Termination of Constructor Systems*

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 method to prove termination of constructor systems automatically. Our approach takes advantage of the special form of these rewrite systems because for constructor systems instead of left- and right-hand sides of rules it is sufficient to compare so-called *dependency pairs* [Art96]. Unfortunately, standard techniques for the generation of well-founded orderings cannot be directly used for the automation of the dependency pair approach. To solve this problem we have developed a transformation technique which enables the application of known synthesis methods for well-founded orderings to prove that dependency pairs are decreasing. In this way termination of many (also non-simply terminating) constructor systems can be proved fully automatically.

1 Introduction

One of the most interesting properties of a term rewriting system is termination, cf. e.g. [DJ90]. While in general this problem is undecidable [HL78], several methods for proving termination have been developed (e.g. path orderings [Pla78, Der82, Ges94, DH95, Ste95b], Knuth-Bendix orderings [KB70, Mar87], semantic interpretations [MN70, Lan79, BCL87, BL93, Ste94, Zan94, Gie95b], transformation orderings [BD86, BL90, Ste95a], semantic labelling [Zan95] etc. — for surveys see e.g. [Der87, Ste95b]).

In this paper we are concerned with the *automation* of termination proofs for *constructor systems* (CS for short). Due to the special form of these rewrite systems it is possible to use a different approach for CSs than is necessary for termination of general rewrite systems. Therefore, in this paper we focus on a technique specially tailored for CSs, viz. the so-called *dependency pair* approach [Art96]. With this approach it is also possible to prove termination of systems where all simplification orderings fail. In Sect. 2 we describe which steps have to be performed (automatically) to verify termination of CSs using this approach. Although the dependency pair approach may be used for arbitrary CSs, in this paper we focus on special hierarchical combinations of CSs ensuring that all steps can be performed automatically.

* Technical Report IBN 96/34, Technische Hochschule Darmstadt. This is an extended version of a paper [AG96] which appeared in the *Proceedings of the 7th International Conference on Rewriting Techniques and Applications*, New Brunswick, NJ, USA, LNCS 1103, Springer-Verlag, 1996.

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

The main task in this approach is to prove that all dependency pairs are decreasing w.r.t. a well-founded ordering. Up to now only some heuristics existed to perform this step automatically. On the other hand, several techniques have been developed to synthesize suitable well-founded orderings for termination proofs of term rewriting systems. Hence, one would like to apply these techniques for the automation of the dependency pair approach. Unfortunately, as we will show in Sect. 3, this is not directly possible.

Therefore in Sect. 4 we suggest a new technique to enable the application of standard methods for the generation of well-founded orderings to prove that dependency pairs are decreasing. For that purpose we transfer a variant of the *estimation* method [Wal94, Gie95c, Gie95d], which was originally developed for termination proofs of functional programs, to rewrite systems.

By the combination of the dependency pair approach and the estimation method we obtain a very powerful technique for automated termination proofs of CSs which can prove termination of numerous CSs whose termination could not be proved automatically before, cf. the appendix.

2 Dependency Pairs

A *constructor system* $(\mathcal{D}, \mathcal{C}, \mathcal{R})$ is a term rewriting system with a set of rules $\mathcal{R}$ and with a signature that can be partitioned into two disjoint sets $\mathcal{D}$ and $\mathcal{C}$ such that for every left-hand side $f(t_1, \ldots, t_n)$ of a rewrite rule of $\mathcal{R}$ the root symbol f is from $\mathcal{D}$ and the terms $t_1, \ldots, t_n$ only contain function symbols from $\mathcal{C}$. Function symbols from $\mathcal{D}$ are called *defined symbols* and function symbols from $\mathcal{C}$ are called *constructors*. As an example consider the following CS:

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

Most methods for automated termination proofs of term rewriting systems are restricted to *simplification orderings* [Der79, Ste95b]. These methods cannot prove termination of the above CS, because no simplification ordering can orient the fourth rule if y is instantiated to $\mathsf{succ}(x)$. The reason is that simplification orderings $\succ$ are monotonic and satisfy the subterm property and this implies $\mathsf{succ}(\mathsf{quot}(\mathsf{minus}(x, \mathsf{succ}(x)), \mathsf{succ}(\mathsf{succ}(x)))) \succ \mathsf{quot}(\mathsf{succ}(x), \mathsf{succ}(\mathsf{succ}(x)))$. All other known techniques for automated termination proofs of non-simply terminating systems [Zan94, Ste95a, Ken95, FZ95] fail with this example, too.

However, with the *dependency pair* approach an automated termination proof of the above CS is possible. The idea of this approach is to use an interpretation on terms which assigns for every rewrite rule of the CS the same value to the left-hand side as to the right-hand side. Then for termination of the CS it is sufficient if there exists a well-founded ordering such that the interpretations of the arguments of all defined symbols are decreasing in each recursive occurrence.

To represent the interpretation another CS $\mathcal{E}$ is used which is *ground-convergent* (i.e. ground-confluent and terminating) and in which the CS $\mathcal{R}$ is *contained*, i.e. $(l\sigma)\downarrow_{\mathcal{E}} = (r\sigma)\downarrow_{\mathcal{E}}$ holds for all rewrite rules $l \to r$ of $\mathcal{R}$ and all ground substitutions σ (where we always assume that there exist ground terms, i.e. there must be a constant in the signature $\mathcal{D} \cup \mathcal{C}$). Then for any ground term t the interpretation is $t\downarrow_{\mathcal{E}}$.

If a term $f(t_1, \ldots, t_n)$ rewrites to another term $C[g(s_1, \ldots, s_m)]$ (where f and g are defined symbols and C denotes some context), then we will try to show that the interpretation of the tuple $t_1, \ldots, t_n$ is greater than the interpretation of the tuple $s_1, \ldots, s_m$. In order to avoid the comparison of *tuples* we extend our signature by a tuple function symbol F for each $f \in \mathcal{D}$ and compare the *terms* $F(t_1, \ldots, t_n)$ and $G(s_1, \ldots, s_m)$ instead. To ease readability we assume that $\mathcal{D} \cup \mathcal{C}$ consists of lower case function symbols only and denote the tuple functions by the corresponding upper case symbols. Pairs of terms that have to be compared are called *dependency pairs*.

Definition 1. Let $(\mathcal{D}, \mathcal{C}, \mathcal{R})$ be a CS. If $f(t_1, \ldots, t_n) \to C[g(s_1, \ldots, s_m)]$ is a rewrite rule of $\mathcal{R}$ and $f, g \in \mathcal{D}$, then $\langle F(t_1, \ldots, t_n), G(s_1, \ldots, s_m)\rangle$ is called a *dependency pair* (of $\mathcal{R}$).

In our example we obtain the following set of dependency pairs (where M and Q denote the tuple function symbols for minus and quot):

$$\langle \mathsf{M}(\mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{M}(x, y)\rangle, \tag{1}$$

$$\langle \mathsf{Q}(\mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{M}(x, y)\rangle, \tag{2}$$

$$\langle \mathsf{Q}(\mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{succ}(y))\rangle\ . \tag{3}$$

The following theorem states that if the interpretations of the dependency pairs are decreasing, then the CS is terminating.

Theorem 2. *Let $(\mathcal{D}, \mathcal{C}, \mathcal{R})$ be a CS and let $(\mathcal{D}, \mathcal{C}, \mathcal{E})$ be a ground-convergent CS such that $\mathcal{R}$ is contained in $\mathcal{E}$. If there exists a well-founded ordering $\succ$ on ground terms such that $(s\sigma)\downarrow_{\mathcal{E}} \succ (t\sigma)\downarrow_{\mathcal{E}}$ holds for all[1] dependency pairs $\langle s, t\rangle$ and all ground substitutions σ, then $\mathcal{R}$ is terminating.*

The proofs of all theorems of this section are based on semantic labelling [Zan95] and can be found in [Art96].

Hence, to prove termination of a CS $\mathcal{R}$ with the dependency pair technique two tasks have to be performed: first, one has to find a ground-convergent CS $\mathcal{E}$ such that $\mathcal{R}$ is contained in $\mathcal{E}$ and second, one has to prove that the $\mathcal{E}$-interpretations of the dependency pairs are decreasing w.r.t. a well-founded ordering.

For the first task, in [Art96] a method is presented to generate suitable CSs $\mathcal{E}$ for a subclass of CSs $\mathcal{R}$ automatically. This subclass consists of non-overlapping[2] hierarchical combinations [KR95] (a CS is a hierarchical combination of two CSs

[1] In many examples it is sufficient if only certain dependency pairs are decreasing and several methods to determine those dependency pairs have been suggested in [Art96].

[2] This requirement can even be weakened to overlay systems with joinable critical pairs.

if defined symbols of the first CS occur as constructors in the second CS, but not vice versa) without nested defined symbols in the second CS (i.e. the rules do not contain subterms of the form $f(\ldots g\ldots)$, where f, g are defined symbols of $\mathcal{R}_1$). We remark that the hierarchical combinations that we focus on, differ from the *proper-extensions* defined by Krishna Rao [KR95].

If $\mathcal{R}$ is such a hierarchical combination of $\mathcal{R}_0$ with $\mathcal{R}_1$ and $\mathcal{R}_0$ is terminating, then it suffices if just the *subsystem* $\mathcal{R}_0$ is contained in $\mathcal{E}$ and hence, one can simply define $\mathcal{E}$ to be $\mathcal{R}_0$. Moreover, one does not have to consider all dependency pairs of $\mathcal{R}$, but it is sufficient 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 termination of hierarchical combinations by successively proving termination of each subsystem and by defining $\mathcal{E}$ to consist of those subsystems whose termination has already been proved before. Thus, we recursively apply the following theorem.

Theorem 3. *Let $(\mathcal{D}, \mathcal{C}, \mathcal{R})$ be a non-overlapping hierarchical combination of $(\mathcal{D}_0, \mathcal{C}, \mathcal{R}_0)$ with $(\mathcal{D}_1, \mathcal{C} \cup \mathcal{D}_0, \mathcal{R}_1)$ such that $\mathcal{R}_0$ is terminating and such that symbols from $\mathcal{D}_1$ do not occur nested in the rules. If there exists a well-founded ordering $\succ$ on ground terms such that $(s\sigma)\downarrow_{\mathcal{R}_0} \succ (t\sigma)\downarrow_{\mathcal{R}_0}$ holds for all dependency pairs $\langle s, t\rangle$ of $\mathcal{R}_1$ and all ground substitutions σ, then $\mathcal{R}$ is terminating.*

For instance, our example is a hierarchical combination of the minus-subsystem with the quot-subsystem. Hence, if we already proved termination of the first two minus-rules[3], then we now only have to prove termination of the quot-rules and let $\mathcal{E}$ consist of the two minus-rules. Now the only dependency pair we have to consider is (3).

Hence, the main problem with automated termination proofs using dependency pairs is the second task, i.e. to find a well-founded ordering such that the interpretations of dependency pairs are decreasing.

3 Using Well-Founded Orderings

Numerous methods for the automated generation of suitable well-founded orderings have been developed to prove termination of term rewriting systems. Hence, for the automation of the dependency pair approach we would like to use these standard methods to prove that dependency pairs are decreasing.

However, we will illustrate in Sect. 3.1 that, unfortunately, the direct application of standard methods for this purpose is unsound. The reason is that arbitrary orderings do not respect the equalities induced by $\mathcal{E}$.

[3] This can for instance be done with standard techniques like e.g. the recursive path ordering [Der82] or again by the dependency pair approach. Then, $\mathcal{E}$ can be chosen to be any ground-convergent CS (even the empty one), because in the CS consisting of the two minus-rules defined symbols do not occur nested and this CS may be regarded as a hierarchical combination where $\mathcal{R}_0$ is empty.

In Sect. 3.2 we show that the straightforward solution of restricting ourselves to orderings that respect the equalities induced by $\mathcal{E}$ results in a method which is not powerful enough.

But in Sect. 3.3 we prove that as long as the dependency pairs do not contain *defined* symbols, the direct approach of Sect. 3.1 is sound. Therefore our aim will be to eliminate all defined symbols in the dependency pairs. A transformation procedure for the elimination of defined symbols will be presented in Sect. 4.

3.1 Direct Application of Well-Founded Orderings

Let $\mathcal{DP}$ be a set of inequalities which represent the constraints that left-hand sides of dependency pairs have to be greater than right-hand sides, i.e. $\mathcal{DP} = \{s \succ t | \langle s, t\rangle \text{ dependency pair}\}$. Now one could use standard methods to generate a well-founded ordering $\succ$ satisfying the constraints $\mathcal{DP}$. But unfortunately, this approach is *unsound*, i.e. it is not sufficient for the termination of the CS $\mathcal{R}$ under consideration. As an example let $\mathcal{R}$ be the CS

$$\begin{aligned} \mathsf{double}(0) &\to 0, \\ \mathsf{double}(\mathsf{succ}(x)) &\to \mathsf{succ}(\mathsf{succ}(\mathsf{double}(x))), \\ \mathsf{f}(\mathsf{succ}(x)) &\to \mathsf{f}(\mathsf{double}(x)). \end{aligned}$$

Assume that we have already proved termination of the double-subsystem. Hence by Thm. 3, we can define $\mathcal{E}$ to consist of the first two rules of $\mathcal{R}$ and we only have to examine the dependency pair $\langle \mathsf{F}(\mathsf{succ}(x)), \mathsf{F}(\mathsf{double}(x))\rangle$. The constraint

$$\mathcal{DP} = \{\mathsf{F}(\mathsf{succ}(x)) \succ \mathsf{F}(\mathsf{double}(x))\}$$

is for instance satisfied by the recursive path ordering $\succ_{rpo}$, cf. [Der82]. Nevertheless, $\mathcal{R}$ is not terminating (e.g. $\mathsf{f}(\mathsf{succ}(\mathsf{succ}(0)))$ starts an infinite reduction).

This direct application of orderings is not possible because the constraints in $\mathcal{DP}$ only compare the terms s and t but not their $\mathcal{E}$-interpretations. However, $s \succ_{rpo} t$ is not sufficient for $(s\sigma)\downarrow_{\mathcal{E}} \succ_{rpo} (t\sigma)\downarrow_{\mathcal{E}}$, because $\succ_{rpo}$ does not respect the equalities induced by $\mathcal{E}$. For instance, $\mathsf{F}(\mathsf{succ}(\mathsf{succ}(0))) \succ_{rpo} \mathsf{F}(\mathsf{double}(\mathsf{succ}(0)))$, but $\mathsf{F}(\mathsf{succ}(\mathsf{succ}(0)))\downarrow_{\mathcal{E}} \not\succ_{rpo} \mathsf{F}(\mathsf{double}(\mathsf{succ}(0)))\downarrow_{\mathcal{E}} = \mathsf{F}(\mathsf{succ}(\mathsf{succ}(0)))$.

So we have to ensure that whenever $s\downarrow_{\mathcal{E}} = t\downarrow_{\mathcal{E}}$ holds for two ground terms s and t, these terms must also be "equivalent" w.r.t. the used ordering. To formalize the notion of "equivalence" we will now regard *quasi*-orderings.

3.2 Quasi-Orderings Respecting $\mathcal{E}$

A *quasi-ordering* $\succsim$ is a reflexive and transitive relation. For every quasi-ordering $\succsim$, let $\sim$ denote the associated equivalence relation (i.e. $s \sim t$ iff $s \succsim t$ and $t \succsim s$) and let $\succ$ denote the strict part of the quasi-ordering (i.e. $s \succ t$ iff $s \succsim t$, but not $t \succsim s$). We say $\succsim$ is well-founded iff the strict part $\succ$ is well-founded. In this paper we restrict ourselves to relations on ground terms and (for notational convenience) we extend every quasi-ordering $\succsim$ to arbitrary terms by defining

$s \gtrsim t$ iff $s\sigma \gtrsim t\sigma$ holds for all ground substitutions σ. Analogously, $s \succ t$ (resp. $s \sim t$) is defined as $s\sigma \succ t\sigma$ (resp. $s\sigma \sim t\sigma$) for all ground substitutions σ.

A straightforward solution for the problem discussed in the preceding section would be to try to find a well-founded quasi-ordering which satisfies both $\mathcal{DP}$ and $\mathcal{EQ}$, where $\mathcal{EQ} = \{s \sim t |\ s,t \text{ ground terms with } s\downarrow_{\mathcal{E}} = t\downarrow_{\mathcal{E}}\}$. Obviously the existence of such a quasi-ordering is sufficient for the termination of the CS $\mathcal{R}$.

Lemma 4. *If there exists a well-founded quasi-ordering satisfying the constraints $\mathcal{DP} \cup \mathcal{EQ}$, then $\mathcal{R}$ is terminating.*

Proof. If $\gtrsim$ satisfies $\mathcal{DP}$, then we have $s\sigma \succ t\sigma$ for each dependency pair $\langle s,t\rangle$ and each ground substitution σ. If $\gtrsim$ also satisfies $\mathcal{EQ}$, then $(s\sigma)\downarrow_{\mathcal{E}} \sim s\sigma \succ t\sigma \sim (t\sigma)\downarrow_{\mathcal{E}}$. Hence, the lemma follows from Thm. 2 (resp. Thm. 3). □

But unfortunately, standard techniques usually cannot be used to find a well-founded quasi-ordering $\gtrsim$ satisfying the constraints $\mathcal{DP} \cup \mathcal{EQ}$. As an example regard the CS for minus and quot again. Assume that we have already proved termination of the minus-subsystem and let us now prove termination of the quot-rules. According to Thm. 3, we can define $\mathcal{E}$ to consist of the two minus-rules and we obtain the constraint

$$\mathcal{DP} = \{\mathsf{Q}(\mathsf{succ}(x),\mathsf{succ}(y)) \succ \mathsf{Q}(\mathsf{minus}(x,y),\mathsf{succ}(y))\}. \tag{4}$$

None of the well-founded quasi-orderings that can be generated automatically by the usual techniques satisfies $\mathcal{DP} \cup \mathcal{EQ}$: Virtually all of those quasi-orderings are quasi-*simplification*-orderings[4] [Der82]. Hence, if $\gtrsim$ is a quasi-simplification-ordering satisfying $\mathcal{EQ}$, then we have

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

(as $\mathsf{minus}(x,y) \sim \mathsf{minus}(\mathsf{succ}(x),\mathsf{succ}(y))$ holds and as quasi-simplification-orderings are (weakly) monotonic). Moreover, we have

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

(as quasi-simplification-orderings satisfy the (weak) subterm property). Hence, $\mathsf{Q}(\mathsf{minus}(x,y),\mathsf{succ}(y)) \gtrsim \mathsf{Q}(\mathsf{succ}(x),\mathsf{succ}(y))$ which is a contradiction to (4).

So the standard techniques for the automated generation of well-founded quasi-orderings fail here (and the same problem appears with most other examples). Hence, demanding $\mathcal{DP} \cup \mathcal{EQ}$ is *too strong*, i.e. in this way most termination proofs will not succeed.

[4] $\mathcal{DP} \cup \mathcal{EQ}$ is not satisfied by polynomial orderings [Lan79] either (which do not have to be quasi-*simplification*-orderings).

3.3 Constraints Without Defined Symbols

In Sect. 3.1 we showed that the existence of a well-founded quasi-ordering $\succsim$ satisfying $\mathcal{DP}$ is in general not sufficient for the termination of $\mathcal{R}$, because $\succsim$ does not necessarily respect the equalities induced by $\mathcal{E}$ (i.e. the equalities $\mathcal{EQ}$).

Nevertheless, if $\mathcal{DP}$ contains no defined symbols (from $\mathcal{D}$) then it is sufficient to find a well-founded quasi-ordering satisfying $\mathcal{DP}$. The reason is that any such quasi-ordering can be transformed into a well-founded quasi-ordering satisfying both $\mathcal{DP}$ and $\mathcal{EQ}$:

Lemma 5. *Let $(\mathcal{D}, \mathcal{C}, \mathcal{E})$ be a ground-convergent CS, let $\mathcal{DP}$ be a set of inequalities containing no defined symbols. If there exists a well-founded quasi-ordering $\succsim$ satisfying $\mathcal{DP}$, then there also exists a well-founded quasi-ordering $\succsim'$ satisfying both $\mathcal{DP}$ and $\mathcal{EQ}$.*

Proof. For two ground terms s, t let $s \succsim' t$ iff $s\downarrow_{\mathcal{E}} \succsim t\downarrow_{\mathcal{E}}$. Since $\succsim$ is a well-founded quasi-ordering, $\succsim'$ is a well-founded quasi-ordering and obviously, $\succsim'$ satisfies $\mathcal{EQ}$.

We will now show that $\succsim'$ satisfies $\mathcal{DP}$: Let s and t be terms without defined symbols. As $\succsim$ satisfies $\mathcal{DP}$, it is sufficient to prove that $s \succsim t$ implies $s \succsim' t$. Note that for terms without defined symbols we have $(s\sigma)\downarrow_{\mathcal{E}} = s(\sigma\downarrow_{\mathcal{E}})$ for each ground substitution σ (where $\sigma\downarrow_{\mathcal{E}}$ denotes the substitution of x by $(\sigma(x))\downarrow_{\mathcal{E}}$ for each $x \in DOM(\sigma)$). Now $s \succsim t$ implies $s(\sigma\downarrow_{\mathcal{E}}) \succsim t(\sigma\downarrow_{\mathcal{E}})$ for all ground substitutions σ or, respectively, $(s\sigma)\downarrow_{\mathcal{E}} \succsim (t\sigma)\downarrow_{\mathcal{E}}$. Hence, $s\sigma \succsim' t\sigma$ holds for all σ and therefore $s \succsim t$ implies $s \succsim' t$. Similarly it can be proved that $s \succ t$ implies $s \succ' t$. □

As an example consider the CS which only consists of the two rules for minus. Here, $\mathcal{DP}$ contains only the inequality $\mathsf{M}(\mathsf{succ}(x), \mathsf{succ}(y)) \succ \mathsf{M}(x, y)$ in which no defined symbol occurs. Of course there exist well-founded quasi-orderings satisfying this constraint (e.g. $\succsim_{rpo}$). For any ground-convergent $\mathcal{E}$ (cf. Footnote 3), $\succsim_{rpo}$ can be transformed into a well-founded quasi-ordering $\succsim'$ (as in the proof of Lemma 5) where $s \succsim' t$ holds iff $s\downarrow_{\mathcal{E}} \succsim_{rpo} t\downarrow_{\mathcal{E}}$. This quasi-ordering satisfies both $\mathcal{DP}$ and $\mathcal{EQ}$. Hence, termination of this CS is proved.

So if $\mathcal{DP}$ does not contain defined symbols we can just use standard techniques to generate a well-founded quasi-ordering satisfying $\mathcal{DP}$. By the two Lemmata 4 and 5 this is sufficient for the termination of $\mathcal{R}$.

To conclude, we have shown that the direct use of well-founded quasi-orderings is unsound (except if $\mathcal{DP}$ does not contain defined symbols) and we have illustrated that the straightforward solution (i.e. the restriction to quasi-orderings which also satisfy $\mathcal{EQ}$) imposes too strong requirements such that termination proofs often fail. In the next section we present a different, powerful approach to deal with CSs where $\mathcal{DP}$ *does* contain defined symbols. (This always happens if defined symbols occur within the arguments of a recursive call in $\mathcal{R}$.)

4 Elimination of Defined Symbols

If we want to prove termination of the quot-subsystem then we have to show that there exists a well-founded quasi-ordering satisfying both $\mathcal{EQ}$ (where $\mathcal{E}$ consists

of the first two minus-rules) and the constraint

$$\mathcal{DP} = \{\mathsf{Q}(\mathsf{succ}(x), \mathsf{succ}(y)) \succ \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{succ}(y))\}. \tag{4}$$

As demonstrated in Sect. 3 the application of methods for the synthesis of well-founded quasi-orderings is only possible if the constraints in $\mathcal{DP}$ do not contain defined symbols (like minus). Therefore our aim is to transform the constraint (4) into new constraints $\mathcal{DP}'$ *without defined symbols*. The invariant of this transformation will be that every quasi-ordering satisfying $\mathcal{EQ}$ and the resulting constraints $\mathcal{DP}'$ also satisfies the original constraints $\mathcal{DP}$. (In fact, this soundness result for our transformation only holds for a certain (slightly restricted) class of quasi-orderings, cf. Sect. 4.2.)

The constraints $\mathcal{DP}'$ resulting from the transformation contain no defined symbols any more. Hence, if we find a well-founded quasi-ordering which satisfies just $\mathcal{DP}'$ (by application of standard methods for the automated generation of such quasi-orderings), then by Lemma 5 there exists a well-founded quasi-ordering satisfying $\mathcal{DP}' \cup \mathcal{EQ}$. Hence, this quasi-ordering also satisfies $\mathcal{DP}$. Thus by Lemma 4, termination is proved. So, existence of a well-founded quasi-ordering satisfying the constraints $\mathcal{DP}'$ suffices for the termination of the CS.

In Sect. 4.1 we introduce the central idea of our transformation, viz. the *estimation technique*. To apply the estimation technique we need so-called *estimation inequalities* and Sect. 4.2 shows how they are computed. This section also contains the soundness theorem for our transformation. For the transformation we have to make a slight restriction on the used quasi-orderings. We present a generalized version of Lemma 5 in Sect. 4.3 which shows how to use methods for the automated generation of well-founded quasi-orderings to synthesize the quasi-orderings we need.

4.1 Estimation

The constraint (4) contains the defined symbol minus. The central idea of our transformation procedure is the *estimation* of defined symbols by new *non-defined* function symbols. For that purpose we extend our signature by a new estimation function $\bar{f}$ for each $f \in \mathcal{D}$. Now minus is replaced by the new non-defined symbol $\overline{\mathsf{minus}}$ and we demand that the result of $\overline{\mathsf{minus}}$ is always greater or equal than the result of minus, i.e. we demand

$$\overline{\mathsf{minus}}(x, y) \succsim \mathsf{minus}(x, y). \tag{5}$$

In contrast to minus the semantics of the non-defined symbol $\overline{\mathsf{minus}}$ are not determined by the equalities in $\mathcal{EQ}$. Our method transforms constraints like (4) into inequalities which contain non-defined symbols like $\overline{\mathsf{minus}}$, but no defined symbols like minus. If these resulting inequalities are satisfied by a well-founded quasi-ordering, then termination of the CS is proved.

Assume for the moment that we know a set of so-called *estimation inequalities* $\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}}$ (without defined symbols) such that every quasi-ordering satisfying

$\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}}$ and $\mathcal{EQ}$ also satisfies (5). Moreover, let us restrict ourselves to quasi-orderings that are weakly monotonic on non-defined symbols (i.e. $s \succsim t$ implies $f(\ldots s \ldots) \succsim f(\ldots t \ldots)$ for all $f \notin \mathcal{D}$). Then $\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}}$ and $\mathcal{EQ}$ do not only imply $\overline{\mathsf{minus}}(x, y) \succsim \mathsf{minus}(x, y)$, but they also ensure

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

Now

$$\mathsf{Q}(\mathsf{succ}(x), \mathsf{succ}(y)) \succ \mathsf{Q}(\overline{\mathsf{minus}}(x, y), \mathsf{succ}(y)) \tag{6}$$

and $\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}}$ are sufficient for the original constraint (4), i.e. every quasi-ordering which satisfies (6), $\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}}$ and $\mathcal{EQ}$ (and is weakly monotonic on non-defined symbols) also satisfies (4).

The restriction to quasi-orderings that are weakly monotonic on non-defined symbols allows to estimate function symbols *within* a term (i.e. function symbols that are not the root symbol of the term). If such a quasi-ordering satisfies $\mathcal{IN}_{\bar{f} \succsim f}$, then it also satisfies $C[\bar{f}(\ldots)] \succsim C[f(\ldots)]$ for all contexts C with no defined symbols above f.

In this way every inequality can be transformed into inequalities without defined symbols: we replace every defined symbol f by the new non-defined symbol $\bar{f}$ and add the estimation inequalities $\mathcal{IN}_{\bar{f} \succsim f}$ to the constraints.

Definition 6. For every term t we define its *estimation* by

$$\mathrm{est}(f(t_1, \ldots, t_n)) = \begin{cases} \bar{f}(\mathrm{est}(t_1), \ldots, \mathrm{est}(t_n)) & \text{if } f \in \mathcal{D} \\ f(\mathrm{est}(t_1), \ldots, \mathrm{est}(t_n)) & \text{if } f \notin \mathcal{D}. \end{cases}$$

Let $\mathcal{DP}$ be a set of inequalities. Then we define

$$\mathcal{DP}' = \{s \succ \mathrm{est}(t) | s \succ t \in \mathcal{DP}\} \cup \bigcup_{f \in \mathcal{D} \text{ occurs in } \mathcal{DP}} \mathcal{IN}_{\bar{f} \succsim f}.$$

In our example, minus is estimated by $\overline{\mathsf{minus}}$ and hence, the resulting set of constraints $\mathcal{DP}'$ consists of (6) and $\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}}$.

4.2 Estimation Inequalities

In this section we show how to compute *estimation inequalities* $\mathcal{IN}_{\bar{f} \succsim f}$ which are needed for the estimation technique of Sect. 4.1 and we prove the soundness of our transformation. The estimation inequalities $\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}}$ have to guarantee that $\overline{\mathsf{minus}}$ really is an upper bound for minus. To compute $\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}}$ we consider each minus-rule of $\mathcal{E}$ separately. Instead of $\overline{\mathsf{minus}}(x, y) \succsim \mathsf{minus}(x, y)$ we therefore demand

$$\overline{\mathsf{minus}}(x, 0) \succsim x, \tag{7}$$

$$\overline{\mathsf{minus}}(\mathsf{succ}(x), \mathsf{succ}(y)) \succsim \mathsf{minus}(x, y). \tag{8}$$

We cannot define $\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}} = \{(7), (8)\}$ because inequality (8) still contains the defined symbol minus. Defined symbols occurring in such formulas have to be eliminated by *estimation* again.

But the problem here is that minus *itself* appears in inequality (8). We cannot use the transformation of Definition 6 for the estimation of minus, because we do not know the estimation inequalities $\mathcal{IN}_{\overline{\mathsf{minus}} \gtrsim \mathsf{minus}}$ yet.

We solve this problem by constructing $\mathcal{IN}_{\overline{\mathsf{minus}} \gtrsim \mathsf{minus}}$ *inductively* with respect to the *computation ordering* of $\mathcal{E}$. The *computation ordering* $>_{\mathcal{E}}$ of a rewrite system $\mathcal{E}$ is a relation on ground terms where $s >_{\mathcal{E}} t$ iff $s \to_{\mathcal{E}}^{+} C[t]$ holds for some (possibly empty) context C. Obviously (as $\mathcal{E}$ is ground-convergent) its computation ordering is well-founded, i.e. inductions w.r.t. such orderings are sound.

The first case of our inductive construction of $\mathcal{IN}_{\overline{\mathsf{minus}} \gtrsim \mathsf{minus}}$ corresponds to the non-recursive first minus-rule. Inequality (7) ensures that for pairs of terms of the form $(t, 0)$, $\overline{\mathsf{minus}}$ is an upper bound for minus.

For the second minus-rule we have to ensure that inequality (8) holds, i.e. for terms of the form $(\mathsf{succ}(t_1), \mathsf{succ}(t_2))$, the result of $\overline{\mathsf{minus}}$ must be greater or equal than the result of minus. As *induction hypothesis* we can now use that this estimation is already correct for (t_1, t_2), because $\mathsf{minus}(\mathsf{succ}(t_1), \mathsf{succ}(t_2)) >_{\mathcal{E}} \mathsf{minus}(t_1, t_2)$. Hence when regarding $\overline{\mathsf{minus}}(\mathsf{succ}(x), \mathsf{succ}(y))$, we can use the induction hypothesis $\overline{\mathsf{minus}}(x, y) \gtrsim \mathsf{minus}(x, y)$. Then it is sufficient for (8) if

$$\overline{\mathsf{minus}}(\mathsf{succ}(x), \mathsf{succ}(y)) \gtrsim \overline{\mathsf{minus}}(x, y) \tag{9}$$

is true. Therefore we can replace (8) by inequality (9) which does not contain defined symbols.

Note that to eliminate the defined symbol minus from (8) due to an inductive argument we could again use the estimation technique. Now we have finished our inductive construction of $\mathcal{IN}_{\overline{\mathsf{minus}} \gtrsim \mathsf{minus}}$ and obtain

$$\begin{aligned} \mathcal{IN}_{\overline{\mathsf{minus}} \gtrsim \mathsf{minus}} = \{&\overline{\mathsf{minus}}(x, 0) \gtrsim x, && (7)\\ &\overline{\mathsf{minus}}(\mathsf{succ}(x), \mathsf{succ}(y)) \gtrsim \overline{\mathsf{minus}}(x, y)\}. && (9) \end{aligned}$$

Definition 7. Let $(\mathcal{D}, \mathcal{C}, \mathcal{E})$ be a ground-convergent CS. For each $f \in \mathcal{D}$ we define the set of *estimation inequalities* $\mathcal{IN}_{\bar{f} \gtrsim f}$ as follows (here, s^* abbreviates a tuple of terms $s_1, \ldots, s_n$):

$$\mathcal{IN}_{\bar{f} \gtrsim f} = \{\bar{f}(s^*) \gtrsim \mathrm{est}(t) \mid s^*, t \text{ are terms, } f(s^*) \to t \in \mathcal{E}\} \cup \bigcup_{\substack{g \in \mathcal{D} \text{ occurs in the} \\ f\text{-rules of } \mathcal{E} \text{ and } g \neq f}} \mathcal{IN}_{\bar{g} \gtrsim g}.$$

But $\mathcal{IN}_{\overline{\mathsf{minus}} \gtrsim \mathsf{minus}}$ is not yet sufficient for $\overline{\mathsf{minus}}(x, y) \gtrsim \mathsf{minus}(x, y)$. The reason is that for the construction of $\mathcal{IN}_{\overline{\mathsf{minus}} \gtrsim \mathsf{minus}}$ we only considered $\overline{\mathsf{minus}}(s_1, s_2)$ for terms s_1, s_2 of the form $(t, 0)$ or $(\mathsf{succ}(t_1), \mathsf{succ}(t_2))$ (i.e. we only considered terms where $\mathsf{minus}(s_1, s_2)$ is $\mathcal{E}$-reducible[5]). But for instance, $\mathcal{IN}_{\overline{\mathsf{minus}} \gtrsim \mathsf{minus}}$ does not guarantee $\overline{\mathsf{minus}}(0, \mathsf{succ}(0)) \gtrsim \mathsf{minus}(0, \mathsf{succ}(0))$.

[5] While in the original estimation method for functional programs [Gie95d] functions had to be completely defined, here we have to extend the estimation method to incompletely defined functions. This allows to prove termination of CSs that are not sufficiently complete [Pla85], too.

Therefore we additionally have to demand that irreducible ground terms with a defined root symbol are minimal, i.e. we also demand the constraints

$$\mathcal{MIN} = \{t \succsim f(r^*) | f \in \mathcal{D}, t, r^* \text{ are ground terms}, f(r^*) \text{ is } \mathcal{E}\text{-normal form}\}.$$

If $\mathcal{MIN}$ is also satisfied, then irreducible terms like $\mathsf{minus}(0, \mathsf{succ}(0))$ are minimal, and hence $\overline{\mathsf{minus}}(0, \mathsf{succ}(0)) \succsim \mathsf{minus}(0, \mathsf{succ}(0))$ obviously holds. Now we can prove the soundness of our transformation:

Theorem 8. *Let $(\mathcal{D}, \mathcal{C}, \mathcal{E})$ be a ground-convergent CS, let $\mathcal{DP}$ be a set of inequalities. Then every quasi-ordering $\succsim$ which is weakly monotonic on non-defined symbols and which satisfies $\mathcal{DP}' \cup \mathcal{EQ} \cup \mathcal{MIN}$ also satisfies $\mathcal{DP}$.*

Proof.

(a) We first prove that all $\mathcal{IN}_{\bar{f} \succsim f}$ for $f \in \mathcal{D}$ are sound. More precisely, if $\succsim$ satisfies $\mathcal{IN}_{\bar{f} \succsim f}$, then $\bar{f}(r^*) \succsim f(r^*)$ holds for all ground terms r^*. The proof is done by induction w.r.t. the computation ordering $>_{\mathcal{E}}$ of $\mathcal{E}$.
If $f(r^*)$ is irreducible then the statement follows from the fact that $\succsim$ satisfies $\mathcal{MIN}$. Otherwise there must be a rule $f(s^*) \to t$ where $r^* = s^*\sigma$ for some σ. Hence, $\mathcal{IN}_{\bar{f} \succsim f}$ contains $\bar{f}(s^*) \succsim \mathrm{est}(t)$ and the inequalities $\mathcal{IN}_{\bar{g} \succsim g}$ for all $g \in \mathcal{D}$ occurring in t.
Note that $\mathrm{est}(t)$ is obtained from t by successively replacing each subterm $g(u^*)$ of t with a defined root symbol $g \in \mathcal{D}$ by $\bar{g}(u^*)$. As the estimation starts with the outermost defined symbol, only such subterms $g(u^*)$ are estimated which have no defined symbol above them any more. Therefore, if $\bar{g}(u^*) \succsim g(u^*)$ holds for all these subterms, then $\mathrm{est}(t) \succsim t$ must obviously be true. Analogously, the instantiation $\mathrm{est}(t)\sigma$ is obtained from $t\sigma$ by replacing subterms $g(u^*)\sigma$ by $\bar{g}(u^*)\sigma$. Hence, if $\bar{g}(u^*)\sigma \succsim g(u^*)\sigma$ holds for all these subterms, then this implies $\mathrm{est}(t)\sigma \succsim t\sigma$.
All subterms $g(u^*)\sigma$ in $t\sigma$ are $>_{\mathcal{E}}$-smaller than $f(r^*)$. If g is a defined symbol ($g = f$ is possible) then $\mathcal{IN}_{\bar{f} \succsim f}$ must contain $\mathcal{IN}_{\bar{g} \succsim g}$ and by the induction hypothesis $\mathcal{IN}_{\bar{g} \succsim g}$ implies $\bar{g}(u^*)\sigma \succsim g(u^*)\sigma$. Hence, we have $\mathrm{est}(t)\sigma \succsim t\sigma$ and (as $\bar{f}(s^*) \succsim \mathrm{est}(t)$ is in $\mathcal{IN}_{\bar{f} \succsim f}$ and as $\succsim$ is closed under substitutions), $\bar{f}(r^*) \succsim \mathrm{est}(t)\sigma \succsim t\sigma$. As $t\sigma \sim f(r^*) \in \mathcal{EQ}$, this implies $\bar{f}(r^*) \succsim f(r^*)$.

(b) Now we can show that $\succsim$ satisfies $\mathcal{DP}$. Let $\mathcal{IN}_{\bar{f} \succsim f}$ hold for all defined symbols f occurring in a term t. Due to (a), this implies $\bar{f}(r^*) \succsim f(r^*)$ for all subterms $f(r^*)$ of t which have a defined root symbol. As illustrated in (a), we therefore can conclude $\mathrm{est}(t) \succsim t$. Hence, $s \succ \mathrm{est}(t)$ implies $s \succ t$. As $\succsim$ satisfies $\mathcal{DP}'$, it must also satisfy $\mathcal{DP}$. □

4.3 Automated Generation of Suitable Quasi-Orderings

Thm. 8 states that if we restrict ourselves to quasi-orderings that are weakly monotonic on non-defined symbols and that satisfy $\mathcal{EQ}$ and $\mathcal{MIN}$, then our transformation is sound, i.e. by application of the estimation technique to $\mathcal{DP}$ we obtain a set of inequalities $\mathcal{DP}'$ without defined symbols, such that every quasi-ordering (as above) satisfying $\mathcal{DP}'$ also satisfies $\mathcal{DP}$.

Recall that the reason for eliminating defined symbols was that we wanted to apply standard techniques to generate well-founded quasi-orderings that satisfy a given set of constraints. If these constraints contain no defined symbols, then by Lemma 5 every such quasi-ordering can be extended to a well-founded quasi-ordering satisfying also the equalities $\mathcal{EQ}$.

To use our transformation procedure we had to restrict ourselves to quasi-orderings which have a certain monotonicity property and which satisfy $\mathcal{MIN}$. Therefore we now have to prove a stronger version of Lemma 5. It must state that if we have a well-founded quasi-ordering of this restricted form which satisfies some constraints $\mathcal{DP}'$ without defined symbols, then we can transform it into one of the same restricted form which additionally satisfies $\mathcal{EQ}$. (Then, by Thm. 8 this quasi-ordering also satisfies $\mathcal{DP}$ and therefore (by Lemma 4) termination of the CS under consideration is proved.)

So with this lemma it would be sufficient to synthesize a well-founded quasi-ordering which is weakly monotonic on non-defined symbols and which satisfies $\mathcal{MIN}$ and $\mathcal{DP}'$. Standard techniques can easily be used to generate suitable quasi-orderings that satisfy the required monotonicity condition, but an automated generation of quasi-orderings satisfying the (infinitely many) constraints in $\mathcal{MIN}$ seems to be hard at first sight.

Here, instead of demanding the constraints $\mathcal{MIN}$ the solution will be to restrict ourselves to quasi-orderings which have a minimal element, i.e. there must be a term m such that $t \gtrsim m$ holds for all ground terms t. Such quasi-orderings can easily be generated automatically (e.g. one could add a constraint of the form $x \gtrsim m$).

We will now prove a variant of Lemma 5 which states that if there is a well-founded quasi-ordering which is weakly monotonic on non-defined symbols, has a minimal element, and satisfies $\mathcal{DP}'$, then there also exists a well-founded quasi-ordering which is weakly monotonic on non-defined symbols and satisfies all $\mathcal{DP}'$, $\mathcal{EQ}$ and $\mathcal{MIN}$. Hence, for termination it is sufficient to find a well-founded quasi-ordering which is weakly monotonic on non-defined symbols, has a minimal element and satisfies $\mathcal{DP}'$. Such quasi-orderings can be generated automatically by standard techniques.

Lemma 9. *Let $(\mathcal{D}, \mathcal{C}, \mathcal{E})$ be a ground-convergent CS, let $\mathcal{DP}'$ be a set of inequalities containing no defined symbols. If there exists a well-founded quasi-ordering $\gtrsim$ which is weakly monotonic on non-defined symbols, has a minimal element, and satisfies $\mathcal{DP}'$, then there also exists a well-founded quasi-ordering $\gtrsim'$ which is weakly monotonic on non-defined symbols and satisfies $\mathcal{DP}' \cup \mathcal{EQ} \cup \mathcal{MIN}$.*

Proof. Let m be the minimal element of $\gtrsim$. For each ground term we define

$$[\![f(t_1,\ldots,t_n)]\!] = \begin{cases} f([\![t_1]\!],\ldots,[\![t_n]\!]) & \text{if } f \notin \mathcal{D} \\ m & \text{if } f \in \mathcal{D}, f(t_1,\ldots,t_n) \text{ is } \mathcal{E}\text{-normal form} \\ [\![f(t_1,\ldots,t_n)\downarrow_{\mathcal{E}}]\!] & \text{otherwise.} \end{cases}$$

For two ground terms s, t let $s \gtrsim' t$ iff $[\![s]\!] \gtrsim [\![t]\!]$. Since $\gtrsim$ is a well-founded quasi-ordering, $\gtrsim'$ is also a well-founded quasi-ordering and obviously, $\gtrsim'$ satisfies $\mathcal{MIN}$ and $\mathcal{EQ}$ (as $[\![t]\!] = [\![t\downarrow_{\mathcal{E}}]\!]$ holds for all ground terms t).

The quasi-ordering $\succsim'$ is weakly monotonic on every non-defined symbol f, because $s \succsim' t$ implies $[\![s\sigma]\!] \succsim [\![t\sigma]\!]$ for all ground substitutions σ, which in turn implies $f([\![\ldots]\!][\![s\sigma]\!][\![\ldots]\!]) \succsim f([\![\ldots]\!][\![t\sigma]\!][\![\ldots]\!])$ as $\succsim$ is weakly monotonic. Note that for $f \notin \mathcal{D}$ we have $f([\![\ldots]\!][\![s\sigma]\!][\![\ldots]\!]) = [\![f(\ldots(s\sigma)\ldots)]\!]$. Hence, $[\![f(\ldots(s\sigma)\ldots)]\!] \succsim [\![f(\ldots(t\sigma)\ldots)]\!]$, resp. $[\![f(\ldots s\ldots)\sigma]\!] \succsim [\![f(\ldots t\ldots)\sigma]\!]$ holds for all ground substitutions σ and therefore $f(\ldots s\ldots) \succsim' f(\ldots t\ldots)$.

That $\succsim'$ also satisfies $\mathcal{DP}'$ can be shown like in the proof of Lemma 5. □

The following final theorem summarizes our approach for termination proofs of constructor systems.

Theorem 10. *If there exists a well-founded quasi-ordering which is weakly monotonic on non-defined symbols, has a minimal element, and satisfies $\mathcal{DP}'$, then $\mathcal{R}$ is terminating.*

Proof. By Lemma 9 every such quasi-ordering can be extended to a well-founded weakly monotonic quasi-ordering which also satisfies $\mathcal{EQ}$ and $\mathcal{MIN}$ and by Thm. 8 this quasi-ordering must also satisfy the original constraints $\mathcal{DP}$. Hence, by Lemma 4 the CS $\mathcal{R}$ is terminating. □

So in our example, it is sufficient to find a well-founded weakly monotonic quasi-ordering which has a minimal element and satisfies the computed constraints (6) and $\mathcal{IN}_{\overline{\mathsf{minus}} \succsim \mathsf{minus}} = \{(7), (9)\}$. For instance, we can use a polynomial ordering [Lan79] where the function symbol 0 is mapped to the number 0, $\mathsf{succ}(x)$ is mapped to $x + 1$ and $\mathsf{Q}(x, y)$ and $\overline{\mathsf{minus}}(x, y)$ are both mapped to the polynomial x. Methods for the automated generation of such polynomial orderings have for instance been developed in [Ste94, Gie95b]. In this way termination of the CS for minus and quot can be proved fully automatically.

5 Conclusion and Further Work

We have developed a method for automated termination proofs of constructor systems which uses an estimation technique to automate the analysis of dependency pairs. Our method works as follows:

- For a CS $\mathcal{R}$ a ground-convergent CS $\mathcal{E}$ is synthesized in which $\mathcal{R}$ is contained. (For CSs that are hierarchical combinations of a certain type, a suitable $\mathcal{E}$ can be immediately obtained automatically, cf. [Art96].)
- Let $\mathcal{DP}$ be the set of inequalities which ensure that all dependency pairs are decreasing. Then by application of the estimation technique $\mathcal{DP}$ is transformed into a new set of inequalities $\mathcal{DP}'$ without defined symbols.
- Standard methods are used to generate a well-founded weakly monotonic quasi-ordering which has a minimal element and satisfies $\mathcal{DP}'$. If there exists such a quasi-ordering, then the CS $\mathcal{R}$ is terminating.

The presented method utilizes the special structure of hierarchical combinations of constructor systems. Therefore in this way termination of many CSs

can be proved automatically where all other known techniques fail. Apart from that, with our approach one can still prove termination of all CSs satisfying the requirements of Thm. 3 that, by any other method, can be oriented by a simplification ordering with a minimal element. Our method has been tested on numerous practically relevant CSs from different areas of computer science (using a system for the automated generation of polynomial orderings [Gie95b]) and proved successful. A collection of examples which demonstrate the power of our method (including arithmetical operations such as gcd and logarithm, several sorting algorithms such as quicksort or selection_sort as well as functions on trees and graphs (e.g. a reachability algorithm)) can be found in the appendix.

Our approach fails if a well-founded quasi-ordering satisfying the generated constraints $\mathcal{DP}'$ cannot be found automatically. Therefore apart from the estimation technique we plan to examine alternative possibilities to derive suitable constraints $\mathcal{DP}'$, which may be advantageous for further sophisticated termination proofs (cf. [BM79, BL93, Wal94, Gie95d]). For that purpose, future work will include an investigation on possible combinations of our method with induction theorem proving systems (e.g. [BM79, BHHW86, KZ89, BHHS90, BKR92]).

Acknowledgements
Thanks are due to Hans Zantema and Thomas Kolbe for the discussions we have had on the subjects described in this paper and for their very helpful criticism.

Appendix

This appendix contains a collection of examples which demonstrate the power of the described method. Several of these examples are not simply terminating. Thus all methods based on simplification orderings fail in proving termination of these (non-simply terminating) constructor systems.

All CSs in this appendix are non-overlapping, hierarchical combinations of constructor systems without nested recursion. Therefore, Thm. 3 can be used to prove termination of the CSs.

Theorem 3. *Let $(\mathcal{D}, \mathcal{C}, \mathcal{R})$ be a non-overlapping hierarchical combination of $(\mathcal{D}_0, \mathcal{C}, \mathcal{R}_0)$ with $(\mathcal{D}_1, \mathcal{C}, \mathcal{R}_1)$ such that $\mathcal{R}_0$ is terminating and such that symbols from $\mathcal{D}_1$ do not occur nested in the rules. If there exists a well-founded ordering $\succ$ on ground terms such that $(s\sigma)\downarrow_{\mathcal{R}_0} \succ (t\sigma)\downarrow_{\mathcal{R}_0}$ holds for all dependency pairs $\langle s, t\rangle$ of $\mathcal{R}_1$ and all ground substitutions σ, then $\mathcal{R}$ is terminating.*

Thus, proving termination of $\mathcal{R}$ is done as follows:

1. prove termination of $\mathcal{R}_0$,
2. prove that there exists a well-founded ordering $\succ$ on ground terms, such that $(s\sigma)\downarrow_{\mathcal{R}_0} \succ (t\sigma)\downarrow_{\mathcal{R}_0}$ for all dependency pairs $\langle s, t\rangle$ of $\mathcal{R}_1$ and all ground substitutions σ.

For proving termination of $\mathcal{R}_0$ we may recursively use Thm. 3, since $\mathcal{R}_0$ is non-overlapping and may again be a hierarchical combination. (If defined symbols

of $\mathcal{R}_0$ do not occur nested, then $\mathcal{R}_0$ can be regarded as a hierarchical combination with the empty CS (no rules).) But also other methods, like the recursive path ordering, may be used to prove termination of $\mathcal{R}_0$.

For proving that there exists a well-founded ordering $\succ$ on ground terms, such that $(s\sigma)\downarrow_{\mathcal{R}_0} \succ (t\sigma)\downarrow_{\mathcal{R}_0}$ for all dependency pairs $\langle s, t\rangle$ of $\mathcal{R}_1$ and all ground substitutions σ, we use the estimation method as described in Sect. 4. The estimation method transforms the dependency pairs of $\mathcal{R}_1$ into a set of inequalities, denoted by $\mathcal{DP}'$, where $\mathcal{R}_0$ is used to construct $\mathcal{DP}'$. This set of inequalities together with Thm. 10 is used to conclude termination of the CS.

Theorem 10. *If there exists a well-founded quasi-ordering which is weakly monotonic on non-defined symbols, has a minimal element, and satisfies $\mathcal{DP}'$, then $\mathcal{R}$ is terminating.*

The set of inequalities $\mathcal{DP}'$ is easily constructed and standard methods are used to find a well-founded quasi-ordering that is weakly monotonic on non-defined symbols, has a minimal element, and satisfies $\mathcal{DP}'$.

An algebra equipped with a well-founded ordering can easily be extended to a well-founded ordering on ground terms by choosing suitable homomorphisms (or interpretations). In all examples, we use the algebra consisting of the natural numbers with the normal ordering on natural numbers. Suitable interpretations of the function symbols lift these orderings to orderings on ground terms. The use, in particular, of *polynomial interpretations* that map terms into the natural numbers was developed by Lankford [Lan79]. These orderings trivially always have a minimal element and the ordering is weakly monotonic as long as the interpreted functions are weakly monotonic. Several techniques exist to derive the interpretations automatically [Gie95b, Ste94].

To easy readability the CSs are presented as two sets of rewrite rules separated by some vertical space. The upper system will always denote $\mathcal{R}_0$, whereas the bottom rules will denote $\mathcal{R}_1$.

For every CS, a set of dependency pairs is given. Note that **not all** dependency pairs are given. Only those dependency pairs that are relevant are listed. For more information about which dependency pairs are relevant and which are not, we refer to [Art96].

1 Division, Version 1

This is the running example of this report. It obviously is not simply terminating.

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

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

The relevant dependency pairs of this CS are

$$\langle \mathsf{M}(\mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{M}(x, y)\rangle$$
$$\langle \mathsf{Q}(\mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{succ}(y))\rangle$$

The CS $\mathcal{R}_0$ (with the minus rules) is terminating, since for the only dependency pair of this CS, viz. $\langle \mathsf{M}(\mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{M}(x, y)\rangle$, we have

$$\mathsf{M}(\mathsf{succ}(x), \mathsf{succ}(y)) \succ \mathsf{M}(x, y)$$

by the embedding ordering. The set of inequalities $\mathcal{DP}'$ is given by

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

$$\overline{\mathsf{minus}}(x, 0) \succsim x$$
$$\overline{\mathsf{minus}}(\mathsf{succ}(x), \mathsf{succ}(y)) \succsim \overline{\mathsf{minus}}(x, y)$$

A suitable quasi-ordering satisfying $\mathcal{DP}'$ is automatically found. The normal ordering on the natural numbers together with the following interpretation of the function symbols satisfies $\mathcal{DP}'$: the function symbol 0 is mapped to the number 0, $\mathsf{succ}(x)$ is mapped to $x + 1$ and $\mathsf{Q}(x, y)$ and $\overline{\mathsf{minus}}(x, y)$ are mapped to x.

2 Division, Version 2

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

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

The relevant dependency pairs of this CS are given by

$$\langle \mathsf{M}(x, \mathsf{succ}(y)), \mathsf{M}(x, y)\rangle$$
$$\langle \mathsf{Q}(\mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{Q}(\mathsf{minus}(x, y), \mathsf{succ}(y))\rangle$$

The CS $\mathcal{R}_0$ is terminating. This can be proved by the recursive path ordering, but also by splitting the system in two CSs and finding a suitable well-founded ordering such that

$$\mathsf{M}(x, \mathsf{succ}(y)) \succ \mathsf{M}(x, y)$$

This can be done automatically.

The set of inequalities $\mathcal{DP}'$ differs from the one in the previous example and is given by

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

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

A suitable quasi-ordering satisfying $\mathcal{DP}'$ is the normal ordering on natural numbers, with an interpretation where the function symbol 0 is mapped to the number 0, $\mathsf{succ}(x)$ is mapped to $x+1$ and $\mathsf{Q}(x,y)$, $\overline{\mathsf{minus}}(x,y)$ and $\overline{\mathsf{pred}}(x)$ are all mapped to x.

3 Division, Version 3

This CS 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.

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

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

The relevant dependency pairs of this CS are given by

$$\begin{aligned}
&\langle \mathsf{LE}(\mathsf{succ}(x),\mathsf{succ}(y)),\mathsf{LE}(x,y)\rangle\\
&\langle \mathsf{M}(\mathsf{succ}(x),y),\mathsf{IF}_{\mathsf{minus}}(\mathsf{le}(\mathsf{succ}(x),y),\mathsf{succ}(x),y)\rangle\\
&\langle \mathsf{IF}_{\mathsf{minus}}(\mathsf{false},x,y),\mathsf{M}(x,y)\rangle\\
&\langle \mathsf{Q}(\mathsf{succ}(x),\mathsf{succ}(y)),\mathsf{Q}(\mathsf{minus}(x,y),\mathsf{succ}(y))\rangle
\end{aligned}$$

The CS $\mathcal{R}_0$ is terminating, this can be proved by a variant of the lexicographic path ordering or by using the dependency pair technique. In the latter proof we split $\mathcal{R}_0$ and use the techniques recursively.

The set of inequalities $\mathcal{DP}'$ is given by

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

$$\begin{array}{l}
\overline{\mathsf{le}}(0,\mathsf{succ}(y)) \succsim \mathsf{true}\\
\overline{\mathsf{le}}(0,0) \succsim \mathsf{true}\\
\overline{\mathsf{le}}(\mathsf{succ}(x),0) \succsim \mathsf{false}\\
\overline{\mathsf{le}}(\mathsf{succ}(x),\mathsf{succ}(y)) \succsim \overline{\mathsf{le}}(x,y)\\
\overline{\mathsf{minus}}(0,y) \succsim 0\\
\overline{\mathsf{minus}}(\mathsf{succ}(x),y) \succsim \overline{\mathsf{if}_{\mathsf{minus}}}(\overline{\mathsf{le}}(\mathsf{succ}(x),y),\mathsf{succ}(x),y)\\
\overline{\mathsf{if}_{\mathsf{minus}}}(\mathsf{true},\mathsf{succ}(x),y) \succsim 0\\
\overline{\mathsf{if}_{\mathsf{minus}}}(\mathsf{false},\mathsf{succ}(x),y) \succsim \mathsf{succ}(\overline{\mathsf{minus}}(x,y))
\end{array}$$

Again, a suitable quasi-ordering satisfying $\mathcal{DP}'$ is the normal ordering on natural numbers with an interpretation on the function symbols, where the function symbol 0 is mapped to the number 0, $\mathsf{succ}(x)$ is mapped to $x+1$, and $\mathsf{Q}(x,y)$, $\overline{\mathsf{minus}}(x,y)$ and $\overline{\mathsf{if}_{\mathsf{minus}}}(b,x,y)$ are mapped to x. All other function symbols (i.e. $\overline{\mathsf{le}}$ true, false) are mapped to the constant 0.

4 Remainder, Version 1 - 3

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

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

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

The relevant dependency pairs of this CS are given by

$$\begin{array}{l}
\langle \mathsf{LE}(\mathsf{succ}(x),\mathsf{succ}(y)),\mathsf{LE}(x,y)\rangle\\
\langle \mathsf{M}(\mathsf{succ}(x),\mathsf{succ}(y)),\mathsf{M}(x,y)\rangle\\
\langle \mathsf{MOD}(\mathsf{succ}(x),\mathsf{succ}(y)),\mathsf{IF}_{\mathsf{mod}}(\mathsf{le}(y,x),\mathsf{succ}(x),\mathsf{succ}(y))\rangle\\
\langle \mathsf{IF}_{\mathsf{mod}}(\mathsf{true},\mathsf{succ}(x),\mathsf{succ}(y)),\mathsf{MOD}(\mathsf{minus}(x,y),\mathsf{succ}(y))\rangle
\end{array}$$

The CS $\mathcal{R}_0$ is terminating. This can be proved by the recursive path ordering or by the dependency pair technique. The set of inequalities $\mathcal{DP}'$ is given by

$$\begin{array}{l}
\mathsf{MOD}(\mathsf{succ}(x),\mathsf{succ}(y)) \succ \mathsf{IF}_{\mathsf{mod}}(\overline{\mathsf{le}}(y,x),\mathsf{succ}(x),\mathsf{succ}(y)) \\
\mathsf{IF}_{\mathsf{mod}}(\mathsf{true},\mathsf{succ}(x),\mathsf{succ}(y)) \succ \mathsf{MOD}(\overline{\mathsf{minus}}(x,y),\mathsf{succ}(y))
\end{array}$$

$$\begin{array}{l}
\overline{\mathsf{le}}(0,\mathsf{succ}(y)) \succsim \mathsf{true} \\
\overline{\mathsf{le}}(0,0) \succsim \mathsf{true} \\
\overline{\mathsf{le}}(\mathsf{succ}(x),0) \succsim \mathsf{false} \\
\overline{\mathsf{le}}(\mathsf{succ}(x),\mathsf{succ}(y)) \succsim \overline{\mathsf{le}}(x,y) \\
\overline{\mathsf{minus}}(x,0) \succsim 0 \\
\overline{\mathsf{minus}}(\mathsf{succ}(x),\mathsf{succ}(y)) \succsim \overline{\mathsf{minus}}(x,y)
\end{array}$$

A suitable quasi-ordering satisfying $\mathcal{DP}'$ is the ordering on natural numbers, where the function symbol 0 is mapped to the number 0, $\mathsf{succ}(x)$ is mapped to $x+2$, $\mathsf{MOD}(x,y)$ is mapped to $x+1$, and $\mathsf{IF}_{\mathsf{mod}}(b,x,y)$ and $\overline{\mathsf{minus}}(x,y)$ are mapped to x. All other function symbols (i.e. $\overline{\mathsf{le}}$, true, false) are mapped to 0.

5 Greatest Common Divisor, Version 1 - 3

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

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

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

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

The relevant dependency pairs of this CS are

$$\langle \mathsf{LE}(\mathsf{succ}(x), \mathsf{succ}(y), \mathsf{LE}(x, y)\rangle$$
$$\langle \mathsf{M}(x, \mathsf{succ}(y)), \mathsf{M}(x, y)\rangle$$
$$\langle \mathsf{GCD}(\mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{IF}_{\mathsf{gcd}}(\mathsf{le}(y, x), \mathsf{succ}(x), \mathsf{succ}(y))\rangle$$
$$\langle \mathsf{IF}_{\mathsf{gcd}}(\mathsf{true}, \mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{GCD}(\mathsf{minus}(x, y), \mathsf{succ}(y))\rangle$$
$$\langle \mathsf{IF}_{\mathsf{gcd}}(\mathsf{false}, \mathsf{succ}(x), \mathsf{succ}(y)), \mathsf{GCD}(\mathsf{minus}(y, x), \mathsf{succ}(x))\rangle$$

Termination of $\mathcal{R}_0$ can be proved by the recursive path ordering or by the dependency pair approach. The set of inequalities $\mathcal{DP}'$ is

$$\mathsf{GCD}(\mathsf{succ}(x), \mathsf{succ}(y)) \succ \mathsf{IF}_{\mathsf{gcd}}(\overline{\mathsf{le}}(y, x), \mathsf{succ}(x), \mathsf{succ}(y))$$
$$\mathsf{IF}_{\mathsf{gcd}}(\mathsf{true}, \mathsf{succ}(x), \mathsf{succ}(y)) \succ \mathsf{GCD}(\overline{\mathsf{minus}}(x, y), \mathsf{succ}(y))$$
$$\mathsf{IF}_{\mathsf{gcd}}(\mathsf{false}, \mathsf{succ}(x), \mathsf{succ}(y)) \succ \mathsf{GCD}(\overline{\mathsf{minus}}(y, x), \mathsf{succ}(x))$$

$$\overline{\mathsf{le}}(0, \mathsf{succ}(y)) \succsim \mathsf{true}$$
$$\overline{\mathsf{le}}(0, 0) \succsim \mathsf{true}$$
$$\overline{\mathsf{le}}(\mathsf{succ}(x), 0) \succsim \mathsf{false}$$
$$\overline{\mathsf{le}}(\mathsf{succ}(x), \mathsf{succ}(y)) \succsim \overline{\mathsf{le}}(x, y)$$
$$\overline{\mathsf{pred}}(\mathsf{succ}(x)) \succsim x$$
$$\overline{\mathsf{minus}}(x, 0) \succsim 0$$
$$\overline{\mathsf{minus}}(x, \mathsf{succ}(y)) \succsim \overline{\mathsf{pred}}(\overline{\mathsf{minus}}(x, y))$$

A suitable quasi-ordering satisfying $\mathcal{DP}'$ is the ordering where the function symbol 0 is mapped to the number 0, $\mathsf{succ}(x)$ is mapped to $x + 2$, $\mathsf{GCD}(x, y)$ is mapped to $x+y+1$, and $\mathsf{IF}_{\mathsf{gcd}}(b, x, y)$ is mapped to $x+y$. The rest of the ordering is as in the preceding examples (i.e. $\overline{\mathsf{pred}}(x)$ and $\overline{\mathsf{minus}}(x, y)$ are mapped to x and all remaining function symbols are mapped to 0).

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.

6 Logarithm, Version 1

The following CS computes the dual logarithm.

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

$$\mathsf{log}(0) \to 0$$
$$\mathsf{log}(\mathsf{succ}(\mathsf{succ}(x))) \to \mathsf{succ}(\mathsf{log}(\mathsf{succ}(\mathsf{half}(x))))$$

The relevant dependency pairs of this CS are

$$\langle \mathsf{HALF}(\mathsf{succ}(\mathsf{succ}(x))), \mathsf{HALF}(x)\rangle$$
$$\langle \mathsf{LOG}(\mathsf{succ}(\mathsf{succ}(x))), \mathsf{LOG}(\mathsf{succ}(\mathsf{half}(x)))\rangle$$

The CS $\mathcal{R}_0$ is terminating. The recursive path ordering or the dependency pair approach directly prove this. The set of inequalities $\mathcal{DP}'$ is given by

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

$$\overline{\mathsf{half}}(0) \succsim 0$$
$$\overline{\mathsf{half}}(\mathsf{succ}(\mathsf{succ}(x))) \succsim \mathsf{succ}(\overline{\mathsf{half}}(x))$$

The interpretation for the function symbols, to derive the suitable quasi-ordering is given by: 0 is mapped to the number 0, $\mathsf{succ}(x)$ is mapped to $x+1$, and $\mathsf{LOG}(x)$ and $\overline{\mathsf{half}}(x)$ are both mapped to x.

7 Logarithm, Version 2 - 4

The following CS 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 CS (all of which are not simply terminating, since the quot CS $\mathcal{R}_{\mathsf{quot}}$ already was not simply terminating).

$$\mathcal{R}_{\mathsf{quot}}$$

$$\mathsf{log}(0, y) \to 0$$
$$\mathsf{log}(\mathsf{succ}(\mathsf{succ}(x))) \to \mathsf{succ}(\mathsf{log}(\mathsf{succ}(\mathsf{quot}(x, \mathsf{succ}(\mathsf{succ}(0))))))$$

The CS $\mathcal{R}_0$, in this case $\mathcal{R}_{\mathsf{quot}}$, is terminating. Termination of all three versions of this CS is proved in the earlier examples. Therefore, we only consider the new dependency pair to be relevant

$$\langle \mathsf{LOG}(\mathsf{succ}(\mathsf{succ}(x))), \mathsf{LOG}(\mathsf{succ}(\mathsf{quot}(x, \mathsf{succ}(\mathsf{succ}(0))))) \rangle$$

The set of inequalities $\mathcal{DP}'$ depends on the version of $\mathcal{R}_{\mathsf{quot}}$, but in all versions we have the inequality

$$\mathsf{LOG}(\mathsf{succ}(\mathsf{succ}(x))) \succ \mathsf{LOG}(\mathsf{succ}(\overline{\mathsf{quot}}(x, \mathsf{succ}(\mathsf{succ}(0)))))$$

The interpretation to derive a quasi-ordering that satisfies all three versions of $\mathcal{DP}'$ is given by: 0 is mapped to the number 0, $\mathsf{succ}(x)$ is mapped to $x+1$, $\mathsf{LOG}(x)$ and $\overline{\mathsf{quot}}(x, y)$ are both mapped to x, and all other function symbols are mapped to the same function as in the example corresponding to the version of $\mathcal{R}_{\mathsf{quot}}$.

8 Eliminating Duplicates

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

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

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

The relevant dependency pairs are

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

Termination of $\mathcal{R}_0$ can be proved with the dependency pair approach by considering this CS as a hierarchical combination of the eq rules and the other rules. The set of inequalities $\mathcal{DP}'$ is given by

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

$$\begin{aligned}
&\overline{\mathsf{eq}}(0,0) \succsim \mathsf{true}\\
&\overline{\mathsf{eq}}(0,\mathsf{succ}(x)) \succsim \mathsf{false}\\
&\overline{\mathsf{eq}}(\mathsf{succ}(x),0) \succsim \mathsf{false}\\
&\overline{\mathsf{eq}}(\mathsf{succ}(x),\mathsf{succ}(y)) \succsim \overline{\mathsf{eq}}(x,y)\\
&\overline{\mathsf{rm}}(n,\mathsf{empty}) \succsim \mathsf{empty}\\
&\overline{\mathsf{rm}}(n,\mathsf{add}(m,x)) \succsim \overline{\mathsf{if}_{\mathsf{rm}}}(\overline{\mathsf{eq}}(n,m),n,\mathsf{add}(m,x))\\
&\overline{\mathsf{if}_{\mathsf{rm}}}(\mathsf{true},n,\mathsf{add}(m,x)) \succsim \overline{\mathsf{rm}}(n,x)\\
&\overline{\mathsf{if}_{\mathsf{rm}}}(\mathsf{false},n,\mathsf{add}(m,x)) \succsim \mathsf{add}(m,\overline{\mathsf{rm}}(n,x))
\end{aligned}$$

This set of inequalities is satisfied by the normal ordering on natural numbers together with the interpretation given by: empty is mapped to 0, $\mathsf{add}(n,x)$ is mapped to $x+1$, $\overline{\mathsf{rm}}(x,y)$ and $\overline{\mathsf{if}_{\mathsf{rm}}}(b,x,y)$ are mapped to y, and $\mathsf{PURGE}(x)$ is mapped to x. All remaining function symbols are mapped to 0.

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 CS, but termination could still be proved with our method in the same way.

9 Selection Sort

The CS below, from [Wal94], is obviously not simply terminating. The CS 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.

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

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

The CS $\mathcal{R}_0$ is terminating, as can be proved fairly easy with the dependency pair approach.

The set of inequalities $\mathcal{DP}'$ is

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

$$\begin{aligned}
&\overline{\mathsf{eq}}(0,0) \succsim \mathsf{true}\\
&\overline{\mathsf{eq}}(0,\mathsf{succ}(x)) \succsim \mathsf{false}\\
&\overline{\mathsf{eq}}(\mathsf{succ}(x),0) \succsim \mathsf{false}\\
&\overline{\mathsf{eq}}(\mathsf{succ}(x),\mathsf{succ}(y)) \succsim \overline{\mathsf{eq}}(x,y)\\
&\overline{\mathsf{le}}(0,\mathsf{succ}(y)) \succsim \mathsf{true}\\
&\overline{\mathsf{le}}(0,0) \succsim \mathsf{true}\\
&\overline{\mathsf{le}}(\mathsf{succ}(x),0) \succsim \mathsf{false}\\
&\overline{\mathsf{le}}(\mathsf{succ}(x),\mathsf{succ}(y)) \succsim \overline{\mathsf{le}}(x,y)\\
&\overline{\mathsf{min}}(\mathsf{add}(0,\mathsf{empty})) \succsim 0\\
&\overline{\mathsf{min}}(\mathsf{add}(\mathsf{succ}(n),\mathsf{empty})) \succsim \mathsf{succ}(n)\\
&\overline{\mathsf{min}}(\mathsf{add}(n,\mathsf{add}(m,x))) \succsim \overline{\mathsf{if}_{\mathsf{min}}}(\overline{\mathsf{le}}(n,m),\mathsf{add}(n,\mathsf{add}(m,x)))\\
&\overline{\mathsf{if}_{\mathsf{min}}}(\mathsf{true},\mathsf{add}(n,\mathsf{add}(m,x))) \succsim \overline{\mathsf{min}}(\mathsf{add}(n,x))\\
&\overline{\mathsf{if}_{\mathsf{min}}}(\mathsf{false},\mathsf{add}(n,\mathsf{add}(m,x))) \succsim \overline{\mathsf{min}}(\mathsf{add}(m,x))\\
&\overline{\mathsf{replace}}(n,m,\mathsf{empty}) \succsim \mathsf{empty}\\
&\overline{\mathsf{replace}}(n,m,\mathsf{add}(k,x)) \succsim \overline{\mathsf{if}_{\mathsf{replace}}}(\overline{\mathsf{eq}}(n,k),n,m,\mathsf{add}(k,x))\\
&\overline{\mathsf{if}_{\mathsf{replace}}}(\mathsf{true},n,m,\mathsf{add}(k,x)) \succsim \mathsf{add}(m,x)\\
&\overline{\mathsf{if}_{\mathsf{replace}}}(\mathsf{false},n,m,\mathsf{add}(k,x)) \succsim \mathsf{add}(k,\overline{\mathsf{replace}}(n,m,x))
\end{aligned}$$

The interpretation in the natural numbers is: empty is mapped to 0, $\mathsf{add}(n,x)$ is mapped to $x+2$, $\mathsf{SELSORT}(x)$ is mapped to $x+1$, $\mathsf{IF}_{\mathsf{selsort}}(b,x)$ is mapped to x, and $\overline{\mathsf{replace}}(n,m,x)$ and $\overline{\mathsf{if}_{\mathsf{replace}}}(b,n,m,x)$ are both mapped to x. All remaining function symbols are mapped to the constant 0.

10 Minimum Sort

This CS 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.)

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

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

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

As in the other examples, the CS $\mathcal{R}_0$ can be proved terminating by recursively applying the technique of the dependency pairs approach to it. The set of inequalities $\mathcal{DP}'$ is

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

$$
\begin{aligned}
&\overline{\mathsf{eq}}(0, 0) \succsim \mathsf{true}\\
&\overline{\mathsf{eq}}(0, \mathsf{succ}(x)) \succsim \mathsf{false}\\
&\overline{\mathsf{eq}}(\mathsf{succ}(x), 0) \succsim \mathsf{false}\\
&\overline{\mathsf{eq}}(\mathsf{succ}(x), \mathsf{succ}(y)) \succsim \overline{\mathsf{eq}}(x, y)\\
&\overline{\mathsf{le}}(0, \mathsf{succ}(y)) \succsim \mathsf{true}\\
&\overline{\mathsf{le}}(0, 0) \succsim \mathsf{true}\\
&\overline{\mathsf{le}}(\mathsf{succ}(x), 0) \succsim \mathsf{false}\\
&\overline{\mathsf{le}}(\mathsf{succ}(x), \mathsf{succ}(y)) \succsim \overline{\mathsf{le}}(x, y)\\
&\overline{\mathsf{app}}(\mathsf{empty}, y) \succsim y\\
&\overline{\mathsf{app}}(\mathsf{add}(n, x), y) \succsim \mathsf{add}(n, \overline{\mathsf{app}}(x, y))\\
&\overline{\mathsf{min}}(\mathsf{add}(0, \mathsf{empty})) \succsim 0\\
&\overline{\mathsf{min}}(\mathsf{add}(\mathsf{succ}(n), \mathsf{empty})) \succsim \mathsf{succ}(n)\\
&\overline{\mathsf{min}}(\mathsf{add}(n, \mathsf{add}(m, x))) \succsim \overline{\mathsf{if}_{\mathsf{min}}}(\overline{\mathsf{le}}(n, m), \mathsf{add}(n, \mathsf{add}(m, x)))\\
&\overline{\mathsf{if}_{\mathsf{min}}}(\mathsf{true}, \mathsf{add}(n, \mathsf{add}(m, x))) \succsim \overline{\mathsf{min}}(\mathsf{add}(n, x))\\
&\overline{\mathsf{if}_{\mathsf{min}}}(\mathsf{false}, \mathsf{add}(n, \mathsf{add}(m, x))) \succsim \overline{\mathsf{min}}(\mathsf{add}(m, x))
\end{aligned}
$$

$$\begin{aligned}
&\overline{\mathsf{rm}}(n, \mathsf{empty}) \gtrsim \mathsf{empty}\\
&\overline{\mathsf{rm}}(n, \mathsf{add}(m, x)) \gtrsim \overline{\mathsf{if}_{\mathsf{rm}}}(\overline{\mathsf{eq}}(n, m), n, \mathsf{add}(m, x))\\
&\overline{\mathsf{if}_{\mathsf{rm}}}(\mathsf{true}, n, \mathsf{add}(m, x)) \gtrsim \overline{\mathsf{rm}}(n, x)\\
&\overline{\mathsf{if}_{\mathsf{rm}}}(\mathsf{false}, n, \mathsf{add}(m, x)) \gtrsim \mathsf{add}(m, \overline{\mathsf{rm}}(n, x))
\end{aligned}$$

A suitable interpretation is: empty is mapped to 0, $\mathsf{add}(n, x)$ is mapped to $x + 2$, $\mathsf{MINSORT}(x, y)$ is mapped to $(x + y)^2 + 2x + y + 1$, $\mathsf{IF}_{\mathsf{minsort}}(b, x, y)$ is mapped to $(x + y)^2 + 2x + y$, $\overline{\mathsf{rm}}(n, x)$ and $\overline{\mathsf{if}_{\mathsf{rm}}}(b, n, x)$ are both mapped to x, and $\mathsf{app}(x, y)$ is mapped to $x + y$. All remaining function symbols are mapped to the constant 0.

This example is inspired by an algorithm from [BM79] and [Wal94]. In the corresponding example from [Ste92] the rules for le, eq, $\mathsf{if}_{\mathsf{rm}}$ and $\mathsf{if}_{\mathsf{min}}$ were missing.

11 Quicksort

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

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

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

The CS $\mathcal{R}_0$ can be proved terminating by recursively applying the described techniques. The set of inequalities $\mathcal{DP}'$ is given by

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

$$\overline{\mathsf{le}}(0,\mathsf{succ}(y)) \succsim \mathsf{true}$$
$$\overline{\mathsf{le}}(0,0) \succsim \mathsf{true}$$
$$\overline{\mathsf{le}}(\mathsf{succ}(x),0) \succsim \mathsf{false}$$
$$\overline{\mathsf{le}}(\mathsf{succ}(x),\mathsf{succ}(y)) \succsim \overline{\mathsf{le}}(x,y)$$
$$\overline{\mathsf{app}}(\mathsf{empty},y) \succsim y$$
$$\overline{\mathsf{app}}(\mathsf{add}(n,x),y) \succsim \mathsf{add}(n,\overline{\mathsf{app}}(x,y))$$
$$\overline{\mathsf{low}}(n,\mathsf{empty}) \succsim \mathsf{empty}$$
$$\overline{\mathsf{low}}(n,\mathsf{add}(m,x)) \succsim \overline{\mathsf{if}_{\mathsf{low}}}(\overline{\mathsf{le}}(m,n),n,\mathsf{add}(m,x))$$
$$\overline{\mathsf{if}_{\mathsf{low}}}(\mathsf{true},n,\mathsf{add}(m,x)) \succsim \mathsf{add}(m,\overline{\mathsf{low}}(n,x))$$
$$\overline{\mathsf{if}_{\mathsf{low}}}(\mathsf{false},n,\mathsf{add}(m,x)) \succsim \overline{\mathsf{low}}(n,x)$$
$$\overline{\mathsf{high}}(n,\mathsf{empty}) \succsim \mathsf{empty}$$
$$\overline{\mathsf{high}}(n,\mathsf{add}(m,x)) \succsim \overline{\mathsf{if}_{\mathsf{high}}}(\overline{\mathsf{le}}(m,n),n,\mathsf{add}(m,x))$$
$$\overline{\mathsf{if}_{\mathsf{high}}}(\mathsf{true},n,\mathsf{add}(m,x)) \succsim \overline{\mathsf{high}}(n,x)$$
$$\overline{\mathsf{if}_{\mathsf{high}}}(\mathsf{false},n,\mathsf{add}(m,x)) \succsim \mathsf{add}(m,\overline{\mathsf{high}}(n,x))$$

A suitable interpretation is: empty is mapped to 0, $\mathsf{add}(n,x)$ is mapped to $x+1$, $\overline{\mathsf{low}}(n,x)$, $\overline{\mathsf{if}_{\mathsf{low}}}(b,n,x)$, $\overline{\mathsf{high}}(n,x)$, $\overline{\mathsf{if}_{\mathsf{high}}}(b,n,x)$ and $\mathsf{QUICKSORT}(x)$ are all mapped to x and $\overline{\mathsf{app}}(x,y)$ is mapped to $x+y$. All remaining function symbols are mapped to the constant 0.

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 CS. With our method termination could still be proved in the same way.

12 Permutation of Lists

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

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

Termination of $\mathcal{R}_0$, the first four rules, can easily be proved by the recursive path ordering or the dependency pair approach. The set $\mathcal{DP}'$ of inequalities is

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

$$\begin{aligned}
&\overline{\mathsf{app}}(\mathsf{empty},y) \succsim y\\
&\overline{\mathsf{app}}(\mathsf{add}(n,x),y) \succsim \mathsf{add}(n,\overline{\mathsf{app}}(x,y))\\
&\overline{\mathsf{reverse}}(\mathsf{empty}) \succsim \mathsf{empty}\\
&\overline{\mathsf{reverse}}(\mathsf{add}(n,x)) \succsim \overline{\mathsf{app}}(\overline{\mathsf{reverse}}(x),\mathsf{add}(n,\mathsf{empty}))
\end{aligned}$$

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

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 CS of this example from [Gie95a]. The fourth argument of reach is used to store edges that have already been examined but that are not included in the actual solution path. If an edge from u to v (with $x \neq u$) is found, then it is rejected at first. If an edge from x to v (with $v \neq y$) is found then one either searches for further edges beginning in x (then one will never need the edge from x to v again) or one tries to find a path from v to y and now all edges that were rejected before have to be considered again.

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

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

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

$$\mathsf{if}_{\mathsf{reach\_2}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) \to \mathsf{or}(\mathsf{reach}(x, y, i, h), \mathsf{reach}(v, y, \mathsf{union}(i, h), \epsilon))$$
$$\mathsf{if}_{\mathsf{reach\_1}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) \to \mathsf{reach}(x, y, i, \mathsf{edge}(u, v, h))$$

The CS $\mathcal{R}_0$ can be proved terminating very easy, for example by the dependency pair approach. The set of inequalities $\mathcal{DP}'$ is

$$\mathsf{REACH}(x, y, \mathsf{edge}(u, v, i), h) \succ \mathsf{IF}_{\mathsf{reach\_1}}(\overline{\mathsf{eq}}(x, u), x, y, \mathsf{edge}(u, v, i), h)$$
$$\mathsf{IF}_{\mathsf{reach\_1}}(\mathsf{true}, x, y, \mathsf{edge}(u, v, i), h) \succ \mathsf{IF}_{\mathsf{reach\_2}}(\overline{\mathsf{eq}}(y, v), x, y, \mathsf{edge}(u, v, i), h)$$
$$\mathsf{IF}_{\mathsf{reach\_2}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) \succ \mathsf{REACH}(x, y, i, h)$$
$$\mathsf{IF}_{\mathsf{reach\_2}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) \succ \mathsf{REACH}(v, y, \overline{\mathsf{union}}(i, h), \epsilon)$$
$$\mathsf{IF}_{\mathsf{reach\_1}}(\mathsf{false}, x, y, \mathsf{edge}(u, v, i), h) \succ \mathsf{REACH}(x, y, i, \mathsf{edge}(u, v, h))$$

$$\overline{\mathsf{eq}}(0, 0) \succsim \mathsf{true}$$
$$\overline{\mathsf{eq}}(0, \mathsf{succ}(x)) \succsim \mathsf{false}$$
$$\overline{\mathsf{eq}}(\mathsf{succ}(x), 0) \succsim \mathsf{false}$$
$$\overline{\mathsf{eq}}(\mathsf{succ}(x), \mathsf{succ}(y)) \succsim \overline{\mathsf{eq}}(x, y)$$
$$\overline{\mathsf{or}}(\mathsf{true}, x) \succsim \mathsf{true}$$
$$\overline{\mathsf{or}}(\mathsf{false}, \mathsf{true}) \succsim \mathsf{true}$$
$$\overline{\mathsf{or}}(\mathsf{false}, \mathsf{false}) \succsim \mathsf{false}$$
$$\overline{\mathsf{union}}(\epsilon, h) \succsim h$$
$$\overline{\mathsf{union}}(\mathsf{edge}(x, y, i), h) \succsim \mathsf{edge}(x, y, \overline{\mathsf{union}}(i, h))$$

A suitable interpretation is: ϵ is mapped to 0, $\mathsf{edge}(x, y, g)$ is mapped to $g+2$, $\mathsf{REACH}(x, y, g, h)$ is mapped to $(g+h)^2+2g+h+2$, $\mathsf{IF}_{\mathsf{reach\_1}}(b, x, y, g, h)$ is mapped to $(g+h)^2+2g+h+1$, $\mathsf{IF}_{\mathsf{reach\_2}}(b, x, y, g, h)$ is mapped to $(g+h)^2+2g+h$ and $\overline{\mathsf{union}}(g, h)$ is mapped to $g+h$. All remaining function symbols are mapped to 0.

14 Comparison of Binary Trees

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

$$\mathsf{concat}(\mathsf{nil}, y) \to y$$
$$\mathsf{concat}(\mathsf{cons}(u, v), y) \to \mathsf{cons}(u, \mathsf{concat}(v, y))$$

$$\mathsf{less\_leafs}(x, \mathsf{nil}) \to \mathsf{false}$$
$$\mathsf{less\_leafs}(\mathsf{nil}, \mathsf{cons}(w, z)) \to \mathsf{true}$$
$$\mathsf{less\_leafs}(\mathsf{cons}(u, v), \mathsf{cons}(w, z)) \to \mathsf{less\_leafs}(\mathsf{concat}(u, v), \mathsf{concat}(w, z))$$

The two rules of $\mathcal{R}_0$ are easily proved terminating. The set of inequalities $\mathcal{DP}'$ is

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

$\overline{\mathsf{concat}}(\mathsf{nil},y) \succsim y$
$\overline{\mathsf{concat}}(\mathsf{cons}(u,v),y) \succsim \mathsf{cons}(u,\overline{\mathsf{concat}}(v,y))$

A suitable interpretation is: nil is mapped to 0, $\mathsf{cons}(u,v)$ is mapped to $1+u+v$, $\mathsf{LESS\_LEAFS}(x,y)$ is mapped to x, and $\overline{\mathsf{concat}}(u,v)$ is mapped to $u+v$.

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

References

[Art96] T. Arts. Termination by absence of infinite chains of dependency pairs. In *Proc. Coll. Trees in Algebra and Programming*, Linköping, Sweden, 1996.

[AG96] T. Arts & J. Giesl. Termination of constructor systems. In *Proceedings of the 7th International Conference on Rewriting Techniques and Applications, LNCS 1103*, New Brunswick, NJ, USA, 1996.

[BD86] L. Bachmair & N. Dershowitz. Commutation, transformation and termination. In *Proc. 8th CADE, LNCS 230*, Oxford, England, 1986.

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

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

[BCL87] A. Ben Cherifa & P. Lescanne. Termination of rewriting systems by polynomial interpretations and its implementation. *Science of Computer Programming*, 9(2):137-159, 1987.

[BHHW86] S. Biundo, B. Hummel, D. Hutter & C. Walther. The Karlsruhe induction theorem proving system. *8th CADE, LNCS 230*, Oxford, England, 1986.

[BKR92] A. Bouhoula, E. Kounalis & M. Rusinowitch. SPIKE: an automatic theorem prover. In *Proceedings of the Conference on Logic Programming and Automated Reasoning, LNAI 624*, St. Petersburg, Russia, 1992.

[BM79] R. S. Boyer & J S. Moore. *A computational logic.* Academic Press, 1979.

[BHHS90] A. Bundy, F. van Harmelen, C. Horn & A. Smaill. The OYSTER-CLAM system. In *Proc. 10th CADE, LNAI 449*, Kaiserslautern, Germany, 1990.

[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, 2):69-115, 1987.

[DJ90] N. Dershowitz & J.-P. Jouannaud. Rewrite systems. *Handbook of Theoret. Comp. Sc.*, J. van Leeuwen, ed., vol. B, ch. 6, pp. 243-320, Elsevier, 1990.

[DH95] N. Dershowitz & C. Hoot. Natural Termination. *Theoretical Computer Science*, 142(2):179-207, 1995.

[FZ95] M. C. F. Ferreira & H. Zantema. Dummy elimination: making termination easier. In *Proceedings of the 10th International Conference on Fundamentals of Computation Theory, LNCS 965*, Dresden, Germany, 1995

[Ges94] A. Geser. An improved general path order. Technical Report MIP-9407, Universität Passau, Germany. To appear in *Applicable Algebra in Engineering, Communication, and Computation.*

[Gie95a] J. Giesl, *Automatisierung von Terminierungsbeweisen für rekursiv definierte Algorithmen.* Doctoral Dissertation, Technische Hochschule Darmstadt, Germany, 1995.

[Gie95b] J. Giesl. Generating polynomial orderings for termination proofs. In *Proc. 6th RTA*, *LNCS 914*, Kaiserslautern, Germany, 1995.

[Gie95c] J. Giesl. Automated termination proofs with measure functions. In *Proc. 19th Annual German Conf. on AI*, *LNAI 981*, Bielefeld, Germany, 1995.

[Gie95d] J. Giesl. Termination analysis for functional programs using term orderings. In *Proceedings of the Second International Static Analysis Symposium*, *LNCS 983*, Glasgow, Scotland, 1995.

[HL78] G. Huet & D. S. Lankford. On the uniform halting problem for term rewriting systems. Rapport Laboria 283, Institut de Recherche d'Informatique et d'Automatique, Le Chesnay, France, 1978.

[KZ89] D. Kapur & H. Zhang. An overview of Rewrite Rule Laboratory (RRL). In *Proc. 3rd RTA*, *LNCS 355*, Chapel Hill, NC, 1989.

[Ken95] R. Kennaway. Complete term rewrite systems for decimal arithmetic and other total recursive functions. Presented at the *Second International Workshop on Termination*, La Bresse, France, 1995.

[KB70] D. E. Knuth & P. B. Bendix. Simple word problems in universal algebras. *Computational Problems in Abstract Algebra*, J. Leech, ed., Pergamon Press, pp. 263-297, 1970.

[KR95] 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. Tech. Report Memo MTP-3, Louisiana Tech. University, Ruston, LA, 1979.

[MN70] Z. Manna & S. Ness. On the termination of Markov algorithms. In *Proc. of the 3rd Hawaii Int. Conf. on System Science*, Honolulu, HI, 1970.

[Mar87] U. Martin. How to choose weights in the Knuth-Bendix ordering. In *Proc. 2nd RTA*, *LNCS 256*, Bordeaux, France, 1987.

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

[Pla85] D. A. Plaisted. Semantic confluence tests and completion methods. *Inform. and Control*, 65(2/3):182-215, 1985.

[Ste92] J. Steinbach. Notes on Transformation Orderings. SEKI-Report SR-92-23, Universität Kaiserslautern, Germany, 1992.

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

[Ste95a] J. Steinbach. Automatic termination proofs with transformation orderings. In *Proc. 6th RTA*, *LNCS 914*, Kaiserslautern, Germany, 1995.

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

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