

Proving Innermost Normalisation Automatically^{*}

Thomas Arts¹ and Jürgen Giesl²

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 redexes 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 [DH95, Ste95b], semantic interpretations [Lan79, BL87, Ste94, Zan94, Gie95], transformation orderings [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.

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

^{*} Appeared in the *Proceedings of the 8th International Conference on Rewriting Techniques and Applications (RTA-97)*, Sitges, Spain, LNCS 1232, 1997. This work was partially supported by the Deutsche Forschungsgemeinschaft under grant no. Wa 652/7-1 as part of the focus program “Deduktion”.

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 [AG96b].

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 $f(\dots)$ 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 $D_{\mathcal{R}} = \{\text{root}(l) \mid l \rightarrow r \in R\}$ 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 0 and s .

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

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 $f(s_1, \dots, s_n) \rightarrow C[g(t_1, \dots, t_m)]$ (where f and g are defined symbols and C denotes some context), we compare the argument tuple s_1, \dots, s_n with the tuple t_1, \dots, 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, \dots, s_n

and t_1, \dots, t_m we compare the *terms* $F(s_1, \dots, s_n)$ and $G(t_1, \dots, t_m)$. 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 $f(s_1, \dots, s_n) \rightarrow C[g(t_1, \dots, t_m)]$ is a rewrite rule of R with $f, g \in D$, then $\langle F(s_1, \dots, s_n), G(t_1, \dots, t_m) \rangle$ is a *dependency pair* of \mathcal{R} .

In the above example we obtain the following dependency pairs:

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

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

$$\langle \mathbf{G}(\mathbf{s}(x)), \mathbf{G}(x) \rangle \quad (3)$$

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, $\mathbf{s}0$ abbreviates $\mathbf{s}(0)$ etc.)

$$f(\mathbf{gs}0, \mathbf{s}0, \mathbf{gs}0) \rightarrow f(\mathbf{gs}0, \mathbf{gs}0, \mathbf{gs}0) \rightarrow f(\mathbf{gs}0, \mathbf{sg}0, \mathbf{gs}0) \rightarrow f(\mathbf{gs}0, \mathbf{s}0, \mathbf{gs}0) \rightarrow \dots$$

However, this reduction is not an innermost reduction, because in the first reduction step the subterm $\mathbf{gs}0$ 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{i} ”.

Every infinite reduction corresponds to an infinite introduction of new redexes. To trace these newly introduced redexes 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 $t_j\sigma \xrightarrow{*}_{\mathcal{R}} s_{j+1}\sigma$ (cf. [AG97]). 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 $t_j\sigma \xrightarrow{*}_{\mathcal{R}} s_{j+1}\sigma$ 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 $t_j\sigma \xrightarrow{i}_{\mathcal{R}} s_{j+1}\sigma$ 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

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

because $\mathbf{G}(x_1)\sigma \xrightarrow{i}_{\mathcal{R}} \mathbf{G}(\mathbf{s}(x_2))\sigma$ and $\mathbf{G}(x_2)\sigma \xrightarrow{i}_{\mathcal{R}} \mathbf{G}(\mathbf{s}(x_3))\sigma$ holds for the substitution σ that replaces x_1 by $\mathbf{s}(\mathbf{s}(x_3))$ and x_2 by $\mathbf{s}(x_3)$. In fact any finite sequence of

the dependency pair $\langle G(s(x)), G(x) \rangle$ 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 subterm¹ $f_1(\mathbf{u}_1)$ 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 $f_1(\mathbf{w}_1) \rightarrow r_1$ is applied to $f_1(\mathbf{v}_1)$, i.e. a substitution σ_1 exists such that $f_1(\mathbf{v}_1) = f_1(\mathbf{w}_1)\sigma_1 \xrightarrow{i} r_1\sigma_1$. Hence, we have $\mathbf{u}_1 \xrightarrow{i}^* \mathbf{w}_1\sigma_1$ 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 $f_1(\mathbf{w}_1) \rightarrow C[f_2(\mathbf{u}_2)]$.

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 $f_i(\mathbf{w}_i) \rightarrow r_i$ 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 $\mathbf{v}_i = \mathbf{w}_i\sigma_i$.

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 $F_i(\mathbf{u}_i\sigma_{i-1}) \xrightarrow{i}^* F_i(\mathbf{v}_i)$ 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 $F_i(\mathbf{u}_i)\sigma \xrightarrow{i}^* F_i(\mathbf{w}_i)\sigma$ for all i . Thus, this sequence is indeed an infinite innermost \mathcal{R} -chain.

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

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

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

resp. $f_j(\mathbf{t}_j)\sigma \xrightarrow{i^*_{\mathcal{R}}} f_j(\mathbf{s}_j)\sigma$, 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 $f(\mathbf{s}) \rightarrow C[g(\mathbf{t})]$ for some context C . Since no redex occurs in $\mathbf{s}\sigma$, this reduction also follows the innermost strategy, i.e. $f(\mathbf{s})\sigma \xrightarrow{i}_{\mathcal{R}} C[g(\mathbf{t})]\sigma$. Therefore, we obtain the following infinite innermost reduction.

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

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-founded² quasi-ordering \succsim such that

$$s_1\sigma \succ t_1\sigma \succsim s_2\sigma \succ t_2\sigma \succsim s_3\sigma \succ t_3\sigma \dots \quad (4)$$

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):

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

Moreover, we have to ensure $t_j\sigma \succsim s_{j+1}\sigma$ whenever $t_j\sigma \xrightarrow{i^*_{\mathcal{R}}} s_{j+1}\sigma$ holds. For that purpose we demand the constraints $l \succsim r$ for all those rules $l \rightarrow r$ that can

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.

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 $\mathbf{G}(x)$ (because \mathbf{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 $\mathbf{F}(y, y, \mathbf{g}(x))$ is \mathbf{g} , the only rules that may be applied on normal instantiations of this term are the two \mathbf{g} -rules of the TRS. Since these \mathbf{g} -rules can never introduce a new redex with root symbol \mathbf{f} , these two \mathbf{g} -rules are the only rules that can be used to reduce any normal instantiation of $\mathbf{F}(y, y, \mathbf{g}(x))$. Hence, in this example we only have to demand that these rules should be weakly decreasing.

$$\mathbf{g}(s(x)) \succcurlyeq s(\mathbf{g}(x)), \quad \mathbf{g}(0) \succcurlyeq 0 \quad (6)$$

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 $Rls(f) = \{f(\mathbf{s}) \rightarrow r \mid f(\mathbf{s}) \rightarrow r \text{ in } R\}$. For any term t , $\mathcal{U}(t)$ is the smallest subset of R such that

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

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

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

$$x_1 \succcurlyeq x_2 \Rightarrow \mathbf{F}(y, y, x_1) \succcurlyeq \mathbf{F}(y, y, x_2). \quad (7)$$

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 \succcurlyeq be a well-founded quasi-ordering where both \succcurlyeq and \succ are closed under substitution. If \succcurlyeq is weakly monotonic on all symbols apart from the tuple symbols and if \succcurlyeq satisfies the following constraints for all dependency pairs $\langle s, t \rangle$*

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

then \mathcal{R} is innermost normalising.

Proof. Suppose $\langle s_1, t_1 \rangle \langle s_2, t_2 \rangle \dots$ 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 $t_j \sigma \succ s_{j+1} \sigma$. This results in an infinite decreasing sequence $s_1 \sigma \succ t_1 \sigma \succ s_2 \sigma \succ t_2 \sigma \succ \dots$ which is a contradiction to the well-foundedness of \succ . Thus, no infinite innermost \mathcal{R} -chain exists and by Thm. 4, the TRS is innermost normalising. \square

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, $s(x)$ is mapped to $x+1$, $g(x)$ is mapped to $x+2$, $F(x, y, z)$ is mapped to $(x-y)^2+1$, and $G(x)$ is mapped to x . Methods to synthesize polynomial orderings automatically have for instance been developed in [Ste94, Gie95]. 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 $F(x, y, z)$ is mapped to a polynomial which is weakly monotonic³ 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 $\text{quot}(x, y, z)$ is used to compute $1 + \lfloor \frac{x-y}{z} \rfloor$, if $x \geq y$ and $z \neq 0$ (i.e. $\text{quot}(x, y, y)$ computes $\lfloor \frac{x}{y} \rfloor$).

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

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 $[F]$, then instead of (7) one could demand that the *partial derivative* of $[F](y, y, x)$ with respect to x should be non-negative, i.e. $\frac{\partial [F](y, y, x)}{\partial x} \geq 0$, cf. [Gie95]. 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).

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 \mathbf{Q} denotes the tuple symbol for `quot`).

$$\langle \mathbf{Q}(s(x), s(y), z), \mathbf{Q}(x, y, z) \rangle \quad (8)$$

$$\langle \mathbf{Q}(x, 0, s(z)), \mathbf{Q}(x, s(z), s(z)) \rangle \quad (9)$$

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 are decreasing. These constraints are for instance satisfied by the polynomial ordering where 0 is mapped to the number 0, $s(x)$ is mapped to $x + 1$, and $\mathbf{Q}(x, y, z)$ 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 \mathbf{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 $\mathbf{Q}(x_1, s(z_1), s(z_1))\sigma$ can never reduce to any instantiation of $\mathbf{Q}(x_2, 0, s(z_2))$. The reason is that the second arguments $s(z_1)$ 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.

$$\mathbf{Q}(s(x), s(y), z) \succ \mathbf{Q}(x, y, z) \quad (10)$$

$$\mathbf{Q}(x, 0, s(z)) \succsim \mathbf{Q}(x, s(z), s(z)) \quad (11)$$

In general, to determine those dependency pairs which may possibly follow each other in innermost chains, we define the following graph⁴.

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

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 $l \rightarrow r$ in $\mathcal{U}(t)$,
- (c) $x_1 \succsim y_1 \wedge \dots \wedge x_n \succsim y_n \Rightarrow C[x_1, \dots, x_n] \succsim C[y_1, \dots, y_n]$, if $t = C[f_1(\mathbf{u}_1), \dots, f_n(\mathbf{u}_n)]$, where C is a context without defined symbols and f_1, \dots, f_n are defined symbols,

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

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.

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. \square

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 $Q(x, y, z)$ 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 $\text{CAP}(t)$ 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 $\text{CAP}(t)$ and v are unifiable. Moreover, the most general unifier (mgu) of $\text{CAP}(t)$ and v must be a normal substitution.

Theorem 9 (Computing Innermost Dependency Graphs). *Let \mathcal{R} be a TRS. If $t\sigma \rightarrow_{\mathcal{R}}^* v\sigma$ holds for some normal substitution σ such that $v\sigma$ is a normal form, then $\text{CAP}(t)$ 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 $t\sigma \rightarrow_{\mathcal{R}}^* u$, then there exists a normal substitution τ (whose domain only includes variables that were newly introduced in the construction of $\text{CAP}(t)$) such that⁵ $\text{CAP}(t)\sigma\tau = u$. Thus in particular, if $t\sigma \rightarrow_{\mathcal{R}}^* v\sigma$, we have $\text{CAP}(t)\sigma\tau = v\sigma (= v\sigma\tau$, because the variables of $v\sigma$ do not occur in the domain of τ). Hence, $\text{CAP}(t)$ and v unify. Moreover, for the mgu μ of $\text{CAP}(t)$ 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 $\text{CAP}(t) = t$. So $\text{CAP}(t)\sigma = u$, i.e. in this case τ is the empty substitution.

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

If $t = c(t_1, \dots, t_k)$ for some constructor $c \in C$, then u has to be of the form $c(u_1, \dots, u_k)$ and $t_i\sigma \rightarrow_{\mathcal{R}}^* u_i$ holds for all i . By the induction hypothesis we obtain that normal substitutions τ_i exist such that $\text{CAP}(t_i)\sigma\tau_i = u_i$ for all i . Note

⁵ Here, “ $t\sigma\tau$ ” is defined as “ $(t\sigma)\tau$ ”, i.e. σ is applied first.

that those variables in $\text{CAP}(t_i)$ that were introduced by CAP are disjoint from the newly introduced variables in $\text{CAP}(t_j)$ (for $i \neq j$). Hence, if $\tau = \tau_1 \circ \dots \circ \tau_k$, then for all i we have $\text{CAP}(t_i)\sigma\tau = u_i$ resp. $\text{CAP}(t)\sigma\tau = c(\text{CAP}(t_1), \dots, \text{CAP}(t_k))\sigma\tau = c(u_1, \dots, u_k) = u$ and again, τ is normal. \square

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 $\text{CAP}(t)$ 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.

$$\begin{array}{l} f(x, \mathbf{g}(x)) \rightarrow f(1, \mathbf{g}(x)) \\ \mathbf{g}(1) \quad \rightarrow \mathbf{g}(0) \end{array}$$

In this example, the first dependency pair $\langle F(x, \mathbf{g}(x)), F(1, \mathbf{g}(x)) \rangle$ does not occur on a cycle in the innermost dependency graph, although $\text{CAP}(F(1, \mathbf{g}(x))) = F(1, y)$ unifies with $F(x, \mathbf{g}(x))$ using a mgu that replaces x by 1 and y by $\mathbf{g}(1)$. However, $\mathbf{g}(1)$ is not a normal form and hence, this mgu is not a normal substitution. The second dependency pair $\langle G(1), G(0) \rangle$ cannot occur on a cycle either, since $G(0)$ does not unify with $G(1)$. 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.

$$\begin{array}{l} \text{quot}(x, 0, s(z)) \rightarrow s(\text{quot}(x, z + s(0), s(z))) \\ 0 + y \rightarrow y \\ s(x) + y \rightarrow s(x + y) \end{array}$$

Now instead of dependency pair (9) we obtain

$$\langle Q(x, 0, s(z)), Q(x, z + s(0), s(z)) \rangle. \quad (12)$$

Note that in our approximation of the innermost dependency graph there would be an arc from (12) to itself, because after replacing $z + s(0)$ 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 $(z + s(0))\sigma$ 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 $\text{PLUS}(x, y)$ 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 $t \rightsquigarrow_{\mathcal{R}} q$), if there exists a nonvariable position p in t , μ is the most general unifier of $t|_p$ and l for some rewrite rule $l \rightarrow r$ of \mathcal{R} , and $q = t\mu[r\mu]_p$. (Here, the variables of $l \rightarrow r$ must have been renamed to fresh variables.)

To find out whether $t\sigma \xrightarrow{i}_{\mathcal{R}}^* v\sigma$ holds for some normal substitution σ , up to now we checked whether $\text{CAP}(t)$ 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, \dots, t_n). Now it suffices to check whether $t_k\sigma$ reduces innermost to $v\sigma$ for one $k \in \{1, \dots, n\}$.

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

$$\begin{aligned} \text{Q}(\dots z + s(0) \dots) &\rightsquigarrow_{\mathcal{R}} \text{Q}(\dots s(0) \dots) \text{ by the first } + \text{ rule, and} \\ \text{Q}(\dots z + s(0) \dots) &\rightsquigarrow_{\mathcal{R}} \text{Q}(\dots s(x + s(0)) \dots) \text{ by the second } + \text{ rule.} \end{aligned}$$

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

In our example, now it suffices to check whether a normal substitution σ exists such that $\text{Q}(\dots s(0) \dots)\sigma$ or $\text{Q}(\dots s(x + s(0)) \dots)\sigma$ reduces innermost to a normal form $\text{Q}(\dots 0 \dots)\sigma$. For that purpose we can use the technique presented in Thm. 9. This technique immediately proves that such a substitution cannot exist because neither $s(0)$ nor $\text{CAP}(s(x + s(0)))$ 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 $\text{Q}(\dots s(x + s(0)) \dots)$ by those terms it narrows to. In general, to determine whether $t\sigma \xrightarrow{i}_{\mathcal{R}}^* v\sigma$ 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 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 $t\sigma \xrightarrow{i}^*_{\mathcal{R}} v\sigma$ 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 $t \rightsquigarrow_{\mathcal{R}} q$, $q\tau \xrightarrow{i}^*_{\mathcal{R}} v\tau$ and $v\tau$ is a normal form.*

Proof. The proof is done by induction on the length of the reduction $t\sigma \xrightarrow{i}^*_{\mathcal{R}} v\sigma$. If the length is zero, then t and v unify. Otherwise we have $t\sigma \xrightarrow{i}_{\mathcal{R}} t' \xrightarrow{i}^*_{\mathcal{R}} v\sigma$ for some term t' . As σ is a normal substitution, the reduction $t\sigma \xrightarrow{i}_{\mathcal{R}} t'$ cannot take place “in σ ”. Hence, t contains some subterm $f(\mathbf{u})$ such that a rule $l \rightarrow r$ 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: $t\sigma = t\sigma[f(\mathbf{u})\sigma]_p = t\sigma[l\rho]_p \xrightarrow{i}_{\mathcal{R}} t\sigma[r\rho]_p = t'$. Similar to Def. 10 we assume that the variables of $l \rightarrow r$ 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 $t \rightsquigarrow_{\mathcal{R}} q$ holds by the definition of narrowing. Moreover we have $q\tau = t\mu\tau[r\mu\tau]_p = t\sigma\rho[r\sigma\rho]_p = t\sigma[r\rho]_p = t' \xrightarrow{i}^*_{\mathcal{R}} v\sigma = v\tau$. \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 [AG96b]. 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 [AG96a]. 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 [AG96a].

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 [AG97] 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 [AG97] 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 the full version of this paper [AG96b].

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. In *Proceedings of RTA-96*, LNCS 1103, pages 63–77, 1996.
- [AG96b] T. Arts and J. Giesl. Proving innermost normalisation automatically. Tech. Report IBN 96/39, TH Darmstadt, 1996. <http://kirmes.inferenzsysteme.informatik.th-darmstadt.de/~reports/notes/ibn-96-39.ps>
- [AG97] T. Arts and J. Giesl. Automatically proving termination where simplification orderings fail. In *Proceedings of CAAP'97*, LNCS, 1997.
- [Art96] T. Arts. Termination by absence of infinite chains of dependency pairs. In *Proceedings of CAAP'96*, LNCS 1059, pages 196–210, 1996.
- [AZ95] T. Arts and H. Zantema. Termination of logic programs using semantic unification. In *Proceedings of LoPSTr'95*, LNCS 1048, pages 219–233, 1995.
- [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.
- [Der79] N. Dershowitz. A note on simplification orderings. *Information Processing Letters*, 9(5):212–215, 1979.
- [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.
- [Gie95] J. Giesl. Generating polynomial orderings for termination proofs. In *Proceedings of RTA-95*, LNCS 914, pages 426–431, 1995.
- [Gra95] B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundam. Informaticae*, 24:3–23, 1995.
- [Gra96] B. Gramlich. On proving termination by innermost termination. In *Proceedings of RTA-96*, LNCS 1103, pages 93–107, 1996.
- [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 *Proceedings of CADE-5*, LNCS 87, pages 318–334, 1980.
- [Kri95] M.R.K. Krishna Rao. Modular proofs for completeness of hierarchical term rewriting systems. *Theoretical Computer Science*, 151:487–512, 1995.
- [Lan79] D.S. Lankford. On proving term rewriting systems are Noetherian. Technical Report Memo MTP-3, Louisiana Technical University, Ruston, LA, 1979.
- [LM78] D.S. Lankford and D.R. Musser. A finite termination criterion, 1978.
- [MOZ96] A. Middeldorp, H. Ohsaki, and H. Zantema. Transforming termination by self-labelling. In *Proceedings of CADE-13*, LNCS 1104, pages 373–387, 1996.
- [Ste94] J. Steinbach. Generating polynomial orderings. *Inf. Pr. Let.*, 49:85–93, 1994.
- [Ste95a] J. Steinbach. Automatic termination proofs with transformation orderings. In *Proceedings of RTA-95*, LNCS 914, pages 11–25, 1995.
- [Ste95b] J. Steinbach. Simplification orderings: history of results. *Fundamenta Informaticae*, 24:47–87, 1995.
- [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.