(\mathsf{add}(n, x)) &\succsim \mathsf{IF}_{\mathsf{selsort}}(\mathsf{eq}(n, \mathsf{min}(\mathsf{add}(n, x))), \mathsf{add}(n, x))\\
\mathsf{IF}_{\mathsf{selsort}}(\mathsf{true}, \mathsf{add}(n, x)) &\succ \mathsf{SELSORT}(x)\\
\mathsf{IF}_{\mathsf{selsort}}(\mathsf{false}, \mathsf{add}(n, x)) &\succ \mathsf{SELSORT}(\mathsf{replace}(\mathsf{min}(\mathsf{add}(n, x)), n, x)).
\end{aligned}$$

Moreover, we have to demand $l \succsim r$ for all rules of $\mathcal{R}_0$, as all these rules are usable and we obtain the following monotonicity constraints

$$\begin{aligned}
x_1 \succsim x_2 &\Rightarrow \mathsf{IF}_{\mathsf{selsort}}(x_1, \mathsf{add}(n, x)) \succsim \mathsf{IF}_{\mathsf{selsort}}(x_2, \mathsf{add}(n, x))\\
x_1 \succsim x_2 &\Rightarrow \mathsf{SELSORT}(x_1) \succsim \mathsf{SELSORT}(x_2)
\end{aligned}$$

A suitable mapping is given by

$$\begin{aligned}
\mathsf{add}(n, x) &\mapsto \mathsf{add}(x)\\
\mathsf{s}(n) &\mapsto \mathsf{s}\\
\mathsf{eq}(x, y) &\mapsto \mathsf{eq}\\
\mathsf{le}(x, y) &\mapsto \mathsf{le}\\
\mathsf{if}_{\mathsf{min}}(b, x) &\mapsto \mathsf{if}_{\mathsf{min}}(x)\\
\mathsf{replace}(x, y, z) &\mapsto z\\
\mathsf{if}_{\mathsf{replace}}(b, x, y, z) &\mapsto z\\
\mathsf{IF}_{\mathsf{selsort}}(b, x) &\mapsto x.
\end{aligned}$$

Then the resulting inequalities are satisfied by the recursive path ordering (where add must be greater than SELSORT in the precedence).

While for all of the preceding examples, innermost normalisation could also be proved without using the modularity result of Thm. 12, in this example the given ordering would not satisfy the constraints resulting from the innermost normalisation proof of the *whole* system. However, if the first two min-rules were replaced by $\mathsf{min}(\mathsf{add}(n, \mathsf{nil})) \to \mathsf{element}(n)$, then a similar ordering (without the mapping $\mathsf{s}(n) \mapsto \mathsf{s}$) would satisfy the constraints obtained for the whole TRS.

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.)

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

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

As in the other examples, the TRS $\mathcal{R}_0$ can be proved terminating by recursively applying our technique. For $\mathcal{R}_1$ we obtain the following inequalities

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

and the following monotonicity constraints (where we neglect monotonicity demands for positions which have a defined symbol above them).

$$x_1 \succsim x_2 \Rightarrow \mathsf{IF}_{\mathsf{minsort}}(x_1, \mathsf{add}(n, x), y) \succsim \mathsf{IF}_{\mathsf{minsort}}(x_2, \mathsf{add}(n, x), y)$$
$$x_1 \succsim x_2 \Rightarrow \mathsf{MINSORT}(x_1, \mathsf{nil}) \succsim \mathsf{MINSORT}(x_2, \mathsf{nil})$$

Moreover, $l \succsim r$ must hold for all rules of $\mathcal{R}_0$.

The synthesized ordering is a (weakly monotonic) polynomial ordering where false, true, 0, nil, eq and le are mapped to 0, $\mathsf{s}(x)$ is mapped to $x+1$, $\mathsf{min}(x)$ and $\mathsf{if}_{\mathsf{min}}(b, x)$ are mapped to x, $\mathsf{add}(n, x)$ is mapped to $n+x+1$, $\mathsf{app}(x, y)$ is mapped to $x+y$, $\mathsf{rm}(n, x)$ and $\mathsf{if}_{\mathsf{rm}}(b, n, x)$ are mapped to x, $\mathsf{MINSORT}(x, y)$ is mapped to $(x+y)^2 + 2x + y + 1$ and $\mathsf{IF}_{\mathsf{minsort}}(b, x, y)$ is mapped to $(x+y)^2 + 2x + y$.

This example is inspired by an algorithm from [BM79] and [Wal94]. In the corresponding example from [Ste95a] the rules for le, eq, $\mathsf{if}_{\mathsf{rm}}$, and $\mathsf{if}_{\mathsf{min}}$ were missing.

8.11 Quicksort

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

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

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

The TRS $\mathcal{R}_0$ can be proved terminating by our approach. For $\mathcal{R}_1$ we obtain $l \succsim r$ for all rules of $\mathcal{R}_0$ (except the app-rules because they are not usable),

$\mathsf{QUICKSORT}$ must be weakly monotonic on its argument, and we have to demand the following constraints.

$$\mathsf{QUICKSORT}(\mathsf{add}(n,x)) \succ \mathsf{QUICKSORT}(\mathsf{low}(n,x))$$
$$\mathsf{QUICKSORT}(\mathsf{add}(n,x)) \succ \mathsf{QUICKSORT}(\mathsf{high}(n,x))$$

A suitable mapping is

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

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 $[1,5,2,4,3]$.

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

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

Termination of $\mathcal{R}_0$, the first four rules, can easily be proved by the recursive path ordering or by our technique. For innermost normalisation we obtain the constraint

$$\mathsf{SHUFFLE}(\mathsf{add}(n,x)) \succ \mathsf{SHUFFLE}(\mathsf{reverse}(x)),$$

$\mathsf{SHUFFLE}$ must be weakly monotonic, and $l \succsim r$ must hold for all rules of $\mathcal{R}_0$.

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

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

$$\begin{aligned}
\mathsf{reach}(x, y, \epsilon, h) &\to \mathsf{false}\\
\mathsf{reach}(x, y, \mathsf{edge}(u, v, i), h) &\to \mathsf{if}_{\mathsf{reach_1}}(\mathsf{eq}(x, u), x, y, \mathsf{edge}(u, v, i), h)\\
\mathsf{if}_{\mathsf{reach_1}}(\mathsf{true}, x, y, \mathsf{edge}(u, v, i), h) &\to \mathsf{if}_{\mathsf{reach_2}}(\mathsf{eq}(y, v), x, y, \mathsf{edge}(u, v, i), h)\\
\mathsf{if}_{\mathsf{reach_2}}(\mathsf{true}, x, y, \mathsf{edge}(u, v, i), h) &\to \mathsf{true}\\
\mathsf{if}_{\mathsf{reach_2}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) &\to \mathsf{or}(\mathsf{reach}(x, y, i, h),\\
&\qquad\quad \mathsf{reach}(v, y, \mathsf{union}(i, h), \epsilon))\\
\mathsf{if}_{\mathsf{reach_1}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) &\to \mathsf{reach}(x, y, i, \mathsf{edge}(u, v, h))
\end{aligned}$$

The TRS $\mathcal{R}_0$ can be proved innermost normalising (and terminating) very easily, e.g. by our technique. For $\mathcal{R}_1$ we obtain

$$\begin{aligned}
\mathsf{REACH}(x, y, \mathsf{edge}(u, v, i), h) &\succsim \mathsf{IF}_{\mathsf{reach_1}}(\mathsf{eq}(x, u), x, y, \mathsf{edge}(u, v, i), h)\\
\mathsf{IF}_{\mathsf{reach_1}}(\mathsf{true}, x, y, \mathsf{edge}(u, v, i), h) &\succsim \mathsf{IF}_{\mathsf{reach_2}}(\mathsf{eq}(y, v), x, y, \mathsf{edge}(u, v, i), h)\\
\mathsf{IF}_{\mathsf{reach_2}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) &\succ \mathsf{REACH}(x, y, i, h)\\
\mathsf{IF}_{\mathsf{reach_2}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) &\succ \mathsf{REACH}(v, y, \mathsf{union}(i, h), \epsilon)\\
\mathsf{IF}_{\mathsf{reach_1}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) &\succ \mathsf{REACH}(x, y, i, \mathsf{edge}(u, v, h)),
\end{aligned}$$

several (weak) monotonicity conditions, and $l \succsim r$ for all rules of $\mathcal{R}_0$ except the or-rules (because these rules are not usable).

A mapping to polynomials results in a suitable ordering. The interpretation is: $\mathsf{eq}(x, y)$, true, false, ϵ, 0, and $\mathsf{s}(x)$ are mapped to 0, $\mathsf{edge}(x, y, g)$ is mapped to $g + 2$, $\mathsf{union}(g, h)$ is mapped to $g + h$, $\mathsf{REACH}(x, y, g, h)$ 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 two rules of $\mathcal{R}_0$ are easily proved terminating. For $\mathcal{R}_1$ we obtain

$$\mathsf{LESS\_LEAVES}(\mathsf{cons}(u, v), \mathsf{cons}(w, z)) \succ \mathsf{LESS\_LEAVES}(\mathsf{concat}(u, v), \mathsf{concat}(w, z)).$$

Moreover, the concat-rules must be weakly decreasing and less_leaves must be weakly monotonic on both arguments.

A suitable (polynomial) interpretation is: leaf is mapped to 0, $\mathsf{cons}(u, v)$ is mapped to $1 + u + v$, $\mathsf{concat}(u, v)$ is mapped to $u + v$, and $\mathsf{LESS\_LEAVES}(x, y)$ is 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 normalisation suffices for proving termination.

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

For proving innermost normalisation of this TRS we have to find a well-founded ordering satisfying the constraints

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

(There are no usable rules.)

In this way, termination of this TRS is easily proved by mapping $\mathsf{s}(x)$ to $x+1$, 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 $\times$.

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

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

The constraints for innermost normalisation of $\mathcal{R}_0$ are

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

which are satisfied by the recursive path ordering.

For $\mathcal{R}_1$ we obtain

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

which is also satisfied by the recursive path ordering.

8.17 Addition with Nested Recursion

The following (non-overlapping) TRS for addition from [Ste95a] has nested recursion.

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

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 constraints generated by our method. (Here all rules are usable.) In this way, innermost normalisation (and thereby, termination) can easily be proved.

8.18 Multiplication and Addition

The following (non-overlapping) system is taken from [Der87, p. 101].

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

The only inequalities resulting from a dependency pair on a cycle of the innermost dependency graph is

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

and all rules are usable (hence, we have to demand $l \succsim r$ for all rules). Moreover, TIMES must be weakly monotonic on its second argument.

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 constraints obtained are satisfied by the natural polynomial ordering, where $\mathsf{TIMES}(x, y)$ is mapped to y.

8.19 Nested Recursion 1

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

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

With our approach, however, an automated innermost normalisation (and hence, 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.

8.20 Nested Recursion 2

This system (by *Christoph Walther*), which is similar to the preceding one, has been examined in [Ste95a].

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

The constraints resulting from our technique 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 0 and s are interpreted as usual.

8.21 Nested Recursion 3

As an example of a string rewriting 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 (as both of them are usable), are

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

and F must be weakly monotonic.

All constraints 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 $x+1$.

8.22 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 generalization 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 $x+y$ satisfies the inequality obtained by the technique.

8.23 Determining Cycles in Innermost Dependency Graphs 1

The following system is from [Ste95a].

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

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

8.24 Determining Cycles in Innermost Dependency Graphs 2

Another example in which the innermost dependency graph plays an important role is a TRS introduced in [FZ95] to demonstrate the technique of 'dummy elimination'.

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

Since $\mathsf{F}(\mathsf{a}(\ldots))$ does not unify with $\mathsf{F}(\mathsf{g}(x))$, the only inequality to satisfy is

$$\mathsf{F}(\mathsf{g}(x)) \succ \mathsf{F}(x)$$

which is easily satisfied by the recursive path ordering.

8.25 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].

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

With our approach, innermost normalisation (resp. termination) of this system is again obvious, because the innermost dependency graph does not contain any cycles (as $\mathsf{F}(\mathsf{b})$ does not unify with $\mathsf{F}(\mathsf{a})$ and $\mathsf{G}(\mathsf{a})$ does not unify with $\mathsf{G}(\mathsf{b})$). Hence, innermost normalisation is proved.

8.26 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:

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

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

8.27 Reversing Lists

The following system is a slight variant of a TRS proposed in [HH82, 'brev']. Here, "$x.l$" represents the insertion of an element x in front of the list l and "$x.y.l$" abbreviates "$x.(y.l)$". 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.

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

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

Termination of $\mathcal{R}_0$ is easily proved (e.g. by the recursive path ordering or by our technique). For innermost normalisation the resulting inequalities from the dependency pairs of $\mathcal{R}_1$ are

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

$l \succsim r$ for all rules and a monotonicity condition. These constraints 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 $\mathsf{REV2}(x, l)$ is mapped to l.

8.28 Even and Odd

The following (non-simply terminating) TRS can be used to compute whether a natural number is even resp. odd. More precisely, $\mathsf{evenodd}(t, 0)$ reduces to true if t is even and $\mathsf{evenodd}(t, \mathsf{s}(0))$ reduces to true if t is odd. (In other words, the second argument of evenodd determines whether evenodd computes the "even" or the "odd" function. Such rewrite systems are often obtained when transforming mutually recursive functions into one function without mutual recursion, cf. [Gie96].)

$$\begin{aligned}\mathsf{not}(\mathsf{true}) &\to \mathsf{false}\\ \mathsf{not}(\mathsf{false}) &\to \mathsf{true}\end{aligned}$$

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

We obtain the following constraints for innermost normalisation (and hence, termination) of $\mathcal{R}$.

$$\begin{aligned}\mathsf{EVENODD}(x,0) &\succsim \mathsf{EVENODD}(x,\mathsf{s}(0))\\ \mathsf{EVENODD}(\mathsf{s}(x),\mathsf{s}(0)) &\succ \mathsf{EVENODD}(x,0)\end{aligned}$$

By mapping $\mathsf{EVENODD}(x,y)$ to x, the recursive path ordering satisfies these constraints.

8.29 Modularity, Version 1

The following example demonstrates the usefulness of modularity results.

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

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

Modularity results (such as Thm. 12 or a result from [MT91] stating that completeness is modular for constructor systems with disjoint sets of defined symbols) allow us to prove innermost normalisation (and thereby, termination) of both rules separately. So we may use different well-founded orderings for the constraint

$$\mathsf{F}(\mathsf{c}(x,\mathsf{s}(y))) \;\succ\; \mathsf{F}(\mathsf{c}(\mathsf{s}(x),y))$$

and the constraint

$$\mathsf{G}(\mathsf{c}(x,\mathsf{s}(y))) \;\succ\; \mathsf{G}(\mathsf{c}(\mathsf{s}(x),y))$$

(i.e. one time we can map $\mathsf{c}(x,y)$ to y and one time we can map it to x). In this way, the termination proof of this system is trivial.

8.30 Modularity, Version 2

A second example for which it really matters that different orderings may be found for different parts of the TRS is the following TRS

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

By using the modularity result of Thm. 12, innermost normalisation of the TRS

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

has to be proved first, which is easily established. It remains to prove that no infinite innermost chain consisting of the dependency pair

$$\langle \mathsf{H}(x.y), \mathsf{H}(\mathsf{g}(x.y))\rangle$$

exists. This is proved by finding a suitable ordering satisfying the inequalities

$$\begin{aligned}
\mathsf{g}(0.y) &\succsim \mathsf{g}(y)\\
\mathsf{g}(\mathsf{s}(x).y) &\succsim \mathsf{s}(x)\\
\mathsf{H}(x.y) &\succ \mathsf{H}(\mathsf{g}(x.y)).
\end{aligned}$$

A suitable well-founded quasi-ordering satisfying these inequalities is given by a polynomial interpretation, where $\mathsf{g}(x)$ and $\mathsf{s}(x)$ are mapped to 0, $x.y$ is mapped to 1 and $\mathsf{H}(x)$ is mapped to x. Hence, by using the modularity result, the TRS can be proved innermost normalising automatically. Since the TRS is non-overlapping, termination of this TRS is thereby proved.

8.31 Second Running Example

The second example of this paper

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

is a non-simply terminating TRS. As is explained in Sect. 4 the only two demands on the ordering are given by

$$\begin{aligned}
\mathsf{Q}(\mathsf{s}(x), \mathsf{s}(y), z) &\succ \mathsf{Q}(x, y, z) \text{ and}\\
\mathsf{Q}(x, 0, \mathsf{s}(z)) &\succsim \mathsf{Q}(x, \mathsf{s}(z), \mathsf{s}(z)).
\end{aligned}$$

The mapping of $\mathsf{Q}(x, y, z)$ to x and the recursive path ordering satisfy these demands.

Termination of this system cannot be proved automatically using the method of [AG97a], as the constraints generated by the technique of [AG97a] are not satisfied by any total or quasi-simplification ordering. The reason is that in the latter method there is no concept of usable rules and that this method is restricted to weakly monotonic orderings.

8.32 Second Running Example with Plus Rules

Changing the last rule of the above example such that $\mathsf{s}(z)$ is computed by adding 1 to z results in the TRS

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

This TRS is, as the previous one, not simply terminating. By performing one narrowing step on the term $\mathsf{QUOT}(x, z + \mathsf{s}(0), \mathsf{s}(z))$ we obtain that there is only one cycle in the approximation of the innermost dependency graph. The only usable rules are the rules for addition. Therefore, the only inequalities to solve are

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

By mapping $\mathsf{QUOT}(x, y, z)$ to x these inequalities are satisfied by the recursive path ordering. Hence, the TRS is terminating.

8.33 A non-totally terminating TRS

The following example is from [Ste95a].

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

This TRS is not totally terminating (and the constraints generated by the method of [AG97a] are not satisfied by any total well-founded quasi-ordering). However, with our method innermost normalisation (and thereby, termination) can easily be proved. The reason is that after applying one narrowing step to $\mathsf{f}(\mathsf{a}, \mathsf{b})$ we obtain $\mathsf{f}(\mathsf{a}, \mathsf{c})$ which is not unifiable with $\mathsf{f}(x, x)$. Hence, there is no cycle in the innermost dependency graph.

8.34 Intervals of Natural Numbers

The following TRS from [Ste95a]

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

is non-overlapping, too. No narrowing is needed to obtain that there are only two cycles in the dependency graph. Furthermore, we obtain that the set of usable rules is empty. The generated inequalities are

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

By mapping $\mathsf{INT}(x, y)$ to y these inequalities are satisfied by the recursive path ordering. Thus, the TRS is terminating. Again, termination of this system cannot be proved automatically using the method of [AG97a].

8.35 Renaming in the Lambda Calculus

The following system is a variation of an algorithm from [MA96]. The purpose of the function $\mathsf{ren}(x, y, t)$ is to replace every free occurrence of the variable x in the term t by the variable y. If the substitution of x by y should be applied to a lambda term $\mathsf{lambda}(z, t)$ (which represents $\lambda z.t$), then we first apply an α-conversion step to $\mathsf{lambda}(z, t)$, i.e. we rename z to a new variable (which is different from x or y and which does not occur in $\mathsf{lambda}(z, t)$). Subsequently, the renaming of x to y is applied to the resulting term. For that reason in this TRS there is a nested recursive call of the function ren.

Variables are represented by $\mathsf{var}(l)$ where l is a list of terms. Therefore, the variable $\mathsf{var}(x.y.\mathsf{lambda}(z, t).\mathsf{nil})$ is distinct from x and y and from all variables occurring in $\mathsf{lambda}(z, t)$.

$$\begin{aligned} \mathsf{and}(\mathsf{true}, y) &\to y \\ \mathsf{and}(\mathsf{false}, y) &\to \mathsf{false} \\ \mathsf{eq}(\mathsf{nil}, \mathsf{nil}) &\to \mathsf{true} \\ \mathsf{eq}(t.l, \mathsf{nil}) &\to \mathsf{false} \\ \mathsf{eq}(\mathsf{nil}, t.l) &\to \mathsf{false} \\ \mathsf{eq}(t.l, t'.l') &\to \mathsf{and}(\mathsf{eq}(t, t'), \mathsf{eq}(l, l')) \\ \mathsf{eq}(\mathsf{var}(l), \mathsf{var}(l')) &\to \mathsf{eq}(l, l') \\ \mathsf{eq}(\mathsf{var}(l), \mathsf{apply}(t, s)) &\to \mathsf{false} \\ \mathsf{eq}(\mathsf{var}(l), \mathsf{lambda}(x, t)) &\to \mathsf{false} \\ \mathsf{eq}(\mathsf{apply}(t, s), \mathsf{var}(l)) &\to \mathsf{false} \\ \mathsf{eq}(\mathsf{apply}(t, s), \mathsf{apply}(t', s')) &\to \mathsf{and}(\mathsf{eq}(t, t'), \mathsf{eq}(s, s')) \\ \mathsf{eq}(\mathsf{apply}(t, s), \mathsf{lambda}(x, t)) &\to \mathsf{false} \\ \mathsf{eq}(\mathsf{lambda}(x, t), \mathsf{var}(l)) &\to \mathsf{false} \\ \mathsf{eq}(\mathsf{lambda}(x, t), \mathsf{apply}(t, s)) &\to \mathsf{false} \\ \mathsf{eq}(\mathsf{lambda}(x, t), \mathsf{lambda}(x', t')) &\to \mathsf{and}(\mathsf{eq}(x, x'), \mathsf{eq}(t, t')) \end{aligned}$$

$$\mathsf{if}(\mathsf{true}, \mathsf{var}(k), \mathsf{var}(l')) \to \mathsf{var}(k)$$
$$\mathsf{if}(\mathsf{false}, \mathsf{var}(k), \mathsf{var}(l')) \to \mathsf{var}(l')$$

$$\mathsf{ren}(\mathsf{var}(l), \mathsf{var}(k), \mathsf{var}(l')) \to \mathsf{if}(\mathsf{eq}(l, l'), \mathsf{var}(k), \mathsf{var}(l'))$$
$$\mathsf{ren}(x, y, \mathsf{apply}(t, s)) \to \mathsf{apply}(\mathsf{ren}(x, y, t), \mathsf{ren}(x, y, s))$$
$$\begin{aligned}\mathsf{ren}(x, y, \mathsf{lambda}(z, t)) \to\ & \mathsf{lambda}(\mathsf{var}(x.y.\mathsf{lambda}(z, t).\mathsf{nil}),\\ & \quad \mathsf{ren}(x, y, \mathsf{ren}(z, \mathsf{var}(x.y.\mathsf{lambda}(z, t).\mathsf{nil}), t)))\end{aligned}$$

Termination of $\mathcal{R}_0$ can for instance be proved by the recursive path ordering (or by our technique). To complete the innermost normalisation proof, we obtain the following constraints for $\mathcal{R}_1$.

$$\mathsf{REN}(x, y, \mathsf{apply}(t, s)) \succ \mathsf{REN}(x, y, t)$$
$$\mathsf{REN}(x, y, \mathsf{apply}(t, s)) \succ \mathsf{REN}(x, y, s)$$
$$\mathsf{REN}(x, y, \mathsf{lambda}(z, t)) \succ \mathsf{REN}(z, \mathsf{var}(x.y.\mathsf{lambda}(z, t).\mathsf{nil}), t)$$
$$\mathsf{REN}(x, y, \mathsf{lambda}(z, t)) \succ \mathsf{REN}(x, y, \mathsf{ren}(z, \mathsf{var}(x.y.\mathsf{lambda}(z, t).\mathsf{nil}), t))$$

Moreover, REN must be weakly monotonic on its third argument and $l \succsim r$ must hold for all rules of the TRS (as all rules are usable).

A well-founded ordering satisfying these constraints can easily be synthesized automatically. For instance, one can use the following polynomial interpretation where $\mathsf{REN}(x, y, t)$ is mapped to t, $\mathsf{ren}(x, y, t)$ is also mapped to t, $\mathsf{lambda}(x, t)$ is mapped to $t + 1$, $\mathsf{apply}(t, s)$ is mapped to $t + s + 1$, $\mathsf{and}(x, y)$ is mapped to y, and where nil, $\mathsf{var}(l)$, true, false, $\mathsf{eq}(t, s)$, and $\mathsf{if}(x, y, z)$ are all mapped to the constant 0.

This TRS is non-simply terminating because the left-hand side of the last rule is embedded in its right-hand side. Since the TRS is a locally confluent overlay system, innermost normalisation suffices for termination. Note that the modularity result of Thm. 12 is essential for this termination proof. If termination of the *whole* system would have to be proved at once, then the resulting inequalities would not be satisfied by any polynomial or path ordering. For that reason the method of [AG97a] (for termination instead of innermost normalisation) cannot handle this example automatically.

Acknowledgements

We would like to thank Hans Zantema, Aart Middeldorp, Thomas Kolbe, and the referees for constructive criticism and many helpful comments.

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. `http://www.cs.ruu.nl/`.

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

[AG97a] T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. In M. Dauchet, editor, *Proceedings of the 22nd International Colloquium on Trees in Algebra and Programming, CAAP'97*, Lecture Notes in Computer Science, Lille, France, April 1997. Springer Verlag, Berlin.

[AG97b] T. Arts and J. Giesl. Proving Innermost Normalisation Automatically. In *Proceedings of the 8th International Conference on Rewriting Techniques and Applications, RTA-97*, Lecture Notes in Computer Science, Sitges, Spain, June 1997. Springer Verlag, Berlin.

[Art96] T. Arts. Termination by absence of infinite chains of dependency pairs. In H. 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.

[AZ95] T. Arts and H. Zantema. Termination of logic programs using semantic unification. In M. Proietti, editor, *Proceedings of the 5th International Workshop on Logic Program Synthesis and Transformation*, volume 1048 of *Lecture Notes in Computer Science*, pages 219–233, Utrecht, September 1995. Springer Verlag, Berlin.

[BD86] L. Bachmair and N. Dershowitz. Commutation, transformation and termination. In J.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.

[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.

[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.

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

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

[Dro89] K. Drosten. Termersetzungssysteme: Grundlagen der Prototyp-Generierung algebraischer Spezifikationen. Springer, Berlin, 1989. In German.

[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 C. 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 H. 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.

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

[Gie95b] J. Giesl. Generating polynomial orderings for termination proofs. In J. 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.

[Gra95] B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.

[Gra96] B. Gramlich. On proving termination by innermost termination. In H. Ganzinger, editor, *Proceedings of the 7th International Conference on Rewriting Techniques and Applications, RTA-96*, volume 1103 of *Lecture Notes in Computer Science*, pages 93–107, New Brunswick, NJ, USA, July 1996. Springer Verlag, Berlin.

[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.

[Hul80] J.M. Hullot. Canonical forms and unification. In W. Bibel and R. Kowalski, editors, *Proceedings of the 5th International Conference on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334, Les Arcs, France, July 1980. Springer Verlag, Berlin.

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

[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 Technical University, Ruston, LA, 1979.

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

[MA96] D. McAllester and K. Arkoudas. Walther recursion. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction, CADE-13*, volume 1104 of *Lecture Notes in Com-*

puter Science, pages 643–657, New Brunswick, NJ, USA, July/August 1996. Springer Verlag, Berlin.

[MOZ96] A. Middeldorp, H. Ohsaki, and H. Zantema. Transforming termination by self-labelling. In M.A. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction, CADE-13*, volume 1104 of *Lecture Notes in Computer Science*, pages 373–387, New Brunswick, NJ, USA, July/August 1996. Springer Verlag, Berlin.

[MT91] A. Middeldorp and Y. Toyama. Completeness of combinations of constructor systems. In R.V. Book, editor, *Proceedings of the 4th International Conference on Rewriting Techniques and Applications, RTA-91*, volume 488 of *Lecture Notes in Computer Science*, pages 188–199, Como, Italy, April 1991. Springer Verlag, Berlin.

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

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

[Ste95a] J. Steinbach. Automatic termination proofs with transformation orderings. In J. 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. Long version also appeared as Technical Report SR-92-23, Universität 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.

[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.