Dissertation

Automated Termination Analysis for Term Rewriting

Nao Hirokawa

Faculty of Mathematics, Computer Science and Physics
University of Innsbruck

May 29, 2006

Acknowledgements

I am grateful to my supervisor Aart Middeldorp for his guidance and valuable advice. Indeed, he was indispensable for the progress of my research. I also thank him for his assistance for living in Innsbruck. Special thanks to my colleagues in the Computational Logic group. I would like to thank to Daniel Laimer for his contribution to improve the web interface of the termination tool TTT, which is explained in this thesis. Last but not least, I thank my family. Some of the results in this thesis are inspired by talking with them.

Contents

Chapter 1

Introduction

Term rewriting is a simple and powerful computational model, which underlies declarative programming and automated theorem proving. Termination ensures that all computation paths end. This thesis is about *automated* termination analysis for term rewriting. All techniques described in this thesis are implemented in T$_\mathsf{T}$T, an automated termination prover for term rewrite systems, and we provide ample experimental data.

The first section is intended to provide a brief and informal introduction to term rewriting, termination, and the dependency pair method. For detailed surveys of term rewriting we refer to [6] and [51]. An overview of the contributions of this thesis is given in the second section.

1.1 Term Rewrite Systems and Termination

Term rewriting is a simple and powerful computational model. A *term rewrite system* (*TRS*) is defined as a directed equational system.

Example 1.1. *Consider the term rewrite system $\mathcal{R}$ consisting of two rules*

$$1\colon\quad 0 + y \to y \qquad\qquad 2\colon\quad \mathsf{s}(x) + y \to \mathsf{s}(x + y)$$

The computation of the term $\mathsf{s}(\mathsf{s}(0)) + \mathsf{s}(0)$ is done by rewriting. For instance, by instantiating x and y to $\mathsf{s}(0)$ in rule 2, the left-hand side is identical to our term, and it can be rewritten to the instantiated right-hand side of the rule, i.e., $\mathsf{s}(\mathsf{s}(0) + \mathsf{s}(0))$. Repeating such an operation, one obtains the following rewrite sequence*:*

$$\mathsf{s}(\mathsf{s}(0)) + \mathsf{s}(0) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(0) + \mathsf{s}(0)) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(0 + \mathsf{s}(0))) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(\mathsf{s}(0)))$$

The last term $\mathsf{s}(\mathsf{s}(\mathsf{s}(0)))$ cannot be rewritten any more, and such terms are called normal forms *of $\mathcal{R}$. This TRS is* terminating *which means there are no infinite rewrite sequences starting from any term.*

The next example is a famous non-terminating TRS by Toyama [54], which illustrates a tricky behavior of rewriting.

Example 1.2. *Consider the TRS $\mathcal{R}_1$:*

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

and the TRS $\mathcal{R}_2$:

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

Both TRSs are terminating, although a formal termination proof of the first TRS is not entirely trivial. The union $\mathcal{R}_1 \cup \mathcal{R}_2$ is not terminating because the following infinite sequence exists:

$$\begin{aligned} & \mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g}(\mathsf{a},\mathsf{b})) \\ \to_{\mathcal{R}_1}\ & \mathsf{f}(\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b})) \\ \to_{\mathcal{R}_2}\ & \mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b})) \\ \to_{\mathcal{R}_2}\ & \mathsf{f}(\mathsf{a},\mathsf{b},\mathsf{g}(\mathsf{a},\mathsf{b})) \\ \to_{\mathcal{R}_1}\ & \cdots \end{aligned}$$

Term rewriting has a close relation with theorem proving and declarative programs. In fact numerous interesting axioms and programs can be encoded as term rewrite systems, either directly or via transformation techniques, and termination is a desired property. Let's take a look at a more practical example.

Example 1.3. *Consider the following TRS $\mathcal{R}$, which is an encoding of Eratosthenes' sieve:*

$$\begin{array}{rlrl}
1\colon & x - \mathsf{0} \to x & 10\colon & \mathsf{filter}(x,\mathsf{nil}) \to \mathsf{nil} \\
2\colon & \mathsf{s}(x) - \mathsf{s}(y) \to x - y & 11\colon & \mathsf{filter}(x, y:ys) \to \mathsf{if_filter}(x \mid y, x, y:ys) \\
3\colon & \mathsf{0} \leqslant y \to \mathsf{true} & 12\colon & \mathsf{if_filter}(\mathsf{true}, x, y:ys) \to \mathsf{filter}(x, ys) \\
4\colon & \mathsf{s}(x) \leqslant \mathsf{0} \to \mathsf{false} & 13\colon & \mathsf{if_filter}(\mathsf{false}, x, y:ys) \to y : \mathsf{filter}(x, ys) \\
5\colon & \mathsf{s}(x) \leqslant \mathsf{s}(y) \to x \leqslant y & 14\colon & \mathsf{sieve}(\mathsf{nil}) \to \mathsf{nil} \\
6\colon & \mathsf{if}(\mathsf{true}, x, y) \to x & 15\colon & \mathsf{sieve}(x:xs) \to x : \mathsf{sieve}(\mathsf{filter}(x, xs)) \\
7\colon & \mathsf{if}(\mathsf{false}, x, y) \to y & & \\
8\colon & x \mid \mathsf{0} \to \mathsf{true} & & \\
9\colon & \mathsf{s}(x) \mid \mathsf{s}(y) \to \mathsf{if}(x \leqslant y, \mathsf{s}(x) \mid (y - x), \mathsf{false}) & &
\end{array}$$

For example,

$$\begin{aligned} \mathsf{sieve}(2:3:4:5:6:\mathsf{nil}) \ \to_{\mathcal{R}}\ & 2 : \mathsf{sieve}(\mathsf{filter}(2, 3:4:5:6:\mathsf{nil})) \\ \to^{+}_{\mathcal{R}}\ & 2 : \mathsf{sieve}(3:5:\mathsf{nil})) \\ \to_{\mathcal{R}}\ & 2:3 : \mathsf{sieve}(\mathsf{filter}(3, 5:\mathsf{nil})) \\ \to^{+}_{\mathcal{R}}\ & 2:3 : \mathsf{sieve}(5:\mathsf{nil}) \\ \to_{\mathcal{R}}\ & 2:3:5 : \mathsf{sieve}(\mathsf{filter}(5, \mathsf{nil})) \\ \to^{+}_{\mathcal{R}}\ & 2:3:5 : \mathsf{sieve}(\mathsf{nil}) \\ \to_{\mathcal{R}}\ & 2:3:5 : \mathsf{nil} \end{aligned}$$

Here 2, 3, ... denotes $\mathsf{s}(\mathsf{0}))$, $\mathsf{s}(\mathsf{s}(\mathsf{0})))$, *etc.*

Not surprisingly, arbitrary Turing machines can be simulated by some term rewrite system. As a consequence, the termination property of TRSs is undecidable in general ([35]). However, in order to prove termination automatically many techniques have been developed.

The traditional method is to check compatibility of the rules with parameterized orderings like the Knuth-Bendix order [37], the multiset path order of Dershowitz [12], the lexicography path order of Kamin and Lévy [36], and the polynomial interpretation orders of Lankford [41]. One may conclude termination of a TRS if all its rewrite rules are oriented from left to right by one of these orders. The automation of these techniques is addressed in a long list of papers including Dick *et al.* [16], Ben Cherifa and Lescanne [9], Steinbach [49], Giesl [20], Hong and Jakuš [34], and Contejean *et al.* [11].

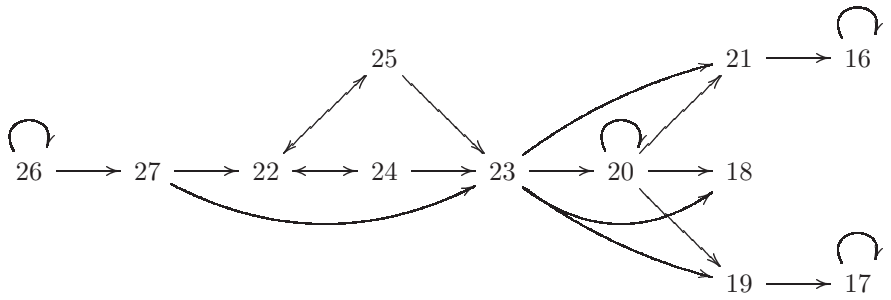
Many recent techniques enhance this method by performing transformations on TRSs. We mention here semantic labeling of Zantema [60], the freezing technique of Xi [58], the work of Borralleras *et al.* [8] on transforming the semantic path order of Kamin and Lévy [36] into a monotonic version that is amenable to automation, and the dependency pair method of Arts and Giesl [5]. The latter method, implemented in several automatic termination provers (cf. Section 6.3), is the starting point of this thesis.

Given a TRS, the method first extracts dependency pairs and constructs its dependency graph. Intuitively, the former represents pairs of tuple of arguments before and after recursive calls, and the latter approximates the recursive call graph. A cycle in the graph represents a potential loop. In the dependency pair method the possibility of the loop is canceled by finding a pair of orderings $\gtrsim$ and $>$ compatible with the TRS and the cycle respectively. The TRS is terminating if all cycles are canceled,

Example 1.4. *There are 12 dependency pairs for the TRS $\mathcal{R}$ in Example 1.3:*

$$
\begin{aligned}
16:&\quad \mathsf{s}(x) -^\sharp \mathsf{s}(y) &&\to x -^\sharp y\\
17:&\quad \mathsf{s}(x) \leqslant^\sharp \mathsf{s}(y) &&\to x \leqslant^\sharp y\\
18:&\quad \mathsf{s}(x) \mid^\sharp \mathsf{s}(y) &&\to \mathsf{if}^\sharp(x \leqslant y, \mathsf{s}(x) \mid (y - x), \mathsf{false})\\
19:&\quad \mathsf{s}(x) \mid^\sharp \mathsf{s}(y) &&\to x \leqslant^\sharp y\\
20:&\quad \mathsf{s}(x) \mid^\sharp \mathsf{s}(y) &&\to \mathsf{s}(x) \mid^\sharp (y - x)\\
21:&\quad \mathsf{s}(x) \mid^\sharp \mathsf{s}(y) &&\to y -^\sharp x\\
22:&\quad \mathsf{filter}^\sharp(x, y : ys) &&\to \mathsf{if\_filter}^\sharp(x \mid y, x, y : ys)\\
23:&\quad \mathsf{filter}^\sharp(x, y : ys) &&\to x \mid^\sharp y\\
24:&\quad \mathsf{if\_filter}^\sharp(\mathsf{true}, x, y : ys) &&\to \mathsf{filter}^\sharp(x, ys)\\
25:&\quad \mathsf{if\_filter}^\sharp(\mathsf{false}, x, y : ys) &&\to \mathsf{filter}^\sharp(x, ys)\\
26:&\quad \mathsf{sieve}^\sharp(x : xs) &&\to \mathsf{sieve}^\sharp(\mathsf{filter}(x, xs))\\
27:&\quad \mathsf{sieve}^\sharp(x : xs) &&\to \mathsf{filter}^\sharp(x, xs)
\end{aligned}
$$

The dependency graph



contains 7 cycles: $\{16\}$, $\{17\}$, $\{20\}$, $\{22,24\}$, $\{22,25\}$, $\{22,24,25\}$, *and* $\{26\}$.

We claim that there is no infinite rewrite sequence corresponding to the cycle $\mathcal{C} = \{26\}$. *This is shown as follows. By taking the polynomial interpretation* $\mathsf{true}_{\mathbb{N}} = \mathsf{nil}_{\mathbb{N}} = \mathsf{false}_{\mathbb{N}} = \mathsf{0}_{\mathbb{N}} = 1$, $-_{\mathbb{N}}(x,y) = x$, $\mathsf{sieve}^{\sharp}{}_{\mathbb{N}}(x) = \mathsf{sieve}_{\mathbb{N}}(x) = \mathsf{s}_{\mathbb{N}}(x) = x{+}1$, $|_{\mathbb{N}}(x,y) = \leqslant_{\mathbb{N}}(x,y) = :_{\mathbb{N}}(x,y) = x{+}y{+}1$, $\mathsf{filter}_{\mathbb{N}}(x,y) = y$, $\mathsf{if}_{\mathbb{N}}(x,y,z) = y{+}z$ *and* $\mathsf{if\_filter}_{\mathbb{N}}(x,y,z) = z$, *the involved rules reduce to the following inequalities:*

$$
\begin{array}{rrcl@{\qquad}rrcl}
1: & x & \gtrsim & x & 9: & x+y+3 & \gtrsim & x+y+3 \\
2: & x+1 & \gtrsim & x & 10: & 1 & \gtrsim & 1 \\
3: & y+2 & \gtrsim & 1 & 11: & y+ys+1 & \gtrsim & y+ys+1 \\
4: & x+3 & \gtrsim & 1 & 12: & y+ys+1 & \gtrsim & ys \\
5: & x+y+3 & \gtrsim & x+y+1 & 13: & y+ys+1 & \gtrsim & y+ys+1 \\
6: & x+y & \gtrsim & x & 14: & 2 & \gtrsim & 1 \\
7: & x+y & \gtrsim & y & 15: & x+xs+2 & \gtrsim & x+xs+2 \\
8: & x+2 & \gtrsim & 1 & 26: & x+xs+2 & > & xs+1
\end{array}
$$

Similarly, one can find orderings for the remaining 6 cycles. Hence, termination of $\mathcal{R}$ *is established.*

The complete termination proof, which is based on new results presented in the thesis and generated by the automatic termination prover T$_{\mathsf{T}}$T, can be found in Appendix A.

We conclude this introductory section with an encoding of *Goldbach's conjecture*. This is one example of an interesting problem that can be reduced to a termination problem. Goldbach's conjecture state that all positive even integers larger than 2 can be expressed as the sum of two primes.

Example 1.5. *We add the following rules to the TRS in Example 1.3:*

$$
\begin{array}{rcl@{\qquad}rcl}
0+y & \to & y & \mathsf{range}(x) & \to & \mathsf{range}'(x,\mathsf{nil}) \\
\mathsf{s}(x)+y & \to & \mathsf{s}(x+y) & \mathsf{range}'(0,ys) & \to & ys \\
0=0 & \to & \mathsf{true} & \mathsf{range}'(\mathsf{s}(0),ys) & \to & ys \\
\mathsf{s}(x)=0 & \to & \mathsf{false} & \mathsf{range}'(\mathsf{s}(\mathsf{s}(x)),ys) & \to & \mathsf{range}'(\mathsf{s}(x),\mathsf{s}(\mathsf{s}(x)):ys) \\
0=\mathsf{s}(x) & \to & \mathsf{false} & \mathsf{test}(n,\mathsf{nil},ys) & \to & \mathsf{false} \\
\mathsf{s}(x)=\mathsf{s}(y) & \to & x=y & \mathsf{test}(n,x:xs,ys) & \to & \mathsf{test}'(n,x,ys) \vee \mathsf{test}(n,xs,ys) \\
\mathsf{true}\vee x & \to & \mathsf{true} & \mathsf{test}'(n,x,\mathsf{nil}) & \to & \mathsf{false} \\
\mathsf{false}\vee x & \to & x & \mathsf{test}'(n,x,y:ys) & \to & (n=x+y)\vee\mathsf{test}'(n,x,ys)
\end{array}
$$

Here the term $\mathsf{range}(n)$ *rewrites to the list* $2 : \ldots : n : \mathsf{nil}$*, and* $\mathsf{test}(n, x_1 : \cdots : x_m : \mathsf{nil}, y_1 : \cdots : y_n : \mathsf{nil})$ *checks whether* $n = x_i + y_j$ *for some* i *and* j*. Finally, we add the following three rules:*

$$\begin{aligned}
\mathsf{goldbach}(\mathsf{s}(\mathsf{s}(n))) &\to \mathsf{loop}(\mathsf{s}(\mathsf{s}(n)), \mathsf{test}(\mathsf{s}(\mathsf{s}(n)) + \mathsf{s}(\mathsf{s}(n)),\\
&\qquad \mathsf{sieve}(\mathsf{range}(\mathsf{s}(\mathsf{s}(n)) + \mathsf{s}(\mathsf{s}(n)))),\\
&\qquad \mathsf{sieve}(\mathsf{range}(\mathsf{s}(\mathsf{s}(n)) + \mathsf{s}(\mathsf{s}(n))))))\\
\mathsf{loop}(\mathsf{s}(\mathsf{s}(n)), \mathsf{true}) &\to \mathsf{goldbach}(\mathsf{s}(n))\\
\mathsf{loop}(\mathsf{s}(\mathsf{s}(n)), \mathsf{false}) &\to \mathsf{goldbach}(\mathsf{s}(\mathsf{s}(n)))
\end{aligned}$$

Since $\mathsf{sieve}(\mathsf{range}(n + n))$ *generates all prime numbers less than or equal to* $2n$*, the normal form of* $\mathsf{test}(n + n, \mathsf{sieve}(\mathsf{range}(n + n)), \mathsf{sieve}(\mathsf{range}(n + n)))$ *is* true *if* $2n = p + q$ *for some prime numbers* p *and* q*, and* false *otherwise. Therefore,* $\mathsf{goldbach}(\mathsf{s}(\mathsf{s}(n)))$ *terminates if and only if Goldbach's conjecture holds for all even numbers* $4, 6, \ldots, 2(n + 2)$*. Hence, if our TRS is terminating then the conjecture is true. How about the converse?* T*T* *can find suitable orderings for all cycles in the dependency graph, except for the one consisting of the following two dependency pairs:*

$$\begin{aligned}
\mathsf{goldbach}^\sharp(\mathsf{s}(\mathsf{s}(n))) &\to \mathsf{loop}^\sharp(\mathsf{s}(\mathsf{s}(n)), \mathsf{test}(\mathsf{s}(\mathsf{s}(n)) + \mathsf{s}(\mathsf{s}(n)),\\
&\qquad \mathsf{sieve}(\mathsf{range}(\mathsf{s}(\mathsf{s}(n)) + \mathsf{s}(\mathsf{s}(n)))),\\
&\qquad \mathsf{sieve}(\mathsf{range}(\mathsf{s}(\mathsf{s}(n)) + \mathsf{s}(\mathsf{s}(n))))))\\
\mathsf{loop}^\sharp(\mathsf{s}(\mathsf{s}(n)), \mathsf{false}) &\to \mathsf{goldbach}^\sharp(\mathsf{s}(\mathsf{s}(n)))
\end{aligned}$$

This means that our TRS is terminating only if for no n *there is an infinite rewrite sequence starting from* $\mathsf{goldbach}(\mathsf{s}(\mathsf{s}(n)))$*. Hence, Goldbach's conjecture is false if our TRS is non-terminating.*

Needless to say, proving or disproving termination of the preceding TRS is beyond all existing (automated) termination techniques, including the ones introduced in this thesis.

1.2 Overview and Results

The aim of this thesis is to develop automated termination techniques based on the dependency pair method. New methods are intended to make termination tools more *powerful* and more *efficient*. In automation we often face large search spaces for parameters required by termination criteria and there is a natural trade-off between power and efficiency. Therefore, developing efficient search techniques are of particular interest. On the other hand, what if one could reduce search spaces without (much) loss of power? Then, one achieves an increase in both practical power and efficiency. For this purpose we go back to the foundations of dependency pair method.

Overview. Starting from scratch, we give a systematic account of the method in Chapter 3. Along the way we derive two new refinements—the *subterm criterion* in Section 3.2 and the *usable rule criterion* for termination in Section 3.4—that are very easy to implement, increase the termination proving power, give rise to simpler termination proofs, and make the method much faster.

Chapter 4 deals with three problems when automating the dependency pair method. The first problem is the undecidability of dependency graphs. In Section 4.1 we present various computable approximations of the dependency graph including the new *estimated* innermost dependency graph approximation*. When a given TRS is non-terminating, its dependency graph always contains cycles. In order to prove termination the standard dependency pair method requires to analyze *all cycles*. However, the number of cycles can be exponential in the number of dependency pairs. That's why one prefers to analyze *all strongly connected components* (SCCs) instead of all cycles. However the latter approach is much weaker than the former. This problem is fully settled by the *recursive SCC algorithm* in Section 4.2.

The third problem is the search space of argument filterings. Typically, the ordering constraints in the obtained groups must be simplified before traditional simplification orders like the recursive path order or the Knuth-Bendix order are applicable. Such simplifications are performed by so-called argument filterings. It is fair to say that the dependency pair method derives much of its power from the ability to use argument filterings to simplify constraints. The finiteness of the argument filtering search space has been stressed in many papers on the dependency pair method, but we do not hesitate to label the enormous size of this search space as the main obstacle for the successful automation of the dependency pair method when using strongly monotone simplification orders. In Section 4.3 techniques for searching suitable argument filterings are described: efficient heuristics and the divide and conquer technique.

In Chapter 5 we explain how to use polynomial interpretation orders with negative coefficients to prove termination. In Section 5.1 we explain how to use polynomial interpretations with *negative constants* (like $x-1$). We do the same for polynomial interpretations with *negative coefficients* (like $x-y$ and $1-x$) in Section 5.2. The usable rule criterion of Section 3.4 is not directly applicable in the latter case. We explain how to restore the criterion.

All techniques developed are implemented in a powerful termination tool T$_\mathsf{T}$T, which is described in Chapter 6. In Section 6.2 we provide experimental results for evaluating the usefulness of all methods described in earlier chapters. Section 6.3 contains a brief comparison with other termination tools.

Contributions. Here we list the main contributions of the thesis.

- The subterm criterion (Section 3.2).
- The usable rule criterion for full termination (Section 3.4).
- The estimated* innermost dependency graph (Section 4.1).
- The recursive SCC algorithm (Section 4.2).
- Heuristics and the divide and conquer algorithm for finding suitable argument filterings (Section 4.3).
- Polynomial interpretations with negative coefficients and a corresponding usable rule criterion (Chapter 5).
- The termination tool T$_\mathsf{T}$T (Section 6.1).

Except for the usable rule criterion for polynomial interpretations with negative coefficients described in Section 5.2.1, all results have been published in various papers ([30, 31, 32, 33]).

Chapter 2

Preliminaries

Term rewrite systems are defined from two simple notions, terms and relations, and termination is characterized by orders. After recalling these three notions in Section 2.1, we introduce term rewrite systems and termination in Section 2.2.

In this thesis $\mathbb{N}$ stands for the set of all natural numbers $\{0, 1, 2, \ldots\}$ and $\mathbb{Z}$ for the set of all integers $\{\ldots, -2, -1, 0, 1, 2, \ldots, \}$.

2.1 Relations, Orders, and Terms

A (binary) *relation* $\to$ on a set A is a subset of $A \times A$. We write $a \to b$ instead of $(a, b) \in \to$. For readability, we use the notational convention that the mirror image $\leftarrow$ of a binary relation symbol $\to$ stands for the inverse relation $\{(b, a) \mid (a, b) \in \to\}$.

Definition 2.1. *Let $\to$, $\to_1$ and $\to_2$ be relations on a set A. The composition $\to_1 \cdot \to_2$ of $\to_1$ and $\to_2$ is the following binary relation on A: $\{(x, z) \in A \times A \mid x \to_1 y$ and $y \to_2 z$ for some $y \in A\}$. Furthermore,*

- *the n-step relation $\to^n$ is defined for all $n \in \mathbb{N}$ as follows:*

$$\to^n = \begin{cases} \{(x, x) \mid x \in A\} & \text{if } n = 0 \\ \to \cdot \to^n & n > 0 \end{cases}$$

- *the* reflexive closure *$\to^=$ of $\to$ is defined as $\to \cup \to^0$,*
- *the* transitive closure *$\to^+$ of $\to$ is defined as $\bigcup_{n>0} \to^n$,*
- *the* reflexive and transitive closure *$\to^*$ of $\to$ is $\bigcup_{n \geqslant 0} \to^n$.*

Definition 2.2. *A binary relation $\to$ on A is* well-founded *if there is no infinite sequence $a_1 \to a_2 \to \cdots$ of elements in A.*

Definition 2.3. *A relation is a* strict order *if it is irreflexive and transitive. A relation is a* preorder *if it is reflexive and transitive. A relation is a* partial order *if it is reflexive, transitive, and anti-symmetric.*

Definition 2.4. *Let $>$ be a strict order on A.*

- $>$ *is a* total order *if* $a > b$, $b > a$, *or* $a = b$ *holds for all* $a, b \in A$.
- $>$ *is a* well-founded order *if it is well-founded.*
- $>$ *is a* well-order *if it is total and well-founded.*

In the following definitions we introduce terms.

Definition 2.5. *A* signature $\mathcal{F}$ *is a set of* function symbols*, where every function symbol* f *is associated with a non-negative number* n*, the* arity *of* f*. We write* $f^{(n)}$ *when we explicitly describe that the arity of* f *is* n*. Function symbols of arity* 0 *are called* constants*.*

Definition 2.6. *Let* $\mathcal{F}$ *be a signature and* $\mathcal{V}$ *a set of countably infinite* variables *with* $\mathcal{F} \cap \mathcal{V} = \varnothing$*. The set* $\mathcal{T}(\mathcal{F}, \mathcal{V})$ *of all* terms *over* $\mathcal{F}$ *is the smallest set such that*

- *if* $x \in \mathcal{V}$ *then* $x \in \mathcal{T}(\mathcal{F}, \mathcal{V})$*, and*
- *if* $f^{(n)} \in \mathcal{F}$ *and* $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ *then* $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$*.*

We sometimes write simply $\mathcal{T}$ *for* $\mathcal{T}(\mathcal{F}, \mathcal{V})$*.*

We use sans-serif font for representing concrete function symbols and *italics* for variables. Binary function symbols are often written in infix notation.

In the following various basic operations on terms are defined.

Definition 2.7. *Let* t *be a term.* $\mathcal{V}\mathsf{ar}(t)$ *denotes the set of variables occurring in* t*, i.e.,*

$$\mathcal{V}\mathsf{ar}(t) = \begin{cases} \{t\} & \text{if } t \text{ is a variable} \\ \bigcup_{i=0}^{n} \mathcal{V}\mathsf{ar}(t_i) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

$\mathcal{F}\mathsf{un}(t)$ *denotes the set of function symbols occurring in* t*, i.e.,*

$$\mathcal{F}\mathsf{un}(t) = \begin{cases} \varnothing & \text{if } t \text{ is a variable} \\ \{f\} \cup \bigcup_{i=0}^{n} \mathcal{F}\mathsf{un}(t_i) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

The set $\mathcal{P}\mathsf{os}(t)$ *of* positions *of* t *is defined as a set of finite sequences of non-negative integers as follows:*

$$\mathcal{P}\mathsf{os}(t) = \begin{cases} \{\epsilon\} & \text{if } t \text{ is a variable} \\ \{\epsilon\} \cup \{iq \mid 1 \leqslant i \leqslant n, p \in \mathcal{P}\mathsf{os}(t_i)\} & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

where ϵ *denotes the empty string. The position* ϵ *is called the* root *position. The* root symbol $\mathrm{root}(t)$ *of* t *is defined as follows:*

$$\mathrm{root}(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ f & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Definition 2.8. *Let t be a term and $p \in \mathcal{P}\mathsf{os}(t)$. The* subterm $t|_p$ *of t at p is defined as follows:*

$$t|_p = \begin{cases} t & \text{if } p = \epsilon \\ t_i|_q & \text{if } p = iq \text{ and } t = f(t_1, \ldots, t_n) \end{cases}$$

$t[s]_p$ denotes the term that is obtained from t by replacing the subterm at p by s, i.e.,

$$t[s]_p = \begin{cases} s & \text{if } p = \epsilon \\ f(t_1, \ldots, t_i[s]_q, \ldots, t_n) & \text{if } p = iq \text{ and } t = f(t_1, \ldots, t_n) \end{cases}$$

We say that s is a subterm *of t (and t is a* superterm *of s) if there is a position $p \in \mathcal{P}\mathsf{os}(t)$ such that $s = t|_p$.*

Definition 2.9. *A term t is* ground *if $\mathcal{V}\mathsf{ar}(t) = \varnothing$. A term t is* linear *if $\{p \in \mathcal{P}\mathsf{os}(t) \mid t|_p = x\}$ is a singleton set for all $x \in \mathcal{V}\mathsf{ar}(t)$. In other words, no variable occurs more than once in t.*

Definition 2.10. *The* superterm relation $\trianglerighteq$ *is defined on terms as follows: $s \trianglerighteq t$ if s is a superterm of t. Its strict part is denoted by $\triangleright$.*

Contexts and substitutions are underlying notions for rewriting terms.

Definition 2.11. *Let $\mathcal{F}$ be a signature and let $\Box$ be a special constant that is called the* hole. *A* context *over $\mathcal{F}$ is a term C over $\mathcal{F} \uplus \{\Box^{(0)}\}$ such that C contains exactly one hole $\Box$. The application $C[t]$ of a context C and a term t is defined as follows:*

$$C[t] = \begin{cases} t & \text{if } C = \Box \\ f(s_1, \ldots, C'[t], \ldots, s_n) & \text{if } C = f(s_1, \ldots, C', \ldots, s_n) \end{cases}$$

where C' is a context. An n-hole content C is a term with n occurrences of the hole. For an n-hole context C and terms $t_1, \ldots, t_n$, we write $C[t_1, \ldots, t_n]$ for the term that is obtained by replacing the holes in C from left to right by $t_1, \ldots, t_n$.

Definition 2.12. *Let $\mathcal{F}$ be a signature and $\mathcal{V}$ a countably infinite set of variables with $\mathcal{F} \cap \mathcal{V} = \varnothing$. A* substitution *$\sigma$ over $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is a function from $\mathcal{V}$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ such that the* domain $\mathrm{dom}(\sigma)$ *of σ is finite, where*

$$\mathrm{dom}(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$$

A substitution σ can be extended to a function $\hat{\sigma}$ from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ as follows:

$$\hat{\sigma}(t) = \begin{cases} \sigma(t) & \text{if } t \text{ is a variable} \\ f(\hat{\sigma}(t_1), \ldots, \hat{\sigma}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

We simply write $t\sigma$ for $\hat{\sigma}(t)$. The composition $\sigma\tau$ of two substations σ and τ is defined as $(\sigma\tau)(x) = (x\sigma)\tau$.

Unifiers are substitutions that make two terms syntactically identical.

Definition 2.13. *A* unifier μ *of* s *and* t *is a substitution that* $s\mu = t\mu$*. A unifier* μ *of* s *and* t *is called a* most general unifier *(mgu) if for every unifier* μ' *of* s *and* t *there is some substitution* σ *such that* $\mu\sigma = \mu'$*. We say that* s *and* t *are* unifiable *if there exists a unifier of* s *and* t*.*

It is well-known that unifiable terms admit an mgu. Moreover, an almost-linear[1] time algorithm is known for computing mgu (cf. [6]).

2.2 Term Rewrite Systems and Termination

First we consider relations on terms.

Definition 2.14. *Let* $\to$ *be a relation on terms. The relation is* closed under contexts *if* $C[s] \to C[t]$ *holds for all terms* s *and* t *with* $s \to t$ *and all contexts* C*. The relation is* closed under substitutions *if* $s\sigma \to t\sigma$ *holds for all terms* s *and* t *with* $s \to t$ *and every substitutions* σ*.*

Definition 2.15. *A relation on terms is a* rewrite relation *if it is closed under contexts and substitutions. A strict order on terms is a* rewrite order *if it is closed under contexts and substitutions. A well-founded rewrite order is a* reduction order*.*

Next we define term rewrite systems and term rewriting.

Definition 2.16. *A pair* (l, r) *of terms is a* rewrite rule *if* l *is not a variable and* $\mathcal{V}\mathsf{ar}(l) \supseteq \mathcal{V}\mathsf{ar}(r)$*. We write* $l \to r$ *instead of* (l, r)*. A* term rewrite system *(*TRS*) is a set of rewrite rules. The* rewrite relation $\to_{\mathcal{R}}$ of *a TRS* $\mathcal{R}$ *is defined on terms as the smallest rewrite relation that contains* $\mathcal{R}$*. A symbol* f *is a* defined symbol *of* $\mathcal{R}$ *if* $f = \mathrm{root}(l)$ *for some* $l \to r \in \mathcal{R}$*.*

Remark that $s \to_{\mathcal{R}} t$ if and only if there exist a rewrite rule $l \to r \in \mathcal{R}$, a context C, and a substitution σ such that $t = C[l\sigma]$ and $u = C[r\sigma]$.

Definition 2.17. *Let* $\mathcal{R}$ *be a TRS. A (possibly infinite) sequence of terms such that* $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$ *is called a* rewrite sequence *of* $\mathcal{R}$*. A term* t *is a* normal form *of* $\mathcal{R}$ *if there is no term* u *such that* $t \to_{\mathcal{R}} u$*. We write* $s \to^{!}_{\mathcal{R}} t$ *if* $s \to^{*}_{\mathcal{R}} t$ *and* t *is a normal form.*

Definition 2.18. *A TRS* $\mathcal{R}$ *is* terminating *if* $\to_{\mathcal{R}}$ *is well-founded.*

Next we introduce the notion of innermost rewriting.

Definition 2.19. *The* innermost rewrite relation $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ *of a TRS* $\mathcal{R}$ *is defined on terms as follows:* $t \xrightarrow{\mathsf{i}}_{\mathcal{R}} u$ *if and only if there exist a rewrite rule* $l \to r \in \mathcal{R}$*, a context* C*, and a substitution* σ *such that* $t = C[l\sigma]$*,* $u = C[r\sigma]$*, and all proper subterms of* $l\sigma$ *are normal forms of* $\mathcal{R}$*.*

Remark that $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is a subset of $\to_{\mathcal{R}}$. Note that $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ is not a rewrite relation in the sense of Definition 2.15 as it is not closed under substitutions.

Definition 2.20. *A TRS* $\mathcal{R}$ *is* innermost terminating *if* $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ *is well-founded.*

[1] Almost-linear means $O(nG(n))$, where $G(n)$ is an extremely slowly growing function with the following properties: $G(1) = 0$ and $G(2^m) = G(m) + 1$ for all $m > 0$.

Termination implies innermost termination but not vice-versa. However, for some class of TRSs innermost termination implies termination.

Definition 2.21. *Let $l \to r$ and $l' \to r'$ be rewrite rules with $\mathcal{V}\mathsf{ar}(l) \cap \mathcal{V}\mathsf{ar}(l') = \varnothing$. We say that $l \to r$ is* overlapping *with $l' \to r'$ if there exists a position $p \in \mathcal{P}\mathsf{os}(l)$ such that*

- *$l|_p$ is not a variable,*
- *$p \neq \epsilon$ if $l \to r$ and $l' \to r'$ are renamings of the same rewrite rules, and*
- *$l|_p$ and l' are unifiable.*

A TRS $\mathcal{R}$ is non-overlapping *if there are no rewrite rules $l \to r$ and $l' \to r'$ in $\mathcal{R}$ such that $l \to r$ is overlapping with $l' \to r'$ where the variables of $l' \to r'$ are possibly renamed in order to satisfy the condition $\mathcal{V}\mathsf{ar}(l) \cap \mathcal{V}\mathsf{ar}(l') = \varnothing$.*

Theorem 2.22 ([28]). *If a TRS $\mathcal{R}$ is non-overlapping and innermost terminating then $\mathcal{R}$ is terminating.* □

The next definition introduces syntactical properties.

Definition 2.23. *A rewrite rule $l \to r$ is* collapsing *if r is a variable. A rewrite rule $l \to r$ is* duplicating *if some variable occurs more often in r than in l. A TRS is called* collapsing *(*duplicating*) if it contains a collapsing (duplicating) rewrite rule.*

The following theorem provides an early characterization of termination.

Theorem 2.24. *A TRS $\mathcal{R}$ over a finite signature is terminating if and only if there exists a reduction order $>$ such that $\mathcal{R} \subseteq >$.* □

We introduce classes of reduction orders: multiset path orders, lexicographic path orders, Knuth-Bendix orders. These are instances of simplification orders, and parameterized by a *(strict) precedence*, which is a strict order on the underlying signature.

Definition 2.25. *A rewrite order $>$ is a* simplification order *if $\rhd \subseteq >$. A rewrite preorder $\gtrsim$ is a* quasi-simplification order *if $\trianglerighteq \subseteq \gtrsim$.*

Obviously, the reflexive closure of a simplification order is a quasi-simplification order.

Theorem 2.26 ([12]). *Let $\mathcal{F}$ be a finite signature and $\mathcal{V}$ a countably infinite set of variables. Every simplification order on $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is a reduction order.* □

We introduce the multiset path order based on strict precedence. As underlying notions of it multisets are used.

Definition 2.27. *Let A be a set. A* multiset *M on A is a function from A to $\mathbb{N}$. The set of all finite multisets on A is denoted by $\mathcal{M}(A)$. Let $a_1, \ldots, a_n$ be elements in A. We write $\{\{a_1, \ldots, a_n\}\}$ for the multiset M defined as follows: $M(a)$ is the size of the set $\{i \mid 1 \leqslant i \leqslant n, a_i = a\}$. Let M and N be multisets on A. We define the following operations:*

- *$x \in M$ if $M(x) > 0$,*

- $M \subseteq N$ *if* $M(x) \leqslant N(x)$ *for all* $x \in A$,
- $M \cup N$ *for the function* $x \mapsto M(x) + N(x)$, *and*
- $M \setminus N$ *for the function* $x \mapsto \max\{0, M(x) - N(x)\}$

Definition 2.28. *Let* $>$ *be an order on a set* A. *The* multiset extension $>^{\mathrm{mul}}$ *of* $>$ *is defined on* $\mathcal{M}(A)$ *as follows:* $M >^{\mathrm{mul}} N$ *if and only if there exist* $X, Y \in \mathcal{M}(A)$ *such that* $\{\{\}\} \neq X \subseteq M$, $N = (M \setminus X) \cup Y$, *and for every* $y \in Y$ *there exists an* $x \in X$ *such that* $x > y$.

Definition 2.29. *Let* $>$ *be a strict precedence. The* multiset path order *(*MPO*)* $>_{\mathrm{mpo}}$ *is defined on terms as follows:* $s >_{\mathrm{mpo}} t$ *if and only if* s *is of the form* $f(s_1, \ldots, s_m)$ *and one of the following statements holds:*

- $s_i \geqslant_{\mathrm{mpo}} t$ *for some* $i \in \{1, \ldots, m\}$, *or*
- t *is of the form* $g(t_1, \ldots, t_n)$, $s >_{\mathrm{mpo}} t_i$ *for all* $1 \leqslant i \leqslant n$, *and*
  - $f > g$, *or*
  - $f = g$ *and* $\{\{s_1, \ldots, s_m\}\} >^{\mathrm{mul}}_{\mathrm{mpo}} \{\{t_1, \ldots, t_m\}\}$.

Here $\geqslant_{\mathrm{mpo}}$ *denotes the reflexive closure of* $>_{\mathrm{mpo}}$.

Theorem 2.30 ([36, 37])**.** *Let* $\mathcal{F}$ *be a finite signature,* $\mathcal{V}$ *a countably infinite set of variables, and* $>$ *a precedence on* $\mathcal{F}$. *Then* $>_{\mathrm{mpo}}$ *is a simplification order on* $\mathcal{T}(\mathcal{F}, \mathcal{V})$. □

By modifying the multiset comparison in the definition of MPO, lexicographic path orders are obtained.

Definition 2.31. *Let* $>$ *be a strict precedence. The* lexicographic path order *(*LPO*)* $>_{\mathrm{lpo}}$ *is defined on terms as follows:* $s >_{\mathrm{lpo}} t$ *if and only if* s *is of the form* $f(s_1, \ldots, s_m)$ *and one of the following statements holds:*

- $s_i \geqslant_{\mathrm{lpo}} t$ *for some* $i \in \{1, \ldots, m\}$,
- t *is of the form* $g(t_1, \ldots, t_n)$, $s >_{\mathrm{lpo}} t_i$ *for all* $i \in \{1, \ldots, n\}$, *and*
  - $f > g$, *or*
  - $f = g$, *and there exists* $j \in \{1, \ldots, n\}$ *such that* $s_1 = t_1, \ldots, s_{j-1} = t_{j-1}$, *and* $s_j >_{\mathrm{lpo}} t_j$.

Theorem 2.32 ([36, 37])**.** *Let* $\mathcal{F}$ *be a finite signature,* $\mathcal{V}$ *a countably infinite set of variables, and* $>$ *a precedence on* $\mathcal{F}$. *Then* $>_{\mathrm{lpo}}$ *is a simplification order on* $\mathcal{T}(\mathcal{F}, \mathcal{V})$. □

Knuth-Bendix orders are based on precedence and a weight function.

Definition 2.33. *Let* $\mathcal{F}$ *be a finite signature,* $\mathcal{V}$ *a countably infinite set of variables, and* $>$ *a precedence on* $\mathcal{F}$. *A* weight function $\mathbf{w} = (w, w_0)$ *is a pair consisting of a function* $w : \mathcal{F} \to \mathbb{N}$ *and a positive integer* w_0 *such that* $w(c) \geqslant w_0$ *for all constants* c, *and* f *is the greatest element in* $\mathcal{F}$ *with respect to* $>$ *for all*

unary function symbols f with $w(f) = 0$. Let $\mathbf{w}$ *be a weight function. We define the extended weight function* $\hat{\mathbf{w}} : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathbb{N}$ *as follows:*

$$\hat{\mathbf{w}}(t) = \begin{cases} w_0 & \text{if } x \in \mathcal{V} \\ w(f) + \hat{\mathbf{w}}(t_1) + \cdots + \hat{\mathbf{w}}(t_n) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

The Knuth-Bendix order *(KBO)* $>_{\mathrm{kbo}}$ *induced by a precedence* $>$ *and a weight function* $\mathbf{w}$ *is defined on* $\mathcal{T}(\mathcal{F}, \mathcal{V})$ *as follows:* $s >_{\mathrm{kbo}} t$ *if and only if* s *is of the form* $f(s_1, \ldots, s_m)$*, the rewrite rule* $s \to t$ *is non-duplicating, and one of the following statements holds:*

- $\hat{\mathbf{w}}(s) > \hat{\mathbf{w}}(t)$,
- $\hat{\mathbf{w}}(s) = \hat{\mathbf{w}}(t)$, t *is of the form* $g(s_1, \ldots, s_n)$*, and* $f > g$,
- $\hat{\mathbf{w}}(s) = \hat{\mathbf{w}}(t)$*, and there exist a unary function symbol* f*, a variable* x*, and an integer* $k > 0$ *such that* $s = f^k(x)$ *and* $t = x$,
- t *is of the form* $f(t_1, \ldots, t_m)$ *and there exists* $j \in \{1, \ldots, n\}$ *such that* $s_1 = t_1, \ldots, s_{j-1} = t_{j-1}$*, and* $s_j >_{\mathrm{kbo}} t_j$.

Theorem 2.34 ([37])**.** *Let* $\mathcal{F}$ *be a finite signature,* $\mathcal{V}$ *a countably infinite set of variables,* $>$ *a precedence on* $\mathcal{F}$*, and* $\mathbf{w}$ *be a weight function. Then* $>_{\mathrm{kbo}}$ *is a simplification order on* $\mathcal{T}(\mathcal{F}, \mathcal{V})$. □

In the literatures one also finds weight functions with positive real numbers. However, Korovin and Voronkov [40] showed that this does not add any power to KBO, i.e., for every finite TRS compatible with KBO based on a weight function on real numbers, one can effectively construct a weight function on natural numbers.

Here we give examples. The rewrite rule $\mathsf{f}(x) \to \mathsf{g}(x, x)$ is oriented by both MPO and LPO with precedence $\mathsf{f} > \mathsf{g}$. But there is no weight function and precedence for KBO that orients this rule. In general any instance of KBO cannot orient duplicating rules. The TRS consisting of two rules $\mathsf{f}(\mathsf{f}(x)) \to \mathsf{g}(\mathsf{f}(x))$ and $\mathsf{g}(\mathsf{g}(x)) \to \mathsf{f}(x)$ is oriented by KBO based on empty precedence and the weight function with $w_0 = 1$, $w(\mathsf{g}) = 2$ and $w(\mathsf{f}) = 3$. But neither MPO nor LPO can orient it.

A preorder on a signature is called a *quasi-precedence*. It is well-known that the preceding orders can be strengthened by adopting quasi-precedences.

Definition 2.35. *Let* $\gtrsim$ *be a quasi-precedence. We write* $\approx$ *for* $\gtrsim \cap \lesssim$*, and* $>$ *for* $\gtrsim \setminus \lesssim$*. Moreover, we extend the binary relation* $\approx$ *to terms as follows:* $s \approx t$ *if one of the following condition holds:*

- s *and* t *are the same variable, or*
- s *is of form* $f(s_1, \ldots, s_m)$*,* t *is of* $g(t_1, \ldots, t_n)$ *such that* $f \approx g$*,* $m = n$*, and* $s_1 \approx t_1, \ldots, s_m \approx t_m$.

Here we only introduce quasi-precedence version of lexicographic path orders.

Definition 2.36. *Let* $\gtrsim$ *be a quasi-precedence. The lexicographic path orders* $>_{\mathrm{lpo}}$ *and* $\gtrsim_{\mathrm{lpo}}$ *with* quasi-precedence $\gtrsim$ *are defined on terms as follows:* $s \gtrsim_{\mathrm{lpo}} t$ *if and only if* $s >_{\mathrm{lpo}} t$ *or* $s \approx_{\mathrm{lpo}} t$*, and* $s >_{\mathrm{lpo}} t$ *if and only if* s *is of the form* $f(s_1, \ldots, s_m)$ *and one of the following statements holds:*

- $s_i \gtrsim_{\text{lpo}} t$ *for some* $i \in \{1, \ldots, m\}$,
- t *is of the form* $g(t_1, \ldots, t_n)$, $s \gtrsim_{\text{lpo}} t_i$ *for all* $i \in \{1, \ldots, n\}$, *and*
  - $f > g$, *or*
  - $f \approx g$, *and there exists* $j \in \{1, \ldots, n\}$ *such that* $s_1 \approx_{\text{lpo}} t_1, \ldots, s_{j-1} \approx_{\text{lpo}} t_{j-1}$, *and* $s_j >_{\text{lpo}} t_j$.
  - $f \approx g$, $n > m$, *and* $s_1 \approx_{\text{lpo}} t_1, \ldots, s_m \approx t_m$.

Theorem 2.37. *Let* $\mathcal{F}$ *be a finite signature,* $\mathcal{V}$ *a countably infinite set of variables,* $\gtrsim$ *a quasi-precedence on* $\mathcal{F}$. *Then* $>_{\text{lpo}}$ *is a simplification order on* $\mathcal{T}(\mathcal{F}, \mathcal{V})$. □

For example, the TRS consisting of the rules $\mathsf{f}(\mathsf{f}(x)) \to \mathsf{g}(x)$ and $\mathsf{g}(\mathsf{g}(x)) \to \mathsf{f}(x)$ is oriented by LPO with quasi-precedence $\mathsf{f} \approx \mathsf{g}$, but cannot be oriented by any LPO based on a strict precedence.

Polynomial interpretation orders are based on the concept of algebra.

Definition 2.38. *Let* $\mathcal{F}$ *be a signature and* $\mathcal{V}$ *a set of variables. An* $\mathcal{F}$-algebra $\mathcal{A}$ *is a pair consisting of a set* A, *called the* carrier, *and a family of functions* $\{f_{\mathcal{A}}\}_{f \in \mathcal{F}}$, *called* interpretations. *Here* $f_{\mathcal{A}}$ *is a function from* A^n *to* A *where* n *is the arity of* f. *Let* $\mathcal{A} = (A, \{f_{\mathcal{A}}\}_{f \in \mathcal{F}})$ *be an* $\mathcal{F}$*-algebra. An* assignment *for* $\mathcal{A}$ *is a function from* $\mathcal{V}$ *to* A. *The* evaluation $[\alpha]_{\mathcal{A}}(t)$ *of a term* t *under an assignment* α *is inductively defined as follows:*

$$[\alpha]_{\mathcal{A}}(t) = \begin{cases} \alpha(t) & \text{if } t \in \mathcal{V} \\ f_{\mathcal{A}}([\alpha]_{\mathcal{A}}(t_1), \ldots, [\alpha]_{\mathcal{A}}(t_n)) & \text{if } t = f(t_1, \ldots, t_n) \end{cases}$$

Let $>$ *be an order on* A. *The relations* $\geqslant_{\mathcal{A}}$ *and* $>_{\mathcal{A}}$ *are defined on terms as follows:*

- $s =_{\mathcal{A}} t$ *if* $[\alpha]_{\mathcal{A}}(s) = [\alpha]_{\mathcal{A}}(t)$ *for all assignments* $\alpha : \mathcal{V} \to A$,
- $s \geqslant_{\mathcal{A}} t$ *if* $[\alpha]_{\mathcal{A}}(s) \geqslant [\alpha]_{\mathcal{A}}(t)$ *for all assignments* $\alpha : \mathcal{V} \to A$,
- $s >_{\mathcal{A}} t$ *if* $[\alpha]_{\mathcal{A}}(s) > [\alpha]_{\mathcal{A}}(t)$ *for all assignments* $\alpha : \mathcal{V} \to A$.

Here $\geqslant$ *is the reflexive closure of* $>$. *A function* ϕ *from* A^n *to* A *is said to be* weakly monotone *with respect to* $>$ *if* $\phi(a_1, \ldots, a_n) \geqslant \phi(b_1, \ldots, b_n)$ *whenever* $a_i \geqslant b_i$ *for all* $1 \leqslant i \leqslant n$. *We call* ϕ strictly monotone *with respect to* $>$ *if for every* $a_1, \ldots, a_n, b \in A$ *and* $1 \leqslant i \leqslant n$,

$$\phi(a_1, \ldots, a_i, \ldots, a_n) > \phi(a_1, \ldots, b, \ldots, a_n)$$

whenever $a_i > b$. *A pair* $(\mathcal{A}, >)$ *is a* weakly monotone algebra *if* $f_{\mathcal{A}}$ *is weakly monotone with respect to* $>$ *for all function symbols* $f \in \mathcal{F}$. *Furthermore,* $(\mathcal{A}, >)$ *is a* well-founded algebra *if* $>$ *is a well-founded order on* A,

Theorem 2.39 ([41, 59]). *Let* $(\mathcal{A}, >)$ *be a well-founded strictly monotone algebra. Then* $>_{\mathcal{A}}$ *is a simplification order.*

Consider the TRS $\mathcal{R}$ of Example 1.1 and the following interpretations over the natural numbers:

$$+_{\mathbb{N}}(x,y) = 2x + y \qquad \mathsf{s}_{\mathbb{N}}(x) = x + 1 \qquad \mathsf{0}_{\mathbb{N}} = 1$$

In order to show $\mathsf{0} + y >_{\mathcal{A}} y$ it is sufficient to show

$$\begin{array}{lcllcl} [\alpha]_{\mathcal{A}}(\mathsf{0}+y) & = & \alpha(y)+2 & > & \alpha(y) & = & [\alpha]_{\mathcal{A}}(y) \\ [\alpha]_{\mathcal{A}}(\mathsf{s}(x)+y) & = & 2\alpha(x)+\alpha(y)+2 & > & 2\alpha(x)+\alpha(y)+1 & = & [\alpha]_{\mathcal{A}}(\mathsf{s}(x+y)) \end{array}$$

for all assignments $\alpha : \mathcal{V} \to \mathbb{N}$. Indeed, these inequalities hold and therefore $\mathcal{R}$ is terminating. When automating this method, polynomials are often used. For polynomials P and Q over polynomial variables $x_1, \ldots, x_n$, we write $P(x_1, \ldots, x_n) > Q(x_1, \ldots, x_n)$ if $P(a_1, \ldots, a_n) > Q(a_1, \ldots, a_n)$ holds for all $a_1, \ldots, a_n \in \mathbb{N}$. The above inequalities are rephrased as the following polynomial inequalities:

$$\begin{array}{rcl} y+2 & > & y \\ 2x+y+2 & > & 2x+y+1 \end{array}$$

In order to check whether the inequality $P > Q$ hold, it is sufficient to test the positiveness of $P - Q$. Clearly, $(y+2)-y = 1$ and $(2x+y+2)-(2x+y+1) = 1$ are positive.

Techniques for finding appropriate polynomials as well as approximating (in general undecidable) polynomial inequalities $P > 0$ are described in several papers (e.g. [9, 11, 20, 34, 49]).

Chapter 3

Dependency Pair Method

In this chapter we present two new termination criteria that are based on the dependency pair method of Arts and Giesl [4]: the subterm criterion in Section 3.2 and the usable rule criterion in Section 3.4. We first recall the basics of the dependency pair method in Section 3.1.

3.1 Dependency Pairs

We use the following TRS from Dershowitz [13] to illustrate the developments in Sections 3.1–3.4:

$$
\begin{array}{lll}
1\colon & \neg\neg x \to x & 4\colon \quad x \wedge (y \vee z) \to (x \wedge y) \vee (x \wedge z) \\
2\colon & \neg(x \vee y) \to \neg x \wedge \neg y & 5\colon \quad (y \vee z) \wedge x \to (x \wedge y) \vee (x \wedge z) \\
3\colon & \neg(x \wedge y) \to \neg x \vee \neg y &
\end{array}
$$

Termination of this TRS is easily shown by the multiset path order. This, however, does not mean that automatic termination tools easily find a termination proof.[1]

Let us start with some easy observations. If a TRS $\mathcal{R}$ is not terminating then there must be a *minimal* non-terminating term, minimal in the sense that all its proper subterms are terminating. Let us denote the set of all minimal non-terminating terms by $\mathcal{T}_\infty$.

Lemma 3.1. *For every term $t \in \mathcal{T}_\infty$ there exists a rewrite rule $l \to r$, a substitution σ, and a non-variable subterm u of r such that $t \xrightarrow{>\epsilon}{}^* l\sigma \xrightarrow{\epsilon} r\sigma \trianglerighteq u\sigma$ and $u\sigma \in \mathcal{T}_\infty$.*

Proof. Let A be an infinite rewrite sequence starting at t. Since all proper subterms of t are terminating, A must contain a root rewrite step. By considering the first root rewrite step in A it follows that there exist a rewrite rule $l \to r$ and a substitution σ such that A starts with $t \xrightarrow{>\epsilon}{}^* l\sigma \xrightarrow{\epsilon} r\sigma$. Write $l = f(l_1, \ldots, l_n)$. Since the rewrite steps in $t \to^* l\sigma$ take place below the root, $t = f(t_1, \ldots, t_n)$ and $t_i \to^* l_i\sigma$ for all $1 \leqslant i \leqslant n$. By assumption the arguments $t_1, \ldots, t_n$ of t are terminating. Hence so are the terms $l_1\sigma, \ldots, l_n\sigma$. It follows that $\sigma(x)$ is terminating for every $x \in \mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l)$. As $r\sigma$ is non-terminating it has a subterm

[1] See the comments in Section 6.3.

$t' \in \mathcal{T}_\infty$. Because non-terminating terms cannot occur in the substitution part, there must be a non-variable subterm u of r such that $t' = u\sigma$. □

Observe that the term $l\sigma$ in Lemma 3.1 belongs to $\mathcal{T}_\infty$ as well. Further note that $u\sigma$ cannot be a proper subterm of $l\sigma$ (since all arguments of $l\sigma$ are terminating).

Corollary 3.2. *Every term in $\mathcal{T}_\infty$ has a defined root symbol.* □

If we were to define a new TRS $\mathcal{S}$ consisting of all rewrite rules $l \to u$ for which there exist a rewrite rule $l \to r \in \mathcal{R}$ and a subterm u of r with defined function symbol, then the sequence in the conclusion of Lemma 3.1 is of the form $\xrightarrow{>\epsilon}{}^*_{\mathcal{R}} \cdot \xrightarrow{\epsilon}_{\mathcal{S}}$. The idea is now to get rid of the position constraints by marking the root symbols of the terms in the rewrite rules of $\mathcal{S}$.

Definition 3.3. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. Let $\mathcal{F}^\sharp$ denote the union of $\mathcal{F}$ and $\{f^\sharp \mid f$ is a defined symbol of $\mathcal{R}\}$ where $f^\sharp$ is a fresh function symbol with the same arity as f. We sometimes write F instead of $f^\sharp$. We call these new symbols* dependency pair symbols *. Given a term $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with f a defined symbol, we write $t^\sharp$ for the term $f^\sharp(t_1, \ldots, t_n)$. If $l \to r \in \mathcal{R}$ and u is a subterm of r with defined root symbol such that u is not a proper subterm of l then the rewrite rule $l^\sharp \to u^\sharp$ is called a* dependency pair *of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted by $\mathsf{DP}(\mathcal{R})$.*

The idea of excluding dependency pairs $l^\sharp \to u^\sharp$ where u is a proper subterm of l is due to Dershowitz [14]. Although dependency pair symbols are defined symbols of $\mathsf{DP}(\mathcal{R})$, they are not defined symbols of the original TRS $\mathcal{R}$. In the following, defined symbols always refer to the original TRS $\mathcal{R}$.

Example 3.4. *The example at the beginning of this section admits the following nine dependency pairs:*

$$
\begin{array}{lll}
6\colon & \neg^\sharp(x \vee y) \to \neg x \wedge^\sharp \neg y & \\
7\colon & \neg^\sharp(x \vee y) \to \neg^\sharp x & 11\colon \quad x \wedge^\sharp (y \vee z) \to x \wedge^\sharp y \\
8\colon & \neg^\sharp(x \vee y) \to \neg^\sharp y & 12\colon \quad x \wedge^\sharp (y \vee z) \to x \wedge^\sharp z \\
9\colon & \neg^\sharp(x \wedge y) \to \neg^\sharp x & 13\colon \quad (y \vee z) \wedge^\sharp x \to x \wedge^\sharp y \\
10\colon & \neg^\sharp(x \wedge y) \to \neg^\sharp y & 14\colon \quad (y \vee z) \wedge^\sharp x \to x \wedge^\sharp z
\end{array}
$$

Lemma 3.5. *For every term $s \in \mathcal{T}_\infty$ there exist terms $t, u \in \mathcal{T}_\infty$ such that $s^\sharp \to^*_{\mathcal{R}} t^\sharp \to_{\mathsf{DP}(\mathcal{R})} u^\sharp$.*

Proof. Immediate from Lemma 3.1, Corollary 3.2, and the preceding definition. □

Definition 3.6. *For any subset $T \subseteq \mathcal{T}$ consisting of terms with a defined root symbol, we denote the set $\{t^\sharp \mid t \in T\}$ by $T^\sharp$.*

An immediate consequence of the previous lemma is that for every non-terminating TRS $\mathcal{R}$ there exists an infinite rewrite sequence of the form

$$t_1 \to^*_{\mathcal{R}} t_2 \to_{\mathsf{DP}(\mathcal{R})} t_3 \to^*_{\mathcal{R}} t_4 \to_{\mathsf{DP}(\mathcal{R})} \cdots$$

with $t_i \in \mathcal{T}^\sharp_\infty$ for all $i \geqslant 1$. Hence, to prove termination of a TRS $\mathcal{R}$ it is sufficient to show that $\mathcal{R} \cup \mathsf{DP}(\mathcal{R})$ does not admit such infinite sequences. For finite $\mathcal{R}$, every such sequence contains a tail in which all applied dependency pairs are used infinitely many times. The set of those dependency pairs forms a cycle in the dependency graph.[2] *From now on, we assume that all TRSs are finite.*

As a side remark, note that all terms in $\mathcal{T}^\sharp_\infty$ are terminating with respect to $\mathcal{R}$ but admit an infinite rewrite sequence with respect to $\mathcal{R} \cup \mathsf{DP}(\mathcal{R})$.

Definition 3.7. *The nodes of the* dependency graph $\mathsf{DG}(\mathcal{R})$ *are the dependency pairs of $\mathcal{R}$ and there is an arrow from $s \to t$ to $u \to v$ if and only if there exist substitutions σ and τ such that $t\sigma \to^*_\mathcal{R} u\tau$. A* cycle *is a nonempty subset $\mathcal{C}$ of dependency pairs of $\mathsf{DP}(\mathcal{R})$ if for every two (not necessarily distinct) pairs $s \to t$ and $u \to v$ in $\mathcal{C}$ there exists a nonempty path in $\mathcal{C}$ from $s \to t$ to $u \to v$.*

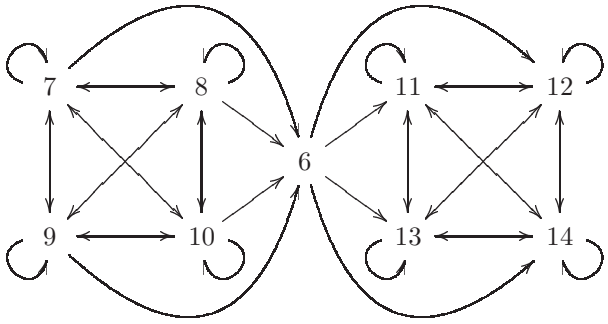
Definition 3.8. *Let $\mathcal{C} \subseteq \mathsf{DP}(\mathcal{R})$. An infinite rewrite sequence in $\mathcal{R} \cup \mathcal{C}$ of the form*

$$t_1 \to^*_\mathcal{R} t_2 \to_\mathcal{C} t_3 \to^*_\mathcal{R} t_4 \to_\mathcal{C} \cdots$$

with $t_1 \in \mathcal{T}^\sharp_\infty$ is called $\mathcal{C}$-minimal *if all rules in $\mathcal{C}$ are applied infinitely often.*

Hence proving termination boils down to proving the absence of $\mathcal{C}$-minimal rewrite sequences, for any cycle $\mathcal{C}$ in the dependency graph $\mathsf{DG}(\mathcal{R})$.

Example 3.9. *Our leading example has the following dependency graph:*



It contains 30 cycles: all nonempty subsets of $\{7, 8, 9, 10\}$ and $\{11, 12, 13, 14\}$.

Although the dependency graph is not computable in general, sound approximations exist that can be computed efficiently (see Section 4.1). Soundness here means that every cycle in the real dependency graph is a cycle in the approximated graph. For the example TRS all known approximations compute the real dependency graph.

How about innermost termination? If $s \xrightarrow{\epsilon} t$ is an innermost rewrite step then all subterms of s are normal forms. In other words $s^\sharp$ is normal form. This yields the next definition.

Definition 3.10. *Let $\mathcal{C} \subseteq \mathsf{DP}(\mathcal{R})$. An infinite rewrite sequence in $\mathcal{R} \cup \mathcal{C}$ of the form*

$$t_1 \xrightarrow{\mathsf{i}}{}^!_\mathcal{R} t_2 \to_\mathcal{C} t_3 \xrightarrow{\mathsf{i}}{}^!_\mathcal{R} t_4 \to_\mathcal{C} \cdots$$

with $t_1 \in \mathcal{T}_\infty$ is called a $\mathcal{C}$-minimal *innermost rewrite sequence if all rules in $\mathcal{C}$ are applied infinitely often.*

[2] But not every cycle in the dependency graph can be obtained in this way.

We have the following characterization of innermost termination: A TRS $\mathcal{R}$ is innermost terminating if and only if there is no $\mathcal{C}$-minimal innermost rewrite sequences, for any cycle $\mathcal{C}$ in the *innermost* dependency graph of $\mathcal{R}$. The innermost dependency graph of the introductory example is the same as the dependency graph illustrated in Example 3.9.

Definition 3.11. *The nodes of the* innermost dependency graph $\mathsf{IDG}(\mathcal{R})$ *are the dependency pairs of $\mathcal{R}$ and there is an arrow from $s \to t$ to $u \to v$ if and only if there exist substitutions σ and τ such that $s\sigma$ and $u\tau$ are normal forms, and $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}}^{*} u\tau$.*

In the remainder of the thesis we introduce methods which prove the absence of $\mathcal{C}$-minimal rewrite sequences. Since $\mathcal{C}$-minimal innermost rewrite sequences are $\mathcal{C}$-minimal rewrite sequences and $\mathsf{IDG}(\mathcal{R})$ is a subgraph of $\mathsf{DG}(\mathcal{R})$, these methods are also applicable for proving the absence of $\mathcal{C}$-minimal innermost rewrite sequences.

3.2 Subterm Criterion

We now present a new criterion which permits us to ignore certain cycles of the dependency graph.

Definition 3.12. *Let $\mathcal{R}$ be a TRS and $\mathcal{C} \subseteq \mathsf{DP}(\mathcal{R})$ such that every dependency pair symbol in $\mathcal{C}$ has positive arity. A* simple projection *for $\mathcal{C}$ is a mapping π that assigns to every n-ary dependency pair symbol $f^\sharp$ in $\mathcal{C}$ an argument position $i \in \{1, \ldots, n\}$. The mapping that assigns to every term $f^\sharp(t_1, \ldots, t_n) \in \mathcal{T}^\sharp$ with $f^\sharp$ a dependency pair symbol in $\mathcal{C}$ its argument at position $\pi(f^\sharp)$ is also denoted by π.*

Theorem 3.13. *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$. If there exists a simple projection π for $\mathcal{C}$ such that $\pi(\mathcal{C}) \subseteq \trianglerighteq$ and $\pi(\mathcal{C}) \cap \triangleright \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.*

Before presenting the proof, let us make some clarifying remarks about the notation. If R is a set of rewrite rules and O is a relation on terms then the expression $\pi(R)$ denotes the set $\{\pi(l) \to \pi(r) \mid l \to r \in R\}$, the inclusion $R \subseteq O$ abbreviates "$(l, r) \in O$ for all $l \to r \in R$", and the inequality $R \cap O \neq \varnothing$ abbreviates "$(l, r) \in R$ for at least one $l \to r \in O$". So the conditions state that after applying the simple projection π, every rule in $\mathcal{C}$ is turned into an identity or a rule whose right-hand side is a proper subterm of the left-hand side. Moreover, the latter case applies at least once.

Proof. Suppose to the contrary that there exists a $\mathcal{C}$-minimal rewrite sequence:

$$t_1 \to_{\mathcal{R}}^{*} u_1 \to_{\mathcal{C}} t_2 \to_{\mathcal{R}}^{*} u_2 \to_{\mathcal{C}} t_3 \to_{\mathcal{R}}^{*} \cdots \tag{3.1}$$

All terms in this sequence have a dependency pair symbol in $\mathcal{C}$ as root symbol. We apply the simple projection π to (3.1). Let $i \geqslant 1$.

- First consider the dependency pair step $u_i \to_{\mathcal{C}} t_{i+1}$. There exist a dependency pair $l \to r \in \mathcal{C}$ and a substitution σ such that $u_i = l\sigma$ and $t_{i+1} = r\sigma$. We have $\pi(u_i) = \pi(l)\sigma$ and $\pi(t_{i+1}) = \pi(r)\sigma$. We have $\pi(l) \trianglerighteq \pi(r)$ by assumption. So $\pi(l) = \pi(r)$ or $\pi(l) \triangleright \pi(r)$. In the former case we trivially

have $\pi(u_i) = \pi(t_{i+1})$. In the latter case the closure under substitutions of $\rhd$ yields $\pi(u_i) \rhd \pi(t_{i+1})$. Because of the assumption $\pi(\mathcal{C}) \cap \rhd \neq \varnothing$, the latter holds for infinitely many i.

- Next consider the rewrite sequence $t_i \to_{\mathcal{R}}^* u_i$. All steps in this sequence take place below the root and thus we obtain the (possibly shorter) sequence $\pi(t_i) \to_{\mathcal{R}}^* \pi(u_i)$.

So by applying the simple projection π, sequence (3.1) is transformed into an infinite $\to_{\mathcal{R}} \cup \rhd$ sequence containing infinitely many $\rhd$ steps, starting from the term $\pi(t_1)$. Since the relation $\rhd$ is well-founded, the infinite sequence must also contain infinitely many $\to_{\mathcal{R}}$ steps. By making repeated use of the well-known relational inclusion $\rhd \cdot \to_{\mathcal{R}} \subseteq \to_{\mathcal{R}} \cdot \rhd$ ($\rhd$ commutes over $\to_{\mathcal{R}}$ in the terminology of [7]), we obtain an infinite $\to_{\mathcal{R}}$ sequence starting from $\pi(t_1)$. In other words, the term $\pi(t_1)$ is non-terminating with respect to $\mathcal{R}$. Let $t_1 = f^\sharp(s_1, \ldots, s_n)$. Because $t_1 \in \mathcal{T}_\infty^\sharp$, $f(s_1, \ldots, s_n)$ is a minimal non-terminating term. Consequently, its argument $\pi(t_1) = s_{\pi(f^\sharp)}$ is terminating with respect to $\mathcal{R}$, providing the desired contradiction. □

The remarkable thing about the above theorem is that it permits us to discard cycles of the dependency graph without considering any rewrite rules. This is extremely useful. Moreover, the criterion is very simple to check.

Example 3.14. *The subterm criterion can handle 18 of the 30 cycles in the dependency graph of our leading example.*

- *Consider the cycle* $\mathcal{C} = \{7, 8, 9, 10\}$ *and all its subcycles. The only dependency pair symbol in* $\mathcal{C}$ *is* $\neg^\sharp$*. Since* $\neg^\sharp$ *is a unary function symbol, there is just one simple projection for* $\mathcal{C}$*:* $\pi(\neg^\sharp) = 1$*. By applying* π *to* $\mathcal{C}$*, we obtain*

$$\begin{array}{llll} 7\colon & x \vee y \to x & 9\colon & x \wedge y \to x \\ 8\colon & x \vee y \to y & 10\colon & x \wedge y \to y \end{array}$$

  We clearly have $\pi(\mathcal{C}) \subseteq \rhd$*. Hence we can handle* $\mathcal{C}$ *and all its subcycles.*

- *The cycle* $\mathcal{C} = \{11, 12\}$ *and all its subcycles are handled by the simple projection* $\pi(\wedge^\sharp) = 2$*:*

$$11, 12\colon \quad y \vee z \to x$$

The only cycles that are not handled by the criterion of Theorem 3.13 are the ones that involve 13 or 14; applying the simple projection $\pi(\wedge^\sharp) = 1$ *produces*

$$13, 14\colon \quad y \vee z \to x$$

whereas $\pi(\wedge^\sharp) = 2$ *gives*

$$13\colon \quad x \to y \qquad\qquad 14\colon \quad x \to z$$

None of these rules are compatible with $\trianglerighteq$*.*

The final two examples illustrate termination proofs by the subterm criterion. The last example furthermore shows that the subterm criterion is capable of proving the termination of TRSs that were considered to be challenging in the termination literature (cf. the remarks in [26, Example 9]).

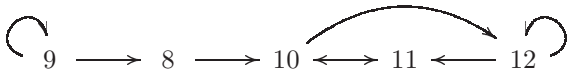
Example 3.15. *Consider the following TRS from [13]:*

$$\begin{aligned}
1\colon&\quad \mathsf{sort}([\,]) \to [\,]\\
2\colon&\quad \mathsf{sort}(x:y) \to \mathsf{insert}(x,\mathsf{sort}(y))\\
3\colon&\quad \mathsf{insert}(x,[\,]) \to x:[\,]\\
4\colon&\quad \mathsf{insert}(x,v:w) \to \mathsf{choose}(x,v:w,x,v)\\
5\colon&\quad \mathsf{choose}(x,v:w,y,\mathsf{0}) \to x:(v:w)\\
6\colon&\quad \mathsf{choose}(x,v:w,\mathsf{0},\mathsf{s}(z)) \to v:\mathsf{insert}(x,w)\\
7\colon&\quad \mathsf{choose}(x,v:w,\mathsf{s}(y),\mathsf{s}(z)) \to \mathsf{choose}(x,v:w,y,z)
\end{aligned}$$

There are 5 dependency pairs:

$$\begin{aligned}
8\colon&\quad \mathsf{sort}^\sharp(x:y) \to \mathsf{insert}^\sharp(x,\mathsf{sort}(y))\\
9\colon&\quad \mathsf{sort}^\sharp(x:y) \to \mathsf{sort}^\sharp(y)\\
10\colon&\quad \mathsf{insert}^\sharp(x,v:w) \to \mathsf{choose}^\sharp(x,v:w,x,v)\\
11\colon&\quad \mathsf{choose}^\sharp(x,v:w,\mathsf{0},\mathsf{s}(z)) \to \mathsf{insert}^\sharp(x,w)\\
12\colon&\quad \mathsf{choose}^\sharp(x,v:w,\mathsf{s}(y),\mathsf{s}(z)) \to \mathsf{choose}^\sharp(x,v:w,y,z)
\end{aligned}$$

The dependency graph



contains 4 cycles: $\{9\}$*,* $\{10,11\}$*,* $\{12\}$*, and* $\{10,11,12\}$*.*

- *The cycle* $\{9\}$ *is handled by the simple projection* $\pi(\mathsf{sort}^\sharp) = 1$*:*

$$9\colon\quad x:y \to y$$

- *The cycles* $\{10,11\}$ *and* $\{10,11,12\}$ *are handled by the simple projection* $\pi(\mathsf{insert}^\sharp) = \pi(\mathsf{choose}^\sharp) = 2$*:*

$$10,12\colon\quad v:w \to v:w \qquad 11\colon\quad v:w \to w$$

- *The cycle* $\{12\}$ *is handled by the simple projection* $\pi(\mathsf{choose}^\sharp) = 3$*:*

$$12\colon\quad s(y) \to y$$

Hence the TRS is terminating.

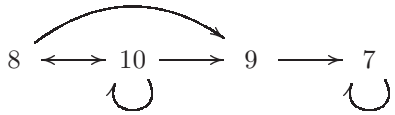
Example 3.16. *Consider the following TRS from [50]:*

$$\begin{aligned}
1\colon&\quad \mathsf{intlist}([\,]) \to [\,]\\
2\colon&\quad \mathsf{intlist}(x:y) \to \mathsf{s}(x):\mathsf{intlist}(y)\\
3\colon&\quad \mathsf{int}(\mathsf{0},\mathsf{0}) \to \mathsf{0}:[\,]\\
4\colon&\quad \mathsf{int}(\mathsf{0},\mathsf{s}(y)) \to \mathsf{0}:\mathsf{int}(\mathsf{s}(\mathsf{0}),\mathsf{s}(y))\\
5\colon&\quad \mathsf{int}(\mathsf{s}(x),\mathsf{0}) \to [\,]\\
6\colon&\quad \mathsf{int}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{intlist}(\mathsf{int}(x,y))
\end{aligned}$$

There are 4 dependency pairs:

$$\begin{aligned}
7\colon&\quad \mathsf{intlist}^\sharp(x:y) \to \mathsf{intlist}^\sharp(y)\\
8\colon&\quad \mathsf{int}^\sharp(0,\mathsf{s}(y)) \to \mathsf{int}^\sharp(\mathsf{s}(0),\mathsf{s}(y))\\
9\colon&\quad \mathsf{int}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{intlist}^\sharp(\mathsf{int}(x,y))\\
10\colon&\quad \mathsf{int}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{int}^\sharp(x,y)
\end{aligned}$$

The dependency graph



contains 3 cycles: $\{7\}$*,* $\{8,10\}$*, and* $\{10\}$*.*

- *The cycle* $\{7\}$ *is handled by the simple projection* $\pi(\mathsf{intlist}^\sharp) = 1$*:*

$$7\colon\quad x:y \to y$$

- *The cycles* $\{8,10\}$ *and* $\{10\}$ *are handled by the simple projection* $\pi(\mathsf{int}^\sharp) = 2$*:*

$$8\colon\quad \mathsf{s}(y) \to \mathsf{s}(y) \qquad\qquad 10\colon\quad \mathsf{s}(y) \to y$$

Hence the TRS is terminating.

An empirical evaluation of the subterm criterion can be found in Section 6.2.

3.3 Reduction Triples and Argument Filterings

What to do with cycles $\mathcal{C}$ of the dependency graph that cannot be handled by the criterion of the preceding section? In the dependency pair approach one uses a pair of orderings $(\gtrsim, >)$ that satisfy the properties stated below such that (1) all rules in $\mathcal{R}$ are oriented by $\gtrsim$, (2) all rules in $\mathcal{C}$ are oriented by $\gtrsim \cup >$, and (3) at least one rule in $\mathcal{C}$ is oriented by $>$. In order to anticipate the developments in Chapter 5 we use triples instead of pairs of orderings to generalize the relation $\gtrsim \cup >$ in (2).

Definition 3.17. *A* reduction triple $(\gtrsim, \geqslant, >)$ *consists of three relations that are closed under substitutions such that* $\gtrsim$ *and* $\geqslant$ *are preorders,* $\gtrsim$ *is closed under contexts,* $>$ *is a well-founded order, and the following compatibility condition holds: both* $\gtrsim \cdot > \subseteq >$ *and* $\geqslant \cdot > \subseteq >$ *or both* $> \cdot \gtrsim \subseteq >$ *and* $> \cdot \geqslant \subseteq >$*. We say that* $(\gtrsim, >)$ *is a reduction* pair *if* $(\gtrsim, \gtrsim \cup >, >)$ *is a reduction triple.*

Since we do not demand that $>$ is the strict part of the preorders $\gtrsim$ or $\geqslant$, the identities $\gtrsim \cdot > = >$ and $\geqslant \cdot > = >$ need not hold.

A typical example of a reduction triple is a combination of a reduction order and its reflexive closure, like $(>_{\mathrm{kbo}}^{=}, >_{\mathrm{kbo}}^{=}, >_{\mathrm{kbo}})$ (corresponding to the reduction pair $(>_{\mathrm{kbo}}^{=}, >_{\mathrm{kbo}})$). Both $>_{\mathrm{kbo}}^{=}$ and $>_{\mathrm{kbo}}$ are closed under contexts and the identity $>_{\mathrm{kbo}}^{=} \cdot >_{\mathrm{kbo}} = >_{\mathrm{kbo}}$ holds. For LPO based on a quasi-precedence $\gtrsim$ the triple $(\gtrsim_{\mathrm{lpo}}, \gtrsim_{\mathrm{lpo}}, >_{\mathrm{lpo}})$ is stronger than $(>_{\mathrm{lpo}}^{=}, >_{\mathrm{lpo}}^{=}, >_{\mathrm{lpo}})$.

Reduction triples based on weakly monotone algebras cover polynomial interpretations. Every algebra weakly monotone algebra $(\mathcal{A}, >)$ with $>$ well-founded gives rise to a reduction triple $(\gtrsim_{\mathcal{A}}, \geqslant_{\mathcal{A}}, >_{\mathcal{A}})$. In general, the relation $>_{\mathcal{A}}$ is not closed under contexts, $\geqslant_{\mathcal{A}}$ is not a partial order, and $>_{\mathcal{A}}$ is not the strict part of $\geqslant_{\mathcal{A}}$. Compatibility holds because of the identity $\geqslant \cdot > = >$ (on A).

In order for reduction triples like $(>^{=}_{\mathrm{lpo}}, >^{=}_{\mathrm{lpo}}, >_{\mathrm{lpo}})$ whose second and third components are closed under contexts to benefit from the fact that closure under contexts is not required, the conditions (1), (2), and (3) mentioned at the beginning of this section may be simplified by deleting certain (arguments of) function symbols occurring in $\mathcal{R}$ and $\mathcal{C}$ before testing orientability.

Definition 3.18. *An* argument filtering *for a signature $\mathcal{F}$ is a mapping π that assigns to every n-ary function symbol $f \in \mathcal{F}$ an argument position $i \in \{1, \ldots, n\}$ or a (possibly empty) list $[i_1, \ldots, i_m]$ of argument positions with $1 \leqslant i_1 < \cdots < i_m \leqslant n$. The signature $\mathcal{F}_\pi$ consists of all function symbols f such that $\pi(f)$ is some list $[i_1, \ldots, i_m]$, where in $\mathcal{F}_\pi$ the arity of f is m. Every argument filtering π induces a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}_\pi, \mathcal{V})$, also denoted by π:*

$$\pi(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ \pi(t_i) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } \pi(f) = i \\ f(\pi(t_{i_1}), \ldots, \pi(t_{i_m})) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } \pi(f) = [i_1, \ldots, i_m] \end{cases}$$

Note that the simple projections of the preceding section can be viewed as special argument filterings.

Example 3.19. *Applying the argument filtering π with $\pi(\wedge) = \pi(\vee) = [\,]$ and $\pi(\neg) = [1]$ to the rewrite rules of our leading example results in the following simplified rules:*

$$\begin{array}{llll} 1\colon & \neg\neg x \to x & 3\colon & \neg(\wedge) \to \vee \\ 2\colon & \neg(\vee) \to \wedge & 4,5\colon & \wedge \to \vee \end{array}$$

These rules are oriented from left to right by the lexicographic path order with precedence $\neg > \wedge > \vee$ (which does not imply termination of the original TRS).

We are now ready to state and prove the standard dependency pair approach to the treatment of cycles in the dependency graph.

Theorem 3.20 ([22]). *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$. If there exist an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ such that $\pi(\mathcal{R}) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, and $\pi(\mathcal{C}) \cap > \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.*

Proof. Suppose to the contrary that there exists a $\mathcal{C}$-minimal rewrite sequence:

$$t_1 \to^*_{\mathcal{R}} u_1 \to_{\mathcal{C}} t_2 \to^*_{\mathcal{R}} u_2 \to_{\mathcal{C}} t_3 \to^*_{\mathcal{R}} \cdots \tag{3.2}$$

We show that after applying the argument filtering π we obtain an infinite descending sequence with respect to the well-founded order $>$. Let $i \geqslant 1$.

- First consider the dependency pair step $u_i \to_{\mathcal{C}} t_{i+1}$. Since $u_i \in \mathcal{T}^\sharp$, the step takes place at the root positions and thus there exist a dependency

pair $l \to r \in \mathcal{C}$ and a substitution σ such that $u_i = l\sigma$ and $t_{i+1} = r\sigma$. Define the substitution σ_π as the composition of σ and π, i.e., $\sigma_\pi(x) = \pi(\sigma(x))$ for every variable x. A straightforward induction proof reveals that $\pi(t\sigma) = \pi(t)\sigma_\pi$ for every term t. Hence $\pi(u_i) = \pi(l)\sigma_\pi$ and $\pi(t_{i+1}) = \pi(r)\sigma_\pi$. From the assumption $\pi(\mathcal{C}) \subseteq \geqslant$ we infer $\pi(l) \geqslant \pi(r)$. Since $\geqslant$ is closed under substitutions, we have $\pi(u_i) \geqslant \pi(t_{i+1})$. As in the proof of Theorem 3.13, $\pi(u_i) > \pi(t_{i+1})$ holds for infinitely many i because of the assumption $\pi(\mathcal{C}) \cap > \neq \varnothing$.

- Next consider the rewrite sequence $t_i \to_{\mathcal{R}}^* u_i$. Using the assumption $\pi(\mathcal{R}) \subseteq \gtrsim$, we obtain $\pi(t_i) \gtrsim^* \pi(u_i)$ and thus $\pi(t_i) \gtrsim \pi(u_i)$ as in the preceding case.

So (3.2) is transformed into an infinite descending sequence consisting of $\gtrsim$, $\geqslant$, and $>$ steps, where there are an infinite number of the latter. Using the compatibility condition we obtain an infinite descending sequence with respect to $>$, providing the desired contradiction. □

The converse of Theorem 3.20 also holds. In fact, if there is no $\mathcal{C}$-minimal rewrite sequence of $\mathcal{R}$, the triple $(\gtrsim, \geqslant, >) = (\to_{\mathcal{R}}^*, \to_{\mathcal{C}}^*, \to_{\mathcal{C}}^+)$ is a reduction triple, and clearly satisfies $\pi(\mathcal{R}) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, and $\pi(\mathcal{C}) \cap > \neq \varnothing$. Here, π is the trivial argument filtering.

Example 3.21. *The argument filtering of Example 3.19 cannot be used to handle the cycle* $\{11, 12, 13, 14\}$ *in our leading example. This can be seen as follows. Because* $\pi(\vee) = [\,]$*, irrespective of the choice of* $\pi(\wedge^\sharp)$*, variables y and z will no longer appear in the left-hand sides of the simplified dependency pairs. Hence they cannot appear in the right-hand sides, and this is only possible if we take* 1*,* $[1]$*, or* $[\,]$ *for* $\pi(\wedge^\sharp)$*. The first two choices transform dependency pairs 13 and 14 into rules in which the variable x appears on the right-hand side but not on the left-hand side, whereas the third choice turns all dependency pairs into the identity* $\wedge^\sharp = \wedge^\sharp$*.*

Since the original TRS is compatible with the multiset path order, it is no surprise that the constraints of Theorem 3.20 for all cycles are satisfied by the full argument filtering π *(that maps every* n*-ary function symbol to* $[1, \ldots, n]$*) and the reduction triple* $(>_{\mathrm{mpo}}^{=}, >_{\mathrm{mpo}}^{=}, >_{\mathrm{mpo}})$ *with the precedence* $\neg > \wedge > \vee$*. However, it can be shown that there is no argument filtering* π *such that the resulting constraints are satisfied by a polynomial interpretation or the lexicographic path order.*

Observe that the proof of Theorem 3.20 does not use the fact that $\mathcal{C}$-minimal rewrite sequences start from terms in $\mathcal{T}_\infty^\sharp$. In the next section we show that by restoring the use of minimality, we can get rid of some of the constraints originating from $\mathcal{R}$.

We conclude this section with the remark that minimal innermost rewrite sequences cannot be characterized by reduction triples. As we mentioned in the last paragraph of Section 3.1, the following corollary for innermost rewriting is derived from Theorem 3.20.

Corollary 3.22. *Let* $\mathcal{R}$ *be a TRS and let* $\mathcal{C}$ *be a cycle in* $\mathsf{IDG}(\mathcal{R})$*. If there exist an argument filtering* π *and a reduction triple* $(\gtrsim, \geqslant, >)$ *such that* $\pi(\mathcal{R}) \subseteq \gtrsim$*,* $\pi(\mathcal{C}) \subseteq \geqslant$*, and* $\pi(\mathcal{C}) \cap > \neq \varnothing$ *then there are no* $\mathcal{C}$*-minimal innermost rewrite sequences.* □

Arts and Giesl [5] showed that the converse of Corollary 3.22 does not hold. The reason is that although we handle innermost steps in $\mathcal{C}$-minimal innermost rewrite sequence with a rewrite preorder $\gtrsim$, which is closed under substitutions, $\xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}}$ is *not*.

Example 3.23 ([5]). *Consider the TRS*

$$\begin{array}{lll} 1\colon & \mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(\mathsf{g}(\mathsf{h}(x))) & 3\colon \quad \mathsf{g}(\mathsf{h}(x)) \to \mathsf{g}(x) \\ 2\colon & \mathsf{h}(\mathsf{0}) \to \mathsf{a} & 4\colon \quad \mathsf{g}(\mathsf{s}(x)) \to \mathsf{s}(x) \\ & & 5\colon \quad \mathsf{g}(\mathsf{0}) \to \mathsf{s}(\mathsf{0}) \end{array}$$

$\mathsf{IDG}(\mathcal{R})$ *contains two cycles:* $\{\mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(x)))\}$ *and* $\{\mathsf{g}^\sharp(\mathsf{h}(x)) \to \mathsf{g}^\sharp(x)\}$. *For both cycles* $\mathcal{C}$ *there is no* $\mathcal{C}$*-minimal innermost rewrite sequence, and therefore the TRS is innermost terminating. The constraints of Corollary 3.22 for the first cycle contains*

$$\begin{array}{ll} \pi(\mathsf{g}(\mathsf{h}(x))) \gtrsim \pi(\mathsf{g}(x)) & \pi(\mathsf{f}^\sharp(\mathsf{s}(x))) > \pi(\mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(x)))) \\ \pi(\mathsf{g}(\mathsf{0})) \gtrsim \pi(\mathsf{s}(\mathsf{0})) & \end{array}$$

If an argument filtering π *and a reduction triple* $(\gtrsim, \geqslant, >)$ *satisfy them, we have the infinite sequence*

$$\pi(\mathsf{f}^\sharp(\mathsf{s}(\mathsf{0}))) > \pi(\mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(\mathsf{0})))) \gtrsim \pi(\mathsf{f}^\sharp(\mathsf{g}(\mathsf{0}))) \gtrsim \pi(\mathsf{f}^\sharp(\mathsf{s}(\mathsf{0}))) > \cdots$$

which contradicts the well-foundedness of $>$. *Note that the corresponding sequence*

$$\mathsf{f}^\sharp(\mathsf{s}(\mathsf{0})) \to_{\mathcal{C}} \mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(\mathsf{0}))) \to_{\mathcal{R}} \mathsf{f}^\sharp(\mathsf{g}(\mathsf{0})) \to_{\mathcal{R}} \mathsf{f}^\sharp(\mathsf{s}(\mathsf{0})) \to_{\mathcal{C}} \cdots$$

is a $\mathcal{C}$*-minimal rewrite sequence (but not a* $\mathcal{C}$*-minimal innermost rewrite sequence).*

3.4 Usable Rules

We show that the concept of *usable* rules which was introduced in [4] to optimize the dependency pair method for *innermost* termination, can also be used for termination. The resulting termination criterion is stronger than previous results in this area ([26, 55]). We start by recalling the definition of usable rules.

Definition 3.24. *We write* $f \rhd_{\mathrm{d}} g$ *if there exists a rewrite rule* $l \to r \in \mathcal{R}$ *such that* $f = \mathrm{root}(l)$ *and* g *is a defined function symbol in* $\mathcal{F}\mathsf{un}(r)$. *For a set* $\mathcal{G}$ *of defined function symbols we denote by* $\mathcal{R}{\restriction}\mathcal{G}$ *the set of rewrite rules* $l \to r \in \mathcal{R}$ *with* $\mathrm{root}(l) \in \mathcal{G}$. *The set* $\mathcal{U}(t)$ *of* usable rules *of a term* t *is defined as* $\mathcal{R}{\restriction}\{g \mid f \rhd_{\mathrm{d}}^* g \text{ for some } f \in \mathcal{F}\mathsf{un}(t)\}$. *Finally, if* $\mathcal{C}$ *is a set of dependency pairs then*

$$\mathcal{U}(\mathcal{C}) = \bigcup_{l \to r \in \mathcal{C}} \mathcal{U}(r)$$

Example 3.25. *None of the dependency pairs that appear in an cycle in our leading example have defined symbols in their right-hand sides, so for all cycles the set of usable rules is empty. The same is true for the TRSs of Examples 3.15 and 3.16.*

Arts and Giesl [5] showed that when proving innermost termination we may replace $\pi(\mathcal{R}) \subseteq \gtrsim$ by $\pi(\mathcal{U}(\mathcal{C}))$ in Theorem 3.20. After reproducing this result, we show that a similar result holds for full termination under an additional condition.

The next lemma states an easy connection between usable rules and defined symbols of the other rules.

Lemma 3.26. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and let $\mathcal{C} \subseteq \mathsf{DP}(\mathcal{R})$. Furthermore, let $\mathcal{G}$ be the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$.*

1. *$\mathcal{R} = \mathcal{U}(\mathcal{C}) \cup (\mathcal{R}{\restriction}\mathcal{G})$.*
2. *If $l \to r \in \mathcal{U}(\mathcal{C})$ then r contains no $\mathcal{G}$-symbols.*

Proof. The first statement is obvious. For the second statement we reason as follows. Suppose to the contrary that r contains a function symbol $g \in \mathcal{G}$. We have $l \to r \in \mathcal{U}(t)$ for some $s \to t \in \mathcal{C}$. So there exists a function symbol $f \in \mathcal{F}\mathsf{un}(t)$ such that $f \rhd_{\mathrm{d}}^* \mathrm{root}(l)$. We have $\mathrm{root}(l) \rhd_{\mathrm{d}} g$ by the definition of $\rhd_{\mathrm{d}}$ and hence also $f \rhd_{\mathrm{d}}^* g$. Therefore $\mathcal{R}{\restriction}\{g\} \subseteq \mathcal{U}(t) \subseteq \mathcal{U}(\mathcal{C})$. So g is a defined symbol of a rule in $\mathcal{U}(\mathcal{C})$. This contradicts the assumption that $g \in \mathcal{G}$. □

The next lemma states a key property for characterizing $\mathcal{C}$-minimal innermost rewrite sequences.

Lemma 3.27. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and let $\mathcal{C} \subseteq \mathsf{DP}(\mathcal{R})$. Furthermore, let $\mathcal{G}$ be the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$. If $t \xrightarrow{\mathsf{i}}_{\mathcal{R}}^* u$ and $t|_p$ is a normal form for all $p \in \mathcal{P}\mathsf{os}_{\mathcal{G}}(t)$ then $t \to_{\mathcal{U}(\mathcal{C})}^* u$.*

Proof. We prove the following claim: If $t \xrightarrow{\mathsf{i}}_{\mathcal{R}} u$ and $t|_p$ is a normal form for all $p \in \mathcal{P}\mathsf{os}_{\mathcal{G}}(t)$ then $t \xrightarrow{\mathsf{i}}_{\mathcal{U}(\mathcal{C})} u$ and $u|_p$ is a normal form for all $p \in \mathcal{P}\mathsf{os}_{\mathcal{G}}(u)$. The result then follows by induction. There exist a rewrite rule $l \to r \in \mathcal{R}$, a substitution σ, and a context C such that $t = C[t_1, \ldots, l\sigma, \ldots, t_n]$, $u = C[t_1, \ldots, r\sigma, \ldots, t_n]$, and C contains no symbol in $\mathcal{G}$. By the assumption, $\mathrm{root}(l) \notin \mathcal{G}$ and therefore $l \to r \in \mathcal{U}(\mathcal{C})$, so we have $t \to_{\mathcal{U}(\mathcal{C})} u$. We show that $u|_p$ is a normal form for all $p \in \mathcal{P}\mathsf{os}_{\mathcal{G}}(u)$. Let $p \in \mathcal{P}\mathsf{os}_{\mathcal{G}}(u)$. Clearly, $u|_p$ is a subterm of either t_i or $r\sigma$. In the former case $u|_p = t|_p$ holds, so $u|_p$ is a normal form. In the latter case r contains no $\mathcal{G}$-symbol by the second part of Lemma 3.26. Therefore, $u|_p$ is a subterm in the image of σ, and thus a normal form. □

We are ready to prove the innermost version of the usable rule criterion.

Theorem 3.28 ([5, 22]). *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{IDG}(\mathcal{R})$. If there exist an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ such that $\pi(\mathcal{U}(\mathcal{C})) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, and $\pi(\mathcal{C}) \cap > \neq \varnothing$ then there are no $\mathcal{C}$-minimal innermost rewrite sequences.*

Proof. Consider a $\mathcal{C}$-minimal innermost rewrite sequence

$$t_1 \xrightarrow{\mathsf{i}!}_{\mathcal{R}} u_1 \to_{\mathcal{C}} t_2 \xrightarrow{\mathsf{i}!}_{\mathcal{R}} u_2 \to_{\mathcal{C}} \cdots$$

Let $\mathcal{G}$ be the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$. We claim that $t_{i+1}|_p$ is a normal form for all $p \in \mathcal{P}\mathsf{os}_{\mathcal{G}}(t_{i+1})$. Then, Lemma 3.27 is applicable to $t_{i+1} \xrightarrow{\mathsf{i}}_{\mathcal{R}}^* u_{i+1}$ and we obtain $t_{i+1} \to_{\mathcal{U}(\mathcal{C})}^* u_{i+1}$ and thus also

$$t_2 \to_{\mathcal{U}(\mathcal{C})}^* u_2 \to_{\mathcal{C}} t_3 \to_{\mathcal{U}(\mathcal{C})}^* u_3 \to_{\mathcal{C}} \cdots$$

Because of the assumptions of this theorem, we can simply reuse the proof of Theorem 3.20 (where $\mathcal{U}(\mathcal{C})$ takes the place of $\mathcal{R}$) and obtain the desired contradiction with the well-foundedness of $>$. The claim is proved as follows. Fix i and let $p \in \mathcal{P}\mathsf{os}_{\mathcal{G}}(t_{i+1})$. There exist a dependency pair $l \to r \in \mathcal{C}$ and a substitution σ such that $u_i = l\sigma$ and $t_{i+1} = r\sigma$. According to the second part of Lemma 3.26, r contains no $\mathcal{G}$-symbols, and therefore $r\sigma|_p$ is a subterm of $\sigma(x)$ for some $x \in \mathcal{V}\mathsf{ar}(r)$. Since $u_i = l\sigma$ is a normal form and $\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l)$, $\sigma(x)$ is a normal form. Hence, $t_{i+1}|_p = r\sigma|_p$ is a normal form. □

Example 3.23 shows that the converse of Theorem 3.28 does not hold; note that $\mathsf{g}(\mathsf{h}(x)) \to \mathsf{g}(x)$ and $\mathsf{g}(\mathsf{0}) \to \mathsf{s}(\mathsf{0})$ belong to $\mathcal{U}(\{\mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(x)))\})$.

The case of termination is more difficult. Indeed, we cannot prove the soundness without imposing a further condition.

Example 3.29. *Consider the non-terminating TRS $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ of Example 1.2. The dependency graph $\mathsf{DG}(\mathcal{R})$ contains one cycle $\mathcal{C} = \{\mathsf{f}^\sharp(\mathsf{a},\mathsf{b},x) \to \mathsf{f}^\sharp(x,x,x)\}$. because of the following $\mathcal{C}$-minimal rewrite sequence:*

$$\begin{aligned}\mathsf{f}^\sharp(\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b})) &\to^2_{\mathcal{R}} \mathsf{f}^\sharp(\mathsf{a},\mathsf{b},\mathsf{g}(\mathsf{a},\mathsf{b}))\\ &\to_{\mathcal{C}} \mathsf{f}^\sharp(\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b}),\mathsf{g}(\mathsf{a},\mathsf{b}))\\ &\to^2_{\mathcal{R}} \cdots\end{aligned}$$

Let $(\gtrsim, \geqslant, >) = (\to^*_{\mathcal{C}}, \to^*_{\mathcal{C}}, \to^+_{\mathcal{C}})$. Since the TRS $\mathcal{C}$ is terminating, $(\gtrsim, \geqslant, >)$ is a reduction triple. It is easy to verify $\mathcal{U}(\mathcal{C}) = \varnothing \subseteq \gtrsim \cup \geqslant$ and $\mathcal{C} \subseteq >$.

Note that the $\mathcal{C}$-minimal rewrite sequence in the preceding example is not a $\mathcal{C}$-minimal innermost rewrite sequence.

The following definition is the key to our result. It is a variation of a similar definition in Urbain [55], which in turn is based on a definition of Gramlich [27].

Definition 3.30. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and let $\mathcal{G} \subseteq \mathcal{F}$. The* interpretation *$I_{\mathcal{G}}$ is a mapping from terminating terms in $\mathcal{T}(\mathcal{F}^\sharp, \mathcal{V})$ to terms in $\mathcal{T}(\mathcal{F}^\sharp \cup \{\mathsf{nil}, \mathsf{cons}\}, \mathcal{V})$, where* nil *and* cons *are fresh function symbols, inductively defined as follows:*

$$I_{\mathcal{G}}(t) = \begin{cases} t & \text{if } t \text{ is a variable}\\ f(I_{\mathcal{G}}(t_1),\ldots,I_{\mathcal{G}}(t_n)) & \text{if } t = f(t_1,\ldots,t_n) \text{ and } f \notin \mathcal{G}\\ \mathsf{cons}(f(I_{\mathcal{G}}(t_1),\ldots,I_{\mathcal{G}}(t_n)), t') & \text{if } t = f(t_1,\ldots,t_n) \text{ and } f \in \mathcal{G}\end{cases}$$

where in the last clause t' denotes the term $\mathrm{order}(\{I_{\mathcal{G}}(u) \mid t \to_{\mathcal{R}} u\})$ *with*

$$\mathrm{order}(T) = \begin{cases} \mathsf{nil} & \text{if } T = \varnothing\\ \mathsf{cons}(t, \mathrm{order}(T \setminus \{t\})) & \text{if } t \text{ is the minimum element of } T\end{cases}$$

Here we assume an arbitrary but fixed total order on $\mathcal{T}(\mathcal{F}^\sharp \cup \{\mathsf{nil}, \mathsf{cons}\}, \mathcal{V})$.

Because we deal with finite TRSs, the relation is $\to_{\mathcal{R}}$ is finitely branching and hence the set $\{u \mid t \to_{\mathcal{R}} u\}$ of one-step reducts of t is finite. Moreover, every term in this set is terminating. The well-definedness of $I_{\mathcal{G}}$ now follows by a straightforward induction argument. The difference with Urbain's definition is

that we insert $f(I_{\mathcal{G}}(t_1), \ldots, I_{\mathcal{G}}(t_n))$ in the list t' when $f \in \mathcal{G}$. This modification is crucial for obtaining Theorem 3.36 below.

In the following $\mathcal{C}_{\mathcal{E}}$ denotes the TRS consisting of the two projection rules

$$\mathsf{cons}(x, y) \to x$$
$$\mathsf{cons}(x, y) \to y$$

These rules are used to extract elements from the lists constructed by the interpretation $I_{\mathcal{G}}$. To improve readability, we abbreviate $\mathsf{cons}(t_1, \ldots \mathsf{cons}(t_n, \mathsf{nil}) \ldots)$ to $[t_1, \ldots, t_n]$ in the next example.

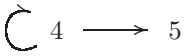
Example 3.31. *Consider the non-terminating TRS $\mathcal{R}$ consisting of the following three rewrite rules:*

$$1\colon\quad x + \mathsf{0} \to x \qquad 2\colon\quad x \times \mathsf{0} \to \mathsf{0} \qquad 3\colon\quad x \times \mathsf{s}(y) \to (x + \mathsf{0}) \times \mathsf{s}(y)$$

There are two dependency pairs:

$$4\colon\quad x \times^\sharp \mathsf{s}(y) \to (x + \mathsf{0}) \times^\sharp \mathsf{s}(y) \qquad\qquad 5\colon\quad x \times^\sharp \mathsf{s}(y) \to x +^\sharp \mathsf{0}$$

The dependency graph



contains 1 cycle: $\mathcal{C} = \{4\}$. The following is a $\mathcal{C}$-minimal rewrite sequence:

$$\begin{aligned}((\mathsf{0} + \mathsf{0}) \times \mathsf{0}) \times^\sharp \mathsf{s}(\mathsf{0}) &\to_{\mathcal{C}} (((\mathsf{0} + \mathsf{0}) \times \mathsf{0}) + \mathsf{0}) \times^\sharp \mathsf{s}(\mathsf{0})\\ &\to_{\mathcal{R}} ((\mathsf{0} + \mathsf{0}) \times \mathsf{0}) \times^\sharp \mathsf{s}(\mathsf{0})\\ &\to_{\mathcal{R}} \mathsf{0} \times^\sharp \mathsf{s}(\mathsf{0})\\ &\to_{\mathcal{C}} (\mathsf{0} + \mathsf{0}) \times^\sharp \mathsf{s}(\mathsf{0})\\ &\to_{\mathcal{R}} \mathsf{0} \times^\sharp \mathsf{s}(\mathsf{0})\\ &\to_{\mathcal{C}} \cdots\end{aligned}$$

We have $\mathcal{U}(\mathcal{C}) = \{1\}$. Let $\mathcal{G}$ be the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$, i.e., $\mathcal{G} = \{\times\}$. Applying the definition of $I_{\mathcal{G}}$ yields

$$\begin{aligned}I_{\mathcal{G}}(\mathsf{0} \times \mathsf{0}) &= \mathsf{cons}(I_{\mathcal{G}}(\mathsf{0}) \times I_{\mathcal{G}}(\mathsf{0}), \mathrm{order}(\{I_{\mathcal{G}}(\mathsf{0})\}))\\ &= \mathsf{cons}(\mathsf{0} \times \mathsf{0}, \mathrm{order}(\{\mathsf{0}\}))\\ &= \mathsf{cons}(\mathsf{0} \times \mathsf{0}, \mathsf{cons}(\mathsf{0}, \mathsf{nil}))\\ &= [\mathsf{0} \times \mathsf{0}, \mathsf{0}]\end{aligned}$$

and

$$\begin{aligned}I_{\mathcal{G}}((\mathsf{0} + \mathsf{0}) \times \mathsf{0}) &= \mathsf{cons}(I_{\mathcal{G}}(\mathsf{0} + \mathsf{0}) \times I_{\mathcal{G}}(\mathsf{0}), \mathrm{order}(\{I_{\mathcal{G}}(\mathsf{0} \times \mathsf{0}), I_{\mathcal{G}}(\mathsf{0})\}))\\ &= \mathsf{cons}((\mathsf{0} + \mathsf{0}) \times \mathsf{0}, \mathrm{order}(\{[\mathsf{0} \times \mathsf{0}, \mathsf{0}], \mathsf{0}\}))\\ &= [(\mathsf{0} + \mathsf{0}) \times \mathsf{0}, \mathsf{0}, [\mathsf{0} \times \mathsf{0}, \mathsf{0}]]\end{aligned}$$

if we assume that $\mathsf{0}$ is smaller than $[\mathsf{0} \times \mathsf{0}, \mathsf{0}]$ in the given total order. Now, by applying $I_{\mathcal{G}}$ to all terms in the above $\mathcal{C}$-minimal rewrite sequence, we obtain the

following infinite rewrite sequence in $\mathcal{U}(\mathcal{C}) \cup \mathcal{C} \cup \mathcal{C}_\mathcal{E}$*:*

$$\begin{aligned}
[(0+0)\times 0, 0, [0\times 0, 0]] \times^\sharp \mathsf{s}(0) &\to_\mathcal{C} ([(0+0)\times 0, 0, [0\times 0, 0]] + 0) \times^\sharp \mathsf{s}(0)\\
&\to_{\mathcal{U}(\mathcal{C})} [(0+0)\times 0, 0, [0\times 0, 0]] \times^\sharp \mathsf{s}(0)\\
&\to^+_{\mathcal{C}_\mathcal{E}} 0 \times^\sharp \mathsf{s}(0)\\
&\to_\mathcal{C} (0+0) \times^\sharp \mathsf{s}(0)\\
&\to_{\mathcal{U}(\mathcal{C})} 0 \times^\sharp \mathsf{s}(0)\\
&\to_\mathcal{C} \cdots
\end{aligned}$$

We start with some preliminary results. The first one addresses the behavior of $I_\mathcal{G}$ on instantiated terms. The second states that $I_\mathcal{G}$ preserves any top part without $\mathcal{G}$-symbols.

Definition 3.32. *If* σ *is a substitution that assigns to every variable in its domain a terminating term then we denote the substitution that assigns to every variable* x *the term* $I_\mathcal{G}(\sigma(x))$ *by* $\sigma_{I_\mathcal{G}}$*.*

Lemma 3.33. *Let* $\mathcal{R}$ *be a TRS over a signature* $\mathcal{F}$ *and let* $\mathcal{G} \subseteq \mathcal{F}$*. Let* t *be a term and* σ *a substitution. If* $t\sigma$ *is terminating then* $I_\mathcal{G}(t\sigma) \to^*_{\mathcal{C}_\mathcal{E}} t\sigma_{I_\mathcal{G}}$ *and, if* t *does not contain* $\mathcal{G}$*-symbols,* $I_\mathcal{G}(t\sigma) = t\sigma_{I_\mathcal{G}}$*.*

Proof. We use induction on t. If t is a variable then $I_\mathcal{G}(t\sigma) = I_\mathcal{G}(\sigma(t)) = t\sigma_{I_\mathcal{G}}$. Let $t = f(t_1, \ldots, t_n)$. We distinguish two cases.

- If $f \notin \mathcal{G}$ then $I_\mathcal{G}(t\sigma) = f(I_\mathcal{G}(t_1\sigma), \ldots, I_\mathcal{G}(t_n\sigma))$. The induction hypothesis yields $I_\mathcal{G}(t_i\sigma) \to^*_{\mathcal{C}_\mathcal{E}} t_i\sigma_{I_\mathcal{G}}$ for $1 \leqslant i \leqslant n$ and thus

  $$I_\mathcal{G}(t\sigma) \to^*_{\mathcal{C}_\mathcal{E}} f(t_1\sigma_{I_\mathcal{G}}, \ldots, t_n\sigma_{I_\mathcal{G}}) = t\sigma_{I_\mathcal{G}}$$

  If there are no $\mathcal{G}$-symbols in $t_1, \ldots, t_n$ then we obtain $I_\mathcal{G}(t_i\sigma) = t_i\sigma_{I_\mathcal{G}}$ for all $1 \leqslant i \leqslant n$ from the induction hypothesis and thus $I_\mathcal{G}(t\sigma) = t\sigma_{I_\mathcal{G}}$.

- If $f \in \mathcal{G}$ then

  $$I_\mathcal{G}(t\sigma) = \mathsf{cons}(f(I_\mathcal{G}(t_1\sigma), \ldots, I_\mathcal{G}(t_n\sigma)), t') \to_{\mathcal{C}_\mathcal{E}} f(I_\mathcal{G}(t_1\sigma), \ldots, I_\mathcal{G}(t_n\sigma))$$

  for some term t'. We obtain $f(I_\mathcal{G}(t_1\sigma), \ldots, I_\mathcal{G}(t_n\sigma)) \to^*_{\mathcal{C}_\mathcal{E}} t\sigma_{I_\mathcal{G}}$ as in the preceding case and thus $I_\mathcal{G}(t\sigma) \to^*_{\mathcal{C}_\mathcal{E}} t\sigma_{I_\mathcal{G}}$ as desired.

□

The preceding lemma is not true for Urbain's interpretation function.

Lemma 3.34. *Let* $\mathcal{R}$ *be a TRS over a signature* $\mathcal{F}$ *and let* $\mathcal{G} \subseteq \mathcal{F}$*. If* $t = C[t_1, \ldots, t_n]$ *is terminating and the context* C *contains no* $\mathcal{G}$*-symbols then* $I_\mathcal{G}(t) = C[I_\mathcal{G}(t_1), \ldots, I_\mathcal{G}(t_n)]$*.*

Proof. Let t' be the term $C[x_1, \ldots, x_n]$ where $x_1, \ldots, x_n$ are fresh variables. We have $t = t'\sigma$ for the substitution $\sigma = \{x_i \mapsto t_i \mid 1 \leqslant i \leqslant n\}$. The preceding lemma yields $I_\mathcal{G}(t) = t'\sigma_{I_\mathcal{G}}$. Clearly $t'\sigma_{I_\mathcal{G}} = C[I_\mathcal{G}(t_1), \ldots, I_\mathcal{G}(t_n)]$. □

The following lemma is the key result for the new termination criterion. It states that rewrite steps in $\mathcal{R}$ are transformed by $I_\mathcal{G}$ into rewrite sequences in $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E}$, provided $\mathcal{G}$ is the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$.

Lemma 3.35. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and let $\mathcal{C} \subseteq \mathsf{DP}(\mathcal{R})$. Furthermore, let $\mathcal{G}$ be the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$. If terms s and t are terminating and $s \to_{\mathcal{R}} t$ then $I_{\mathcal{G}}(s) \to^+_{\mathcal{U}(\mathcal{C})\cup\mathcal{C}_\mathcal{E}} I_{\mathcal{G}}(t)$.*

Proof. Let p be the position of the rewrite step $s \to_{\mathcal{R}} t$. We distinguish two cases.

- First suppose that there is a function symbol from $\mathcal{G}$ at a position $q \leqslant p$. In this case we may write $s = C[s_1, \ldots, s_i, \ldots, s_n]$ and $t = C[s_1, \ldots, t_i, \ldots, s_n]$ with $s_i \to_{\mathcal{R}} t_i$, where $\mathrm{root}(s_i) \in \mathcal{G}$ and the context C contains no $\mathcal{G}$-symbols. We have $I_{\mathcal{G}}(s_i) \to_{\mathcal{C}_\mathcal{E}} \mathrm{order}(\{I_{\mathcal{G}}(u) \mid s_i \to_{\mathcal{R}} u\})$. Since $s_i \to_{\mathcal{R}} t_i$, we can extract $I_{\mathcal{G}}(t_i)$ from the term $\mathrm{order}(\{I_{\mathcal{G}}(u) \mid s_i \to_{\mathcal{R}} u\})$ by appropriate $\mathcal{C}_\mathcal{E}$ steps, so $I_{\mathcal{G}}(s_i) \to^+_{\mathcal{C}_\mathcal{E}} I_{\mathcal{G}}(t_i)$. We now obtain $I_{\mathcal{G}}(s) \to^+_{\mathcal{C}_\mathcal{E}} I_{\mathcal{G}}(t)$ from Lemma 3.34.

- In the other case $s = C[s_1, \ldots, s_i, \ldots, s_n]$ and $t = C[s_1, \ldots, t_i, \ldots, s_n]$ with $s_i \xrightarrow{\epsilon}_{\mathcal{R}} t_i$, where $\mathrm{root}(s_i) \notin \mathcal{G}$ and the context C contains no $\mathcal{G}$-symbols. Since $\mathrm{root}(s_i) \notin \mathcal{G}$ the applied rewrite rule $l \to r$ in the step $s_i \xrightarrow{\epsilon}_{\mathcal{R}} t_i$ must come from $\mathcal{U}(\mathcal{C})$ according to part 1 of Lemma 3.26. Let σ be the substitution with $\mathcal{D}\mathsf{om}(\sigma) \subseteq \mathcal{V}\mathsf{ar}(l)$ such that $s_i = l\sigma$ and $t_i = r\sigma$. According to part 2 of Lemma 3.26, r contains no $\mathcal{G}$-symbols and thus we obtain $I_{\mathcal{G}}(s_i) \to^*_{\mathcal{C}_\mathcal{E}} l\sigma_{I_\mathcal{G}}$ and $I_{\mathcal{G}}(t_i) = r\sigma_{I_\mathcal{G}}$ from Lemma 3.33. Clearly $l\sigma_{I_\mathcal{G}} \to_{\mathcal{U}(\mathcal{C})} r\sigma_{I_\mathcal{G}}$ and thus $I_{\mathcal{G}}(s_i) \to^+_{\mathcal{U}(\mathcal{C})\cup\mathcal{C}_\mathcal{E}} I_{\mathcal{G}}(t_i)$. Lemma 3.34 now yields the desired $I_{\mathcal{G}}(s) \to^+_{\mathcal{U}(\mathcal{C})\cup\mathcal{C}_\mathcal{E}} I_{\mathcal{G}}(t)$.

□

After these preparations, the main result[3] of this section is now easily proved.

Theorem 3.36. *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$. If there exist an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ such that $\pi(\mathcal{U}(\mathcal{C})\cup\mathcal{C}_\mathcal{E}) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, and $\pi(\mathcal{C}) \cap > \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.*

Proof. Suppose to the contrary that there exists a $\mathcal{C}$-minimal rewrite sequence:

$$t_1 \to^*_{\mathcal{R}} u_1 \to_{\mathcal{C}} t_2 \to^*_{\mathcal{R}} u_2 \to_{\mathcal{C}} t_3 \to^*_{\mathcal{R}} \cdots \tag{3.3}$$

Let $\mathcal{G}$ be the set of defined symbols of $\mathcal{R} \setminus \mathcal{U}(\mathcal{C})$. We show that after applying the interpretation $I_\mathcal{G}$ we obtain an infinite rewrite sequence in $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E} \cup \mathcal{C}$ in which every rule of $\mathcal{C}$ is used infinitely often. Since all terms in (3.3) belong to $\mathcal{T}^\sharp_\infty$, they are terminating with respect to $\mathcal{R}$ and hence we can indeed apply the interpretation $I_\mathcal{G}$. Let $i \geqslant 1$.

- First consider the dependency pair step $u_i \to_{\mathcal{C}} t_{i+1}$. There exist a dependency pair $l \to r \in \mathcal{C}$ and a substitution σ such that $u_i = l\sigma$ and $t_{i+1} = r\sigma$. We may assume that $\mathcal{D}\mathsf{om}(\sigma) \subseteq \mathcal{V}\mathsf{ar}(l)$. Since $u_i \in \mathcal{T}^\sharp_\infty$, $\sigma(x)$ is terminating for every variable $x \in \mathcal{V}\mathsf{ar}(l)$. Hence the substitution $\sigma_{I_\mathcal{G}}$ is well-defined. Since r lacks $\mathcal{G}$-symbols by Lemma 3.26, we have $I_{\mathcal{G}}(r\sigma) = r\sigma_{I_\mathcal{G}}$ by Lemma 3.33. Furthermore, $I_{\mathcal{G}}(l\sigma) \to^*_{\mathcal{C}_\mathcal{E}} l\sigma_{I_\mathcal{G}}$ by Lemma 3.33. Hence

$$I_{\mathcal{G}}(u_i) \to^*_{\mathcal{C}_\mathcal{E}} l\sigma_{I_\mathcal{G}} \to_{\mathcal{C}} r\sigma_{I_\mathcal{G}} = I_{\mathcal{G}}(t_{i+1})$$

[3]This result has been independently obtained by Thiemann *et al.* [53].

- Next consider the rewrite sequence $t_i \to_{\mathcal{R}}^* u_i$. Because all terms in this sequence are terminating, we obtain $I_{\mathcal{G}}(t_i) \to_{\mathcal{U}(\mathcal{C})\cup\mathcal{C}_\mathcal{E}}^* I_{\mathcal{G}}(u_i)$ by repeated applications of Lemma 3.35.

Next we apply the argument filtering π to all terms in the resulting infinite rewrite sequence in $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E} \cup \mathcal{C}$. Because of the assumptions of this theorem, we can simply reuse the proof of Theorem 3.20 (where $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E}$ takes the place of $\mathcal{R}$) and obtain the desired contradiction with the well-foundedness of $>$. □

Example 3.37. *Let us take a final look at the cycle* $\{11, 12, 13, 14\}$ *in our leading example. There are no usable rules. By taking the linear polynomial interpretation* $\wedge^\sharp_{\mathbb{N}}(x, y) = x+y$ *and* $\vee_{\mathbb{N}}(x, y) = x+y+1$ *the involved dependency pairs reduce the following inequalities:*

$$\begin{aligned} 11, 13&: \quad x + y + z + 1 > x + y \\ 12, 14&: \quad x + y + z + 1 > x + z \end{aligned}$$

Hence there are no $\mathcal{C}$*-minimal rewrite sequences for any nonempty subset* $\mathcal{C} \subseteq \{11, 12, 13, 14\}$ *and we conclude that the TRS is terminating.*

Since $\mathcal{U}(\mathcal{C})$ in general is a proper subset of $\mathcal{R}$, the condition $\pi(\mathcal{U}(\mathcal{C})) \subseteq \gtrsim$ is easier to satisfy than the condition $\pi(\mathcal{R}) \subseteq \gtrsim$ of Theorem 3.20 and Corollary 3.22. Therefore, Theorem 3.28 is a better criterion for innermost termination, also because $\mathsf{IDG}(\mathcal{R})$ is a subgraph of $\mathsf{DG}(\mathcal{R})$. What about the additional condition $\pi(\mathcal{C}_\mathcal{E}) \subseteq \gtrsim$ of Theorem 3.36? By choosing $\pi(\mathsf{cons}) = [1, 2]$ the condition reduces to $\mathsf{cons}(x, y) \gtrsim x$ and $\mathsf{cons}(x, y) \gtrsim y$. Almost all reduction triples (pairs) that are used in termination tools can be extended to satisfy this condition. For reduction triples that are based on simplification orders, like $(>^{=}_{\mathrm{lpo}}, >^{=}_{\mathrm{lpo}}, >_{\mathrm{lpo}})$, this is clear. A sufficient condition that makes the semantic construction described in Section 3.3 for generating reduction triples work is that each pair of elements of the carrier has a least upper bound. For interpretations in the set $\mathbb{N}$ of natural numbers equipped with the standard order this is obviously satisfied. The necessity of the least upper bound condition follows by considering the term algebra associated with the TRS $\mathcal{R}_1$ of Example 1.2 equipped with the well-founded order $\to^+$. In Chapter 5 we introduce reduction triples based on polynomial interpretations with negative coefficients that are *not* compatible with $\mathcal{C}_\mathcal{E}$.

As a matter of fact, due to the condition $\pi(\mathcal{C}_\mathcal{E}) \subseteq \gtrsim$, Theorem 3.36 provides only a sufficient condition for the absence of $\mathcal{C}$-minimal rewrite sequences. This is in contrast to Theorem 3.20, which provides a sufficient and necessary condition for termination. The reason is that termination of a TRS $\mathcal{R}$ is equivalent to the termination of $\mathcal{R} \cup \mathsf{DP}(\mathcal{R})$, a result due to [4] (see [47] for a simple proof based on type introduction). A concrete example of a terminating TRS that cannot be proved terminating by the criterion of Theorem 3.36 is presented below.

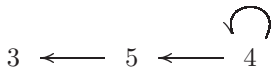
Example 3.38. *Consider the terminating TRS* $\mathcal{R}$ *consisting of the following two rewrite rules:*

$$\begin{aligned} 1&: \quad \mathsf{f}(\mathsf{s}(\mathsf{a}), \mathsf{s}(\mathsf{b}), x) \to \mathsf{f}(x, x, x) \\ 2&: \quad \mathsf{g}(\mathsf{f}(\mathsf{s}(x), \mathsf{s}(y), z)) \to \mathsf{g}(\mathsf{f}(x, y, z)) \end{aligned}$$

There are three dependency pairs:

$$\begin{aligned}
3:&\quad \mathsf{f}^\sharp(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b}),x) \to \mathsf{f}^\sharp(x,x,x)\\
4:&\quad \mathsf{g}^\sharp(\mathsf{f}(\mathsf{s}(x),\mathsf{s}(y),z)) \to \mathsf{g}^\sharp(\mathsf{f}(x,y,z))\\
5:&\quad \mathsf{g}^\sharp(\mathsf{f}(\mathsf{s}(x),\mathsf{s}(y),z)) \to \mathsf{f}^\sharp(x,y,z)
\end{aligned}$$

The dependency graph



contains 1 cycle: $\mathcal{C} = \{4\}$. *The only simple projection for* $\mathsf{g}^\sharp$ *transforms 4 into*

$$4:\quad \mathsf{f}(\mathsf{s}(x),\mathsf{s}(y),z) \to \mathsf{f}(x,y,z)$$

and $\mathsf{f}(x,y,z)$ *is not a proper subterm of* $\mathsf{f}(\mathsf{s}(x),\mathsf{s}(y),z)$. *We have* $\mathcal{U}(\mathcal{C}) = \{1\}$. *We claim that the conditions* $\pi(\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E}) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, *and* $\pi(\mathcal{C}) \cap > \neq \varnothing$ *are not satisfied for any argument filtering* π *and reduction triple* $(\gtrsim, \geqslant, >)$. *The reason is simply that the term* $t = \mathsf{g}^\sharp(\mathsf{f}(u,u,u))$ *with* $u = \mathsf{s}(\mathsf{cons}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b})))$ *admits the following cyclic reduction in* $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E} \cup \mathcal{C}$:

$$\begin{aligned}
t \to_{\mathcal{C}}\ & \mathsf{g}^\sharp(\mathsf{f}(\mathsf{cons}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b})),\mathsf{cons}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b})),u))\\
\to_{\mathcal{C}_\mathcal{E}}\ & \mathsf{g}^\sharp(\mathsf{f}(\mathsf{s}(\mathsf{a}),\mathsf{cons}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b})),u))\\
\to_{\mathcal{C}_\mathcal{E}}\ & \mathsf{g}^\sharp(\mathsf{f}(\mathsf{s}(\mathsf{a}),\mathsf{s}(\mathsf{b}),u))\\
\to_{\mathcal{U}(\mathcal{C})}\ & t
\end{aligned}$$

Combining the usable rule and subterm criteria, we arrive at the following result.

Corollary 3.39. *A TRS* $\mathcal{R}$ *is terminating if for every cycle* $\mathcal{C}$ *in* $\mathsf{DG}(\mathcal{R})$ *one of the following two conditions holds:*

- *there exists a simple projection* π *for* $\mathcal{C}$ *such that* $\pi(\mathcal{C}) \subseteq \trianglerighteq$ *and* $\pi(\mathcal{C}) \cap \triangleright \neq \varnothing$,
- *there exist an argument filtering* π *and a reduction triple* $(\gtrsim, \geqslant, >)$ *such that* $\pi(\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E}) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, *and* $\pi(\mathcal{C}) \cap > \neq \varnothing$.

For completeness sake, we state the corresponding result for innermost termination.

Corollary 3.40. *A TRS* $\mathcal{R}$ *is innermost terminating if for every cycle* $\mathcal{C}$ *in* $\mathsf{IDG}(\mathcal{R})$ *one of the following two conditions holds:*

- *there exists a simple projection* π *for* $\mathcal{C}$ *such that* $\pi(\mathcal{C}) \subseteq \trianglerighteq$ *and* $\pi(\mathcal{C}) \cap \triangleright \neq \varnothing$,
- *there exist an argument filtering* π *and a reduction triple* $(\gtrsim, \geqslant, >)$ *such that* $\pi(\mathcal{U}(\mathcal{C})) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, *and* $\pi(\mathcal{C}) \cap > \neq \varnothing$.

Apart from the innermost dependency graph, which is a subgraph of the dependency graph, the difference with Corollary 3.39 is that $\mathcal{C}_\mathcal{E}$-compatibility (i.e., $\pi(\mathcal{C}_\mathcal{E}) \subseteq \gtrsim$) is not required.

The final result of this section gives two sufficient conditions that allow us to ignore $\mathcal{C}_\mathcal{E}$ for cycles $\mathcal{C}$ in Theorem 3.36. This may be useful for the polynomial interpretations with negative coefficients of Chapter 5 which give rise to reduction triples that do not satisfy $\pi(\mathcal{C}_\mathcal{E}) \subseteq \gtrsim$.

Lemma 3.41. *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$ such that $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ is non-duplicating or $\mathcal{C}$ contains a right-ground rule. If there exist an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ such that $\pi(\mathcal{U}(\mathcal{C})) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, and $\pi(\mathcal{C}) \cap > \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.* □

Proof. If there are $\mathcal{C}$-minimal rewrite sequences then we obtain an infinite rewrite sequence

$$t_1 \to^*_{\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E}} \to t_2 \to_\mathcal{C} t_3 \to^*_{\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E}} \to t_4 \to_\mathcal{C} \cdots$$

as in the proof of Theorem 3.36. If $\mathcal{U}(\mathcal{C}) \cup \mathcal{C} \cup \mathcal{C}_\mathcal{E}$ is non-duplicating then, because the rules in $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ do not introduce cons symbols, there can be only finitely many $\mathcal{C}_\mathcal{E}$-steps in this sequence. The same holds if $\mathcal{C}$ contains a right-ground rule because after applying this rule there are no cons symbols left. So there is an index n such that

$$t_n \to^*_{\mathcal{U}(\mathcal{C})} t_{n+1} \to_\mathcal{C} t_{n+2} \to^*_{\mathcal{U}(\mathcal{C})} t_{n+3} \to_\mathcal{C} \cdots,$$

contradicting the assumptions. □

Chapter 4

Automating the Dependency Pair Method

When automating the methods in Chapter 3 several obstacles arise:

1. How to estimate uncomputable (innermost) dependency graphs?
2. How to handle the large (exponentially many in the worst case) number of cycles in the (innermost) dependency graphs?
3. How to find a suitable argument filtering from an exponential number of possible candidates.

In order to address the first problem we recall existing computable approximations and derive a new estimation of the innermost dependency graph in Section 4.1. The second problem is settled by the recursive SCC algorithm in Section 4.2. In Section 4.3 we present heuristics and a divide and conquer approach to reduce the search space of the third problem.

4.1 Dependency Graph Approximations

The (innermost) dependency graph cannot be computed in general because it is undecidable whether there exist substitutions σ and τ such that $t\sigma \to_{\mathcal{R}}^* u\tau$ ($t\sigma \overset{\text{i}}{\to}_{\mathcal{R}}^* u\tau$). In order to mechanize the (innermost) termination criterion of Corollary 3.39 (3.40) one has to approximate the (innermost) dependency graph. It is easy to see that we are allowed to use any supergraph of $\mathsf{DG}(\mathcal{G})$ ($\mathsf{IDG}(\mathcal{G})$). The challenge is to develop an approximation that yields a small supergraph of the (innermost) dependency graph. After introducing known (innermost) dependency graph approximations, we present a new approximation of the innermost dependency graph.

Arts and Giesl [4] proposed simple approximations based on syntactic unification.

Definition 4.1. *Let $\mathcal{R}$ be a TRS. The nodes of the* estimated *dependency graph* $\mathsf{EDG}(\mathcal{R})$ *are the dependency pairs of $\mathcal{R}$ and there is an arrow from $s \to t$ to $u \to v$ if and only if* $\mathsf{REN}(\mathsf{CAP}(t))$ *and u are unifiable. Here* CAP *replaces all*

outermost subterms with a defined root symbol by distinct fresh variables and REN *replaces all occurrences of variables by distinct fresh variables.*

Definition 4.2. *Let* $\mathcal{R}$ *be a TRS. The nodes of the* estimated *innermost dependency graph* $\mathsf{EIDG}(\mathcal{R})$ *are the dependency pairs of* $\mathcal{R}$ *and there is an arrow from* $s \to t$ *to* $u \to v$ *if and only if* $\mathsf{CAP}_s(t)$ *and* u *are unifiable with mgu* σ *such that* $s\sigma$ *and* $u\sigma$ *are in normal form. Here* $\mathsf{CAP}_s(t)$ *replaces all outermost subterms of* t different from all subterms of s *with a defined root symbol by distinct fresh variables.*

Middeldorp [45] showed that better approximations of the dependency graph are obtained by adopting tree automata techniques. These techniques are however computationally expensive. In [46] Middeldorp showed that the approximation of Definition 4.1 can be improved by symmetry considerations without incurring the overhead of tree automata techniques.

Definition 4.3. *Let* $\mathcal{R}$ *be a TRS over a signature* $\mathcal{F}$*. The result of replacing all outermost subterms of a term* t *with a root symbol in* $\mathcal{D}_{\mathcal{S}}^{-1}$ *by distinct fresh variables is denoted by* $\mathsf{CAP}_{\mathcal{S}}^{-1}(t)$*. Here* $\mathcal{D}_{\mathcal{S}}^{-1} = \{\mathrm{root}(r) \mid l \to r \in \mathcal{S}\}$ *if* $\mathcal{S}$ *is non-collapsing and* $\mathcal{D}_{\mathcal{S}}^{-1} = \mathcal{F}$ *otherwise. The nodes of the* estimated* *dependency graph* $\mathsf{EDG}^*(\mathcal{R})$ *are the dependency pairs of* $\mathcal{R}$ *and there is an arrow from* $s \to t$ *to* $u \to v$ *if and only if both* $\mathsf{REN}(\mathsf{CAP}(t))$ *and* u *are unifiable, and* t *and* $\mathsf{REN}(\mathsf{CAP}_{\mathcal{R}}^{-1}(u))$ *are unifiable.*

A comparison between the new estimation and the tree automata based approximations described in [45] can be found in [46]. From the latter paper we recall the identity $\mathsf{EDG}(\mathcal{R}) = \mathsf{EDG}^*(\mathcal{R})$ for collapsing $\mathcal{R}$. This explains why for most examples the new estimation does not improve upon the one of Arts of Giesl. However, when the two approximations do differ, the difference can be substantial.

Example 4.4. *Using the new estimation, automatically proving termination of notorious TRSs like the famous rule* $\mathsf{f}(\mathsf{a},\mathsf{b},x) \to \mathsf{f}(x,x,x)$ *of Toyama (cf. Example 1.2) becomes trivial, as in this case the estimated** *dependency graph coincides with the real dependency graph, and the latter is empty since no instance of* $\mathsf{f}^\sharp(x,x,x)$ *rewrites to an instance of* $\mathsf{f}^\sharp(\mathsf{a},\mathsf{b},x)$*. On the other hand, the estimated dependency graph contains a cycle and the constraints resulting from Corollary 3.39 cannot be solved by any quasi-simplification order.*

Next we introduce a new estimation of the innermost dependency graph.

Definition 4.5. *Let* $\mathcal{R}$ *be a TRS. The nodes of the* estimated* *innermost dependency graph* $\mathsf{EIDG}^*(\mathcal{R})$ *are the dependency pairs of* $\mathcal{R}$ *and there is an arrow from* $s \to t$ *to* $u \to v$ *if and only if both* $\mathsf{CAP}_s(t)$ *and* u *are unifiable with mgu* σ *such that* $s\sigma$ *and* $u\sigma$ *are normal forms, and* t *and* $\mathsf{REN}(\mathsf{CAP}_{\mathcal{U}(t)}^{-1}(u))$ *are unifiable with mgu* τ *such that* $s\tau$ *is a normal form.*

The following example shows that the new estimation is strictly more powerful than the one of Arts and Giesl if one uses reduction pairs based on quasi-simplification orders.

Example 4.6. *Consider the non-terminating TRS* $\mathcal{R}$ *consisting of the rules*

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

There are two dependency pairs:

$$1\colon \mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},x)) \to \mathsf{h}^\sharp(\mathsf{f}(x,x,x)) \qquad 2\colon \mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},x)) \to \mathsf{f}^\sharp(x,x,x)$$

Because there are no terms s and t such that $\mathsf{h}^\sharp(\mathsf{f}(s,s,s)) \xrightarrow{\mathsf{i}}{}^* \mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},t))$, the innermost dependency graph $\mathsf{IDG}(\mathcal{R})$ contains no arrows. Hence $\mathcal{R}$ is innermost terminating. We have $\mathcal{U}(\mathsf{h}^\sharp(\mathsf{f}(x,x,x))) = \{\mathsf{f}(x,x,x) \to \mathsf{c}\}$ and thus $\mathsf{h}^\sharp(\mathsf{f}(x,x,x))$ and

$$\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(\mathsf{h}^\sharp(\mathsf{f}(x,x,x)))}(\mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},x')))) = \mathsf{REN}(\mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},x'))) = \mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},x''))$$

are not unifiable. Hence $\mathsf{EIDG}^*(\mathcal{R})$ coincides with $\mathsf{IDG}(\mathcal{R})$. The estimated innermost dependency graph $\mathsf{EIDG}(\mathcal{R})$ contains arrows from 1 to 1 and 2 as $\mathsf{CAP}_{\mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},x))}(\mathsf{h}^\sharp(\mathsf{f}(x,x,x))) = \mathsf{h}^\sharp(y)$ unifies with $\mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},x'))$. However, the constraints for the resulting cycle $\{1\}$ cannot be solved by any combination of an argument filtering π and a reduction pair $(\gtrsim, >)$ based on a quasi-simplification order $\gtrsim$. Suppose to the contrary that $\pi(\mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},x))) > \pi(\mathsf{h}^\sharp(\mathsf{f}(x,x,x)))$ and $\pi(\mathsf{f}(x,x,x)) \gtrsim \pi(\mathsf{c})$. The first condition can only be satisfied if $\pi(\mathsf{h}^\sharp) \in \{1,[1]\}$ and $\pi(\mathsf{f}) \in \{[1,3],[2,3],[1,2,3]\}$. Let $t = \mathsf{f}(\mathsf{a},\mathsf{a},\mathsf{b})$. Because $\gtrsim$ is a quasi-simplification order, $\pi(t) \gtrsim \mathsf{a}$ and $\pi(t) \gtrsim \mathsf{b}$. We obtain $\pi(\mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},t))) > \pi(\mathsf{h}^\sharp(\mathsf{f}(t,t,t)))$ as $>$ is closed under substitutions. Closure under contexts of $\gtrsim$ yields $\pi(\mathsf{h}^\sharp(\mathsf{f}(t,t,t))) \gtrsim \pi(\mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},t)))$. Hence

$$\pi(\mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},t))) > \pi(\mathsf{h}^\sharp(\mathsf{f}(\mathsf{a},\mathsf{b},t)))$$

by the compatibility of $>$ and $\gtrsim$, contradicting the well-foundedness of $>$.

The following lemma states the soundness of the new approximation.

Lemma 4.7. *For a TRS $\mathcal{R}$, $\mathsf{IDG}(\mathcal{R}) \subseteq \mathsf{EIDG}^*(\mathcal{R})$.*

Proof. Suppose there is an arrow from $s \to t$ to $u \to v$ in $\mathsf{IDG}(\mathcal{R})$. So there exists a substitution σ such that $t\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} u\sigma$ and $s\sigma$ and $u\sigma$ are normal forms. The first condition of the definition of EIDG^* holds because $\mathsf{IDG}(\mathcal{R}) \subseteq \mathsf{EIDG}(\mathcal{R})$. We claim that $t\sigma = \mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))\mu$ for some substitution μ. Since t and $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))$ do not share variables, the substitution $\tau = \sigma \uplus \mu$ is well-defined and clearly a unifier of t and $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))$. Hence these two terms admit an mgu τ' which subsumes τ. We have $s\tau = s\sigma$. The latter term is a normal form by assumption and hence so is $s\tau'$. Consequently, the second condition of the definition of EIDG^* holds as well. We prove the claim by induction on u. If u is a variable or $\mathrm{root}(u) \in \mathcal{D}^{-1}_{\mathcal{U}(t)}$ then $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))$ is a fresh variable, say x, and we can take $\mu = \{x \mapsto t\sigma\}$. If $u = f(u_1,\dots,u_n)$ with $f \notin \mathcal{D}^{-1}_{\mathcal{U}(t)}$ then $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u)) = f(\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u_1)),\dots,\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u_n)))$. Because $s\sigma$ is a normal form and $\mathcal{V}\mathsf{ar}(t) \subseteq \mathcal{V}\mathsf{ar}(s)$, $t\sigma \to^*_{\mathcal{R}} u\sigma$ if and only if $t\sigma \to^*_{\mathcal{U}(t)} u\sigma$. The latter is equivalent to $u\sigma \to^*_{\mathcal{U}(t)^{-1}} t\sigma$. We distinguish two cases. If t is a variable then $t\sigma$ is a subterm of $s\sigma$ and thus a normal form. Hence $t\sigma = u\sigma$ and since $u\sigma$ is an instance of $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))$, we are done. Otherwise, since $\mathrm{root}(u) = f \notin \mathcal{D}^{-1}_{\mathcal{U}(t)}$, $t\sigma$ must be of the form $f(t_1\sigma,\dots,t_n\sigma)$ and we have $u_i\sigma \to^*_{\mathcal{U}(t)^{-1}} t_i\sigma$ for each $i \in \{1,\dots,n\}$. The induction hypothesis yields for each i a substitution μ_i such that $t_i\sigma = \mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u_i))\mu_i$. Since different $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u_i))$ do not share variables, the substitution $\mu = \mu_1 \uplus \dots \uplus \mu_n$ is well-defined and clearly satisfies $t\sigma = \mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))\mu$. □

The next example shows that we cannot omit REN from $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))$ without violating the soundness condition $\mathsf{IDG}(\mathcal{R}) \subseteq \mathsf{EIDG}^*(\mathcal{R})$ of Lemma 4.7, which is essential for inferring innermost termination.

Example 4.8. *Consider the TRS $\mathcal{R}$ consisting of the rules*

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

There are two dependency pairs:

$$1\colon \mathsf{f}^\sharp(x,x) \to \mathsf{f}^\sharp(\mathsf{g}(x),x) \qquad\qquad 2\colon \mathsf{f}^\sharp(x,x) \to \mathsf{g}^\sharp(x)$$

Since $\mathsf{f}^\sharp(\mathsf{g}(\mathsf{h}(x)),\mathsf{h}(x)) \xrightarrow{\mathsf{i}} \mathsf{f}^\sharp(\mathsf{h}(x),\mathsf{h}(x))$, $\mathsf{IDG}(\mathcal{R})$ contains arrows from 1 to 1 and 2. However, $\mathsf{CAP}^{-1}_{\mathcal{U}(\mathsf{f}^\sharp(\mathsf{g}(x),x))}(\mathsf{f}^\sharp(x',x')) = \mathsf{CAP}^{-1}_{\{\mathsf{g}(\mathsf{h}(x))\to\mathsf{h}(x)\}}(\mathsf{f}^\sharp(x',x')) = \mathsf{f}^\sharp(x',x')$ does not unify with $\mathsf{f}^\sharp(\mathsf{g}(x),x)$. Thus, by replacing $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))$ with $\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u)$ in the definition of $\mathsf{EIDG}^*(\mathcal{R})$, we would obtain graph without cycles and hence wrongly conclude innermost termination.

Note that in the above example $\xrightarrow{\mathsf{i}}$ differs from $(\xleftarrow{\mathsf{i}})^{-1}$. Replacing CAP^{-1} by CAP^{-1}_s (or CAP^{-1}_v) in Definition 4.5 would make the approximation unsound. Here CAP^{-1}_s replaces all outermost subterms different from subterms of s with a root symbol in $\mathcal{D}^{-1}_{\mathcal{R}}$ by distinct fresh variables.

Example 4.9. *Consider the non-innermost terminating TRS $\mathcal{R}$ consisting of the rules $\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{b})$ and $\mathsf{b} \to \mathsf{a}$. There is one dependency pair: $\mathsf{f}^\sharp(\mathsf{a}) \to \mathsf{f}^\sharp(\mathsf{b})$. Because of $\mathsf{f}^\sharp(\mathsf{b}) \xrightarrow{\mathsf{i}} \mathsf{f}^\sharp(\mathsf{a})$, $\mathsf{IDG}(\mathcal{R})$ contains a cycle. This cycle would not be detected if CAP^{-1} is replaced by CAP^{-1}_s: $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathsf{f}^\sharp(\mathsf{a})}(\mathsf{f}^\sharp(\mathsf{a}))) = \mathsf{f}^\sharp(\mathsf{a})$ does not unify with $\mathsf{f}^\sharp(\mathsf{b})$.*

The following theorem summarizes the relationships between the various approximations. The only non-trivial inclusions are $\mathsf{DG}(\mathcal{R}) \subseteq \mathsf{EDG}^*(\mathcal{R})$ ([46]) and $\mathsf{IDG}(\mathcal{R}) \subseteq \mathsf{EIDG}^*(\mathcal{R})$ (Lemma 4.7).

Theorem 4.10. *For any TRS $\mathcal{R}$, the following inclusions hold:*

$$\begin{array}{ccccc} \mathsf{DG}(\mathcal{R}) & \subseteq & \mathsf{EDG}^*(\mathcal{R}) & \subseteq & \mathsf{EDG}(\mathcal{R}) \\ \cup\mathsf{I} & & \cup\mathsf{I} & & \cup\mathsf{I} \\ \mathsf{IDG}(\mathcal{R}) & \subseteq & \mathsf{EIDG}^*(\mathcal{R}) & \subseteq & \mathsf{EIDG}(\mathcal{R}) \end{array}$$

□

Unlike the inclusion $\mathsf{EDG}^*(\mathcal{R}) \subseteq \mathsf{EDG}(\mathcal{R})$, the inclusion $\mathsf{EIDG}^*(\mathcal{R}) \subseteq \mathsf{EIDG}(\mathcal{R})$ need not become an equality for collapsing $\mathcal{R}$, due to the use of usable rules in the second part of Definition 4.5. This can be seen from Example 4.6.

We conclude this section by showing that for every finite graph G there is a terminating TRS such that its dependency graph is isomorphic to G. This implies that the dependency graph may form a large clique or even a complete graph. We discuss the implications in the next section.

Definition 4.11. *Let $G = (\{1,\dots,n\}, E)$ be a finite graph. The TRS $\mathcal{R}_G$ is the set of rewrite rules $\{l_i \to r_i \mid 1 \leqslant i \leqslant n\}$ with*

$$\begin{aligned} l_i &= \mathsf{f}(x_1,\dots,x_{i-1},\mathsf{a},x_{i+1},\dots,x_n,\mathsf{s}(y)) \\ r_i &= \mathsf{f}(r_{i1},\dots,r_{in},y) \end{aligned}$$

where

$$r_{ij} = \begin{cases} \mathsf{a} & \text{if } (i,j) \in E \\ \mathsf{b} & \text{if } (i,j) \notin E \end{cases}$$

Lemma 4.12. *Let $G = (\{1,\dots,n\}, E)$ be a finite graph.*

1. *$\mathsf{DG}(\mathcal{R}_G)$, $\mathsf{EDG}(\mathcal{R}_G)$, $\mathsf{EDG}^*(\mathcal{R}_G)$, $\mathsf{IDG}(\mathcal{R}_G)$, $\mathsf{EIDG}(\mathcal{R}_G)$, and $\mathsf{EIDG}^*(\mathcal{R}_G)$ are identical and isomorphic to G.*
2. *$\mathcal{R}_G$ is terminating.*

Proof. We have $\mathsf{DP}(\mathcal{R}_G) = \{l_i^\sharp \to r_i^\sharp \mid 1 \leqslant i \leqslant n\}$.

1. Suppose $(i,j) \in E$. By taking the substitutions $\sigma = \{y \mapsto \mathsf{s}(y)\}$ and $\tau = \{x_k \mapsto r_{ik} \mid k \in \{1,\dots,n\} \setminus \{j\}\}$, we have $r_i^\sharp\sigma = l_j^\sharp\tau$ and the terms $l_i^\sharp\sigma$ and $l_j^\sharp\tau$ are normal forms. Therefore, $\mathsf{IDG}(\mathcal{R}_G)$ contains the arrow from $l_i^\sharp \to r_i^\sharp$ to $l_j^\sharp \to r_j^\sharp$. Suppose $(i,j) \notin E$. $\mathsf{REN}(r_i^\sharp)$ and $l_j^\sharp$ are not unifiable because their i-th arguments are b and a. Therefore, $\mathsf{EDG}(\mathcal{R}_G)$ does not contain the arrow from $l_i^\sharp \to r_i^\sharp$ to $l_j^\sharp \to r_j^\sharp$. By Theorem 4.10 we obtain the desired result.
2. By taking the simple projection $\pi(\mathsf{f}^\sharp) = n+1$, all dependency pairs are simplified to $\mathsf{s}(y) \to y$. Therefore, the subterm criterion (Theorem 3.13) applies and hence $\mathcal{R}_G$ is terminating.

□

4.2 Cycle Analysis

The use of Corollary 3.39 (3.40) for ensuring (innermost) termination requires that *all* cycles have to be considered.

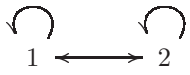
Example 4.13. *Consider the TRS from [21] consisting of the two rules*

$$\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(\mathsf{s}(x)) \qquad\qquad \mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(x)$$

There are two dependency pairs

$$1\colon \mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(\mathsf{s}(x)) \qquad 2\colon \mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(x)$$

and the dependency graph



has three cycles. The cycles $\{2\}$ and $\{1,2\}$ are readily handled by the subterm criterion, but the constraints for cycle $\{1\}$ cannot be solved. Note that the TRS is not (innermost) terminating.

Unfortunately, the number of cycles can be very large, even if the number of dependency pairs is small. In the worst case, there are $2^n - 1$ cycles for n dependency pairs. This explains why in early implementations ([3, 10]) of the dependency pair method, strongly connected components rather than cycles are computed. A *strongly connected component* (SCC) is a maximal (with respect

to the inclusion relation) cycle. Note that the number of SCCs for n dependency pairs is at most n, since every dependency pair belongs to at most one SCC.

We introduce convenient notations to abstract from the combination of conditions in Theorems 3.13, 3.20 and 3.36.

Definition 4.14. *Let $\mathcal{R}$ be a TRS, $\mathcal{C}$ a subset of $\mathsf{DP}(\mathcal{R})$, and $\mathcal{D}$ a subset of $\mathcal{C}$. We write $\vDash (\mathcal{C}, \mathcal{D})$ if one of the following statements holds:*

- *there exists a simple projection π such that $\pi(\mathcal{C}) \subseteq \trianglerighteq$, and $\pi(\mathcal{D}) \subseteq \triangleright$,*
- *there exist an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ such that $\pi(\mathcal{R}) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \trianglerighteq$, and $\pi(\mathcal{D}) \subseteq >$,*
- *there exist an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ such that $\pi(\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E}) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, and $\pi(\mathcal{D}) \subseteq >$.*

Moreover, we write $\vDash_\exists \mathcal{C}$ if there exists a non-empty subset $\mathcal{D}$ of $\mathcal{C}$ such that $\vDash (\mathcal{C}, \mathcal{D})$, and we write $\vDash_\forall \mathcal{C}$ if $\vDash (\mathcal{C}, \mathcal{C})$.

In the notation the existential quantifier indicates that *some* pair in $\mathcal{C}$ should be strictly decreasing, and the universal quantifier indicates that *all* pairs in $\mathcal{C}$ should be strictly decreasing. The next two statements are immediate consequences of Theorems 3.13, 3.20, and 3.36.

Corollary 4.15. *A TRS $\mathcal{R}$ is terminating if $\vDash_\exists \mathcal{C}$ for every cycle $\mathcal{C}$ in $\mathsf{DG}(\mathcal{R})$.* □

Corollary 4.16. *A TRS $\mathcal{R}$ is terminating if $\vDash_\forall \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$.* □

The difference with Corollary 4.16 is that all pairs in an SCC must be strictly decreasing. This, however, makes the termination criterion of Corollary 4.15 strictly weaker than the one of Theorem 4.16, if we employ reduction triples based on simplification orders.

Example 4.17. *Consider the following TRS (from [5]):*

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

There are three dependency pairs:

$$\begin{aligned}
1\colon&\quad \mathsf{evenodd}^\sharp(x, \mathsf{0}) \to \mathsf{NOT}(\mathsf{evenodd}(x, \mathsf{s}(\mathsf{0})))\\
2\colon&\quad \mathsf{evenodd}^\sharp(x, \mathsf{0}) \to \mathsf{evenodd}^\sharp(x, \mathsf{s}(\mathsf{0}))\\
3\colon&\ \mathsf{evenodd}^\sharp(\mathsf{s}(x), \mathsf{s}(\mathsf{0})) \to \mathsf{evenodd}^\sharp(x, \mathsf{0})
\end{aligned}$$

The dependency graph

$$1 \longleftarrow 3 \longleftrightarrow 2$$

contains one cycle: $\{2, 3\}$. The subterm criterion handles the cycle by taking the simple projection π with $\pi(\mathsf{evenodd}^\sharp) = 1$.

However, if one uses Corollary 4.15 the cycle (SCC) cannot be solved by the subterm criterion. Even a combination of an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ based on a quasi-simplification order $\gtrsim$ fails. To see this, suppose that both

$$\pi(\mathsf{evenodd}^\sharp(x, \mathsf{0})) > \pi(\mathsf{evenodd}^\sharp(x, \mathsf{s}(\mathsf{0})))$$

and

$$\pi(\mathsf{evenodd}^\sharp(\mathsf{s}(x), \mathsf{s}(\mathsf{0}))) > \pi(\mathsf{evenodd}^\sharp(x, \mathsf{0}))$$

Since every quasi-simplification order satisfies $\mathsf{s}(\mathsf{0}) \gtrsim \mathsf{0}$, the first constraint requires $\pi(\mathsf{s}) = [\,]$, $\pi(\mathsf{evenodd}^\sharp) \in \{[1, 2], [2], 2\}$, and $\mathsf{0} > \mathsf{s}$. So the second constraint reduces to

$$\pi(\mathsf{evenodd}^\sharp(\mathsf{s}, \mathsf{s})) > \pi(\mathsf{evenodd}^\sharp(x, \mathsf{0}))$$

and this latter constraint can only be satisfied if $\pi(\mathsf{evenodd}^\sharp) \in \{[2], 2\}$ and $\mathsf{s} > \mathsf{0}$. This is clearly impossible.

In order to cope with this problem, we propose a new *recursive SCC algorithm* to compute and solve SCCs. More precisely, if $\mathcal{S}$ is the current SCC then we first use a method in Definition 4.14 to show $\vDash (\mathcal{S}, \mathcal{D})$ for some non-empty subset $\mathcal{D}$ of $\mathcal{S}$. Then we compute the SCCs of the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by $\mathcal{S} \setminus \mathcal{D}$. These new SCCs are added to the list of SCCs that have to be solved. It turns out that this new approach has the termination proving power of Corollary 4.16 and the efficiency of Corollary 4.15. The former is proved below and the latter is confirmed by extensive experiments (see Section 6.2) and explained in the paragraph following Theorem 4.22.

In order to ensure that ignored subcycles are actually harmless, the individual theorems used in the algorithm must satisfy the following *subcycle condition*. Fortunately, all methods introduced in this thesis satisfy it.

Lemma 4.18. *Let $\mathcal{R}$ be a TRS, $\mathcal{S}$ a subset of the dependency pairs, and $\mathcal{C}$ and $\mathcal{D}$ subsets of $\mathcal{S}$. If $\vDash (\mathcal{S}, \mathcal{D})$ then $\vDash (\mathcal{C}, \mathcal{C} \cap \mathcal{D})$.*

Proof. Suppose $\vDash (\mathcal{S}, \mathcal{D})$. Based on the definition we perform case analysis.

- Suppose that there exists a simple projection π such that $\pi(\mathcal{S}) \subseteq \trianglerighteq$ and $\pi(\mathcal{D}) \subseteq \triangleright$. Since $\mathcal{C} \subseteq \mathcal{S}$, we have $\pi(\mathcal{C}) \subseteq \pi(\mathcal{S})$ and $\pi(\mathcal{C} \cap \mathcal{D}) \subseteq \pi(\mathcal{D})$. Therefore, $\pi(\mathcal{C}) \subseteq \trianglerighteq$ and $\pi(\mathcal{C} \cap \mathcal{D}) \subseteq \triangleright$ hold. Hence, the first condition of $\vDash$ holds.

- Suppose that there exist an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ such that $\pi(\mathcal{R}) \subseteq \gtrsim$, $\pi(\mathcal{S}) \subseteq \geqslant$, and $\pi(\mathcal{D}) \subseteq >$. We obtain $\pi(\mathcal{C}) \subseteq \geqslant$ and $\pi(\mathcal{C} \cap \mathcal{D}) \subseteq >$ as in the previous case. Hence, the second condition of $\vDash$ holds.

- Suppose that there exist an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ such that $\pi(\mathcal{U}(\mathcal{S}) \cup \mathcal{C}_\mathcal{E}) \subseteq \gtrsim$, $\pi(\mathcal{S}) \subseteq \geqslant$, and $\pi(\mathcal{D}) \subseteq \triangleright$. From $\mathcal{C} \subseteq \mathcal{S}$ we obtain $\mathcal{U}(\mathcal{C}) \subseteq \mathcal{U}(\mathcal{S})$, so $\pi(\mathcal{U}(\mathcal{C}) \cup \mathcal{C}_\mathcal{E}) \subseteq \pi(\mathcal{U}(\mathcal{S}) \cup \mathcal{C}_\mathcal{E}) \subseteq \gtrsim$ holds. We obtain $\pi(\mathcal{C}) \subseteq \geqslant$ and $\pi(\mathcal{C} \cap \mathcal{D}) \subseteq >$ as before. Hence, the third condition of $\vDash$ holds.

□

Now we are ready to prove the correctness of the recursive SCC algorithm.

Definition 4.19. *Let $\mathcal{R}$ be a TRS and $\mathcal{S}$ a subset of the dependency pairs in $\mathsf{DP}(\mathcal{R})$. We write $\vDash \mathcal{S}$ if there exists a non-empty subset $\mathcal{D}$ of $\mathcal{S}$ such that $\vDash (\mathcal{S}, \mathcal{D})$ and $\vDash \mathcal{S}'$ for all SCCs $\mathcal{S}'$ of the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by $\mathcal{S} \setminus \mathcal{D}$.*

Theorem 4.20. *Let $\mathcal{R}$ be a TRS. The following conditions are equivalent:*

1. *$\vDash \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$,*
2. *$\vDash_\exists \mathcal{C}$ for every cycle $\mathcal{C}$ in $\mathsf{DG}(\mathcal{R})$.*

Proof. First suppose $\vDash \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$ and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$. We show that $\vDash_\exists \mathcal{C}$. Let $\mathcal{S}$ be the SCC that contains $\mathcal{C}$. We use induction on the size of $\mathcal{S}$. We have $\vDash \mathcal{S}$ by assumption. There exists a non-empty subset $\mathcal{D}$ of $\mathcal{S}$ such that $\vDash \mathcal{S}'$ for all SCCs $\mathcal{S}'$ of the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by $\mathcal{S} \setminus \mathcal{D}$. Since $\mathcal{C}$ and $\mathcal{D}$ are subsets of $\mathcal{S}$, Lemma 4.18 yields $\vDash (\mathcal{C}, \mathcal{C} \cap \mathcal{D})$. If $\mathcal{C} \cap \mathcal{D} \neq \varnothing$ then $\vDash_\exists \mathcal{C}$. Otherwise, all pairs in $\mathcal{C}$ belong to $\mathcal{S} \setminus \mathcal{D}$ and thus $\mathcal{C}$ is a cycle in the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by $\mathcal{S} \setminus \mathcal{D}$. Hence $\mathcal{C}$ is contained in an SCC $\mathcal{S}'$ of this subgraph. We have $\vDash \mathcal{S}'$ by assumption. Since $|\mathcal{S}'| < |\mathcal{S}|$ we can apply the induction hypothesis to obtain the desired $\vDash_\exists \mathcal{C}$.

Next we suppose that $\vDash_\exists \mathcal{C}$ for every cycle $\mathcal{C}$ in $\mathsf{DG}(\mathcal{R})$. Let $\mathcal{S}$ be an SCC in $\mathsf{DG}(\mathcal{R})$. We have to show that $\vDash \mathcal{S}$. We use induction on the size of $\mathcal{S}$. Since $\mathcal{S}$ is also a cycle, $\vDash_\exists \mathcal{S}$, i.e., there exists a non-empty subset $\mathcal{D}$ of $\mathcal{S}$ such that $\vDash (\mathcal{S}, \mathcal{D})$. $\mathcal{S} \setminus \mathcal{D}$ is a proper subset of $\mathcal{S}$, and therefore every SCC $\mathcal{S}'$ in the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by $\mathcal{S} \setminus \mathcal{D}$ is smaller than $\mathcal{S}$, and thus $\vDash \mathcal{S}'$ by the induction hypothesis. Consequently, $\vDash \mathcal{S}$. □

The above proof provides quite a bit more information than the statement of Theorem 4.20 suggests. As a matter of fact, both conditions are equivalent to termination of $\mathcal{R}$, and also equivalent to the criterion "$\vDash_\forall \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$" of Corollary 4.15. However, from the proof of Theorem 4.20 we learn that a termination proof based on "$\vDash \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$" can be directly transformed into a termination proof based on "$\vDash_\exists \mathcal{C}$ for every cycle $\mathcal{C}$ in $\mathsf{DG}(\mathcal{R})$" and vice-versa; there is no need to search for new argument filterings and reduction pairs. This is not true for the criterion of Corollary 4.15.

Theorem 4.20 and the discussion following it easily generalize to the innermost case.

Definition 4.21. *Let $\mathcal{R}$ be a TRS, $\mathcal{C}$ a subset of $\mathsf{DP}(\mathcal{R})$, and $\mathcal{D}$ a subset of $\mathcal{C}$. We write $\vDash_{\mathsf{i}} (\mathcal{C}, \mathcal{D})$ if one of the following statements holds:*

- *there exists a simple projection π such that $\pi(\mathcal{C}) \subseteq \trianglerighteq$, and $\pi(\mathcal{D}) \subseteq \triangleright$.*
- *there exist an argument filtering π and a reduction triple $(\gtrsim, \geqslant, >)$ such that $\pi(\mathcal{U}(\mathcal{C})) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \geqslant$, and $\pi(\mathcal{D}) \subseteq >$.*

We write $\vDash_{\mathsf{i}} \mathcal{S}$ if there exists a non-empty subset $\mathcal{D}$ of $\mathcal{S}$ such that $\vDash \mathcal{S}, \mathcal{D}$ and $\vDash_{\mathsf{i}} \mathcal{S}'$ for all SCCs $\mathcal{S}'$ of the subgraph of $\mathsf{IDG}(\mathcal{R})$ induced by $\mathcal{S} \setminus \mathcal{D}$. Moreover, we write $\vDash_{\mathsf{i}\exists} \mathcal{C}$ if there exists a non-empty subset $\mathcal{D}$ of $\mathcal{C}$ such that $\vDash_{\mathsf{i}} (\mathcal{C}, \mathcal{D})$.

Theorem 4.22. *Let $\mathcal{R}$ be a TRS. The following conditions are equivalent:*

1. *$\vDash_{\mathsf{i}} \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{IDG}(\mathcal{R})$,*

2. $\vDash_{i\exists} \mathcal{C}$ *for every cycle* $\mathcal{C}$ *in* $\mathsf{IDG}(\mathcal{R})$.

□

A dependency graph with n dependency pairs has at most n SCCs. So the number of groups of ordering constraints that need to be solved in order to ensure (innermost) termination according to Corollary 4.15 is bounded by n. We already remarked that the number of cycles and hence the number of groups generated by the cycle approach of Corollary 4.16 is at most $2^n - 1$. This upper bound cannot be improved, because a dependency graph may be a complete graph (cf. Lemma 4.12), which contains $2^n - 1$ cycles. It is easy to see that the recursive SCC algorithm of Theorem 4.20 (4.22) generates at most n groups. This explains why the efficiency of the new approach is comparable to the SCC approach and better than the cycle approach. It also explains why (human or machine) verification of the (innermost) termination proof generated by the new algorithm involves (much) less work than the one generated by the approach based on Corollary 4.16.

We illustrate the recursive SCC algorithm of Theorem 4.20 on two relatively small TRSs that contain very many cycles.

Example 4.23. *As an extreme example, consider the TRS* $\mathcal{R}$ *(Example 11 in [13]) consisting of the rules*

$$\begin{aligned}
\mathsf{D}(\mathsf{t}) &\to 1 & \mathsf{D}(x+y) &\to \mathsf{D}(x) + \mathsf{D}(y)\\
\mathsf{D}(\mathsf{c}) &\to 0 & \mathsf{D}(x \times y) &\to (y \times \mathsf{D}(x)) + (x \times \mathsf{D}(y))\\
\mathsf{D}(-x) &\to -\mathsf{D}(x) & \mathsf{D}(x-y) &\to \mathsf{D}(x) - \mathsf{D}(y)\\
\mathsf{D}(\mathsf{ln}\, x) &\to \mathsf{D}(x)/x & \mathsf{D}(x/y) &\to (\mathsf{D}(x)/y) - ((x \times \mathsf{D}(y))/y^2)\\
\mathsf{D}(x^y) &\to ((y \times x^{y-1}) \times \mathsf{D}(x)) + ((x^y \times \mathsf{ln}\, x) \times \mathsf{D}(y))
\end{aligned}$$

The only defined symbol, D*, occurs 12 times in the right-hand sides of the rules, so* $\mathsf{DP}(\mathcal{R})$ *consists of 12 dependency pairs:*

$$\begin{array}{lll}
\mathsf{D}^\sharp(-x) \to \mathsf{D}^\sharp(x) & \mathsf{D}^\sharp(x+y) \to \mathsf{D}^\sharp(x) & \mathsf{D}^\sharp(x+y) \to \mathsf{D}^\sharp(y)\\
\mathsf{D}^\sharp(\mathsf{ln}\, x) \to \mathsf{D}^\sharp(x) & \mathsf{D}^\sharp(x-y) \to \mathsf{D}^\sharp(x) & \mathsf{D}^\sharp(x-y) \to \mathsf{D}^\sharp(y)\\
\mathsf{D}^\sharp(x^y) \to \mathsf{D}^\sharp(x) & \mathsf{D}^\sharp(x \times y) \to \mathsf{D}^\sharp(x) & \mathsf{D}^\sharp(x \times y) \to \mathsf{D}^\sharp(y)\\
\mathsf{D}^\sharp(x^y) \to \mathsf{D}^\sharp(y) & \mathsf{D}^\sharp(x/y) \to \mathsf{D}^\sharp(x) & \mathsf{D}^\sharp(x/y) \to \mathsf{D}^\sharp(y)
\end{array}$$

All these dependency pairs have a right-hand side $\mathsf{D}^\sharp(t)$ *with* t *a variable. It follows that the dependency graph* $\mathsf{DG}(\mathcal{R})$ *is a complete graph. Consequently, there are* $2^{12} - 1 = 4095$ *cycles but just 1 SCC* $\mathcal{S} = \mathsf{DP}(\mathcal{R})$*. We show the termination of* $\mathcal{R}$*. The subterm criterion takes care of all dependency pairs, i.e.,* $\vDash (\mathcal{S}, \mathcal{S})$ *and thus also* $\vDash \mathcal{S}$*. Hence,* $\mathcal{R}$ *is terminating.*

The next TRS is obtained by applying the first transformation of Giesl and Middeldorp [23, Definition 12] to a context-sensitive rewrite system that approximates the infinite sequence $\frac{1}{1}, \frac{1}{4}, \frac{1}{9}, \ldots, \frac{1}{n^2}$ whose partial sums converge to $\frac{\pi^2}{6}$ (Lucas [42, Example 2]). In the termination proof we use LPO with quasi-precedence and linear polynomial interpretations with coefficients in $\{0, 1\}$ as base orders.

Example 4.24. *Consider the following TRS $\mathcal{R}$:*

$$
\begin{aligned}
1:&\ \mathsf{terms_a}(x) \to \mathsf{recip}(\mathsf{sqr_a}(\mathsf{m}(x))) : \mathsf{terms}(\mathsf{s}(x))\\
2:&\ \mathsf{sqr_a}(0) \to 0\\
3:&\ \mathsf{sqr_a}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{sqr_a}(\mathsf{m}(x)) +_\mathsf{a} \mathsf{dbl_a}(\mathsf{m}(x)))\\
4:&\ \mathsf{dbl_a}(0) \to 0\\
5:&\ \mathsf{dbl_a}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{dbl_a}(\mathsf{m}(x))))\\
6:&\ 0 +_\mathsf{a} y \to \mathsf{m}(y)\\
7:&\ \mathsf{s}(x) +_\mathsf{a} y \to \mathsf{s}(\mathsf{m}(x) +_\mathsf{a} \mathsf{m}(y))\\
8:&\ \mathsf{first_a}(0, z) \to \mathsf{nil}\\
9:&\ \mathsf{first_a}(\mathsf{s}(x), y : z) \to \mathsf{m}(y) : \mathsf{first}(x, z)\\
10:&\ \mathsf{half_a}(0) \to 0\\
11:&\ \mathsf{half_a}(\mathsf{s}(0)) \to 0\\
12:&\ \mathsf{half_a}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{half_a}(\mathsf{m}(x)))\\
13:&\ \mathsf{half_a}(\mathsf{dbl}(x)) \to \mathsf{m}(x)\\
14:&\ \mathsf{m}(\mathsf{terms}(x)) \to \mathsf{terms_a}(\mathsf{m}(x))\\
15:&\ \mathsf{m}(\mathsf{sqr}(x)) \to \mathsf{sqr_a}(\mathsf{m}(x))\\
16:&\ \mathsf{m}(x + y) \to \mathsf{m}(x) +_\mathsf{a} \mathsf{m}(y)\\
17:&\ \mathsf{m}(\mathsf{dbl}(x)) \to \mathsf{dbl_a}(\mathsf{m}(x))\\
18:&\ \mathsf{m}(\mathsf{first}(x, y)) \to \mathsf{first_a}(\mathsf{m}(x), \mathsf{m}(y))\\
19:&\ \mathsf{m}(\mathsf{half}(x)) \to \mathsf{half_a}(\mathsf{m}(x))\\
20:&\ \mathsf{m}(x : y) \to \mathsf{m}(x) : y\\
21:&\ \mathsf{m}(\mathsf{recip}(x)) \to \mathsf{recip}(\mathsf{m}(x))\\
22:&\ \mathsf{m}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{m}(x))\\
23:&\ \mathsf{m}(0) \to 0\\
24:&\ \mathsf{m}(\mathsf{nil}) \to \mathsf{nil}
\end{aligned}
\qquad
\begin{aligned}
25:&\ \mathsf{terms_a}(x) \to \mathsf{terms}(x)\\
26:&\ \mathsf{sqr_a}(x) \to \mathsf{sqr}(x)\\
27:&\ x +_\mathsf{a} y \to x + y\\
28:&\ \mathsf{dbl_a}(x) \to \mathsf{dbl}(x)\\
29:&\ \mathsf{first_a}(x, y) \to \mathsf{first}(x, y)\\
30:&\ \mathsf{half_a}(x) \to \mathsf{half}(x)
\end{aligned}
$$

There are 33 dependency pairs:

$$
\begin{aligned}
31:&\ \mathsf{terms}^\sharp_\mathsf{a}(x) \to \mathsf{sqr}^\sharp_\mathsf{a}(\mathsf{m}(x))\\
32:&\ \mathsf{terms}^\sharp_\mathsf{a}(x) \to \mathsf{m}^\sharp(x)\\
33:&\ \mathsf{sqr}^\sharp_\mathsf{a}(\mathsf{s}(x)) \to \mathsf{sqr_a}(\mathsf{m}(x)) +^\sharp_\mathsf{a} \mathsf{dbl_a}(\mathsf{m}(x))\\
34:&\ \mathsf{sqr}^\sharp_\mathsf{a}(\mathsf{s}(x)) \to \mathsf{sqr}^\sharp_\mathsf{a}(\mathsf{m}(x))\\
35:&\ \mathsf{sqr}^\sharp_\mathsf{a}(\mathsf{s}(x)) \to \mathsf{m}^\sharp(x)\\
36:&\ \mathsf{sqr}^\sharp_\mathsf{a}(\mathsf{s}(x)) \to \mathsf{dbl}^\sharp_\mathsf{a}(\mathsf{m}(x))\\
37:&\ \mathsf{dbl}^\sharp_\mathsf{a}(\mathsf{s}(x)) \to \mathsf{dbl}^\sharp_\mathsf{a}(\mathsf{m}(x))\\
38:&\ \mathsf{dbl}^\sharp_\mathsf{a}(\mathsf{s}(x)) \to \mathsf{m}^\sharp(x)\\
39:&\ 0 +^\sharp_\mathsf{a} y \to \mathsf{m}^\sharp(y)\\
40:&\ \mathsf{s}(x) +^\sharp_\mathsf{a} y \to \mathsf{m}(x) +^\sharp_\mathsf{a} \mathsf{m}(y)\\
41:&\ \mathsf{s}(x) +^\sharp_\mathsf{a} y \to \mathsf{m}^\sharp(x)\\
42:&\ \mathsf{s}(x) +^\sharp_\mathsf{a} y \to \mathsf{m}^\sharp(y)
\end{aligned}
$$

$$
\begin{aligned}
43&: \mathsf{first}^\sharp_\mathsf{a}(\mathsf{s}(x), y : z) \to \mathsf{m}^\sharp(y)\\
44&: \mathsf{half}^\sharp_\mathsf{a}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{half}^\sharp_\mathsf{a}(\mathsf{m}(x))\\
45&: \mathsf{half}^\sharp_\mathsf{a}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{m}^\sharp(x)\\
46&: \mathsf{half}^\sharp_\mathsf{a}(\mathsf{dbl}(x)) \to \mathsf{m}^\sharp(x)\\
47&: \mathsf{m}^\sharp(\mathsf{terms}(x)) \to \mathsf{terms}^\sharp_\mathsf{a}(\mathsf{m}(x))\\
48&: \mathsf{m}^\sharp(\mathsf{terms}(x)) \to \mathsf{m}^\sharp(x)\\
49&: \mathsf{m}^\sharp(\mathsf{sqr}(x)) \to \mathsf{sqr}^\sharp_\mathsf{a}(\mathsf{m}(x))\\
50&: \mathsf{m}^\sharp(\mathsf{sqr}(x)) \to \mathsf{m}^\sharp(x)\\
51&: \mathsf{m}^\sharp(x + y) \to \mathsf{m}(x) +^\sharp_\mathsf{a} \mathsf{m}(y)\\
52&: \mathsf{m}^\sharp(x + y) \to \mathsf{m}^\sharp(x)\\
53&: \mathsf{m}^\sharp(x + y) \to \mathsf{m}^\sharp(y)\\
54&: \mathsf{m}^\sharp(\mathsf{dbl}(x)) \to \mathsf{dbl}^\sharp_\mathsf{a}(\mathsf{m}(x))\\
55&: \mathsf{m}^\sharp(\mathsf{dbl}(x)) \to \mathsf{m}^\sharp(x)\\
56&: \mathsf{m}^\sharp(\mathsf{first}(x, y)) \to \mathsf{first}^\sharp_\mathsf{a}(\mathsf{m}(x), \mathsf{m}(y))\\
57&: \mathsf{m}^\sharp(\mathsf{first}(x, y)) \to \mathsf{m}^\sharp(x)\\
58&: \mathsf{m}^\sharp(\mathsf{first}(x, y)) \to \mathsf{m}^\sharp(y)\\
59&: \mathsf{m}^\sharp(\mathsf{half}(x)) \to \mathsf{half}^\sharp_\mathsf{a}(\mathsf{m}(x))\\
60&: \mathsf{m}^\sharp(\mathsf{half}(x)) \to \mathsf{m}^\sharp(x)\\
61&: \mathsf{m}^\sharp(x : y) \to \mathsf{m}^\sharp(x)\\
62&: \mathsf{m}^\sharp(\mathsf{recip}(x)) \to \mathsf{m}^\sharp(x)\\
63&: \mathsf{m}^\sharp(\mathsf{s}(x)) \to \mathsf{m}^\sharp(x)
\end{aligned}
$$

The dependency graph contains a single SCC that consists of all dependency pairs. The usable rules are all rules in $\mathcal{R}$. By taking the argument filtering π with $\pi(\mathsf{dbl}^\sharp_\mathsf{a}) = \pi(\mathsf{half}^\sharp_\mathsf{a}) = \pi(\mathsf{m}^\sharp) = \pi(\mathsf{half}_\mathsf{a}) = \pi(:) = \pi(\mathsf{half}) = \pi(\mathsf{m}) = \pi(\mathsf{recip}) = 1$ and $\pi(\mathsf{first}^\sharp_\mathsf{a}) = 2$ together with LPO with quasi-precedence $0 > \mathsf{nil}$, $\mathsf{terms}_\mathsf{a} \approx \mathsf{terms} > \mathsf{terms}^\sharp_\mathsf{a} > \mathsf{sqr}^\sharp_\mathsf{a} \approx \mathsf{sqr}_\mathsf{a} \approx \mathsf{sqr} > + \approx +_\mathsf{a} > +^\sharp_\mathsf{a}$, $\mathsf{sqr} > \mathsf{dbl} \approx \mathsf{dbl}_\mathsf{a} > \mathsf{s}$, $+ > \mathsf{s}$, and $\mathsf{first}_\mathsf{a} \approx \mathsf{first}$, all rewrite rules are (weakly) decreasing, the dependency pairs in $\{31\text{–}42, 44\text{–}48, 50\text{–}58, 63\}$ are strictly decreasing and the remaining dependency pairs 43, 49, and 59–62 are weakly decreasing.

- *The dependency graph restricted to these dependency pairs contains one SCC: $\{60, 61, 62\}$. By taking the simple projection π with $\pi(\mathsf{m}) = 1$, all dependency pairs are strictly decreasing. We obtain $\vDash \{60, 61, 62\}$.*

Therefore, $\vDash \mathsf{DP}(\mathcal{R})$. Hence the termination of $\mathcal{R}$ is proved. The proof took 9.78 seconds. Using either LPO with quasi-precedence or linear polynomial interpretations with coefficients in $\{0, 1\}$ as base order will fail. The point we want to stress, however, is that computing all cycles is doomed to fail. The dependency graph contains at least 11,004,672 cycles but 24 hours of CPU time was insufficient to compute the exact number.

In order to find a suitable argument filtering and LPO precedence, we used the some *heuristic and the divide and conquer technique with dynamic programming, which are introduced in the next section.*

To conclude this section, we can safely state that every implementation of the dependency pair method should use the recursive SCC algorithm for cycle analysis.

4.3 Argument Filterings

The search for a suitable argument filtering that enables the simplified constraints to be solved by a reduction pair based on a strongly monotone simplification order is the main bottleneck of the dependency pair method. The standard approach is to enumerate all possible argument filterings until one is encountered that enables the resulting constraints to be solved. However, since a single function symbol of arity n already gives rise to $2^n + n$ different argument filterings, enumeration is impractical except for small examples. In this chapter we present two new ideas to reduce the number of computed argument filterings.

4.3.1 Heuristics

We propose two simple heuristics that significantly reduce the number of argument filterings:

- In the *some* heuristic we consider for an n-ary function symbol f only the 'full' argument filtering $\pi(f) = [1, \ldots, n]$ and the n 'collapsing' argument filterings $\pi(f) = i$ for $i = 1, \ldots, n$.
- In the *some more* heuristic we consider additionally the argument filtering $\pi(f) = [\,]$ (when $n > 0$).

Clearly, an n-ary function symbol admits $n + 1$ argument filterings in the *some* heuristic and $n + 2$ (1 if $n = 0$) in the *some more* heuristic. The following example shows that even if the total number of function symbols is relatively small, the savings made by these heuristics is significant.

Example 4.25. *Consider the following TRS (from [5]), encoding the quicksort algorithm:*

$$
\begin{array}{rlrl}
1\colon & \mathsf{high}(n, \mathsf{nil}) \to \mathsf{nil} & 9\colon & \mathsf{ifHigh}(\mathsf{false}, n, m : x) \to m : \mathsf{high}(n, x) \\
2\colon & \mathsf{high}(n, m : x) \to \mathsf{ifHigh}(m \leq n, n, m : x) & 10\colon & \mathsf{ifHigh}(\mathsf{true}, n, m : x) \to \mathsf{high}(n, x) \\
3\colon & \mathsf{low}(n, \mathsf{nil}) \to \mathsf{nil} & 11\colon & \mathsf{ifLow}(\mathsf{false}, n, m : x) \to \mathsf{low}(n, x) \\
4\colon & \mathsf{low}(n, m : x) \to \mathsf{ifLow}(m \leq n, n, m : x) & 12\colon & \mathsf{ifLow}(\mathsf{true}, n, m : x) \to m : \mathsf{low}(n, x) \\
5\colon & \mathsf{nil} +\!\!+ y \to y & 13\colon & 0 \leq y \to \mathsf{true} \\
6\colon & (n : x) +\!\!+ y \to n : (x +\!\!+ y) & 14\colon & \mathsf{s}(x) \leq 0 \to \mathsf{false} \\
7\colon & \mathsf{qsort}(\mathsf{nil}) \to \mathsf{nil} & 15\colon & \mathsf{s}(x) \leq \mathsf{s}(y) \to x \leq y \\
8\colon & \mathsf{qsort}(n : x) \to \mathsf{qsort}(\mathsf{low}(n, x)) +\!\!+ (n : \mathsf{qsort}(\mathsf{high}(n, x))) & &
\end{array}
$$

There are 2 function symbols of arity 3, 5 function symbols of arity 2, 2 function symbols of arity 1, and 2 function symbols of arity 0, resulting in $(2^3 + 3)^2 \times (2^2 + 2)^5 \times (2^1 + 1)^2 \times (2^0 + 0)^2 = 8468064$ *argument filterings for just the rule constraints. The* some more *heuristic produces only* 230400 *possible argument filterings and the* some *heuristic reduces this number further to* 15552.

One can imagine several other heuristics, like computing all argument filterings for function symbols of arity $n \leqslant 2$ but only some for function symbols of higher arity. Needless to say, adopting any of these heuristics reduces the class of TRSs that can be proved (innermost) terminating automatically. Nevertheless, the experiments reported in Section 6.2 reveal that the two heuristics described above are surprisingly effective. The reason is that termination is often caused by a decrease in one argument of a recursive call, which can be captured by a suitable 'collapsing' argument filtering. Moreover, the new recursive algorithm for cycle analysis described in Section 4.2 supports the situation where different recursive calls of the same function depend on a decrease of different arguments.

4.3.2 Divide and Conquer

In this subsection we propose a new divide and conquer approach for finding *all* suitable argument filterings while avoiding enumeration. In the following we develop this approach in a stepwise fashion.

The first observation is that argument filterings should be computed for terms rather than for function symbols. Consider e.g. the term $t = \mathsf{f}(\mathsf{g}(\mathsf{h}(x)), y)$. There are $6 \times 3 \times 3 = 54$ possible argument filterings for the function symbols f, g, and h. Many of these argument filterings contain redundant information. For instance, if $\pi(\mathsf{f}) = [2]$ then it does not matter how $\pi(\mathsf{g})$ and $\pi(\mathsf{h})$ are defined since g and h no longer appear in $\pi(t) = \mathsf{f}(y)$; likewise for $\pi(\mathsf{f}) = 2$ or $\pi(\mathsf{f}) = [\,]$. If $\pi(\mathsf{f}) \in \{[1,2],[1],1\}$ and $\pi(\mathsf{g}) = [\,]$ then the value of $\pi(\mathsf{h})$ is irrelevant. It follows that there are only 24 'minimal' argument filterings for t. The following definitions explains how these minimal argument filterings can be computed.

Definition 4.26. *Let $\mathcal{F}$ be a signature. We consider partial argument filterings that need not be defined for all function symbols in $\mathcal{F}$. The completely undefined argument filtering will be denoted by ϵ. Let π be a (partial) argument filtering and t a term in $\mathcal{T}(\mathcal{F},\mathcal{V})$. The* domain $\mathrm{dom}(\pi)$ *is the set of function symbols on which π is defined. We define $\mathsf{outer}(t,\pi)$ as the subset of $\mathcal{F}$ consisting of those function symbols in t where the computation of $\pi(t)$ gets stuck: $\mathsf{outer}(t,\pi) = \varnothing$ when $t \in \mathcal{V}$ and if $t = f(t_1,\ldots,t_n)$ then $\mathsf{outer}(t,\pi) = \mathsf{outer}(t_i,\pi)$ when $\pi(f) = i$, $\mathsf{outer}(t,\pi) = \bigcup_{j=1}^{m} \mathsf{outer}(t_{i_j},\pi)$ when $\pi(f) = [i_1,\ldots,i_m]$, and $\mathsf{outer}(t_i,\pi) = \{f\}$ when $\pi(f)$ is undefined. Let π and π' be argument filterings. We say that π' is an* extension *of π and write $\pi \subseteq \pi'$ if $\mathrm{dom}(\pi) \subseteq \mathrm{dom}(\pi')$ and $\pi(f) = \pi'(f)$ for all $f \in \mathrm{dom}(\pi)$. Finally, if $\mathcal{G} \subseteq \mathcal{F}$ then $\mathsf{AF}(\mathcal{G})$ denotes the set of all argument filterings whose domain coincides with $\mathcal{G}$.*

The next definition introduces a set $\mathsf{AF}(t,\pi)$ of argument filterings that extend π and permit the term t to be completely evaluated.

Definition 4.27. *Let $\mathcal{F}$ be a signature, $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$, and π an argument filtering. We define a set $\mathsf{AF}(t,\pi)$ of argument filterings as follows: $\mathsf{AF}(t,\pi) = \{\pi\}$ if $\mathsf{outer}(t,\pi) = \varnothing$ and $\mathsf{AF}(t,\pi) = \bigcup\{\mathsf{AF}(t,\pi') \mid \pi' \in \mathsf{AF}(\mathsf{outer}(t,\pi)) \times \pi\}$ if $\mathsf{outer}(t,\pi) \neq \varnothing$. Here $\mathsf{AF}(\mathsf{outer}(t,\pi))$ returns the set of all argument filterings whose domain coincide with $\mathsf{outer}(t,\pi)$ and $\mathsf{AF}(\mathsf{outer}(t,\pi)) \times \pi$ extends each of these argument filterings with π.*

Note that the recursion in the definition of $\mathsf{AF}(t,\pi)$ terminates since its second argument enables more and more of t to be evaluated, until $\pi(t)$ can be

completely computed, i.e., until $\mathsf{outer}(t,\pi) = \varnothing$. Next we present an equivalent non-recursive definition of $\mathsf{AF}(t,\pi)$.

Definition 4.28. *For a term t and an argument filtering π we denote by $\mathsf{AF}'(t,\pi)$ the set of minimal extensions π' of π such that $\mathsf{outer}(t,\pi') = \varnothing$. Minimality here means that if $\mathsf{outer}(t,\pi'') = \varnothing$ and $\pi \subseteq \pi'' \subseteq \pi'$ then $\pi'' = \pi'$.*

Lemma 4.29. *For all terms t and argument filterings π, $\mathsf{AF}(t,\pi) = \mathsf{AF}'(t,\pi)$.*

Proof. We use induction on $n = |\mathcal{F}\mathsf{un}(t) \setminus \mathrm{dom}(\pi)|$. If $n = 0$ then $\mathcal{F}\mathsf{un}(t) \setminus \mathrm{dom}(\pi) = \varnothing$ and thus $\mathsf{outer}(t,\pi) = \varnothing$. Hence $\mathsf{AF}(t,\pi) = \{\pi\} = \mathsf{AF}'(t,\pi)$. Suppose $n > 0$. We have $\mathsf{AF}(t,\pi) = \bigcup\{\mathsf{AF}(t,\pi') \mid \pi' \in \mathsf{AF}(\mathsf{outer}(t,\pi)) \times \pi\}$. For every $\pi' \in \mathsf{AF}(\mathsf{outer}(t,\pi)) \times \pi$, $|\mathcal{F}\mathsf{un}(t) \setminus \mathrm{dom}(\pi')| < n$ and thus $\mathsf{AF}(t,\pi') = \mathsf{AF}'(t,\pi')$ by the induction hypothesis. So it remains to show that

$$\mathsf{AF}'(t,\pi) = \bigcup\{\mathsf{AF}'(t,\pi') \mid \pi' \in \mathsf{AF}(\mathsf{outer}(t,\pi)) \times \pi\}.$$

First suppose that $\pi'' \in \mathsf{AF}'(t,\pi)$. So $\pi \subseteq \pi''$ and $\mathsf{outer}(t,\pi'') = \varnothing$. Hence there exists an argument filtering $\pi' \in \mathsf{AF}(\mathsf{outer}(t,\pi)) \times \pi$ such that $\pi' \subseteq \pi''$. To conclude that $\pi'' \in \mathsf{AF}'(t,\pi')$ we have to show that $\pi'' = \bar\pi$ whenever $\pi' \subseteq \bar\pi \subseteq \pi''$ and $\mathsf{outer}(t,\bar\pi) = \varnothing$. Clearly $\pi \subseteq \bar\pi \subseteq \pi''$ for any such $\bar\pi$ and thus $\pi'' = \bar\pi$ by the assumption $\pi'' \in \mathsf{AF}'(t,\pi)$.

Next suppose that $\pi'' \in \mathsf{AF}'(t,\pi')$ for some $\pi' \in \mathsf{AF}(\mathsf{outer}(t,\pi)) \times \pi$. We have $\mathsf{outer}(t,\pi'') = \varnothing$, $\pi \subseteq \pi' \subseteq \pi''$, and $\mathrm{dom}(\pi') = \mathrm{dom}(\pi) \cup \mathsf{outer}(t,\pi)$. To conclude that $\pi'' \in \mathsf{AF}'(t,\pi)$ it remains to show that $\pi'' = \bar\pi$ whenever $\pi \subseteq \bar\pi \subseteq \pi''$ and $\mathsf{outer}(t,\bar\pi) = \varnothing$. Any such $\bar\pi$ satisfies $\mathrm{dom}(\pi) \cup \mathsf{outer}(t,\pi) \subseteq \mathrm{dom}(\bar\pi)$ and hence, as $\bar\pi \subseteq \pi''$ and $\pi' \subseteq \pi''$, $\bar\pi$ and π' agree on the function symbols in $\mathsf{outer}(t,\pi)$. Consequently, $\pi' \subseteq \bar\pi$ and thus $\pi'' = \bar\pi$ by the assumption $\pi'' \in \mathsf{AF}'(t,\pi')$. □

Since a term t can be completely evaluated by an argument filtering π if and only if $\mathsf{outer}(t,\pi) = \varnothing$, the next result is an immediate consequence of Lemma 4.29.

Corollary 4.30. *$\mathsf{AF}(t,\epsilon)$ is the set of all minimal argument filterings π such that $\pi(t)$ can be completely evaluated.* □

We now explain how to compute the set of minimal argument filterings for a set of terms.

Definition 4.31. *Let T be a set of terms. We denote by $\mathsf{AF}(T)$ the set of all minimal argument filterings that completely evaluate each term in T. In particular, we define $\mathsf{AF}(\varnothing) = \{\epsilon\}$.*

Definition 4.32. *Two argument filterings π_1 and π_2 are said to be* compatible *if they agree on the function symbols on which both are defined, in which case their union $\pi_1 \cup \pi_2$ is defined in the obvious way. If A_1 and A_2 are sets of argument filterings then $A_1 \otimes A_2 = \{\pi_1 \cup \pi_2 \mid \pi_1 \in A_1$ and $\pi_2 \in A_2$ are compatible$\}$.*

Note that $\{\epsilon\}$ is the identity of the *merge* operation $\otimes$. The following lemma expresses the fact that merging preserves the minimality property.

Lemma 4.33. *If T_1, T_2 are sets of terms then $\mathsf{AF}(T_1 \cup T_2) = \mathsf{AF}(T_1) \otimes \mathsf{AF}(T_2)$.*

Proof. First we show that $\mathsf{AF}(T_1 \cup T_2) \subseteq \mathsf{AF}(T_1) \otimes \mathsf{AF}(T_2)$. Let $\pi \in \mathsf{AF}(T_1 \cup T_2)$. Let π_1 and π_2 be the minimum restrictions of π that completely evaluate every term in T_1 and T_2, respectively. We have $\pi_1 \in \mathsf{AF}(T_1)$ and $\pi_2 \in \mathsf{AF}(T_2)$ by definition. Since π_1 and π_2 are compatible, $\pi_1 \cup \pi_2 \in \mathsf{AF}(T_1) \otimes \mathsf{AF}(T_2)$. Since $\pi_1 \cup \pi_2$ completely evaluates every term in $T_1 \cup T_2$, we obtain $\pi = \pi_1 \cup \pi_2$ from the minimality of π.

Next we show that $\mathsf{AF}(T_1) \otimes \mathsf{AF}(T_2) \subseteq \mathsf{AF}(T_1 \cup T_2)$. Let $\pi \in \mathsf{AF}(T_1) \otimes \mathsf{AF}(T_2)$. So there exist compatible $\pi_1 \in \mathsf{AF}(T_1)$ and $\pi_2 \in \mathsf{AF}(T_2)$ such that $\pi = \pi_1 \cup \pi_2$. Since π completely evaluates every term in $T_1 \cup T_2$, there must be a $\pi' \in \mathsf{AF}(T_1 \cup T_2)$ such that $\pi' \subseteq \pi$. Because π' completely evaluates every term in T_1 and T_2, the minimality of π_1 and π_2 yields $\pi_1 \subseteq \pi'$ and $\pi_2 \subseteq \pi'$. Hence $\pi = \pi_1 \cup \pi_2 \subseteq \pi'$ and thus $\pi = \pi'$. □

The combination of Corollary 4.30 and Lemma 4.33 yields that $\mathsf{AF}(T)$ can be computed as $\bigotimes\{\mathsf{AF}(t, \epsilon) \mid t \in T\}$.

Definition 4.27 (for $\pi = \epsilon$) is easily extended to rewrite rules.

Definition 4.34. *For a rewrite rule $l \to r$ we define $\mathsf{AF}(l \to r) = \mathsf{AF}(\{l, r\})$ and $\mathsf{AF}_{\mathrm{vc}}(l \to r) = \{\pi \in \mathsf{AF}(l \to r) \mid \mathcal{V}\mathsf{ar}(\pi(r)) \subseteq \mathcal{V}\mathsf{ar}(\pi(l))\}$.*

The reason for excluding, in the definition of $\mathsf{AF}_{\mathrm{vc}}(l \to r)$, argument filterings π from $\mathsf{AF}(l \to r)$ that violate the variable condition $\mathcal{V}\mathsf{ar}(\pi(r)) \subseteq \mathcal{V}\mathsf{ar}(\pi(l))$ is simply that no simplification order $>$ satisfies $\pi(l) \gtrsim \pi(r)$ if some variable in $\pi(r)$ does not also occur in $\pi(l)$. If we know in advance which base order will be used to satisfy the simplified constraints, then we can do even better. In the following definition we illustrate this for LPO with strict precedence.

Definition 4.35. *Let $l \to r$ be a rewrite rule. We define $\mathsf{AF}_{\mathrm{lpo}}(l \to r) = \{\pi \in \mathsf{AF}(l \to r) \mid \pi(l) >^{=}_{\mathrm{lpo}} \pi(r)$ for some precedence $>\}$.*

The next example shows the effectiveness of (restricted) partial argument filterings.

Example 4.36. *Table 4.1 shows for each rule $l \to r$ the number of argument filterings in $\mathsf{AF}(\mathcal{F}\mathsf{un}(l \to r))$, $\mathsf{AF}(l \to r)$, $\mathsf{AF}_{\mathrm{vc}}(l \to r)$, $\mathsf{AF}_{\mathrm{lpo}}(l \to r)$, and $\mathsf{AF}_{\mathrm{lpo}}(l \to r)$.*

The idea is now to (1) compute all argument filterings for each constraint *separately* and (2) subsequently *merge* them to obtain the argument filterings of the full set of constraints.

Definition 4.37. *We define $\mathsf{AF}(\mathcal{R}) = \bigotimes\{\mathsf{AF}(l \to r) \mid l \to r \in \mathcal{R}\}$ for a set of rewrite rules $\mathcal{R}$. Furthermore, if A is a set of argument filterings then $A_{\mathrm{lpo}(\mathcal{R})} = \{\pi \in A \mid \pi(\mathcal{R}) \subseteq >^{=}_{\mathrm{lpo}}$ for some precedence $>\}$.*

From the previous lemma we obtain the following equality:

$$\mathsf{AF}(\mathcal{R}_1 \cup \mathcal{R}_2)_{\mathrm{lpo}(\mathcal{R}_1 \cup \mathcal{R}_2)} = (\mathsf{AF}(\mathcal{R}_1)_{\mathrm{lpo}(\mathcal{R}_1)} \otimes \mathsf{AF}(\mathcal{R}_2)_{\mathrm{lpo}(\mathcal{R}_2)})_{\mathrm{lpo}(\mathcal{R}_1 \cup \mathcal{R}_2)}$$

The divide and conquer approach is based on the observation that the right-hand side can be computed faster than a direct computation of the left-hand side. By using the equality repeatedly, $\mathcal{R}$ is eventually divided into sets of single

Table 4.1: Divide and conquer: quicksort (I).

$l \to r$	$\mathsf{AF}(\mathcal{F}\mathsf{un}(l \to r))$	$\mathsf{AF}(l \to r)$	$\mathsf{AF}_{\mathrm{vc}}(l \to r)$	$\mathsf{AF}_{\mathrm{lpo}}(l \to r)$
1	6	6	6	5
2	2376	981	327	281
3	6	6	6	5
4	2376	981	327	281
5	6	6	3	3
6	36	36	27	23
7	3	3	3	3
8	3888	513	282	151
9	396	231	108	96
10	396	216	102	97
11	396	216	102	97
12	396	231	108	96
13	6	6	6	5
14	18	12	12	11
15	18	16	11	11

rules, but the form is not unique. For example, if 1, 2, and 3 are rewrite rules then $\mathsf{AF}(\{1,2,3\})_{\mathrm{lpo}(\{1,2,3\})}$ can be divided in three different ways:

$$((\mathsf{AF}(\{1\})_{\mathrm{lpo}(\{1\})} \otimes \mathsf{AF}(\{2\})_{\mathrm{lpo}(\{2\})})_{\mathrm{lpo}(\{1,2\})}) \otimes \mathsf{AF}(\{3\})_{\mathrm{lpo}(\{3\})})_{\mathrm{lpo}(\{1,2,3\})}$$
$$((\mathsf{AF}(\{1\})_{\mathrm{lpo}(\{1\})} \otimes \mathsf{AF}(\{3\})_{\mathrm{lpo}(\{3\})})_{\mathrm{lpo}(\{1,3\})}) \otimes \mathsf{AF}(\{2\})_{\mathrm{lpo}(\{2\})})_{\mathrm{lpo}(\{1,2,3\})}$$
$$((\mathsf{AF}(\{2\})_{\mathrm{lpo}(\{2\})} \otimes \mathsf{AF}(\{3\})_{\mathrm{lpo}(\{3\})})_{\mathrm{lpo}(\{2,3\})}) \otimes \mathsf{AF}(\{1\})_{\mathrm{lpo}(\{1\})})_{\mathrm{lpo}(\{1,2,3\})}$$

We illustrate the divide and conquer approach on the TRS of Example 4.25. Here we use the merge order corresponding to the numbering of the rewrite rules.

Example 4.38. *Table 4.2 shows the cumulative effect of the merge operation. For instance, merging the 5 argument filterings for rule 1 with the 281 for rule 2 produces 279 argument filterings for the combination of rules 1 and 2. From the last entry in the table we see that only* 40 *out of* 8468064 *argument filterings enable the rule constraints to be solved by LPO with strict precedence.*

The divide and conquer approach can easily be combined with the heuristics of the previous subsection, just replace $\mathsf{AF}(\mathsf{outer}(t,\pi))$ in Definition 4.27 by $\mathsf{AF}^h(\mathsf{outer}(t,\pi))$ where h is the heuristic. With respect to Example 4.38, the *some more* heuristic would produce 16 and the *some* heuristic just 9 suitable argument filterings. This can be inferred from Table 4.2. An additional advantage of the divide and conquer approach is that the argument filterings for the usable rule constraints can be shared and reused among different cycles (SCCs) in a dependency graph.

In the above we restricted the sets that were computed after *every* merge operation by incorporating the underlying order. However, the cost of the orientability check can be high. This is especially true if one uses KBO as base

Table 4.2: Divide and conquer: quicksort (II).

h	1	2	3	4	5	6	7	8	9	10
all	5	279	1395	11579	34737	17368	52104	9637	5135	530
some more	3	49	147	581	1162	681	2043	333	75	57
some	2	25	50	161	322	186	372	78	20	20
h	11	12	13	14	15					
all	65	49	25	50	40					
some more	11	10	12	24	16					
some	3	3	6	9	9					

Table 4.3: Divide and conquer: quicksort (III).

h	1	3	5	7	9	11	13	15	2	4	6	8	10	12	14
all	5	165	104	10	50	218	44	28	84	45	25	50	150	120	40
some more	3	28	20	4	12	23	7	6	12	12	9	18	54	36	16
some	2	10	7	2	4	6	2	2	4	8	6	9	18	18	9

order, so we suggest to compute $\bigotimes\{\mathsf{AF}^h_{\mathrm{kbo}}(l \to r) \mid l \to r \in \mathcal{R}\}$. The experimental results reveal that this significantly improves the computation times (cf. Table 6.4 in Section 6.2) .

4.3.3 Dynamic Programming

The effectiveness of the divide and conquer approach depends very much on the merge order. Table 4.3 shows a different merge order for the rules of the quicksort example. Although the final outcome is the same, the intermediate results differ greatly.

In order to determine a good merge order, we use a dynamic programming technique.

Definition 4.39. *Let $\mathcal{R}$ be a set of rules over a signature $\mathcal{F}$. We put* $\mathrm{root}(\mathcal{R}) = \{\mathrm{root}(l), \mathrm{root}(r) \mid l \to r \in \mathcal{R}\} \cap \mathcal{F}$.

The key observation is that when merging two sets of argument filterings A_1 for $\mathcal{R}_1$ and A_2 for $\mathcal{R}_2$, the size of $A_1 \otimes A_2$ often decreases when $\mathrm{root}(\mathcal{R}_1) = \mathrm{root}(\mathcal{R}_2)$. In general, the size of $A_1 \otimes A_2$ increases with the size of $\mathrm{root}(\mathcal{R}_1 \cup \mathcal{R}_2)$. An argument filtering in A_1 cannot be combined with an argument filtering in A_2 if the compatibility condition in the definition of the merge operation (Definition 4.32) is not satisfied or if the orientability condition of the employed base order is not satisfied (cf. Definition 4.37). Obviously, the first possibility is more likely to happen if the domains of the two argument filterings have a large intersection. For the second condition, function symbols that appear at the root of terms in $\mathcal{R}_1 \cup \mathcal{R}_2$ have a larger impact than function symbols

Table 4.4: Divide and conquer: quicksort (IV).

$\mathcal{R}'$	$\varnothing$	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$	$\{5\}$	$\{6\}$	$\{7\}$	$\{8\}$	$\{9\}$	$\{10\}$
$\lvert\mathrm{A}(\mathcal{R}')\rvert$	1	5	281	5	281	3	23	3	151	96	97
$\lvert\mathrm{root}(\mathcal{R}')\rvert$	–	–	–	–	–	–	–	–	–	–	–

$\mathcal{R}'$	$\{11\}$	$\{12\}$	$\{13\}$	$\{14\}$	$\{15\}$	$\{14,15\}$	$\{2,10\}$	$\{4,11\}$
$\lvert\mathrm{A}(\mathcal{R}')\rvert$	97	96	5	11	11	10	15	15
$\lvert\mathrm{root}(\mathcal{R}')\rvert$	–	–	–	–	–	2	2	2

$\mathcal{R}'$	$\{5,8\}$	$\{3,7\}$	$\{13,14,15\}$	$\{2,9,10\}$	$\{4,11,12\}$	$\{5,6,8\}$
$\lvert\mathrm{A}(\mathcal{R}')\rvert$	83	15	8	11	11	54
$\lvert\mathrm{root}(\mathcal{R}')\rvert$	2	3	3	3	3	3

$\mathcal{R}'$	$\{1,3,7\}$	$\{2,4,9,10,11,12\}$	$\{5,6,8,13,14,15\}$
$\lvert\mathrm{A}(\mathcal{R}')\rvert$	75	35	432
$\lvert\mathrm{root}(\mathcal{R}')\rvert$	4	6	6

$\mathcal{R}'$	$\{1,2,3,4,7,9,10,11,12\}$	$\mathcal{R}$
$\lvert\mathrm{A}(\mathcal{R}')\rvert$	84	40
$\lvert\mathrm{root}(\mathcal{R}')\rvert$	7	11

that appear only below the root since the latter might disappear. Based on these observations, we now explain in some detail how the divide and conquer approach is implemented in our termination prover T$_\text{T}$T (cf. Chapter 6).

First we consider automation of Theorem 3.20, which is easier than Theorems 3.36 and 3.28. Suppose we want to compute $\mathsf{AF}(\mathcal{R})_{\mathrm{lpo}(\mathcal{R})}$. We create a table A consisting of pairs of sets of rewrite rules $\mathcal{R}' \subseteq \mathcal{R}$ and the corresponding sets of partial argument filterings $\mathsf{AF}(\mathcal{R}')_{\mathrm{lpo}(\mathcal{R}')}$. The table is initialized as follows:

$$\mathrm{A}(\varnothing) = \mathsf{AF}(\varnothing) \qquad\qquad \mathrm{A}(\{l \to r\}) = \mathsf{AF}(\{l \to r\})_{\mathrm{lpo}(\{l \to r\})}$$

for all $l \to r \in \mathcal{R}$. Let us write $\max_{\mathrm{A}}(\mathcal{R})$ for the set of maximal subsets $\mathcal{S} \subseteq \mathcal{R}$ such that $\mathrm{A}(\mathcal{S})$ is defined. So initially $\max_{\mathrm{A}}(\mathcal{R})$ consists of the set of all singleton subsets of $\mathcal{R}$. As long as $\max_{\mathrm{A}}(\mathcal{R})$ contains at least two sets, we choose two distinct sets $\mathcal{R}_1$ and $\mathcal{R}_2$ from $\max_{\mathrm{A}}(\mathcal{R})$ such that the size of $\mathrm{root}(\mathcal{R}_1 \cup \mathcal{R}_2)$ is minimal and we add the following entry to the table:

$$\mathrm{A}(\mathcal{R}_1 \cup \mathcal{R}_2) = (\mathrm{A}(\mathcal{R}_1) \otimes \mathrm{A}(\mathcal{R}_2))_{\mathrm{lpo}(\mathcal{R}_1 \cup \mathcal{R}_2)}$$

This process terminates if $\max_{\mathrm{A}}(\mathcal{R})$ equals $\{\mathcal{R}\}$, which means that $\mathrm{A}(\mathcal{R}) = \mathsf{AF}(\mathcal{R})_{\mathrm{lpo}(\mathcal{R})}$ has been computed.

Example 4.40. *For the fifteen rewrite rules of the TRS $\mathcal{R}$ of Example 4.25, after initializing the table, it turns out that $\lvert\mathrm{root}(\{14,15\})\rvert = \lvert\{\leq, \mathsf{false}\}\rvert = 2$ is minimal, so we add*

$$\mathrm{A}(\{14,15\}) = (\mathrm{A}(14) \otimes \mathrm{A}(15))_{\mathrm{lpo}(\{14,15\})}$$

to the table. Next the pair of $\{2\}$ and $\{10\}$ is selected. Continuing in this fashion, the data in Table 4.4 is computed (left to right, top to bottom).

When using the condition "$\models \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$" of Theorem 4.20 for proving termination, for the first SCC $\mathcal{S}$ we compute $\mathrm{A}(\mathcal{R} \cup \mathcal{S})$ by first computing $\mathrm{A}(\mathcal{S})$ and, if this set is non-empty, then computing $\mathrm{A}(\mathcal{R})$ before merging these two sets to get $\mathrm{A}(\mathcal{R} \cup \mathcal{S}) = (\mathrm{A}(\mathcal{R}) \otimes \mathrm{A}(\mathcal{S}))_{\mathrm{lpo}(\mathcal{R} \cup \mathcal{S})}$. The obvious reason is that the result of the computation of $\mathrm{A}(\mathcal{R})$ can be reused in combination with other SCCs, including newly generated ones.

If one incorporates the usable rules of Theorems 3.36 and 3.28 with the recursive SCC algorithm, it does not make sense to compute $\mathrm{A}(\mathcal{R})$, because different SCCs (may) have different usable rules and some rewrite rules may not be usable at all. Rather, we compute $\mathrm{A}(\mathcal{R}')$ for suitable subsets of $\mathcal{R}$ on demand. This is illustrated in the following example.

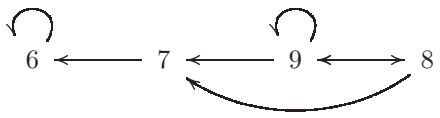
Example 4.41. *Consider the following TRS $\mathcal{R}$ (from [5]):*

$$\begin{aligned}
&1\colon\ \mathsf{0} + y \to y && 3\colon\ \mathsf{quot}(x, \mathsf{0}, \mathsf{s}(z)) \to \mathsf{s}(\mathsf{quot}(x, z + \mathsf{s}(\mathsf{0}), \mathsf{s}(z)))\\
&2\colon\ \mathsf{s}(x) + y \to \mathsf{s}(x + y) && 4\colon\ \mathsf{quot}(\mathsf{0}, \mathsf{s}(y), \mathsf{s}(z)) \to \mathsf{0}\\
& && 5\colon\ \mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y), z) \to \mathsf{quot}(x, y, z)
\end{aligned}$$

Since $\mathcal{R}$ is non-overlapping, it is sufficient to prove innermost termination by Theorem 2.22. There are four dependency pairs:

$$\begin{aligned}
6\colon&\ \mathsf{s}(x) +^\sharp y \to x +^\sharp y\\
7\colon&\ \mathsf{quot}^\sharp(x, \mathsf{0}, \mathsf{s}(z)) \to z +^\sharp \mathsf{s}(\mathsf{0})\\
8\colon&\ \mathsf{quot}^\sharp(x, \mathsf{0}, \mathsf{s}(z)) \to \mathsf{quot}^\sharp(x, z + \mathsf{s}(\mathsf{0}), \mathsf{s}(z))\\
9\colon&\ \mathsf{quot}^\sharp(\mathsf{s}(x), \mathsf{s}(y), z) \to \mathsf{quot}^\sharp(x, y, z)
\end{aligned}$$

The EIDG *or* EIDG* *approximated innermost dependency graph*



contains two SCCs: $\mathcal{S}_1 = \{6\}$, $\mathcal{S}_2 = \{8, 9\}$.

- *First consider the SCC $\mathcal{S}_1$. Since $\mathcal{U}(\mathcal{S}_1) = \varnothing$ and $\mathcal{S}_1$ has just one element, initializing the table* A *will produce*

  $$\mathrm{A}(\varnothing) = \mathsf{AF}(\varnothing) \qquad \mathrm{A}(\{6\}) = \mathsf{AF}(\{6\})_{\mathrm{lpo}(\{6\})}$$

  Since $\mathrm{A}(\{6\})$ contains a suitable argument filtering, we have $\models_{\mathsf{i}} S_1$.

- *Next we consider the constraints for SCC $\mathcal{S}_2$. We have $\mathcal{U}(\mathcal{S}_2) = \{1, 2\}$, so we add the following entries to our table:*

  $$\begin{aligned}
  \mathrm{A}(\{1\}) &= \mathsf{AF}(\{1\})_{\mathrm{lpo}(\{1\})} & \mathrm{A}(\{8\}) &= \mathsf{AF}(\{8\})_{\mathrm{lpo}(\{8\})}\\
  \mathrm{A}(\{2\}) &= \mathsf{AF}(\{2\})_{\mathrm{lpo}(\{2\})} & \mathrm{A}(\{9\}) &= \mathsf{AF}(\{9\})_{\mathrm{lpo}(\{9\})}
  \end{aligned}$$

  We want to compute $\mathrm{A}(\mathcal{U}(\mathcal{S}_2) \cup \mathcal{S}_2)$ by merging $\mathrm{A}(\mathcal{U}(\mathcal{S}_2))$ and $\mathrm{A}(\mathcal{S}_2)$ since it is more likely that the two partial results can be reused than some mixture of elements of both $\mathrm{A}(\mathcal{U}(\mathcal{S}_2))$ and $\mathrm{A}(\mathcal{S}_2)$. So we compute $\mathrm{A}(\{1, 2\})$ and $\mathrm{A}(\{8, 9\})$:

  $$\begin{aligned}
  \mathrm{A}(\{1, 2\}) &= (\mathrm{A}(\{1\}) \otimes \mathrm{A}(\{2\}))_{\mathrm{lpo}(\{1,2\})}\\
  \mathrm{A}(\{8, 9\}) &= (\mathrm{A}(\{8\}) \otimes \mathrm{A}(\{9\}))_{\mathrm{lpo}(\{8,9\})}
  \end{aligned}$$

and then we compute $\mathrm{A}(\{1,2,8,9\})$ *by merging the results:*

$$\mathrm{A}(\{1,2,8,9\}) = (\mathrm{A}(\{1,2\}) \otimes \mathrm{A}(\{8,9\}))_{\mathrm{lpo}(\{1,2,8,9\})}$$

In $\mathrm{A}(\{1,2,8,9\})$ *we find an argument filtering that makes rule* 9 *strictly decreasing. By the construction of* A*, all (other) rules are weakly decreasing, so SCC* $\mathcal{S}_2$ *gives rise to the new SCC* $\mathcal{S}_3 = \{8\}$*.*

- *We have* $\mathcal{U}(\mathcal{S}_3) = \{1,2\}$*, so we have to compute* $\mathrm{A}(\{1,2,8\})$*. An obvious search through the table reveals that* $\max_{\mathrm{A}}(\{1,2,8\}) = \{\{1,2\},\{8\}\}$*. So the computation of* $\mathrm{A}(\{1,2,8\})$ *involves just one merge operation:*

$$\mathrm{A}(\{1,2,8\}) = (\mathrm{A}(\{1,2\}) \otimes \mathrm{A}(\{8\}))_{\mathrm{lpo}(\{1,2,8\})}$$

Since $\mathrm{A}(\{1,2,8\})$ *contains a suitable argument filtering (i.e., an argument filtering that makes rule* 8 *strictly decreasing), the constraints for SCC* $\mathcal{S}_3$ *are solved, i.e., we have* $\models_{\mathsf{i}} \mathcal{S}_3$ *and thus also* $\models_{\mathsf{i}} \mathcal{S}_2$*.*

Hence $\mathcal{R}$ *is (innermost) terminating. The following table summarizes the divide and conquer process:*

$\mathcal{R}'$	$\varnothing$	$\{6\}$	$\{1\}$	$\{2\}$	$\{8\}$	$\{9\}$	$\{1,2\}$	$\{8,9\}$	$\{1,2,8,9\}$	$\{1,2,8\}$
$\lvert\mathrm{A}(\mathcal{R}')\rvert$	1	9	3	14	36	19	6	28	35	46

We conclude this section by mentioning a different approach to search for suitable argument filterings. In [26] one always starts with the dependency pairs. Given an SCC $\mathcal{S}$, a single argument filtering π is selected that makes one pair in $\mathcal{S}$ strictly decreasing and all other pairs weakly decreasing. This argument filtering is then extended to handle the rule constraints in a step-wise fashion. A depth-first search algorithm is used to explore the search space. The advantage of this approach is that the computationally expensive merge operation is avoided, but we see two disadvantages. First of all, a wrong choice in the selection of the dependency pair that must be strictly decreasing causes backtracking. Secondly, if there is no solution the whole search space must be explored before this is detected whereas in the divide and conquer approach the search is terminated as soon as an empty set of argument filterings is produced.

Chapter 5

Polynomial Orders

In Chapter 3 we used polynomial interpretations with non-negative integer coefficients to construct reduction triples. In this chapter we show that polynomial interpretations over the integers with negative coefficients like $x-1$ and $x-y+1$ can also be used for constructing reduction triples. In Section 5.1 we consider polynomials with negative constants. In Section 5.2 we show how non-constant coefficients can be negative integers by using a different construction of reduction triples.

5.1 Negative Constants

To make the discussion more concrete, let us consider a somewhat artificial example: the recursive definition

$$f(x) = \text{if } x > 0 \text{ then } f(f(x-1)) + 1 \text{ else } 0$$

from [15]. It computes the identity function over the natural numbers. Termination of the rewrite system

$$\begin{aligned} 1\colon&\ \mathsf{f}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{f}(\mathsf{f}(\mathsf{p}(\mathsf{s}(x))))) \\ 2\colon&\ \mathsf{f}(\mathsf{0}) \to \mathsf{0} \\ 3\colon&\ \mathsf{p}(\mathsf{s}(x)) \to x \end{aligned}$$

obtained after the obvious translation is not easily proved. The (manual) proof in [15] relies on forward closures whereas Theorem 3.36 with any reduction triple based on a simplification order fails.[1] There are three dependency pairs:

$$\begin{aligned} 4\colon&\ \mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(\mathsf{f}(\mathsf{p}(\mathsf{s}(x)))) \\ 5\colon&\ \mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{f}^\sharp(\mathsf{p}(\mathsf{s}(x))) \\ 6\colon&\ \mathsf{f}^\sharp(\mathsf{s}(x)) \to \mathsf{p}^\sharp(\mathsf{s}(x)) \end{aligned}$$

and the dependency graph contains one SCC: $\mathcal{C} = \{4, 5\}$. All rules of $\mathcal{R}$ are usable. By taking the *natural* polynomial interpretation

$$\mathsf{f}_\mathbb{Z}(x) = \mathsf{f}^\sharp_\mathbb{Z}(x) = x \qquad \mathsf{s}_\mathbb{Z}(x) = x + 1 \qquad \mathsf{0}_\mathbb{Z} = 0 \qquad \mathsf{p}_\mathbb{Z}(x) = x - 1$$

[1] See Section 6.3 for a discussion how existing automatic tools perform on this example.

over the integers, the rule and dependency pair constraints reduce to the following inequalities:

$$\begin{array}{llll} 1\colon & x+1 \geqslant x+1 & 3\colon & x \geqslant x \\ 2\colon & 0 \geqslant 0 & 4,5\colon & x+1 > x \end{array}$$

These constraints are obviously satisfied. The question is whether we are allowed to conclude termination at this point. We will argue that the answer is affirmative and, moreover, that the search for appropriate natural interpretations can be efficiently implemented.

5.1.1 Theoretical Framework

When using polynomial interpretations with negative constants, the first challenge we face is that the standard order $>$ on $\mathbb{Z}$ is not well-founded. Restricting the domain to the set $\mathbb{N}$ of natural numbers makes an interpretation like $\mathsf{p}_{\mathbb{Z}}(x) = x - 1$ ill-defined. Dershowitz and Hoot observe in [15] that if all (instantiated) subterms in the rules of the TRS are interpreted as non-negative integers, such interpretations can work correctly. Following their observation, we propose to modify the interpretation of p to $\mathsf{p}_{\mathbb{N}}(x) = \max\{0, x-1\}$.

Definition 5.1. *Let $\mathcal{F}$ be a signature and let $(\mathbb{Z}, \{f_{\mathbb{Z}}\}_{f\in\mathcal{F}})$ be an $\mathcal{F}$-algebra such that every interpretation function $f_{\mathbb{Z}}$ is weakly monotone in all its arguments. The interpretation functions of the* induced *algebra $(\mathbb{N}, \{f_{\mathbb{N}}\}_{f\in\mathcal{F}})$ are defined as follows: $f_{\mathbb{N}}(x_1, \ldots, x_n) = \max\{0, f_{\mathbb{Z}}(x_1, \ldots, x_n)\}$ for all $x_1, \ldots, x_n \in \mathbb{N}$.*

With respect to the interpretations in the example at the beginning of Section 5.1, we obtain $\mathsf{s}_{\mathbb{N}}(\mathsf{p}_{\mathbb{N}}(x)) = \max\{0, \max\{0, x-1\}+1\} = \max\{0, x-1\}+1$, $\mathsf{p}_{\mathbb{N}}(\mathsf{0}_{\mathbb{N}}) = \max\{0,0\} = 0$, and $\mathsf{p}_{\mathbb{N}}(\mathsf{s}_{\mathbb{N}}(x)) = \max\{0, \max\{0, x+1\}-1\} = x$.

Lemma 5.2. *If $(\mathbb{Z}, \{f_{\mathbb{Z}}\}_{f\in\mathcal{F}})$ is an $\mathcal{F}$-algebra with weakly monotone interpretations then $(\geqslant_{\mathbb{N}}, \geqslant_{\mathbb{N}}, >_{\mathbb{N}})$ is a reduction triple.*

Proof. It is easy to show that the interpretation functions of the induced algebra are weakly monotone in all arguments. Routine arguments reveal that the relation $>_{\mathbb{N}}$ is a well-founded order which is closed under substitutions and that $\geqslant_{\mathbb{N}}$ is a preorder closed under contexts and substitutions. Moreover, the identity $>_{\mathbb{N}} \cdot \geqslant_{\mathbb{N}} = >_{\mathbb{N}}$ holds. Hence $(\geqslant_{\mathbb{N}}, \geqslant_{\mathbb{N}}, >_{\mathbb{N}})$ is a reduction triple. □

The reduction triples $(\geqslant_{\mathbb{N}}, \geqslant_{\mathbb{N}}, >_{\mathbb{N}})$ of this section can be used in connection with Theorem 3.36 because they can be made $\mathcal{C}_{\mathcal{E}}$-compatible by simply defining $\mathsf{cons}_{\mathbb{N}}(x, y) = \max\{x, y\}$.

Corollary 5.3. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and let $\mathcal{C}$ be a cycle in $\mathsf{DG}(\mathcal{R})$. If there exist an $\mathcal{F}$-algebra $(\mathbb{Z}, \{f_{\mathbb{Z}}\}_{f\in\mathcal{F}})$ with weakly monotone interpretations such that $\mathcal{U}(\mathcal{C}) \cup \mathcal{C} \subseteq \geqslant_{\mathbb{N}}$ and $\mathcal{C} \cap >_{\mathbb{N}} \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.* □

Note that there is no need for argument filterings here since their effect can be incorporated in the definition of the interpretation functions.

It is interesting to remark that unlike usual polynomial interpretations, the relation $>_{\mathbb{N}}$ does not have the (weak) subterm property. For instance, with respect to the interpretations in the example at the beginning of Section 5.1, we have $\mathsf{s}(\mathsf{0}) >_{\mathbb{N}} \mathsf{p}(\mathsf{s}(\mathsf{0}))$ and not $\mathsf{p}(\mathsf{s}(\mathsf{0})) >_{\mathbb{N}} \mathsf{p}(\mathsf{0})$.

Example 5.4. *Consider the TRS consisting of the following rewrite rules:*

$$\begin{array}{lll} 1: & \mathsf{half}(0) \to 0 & 4: \quad \mathsf{bits}(0) \to 0 \\ 2: & \mathsf{half}(\mathsf{s}(0)) \to 0 & 5: \quad \mathsf{bits}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{bits}(\mathsf{half}(\mathsf{s}(x)))) \\ 3: & \mathsf{half}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{half}(x)) & \end{array}$$

The function $\mathsf{half}(x)$ *computes* $\lceil \frac{x}{2} \rceil$ *and* $\mathsf{bits}(x)$ *computes the number of bits that are needed to represent all numbers less than or equal to* x*. Termination of this TRS is proved in [5] by using the dependency pair method together with the narrowing refinement. There are three dependency pairs:*

$$\begin{array}{ll} 6: & \mathsf{half}^\sharp(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{half}^\sharp(x) \\ 7: & \mathsf{bits}^\sharp(\mathsf{s}(x)) \to \mathsf{bits}^\sharp(\mathsf{half}(\mathsf{s}(x))) \\ 8: & \mathsf{bits}^\sharp(\mathsf{s}(x)) \to \mathsf{half}^\sharp(\mathsf{s}(x)) \end{array}$$

and $\mathsf{DG}(\mathcal{R})$ *contains two SCCs:* $\{6\}$ *and* $\{7\}$*. The SCC* $\{6\}$ *is handled by the subterm criterion with the simple projection* $\pi(\mathsf{half}^\sharp) = 1$*. Consider the SCC* $\{7\}$*. The usable rules are* $\mathcal{U}(\{7\}) = \{1, 2, 3\}$*. By taking the interpretations* $0_\mathbb{Z} = 0$*,* $\mathsf{half}_\mathbb{Z}(x) = x - 1$*,* $\mathsf{bits}_\mathbb{Z}(x) = \mathsf{half}^\sharp_\mathbb{Z}(x) = x$*, and* $\mathsf{s}_\mathbb{Z}(x) = \mathsf{bits}^\sharp_\mathbb{Z}(x) = x + 1$*, we obtain the following constraints over* $\mathbb{N}$*:*

$$1, 2: \quad 0 \geqslant 0 \qquad 3: \quad x + 1 \geqslant \max\{0, x - 1\} + 1 \qquad 7: \quad x + 2 > x + 1$$

5.1.2 Towards Automation

The constraints in Example 5.4 are satisfied. However, how can an inequality like $x + 1 \geqslant \max\{0, x - 1\} + 1$ be verified automatically? Because the inequalities resulting from interpretations with negative constants may contain the max operator, we cannot use standard techniques for comparing polynomial expressions. In order to avoid reasoning by case analysis ($x - 1 > 0$ or $x - 1 \leqslant 0$ for constraint 3 in Example 5.4), we approximate the evaluation function of the induced algebra.

Definition 5.5. *Given a polynomial* P *with coefficients in* $\mathbb{Z}$*, we denote the constant part by* $c(P)$ *and the non-constant part* $P - c(P)$ *by* $n(P)$*. Let* $(\mathbb{Z}, \{f_\mathbb{Z}\}_{f \in \mathcal{F}})$ *be an* $\mathcal{F}$*-algebra such that every* $f_\mathbb{Z}$ *is a weakly monotone polynomial. With every term* t *we associate polynomials* $P_{left}(t)$ *and* $P_{right}(t)$ *with coefficients in* $\mathbb{Z}$ *and variables in* t *as indeterminates:*

$$P_{left}(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ 0 & \text{if } t = f(t_1, \ldots, t_n),\ n(P_1) = 0, \text{ and } c(P_1) < 0 \\ P_1 & \text{otherwise} \end{cases}$$

where $P_1 = f_\mathbb{Z}(P_{left}(t_1), \ldots, P_{left}(t_n))$ *and*

$$P_{right}(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ n(P_2) & \text{if } t = f(t_1, \ldots, t_n) \text{ and } c(P_2) < 0 \\ P_2 & \text{otherwise} \end{cases}$$

where $P_2 = f_{\mathbb{Z}}(P_{right}(t_1), \ldots, P_{right}(t_n))$. *Let* $\alpha\colon \mathcal{V} \to \mathbb{N}$ *be an assignment. The result of evaluating* $P_{left}(t)$ *and* $P_{right}(t)$ *under* α *is denoted by* $[\alpha]^l_{\mathbb{Z}}(t)$ *and* $[\alpha]^r_{\mathbb{Z}}(t)$. *Furthermore, the result of evaluating a polynomial* P *under* α *is denoted by* $\alpha(P)$.

According the following lemma, $P_{left}(t)$ is a lower bound and $P_{right}(t)$ is an upper bound of the interpretation of t in the induced algebra.

Lemma 5.6. *Let* $(\mathbb{Z}, \{f_{\mathbb{Z}}\}_{f\in\mathcal{F}})$ *be an* $\mathcal{F}$*-algebra such that every* $f_{\mathbb{Z}}$ *is a weakly monotone polynomial. Let* t *be a term. For every assignment* $\alpha\colon \mathcal{V} \to \mathbb{N}$ *we have* $[\alpha]^r_{\mathbb{Z}}(t) \geqslant [\alpha]_{\mathbb{N}}(t) \geqslant [\alpha]^l_{\mathbb{Z}}(t)$.

Proof. By induction on the structure of t. If $t \in \mathcal{V}$ then $[\alpha]^r_{\mathbb{Z}}(t) = [\alpha]^l_{\mathbb{Z}}(t) = \alpha(t) = [\alpha]_{\mathbb{N}}(t)$. Suppose $t = f(t_1, \ldots, t_n)$. According to the induction hypothesis, $[\alpha]^r_{\mathbb{Z}}(t_i) \geqslant [\alpha]_{\mathbb{N}}(t_i) \geqslant [\alpha]^l_{\mathbb{Z}}(t_i)$ for all i. Since $f_{\mathbb{Z}}$ is weakly monotone,

$$\begin{aligned} f_{\mathbb{Z}}([\alpha]^r_{\mathbb{Z}}(t_1), \ldots, [\alpha]^r_{\mathbb{Z}}(t_n)) &\geqslant f_{\mathbb{Z}}([\alpha]_{\mathbb{N}}(t_1), \ldots, [\alpha]_{\mathbb{N}}(t_n)) \\ &\geqslant f_{\mathbb{Z}}([\alpha]^l_{\mathbb{Z}}(t_1), \ldots, [\alpha]^l_{\mathbb{Z}}(t_n)) \end{aligned}$$

By applying the weakly monotone function $\max\{0, \cdot\}$ we obtain $\max\{0, \alpha(P_2)\} \geqslant [\alpha]_{\mathbb{N}}(t) \geqslant \max\{0, \alpha(P_1)\}$ where $P_1 = f_{\mathbb{Z}}(P_{left}(t_1), \ldots, P_{left}(t_n))$ and $P_2 = f_{\mathbb{Z}}(P_{right}(t_1), \ldots, P_{right}(t_n))$. We have

$$[\alpha]^l_{\mathbb{Z}}(t) = \begin{cases} 0 & \text{if } n(P_1) = 0 \text{ and } c(P_1) < 0 \\ \alpha(P_1) & \text{otherwise} \end{cases}$$

and thus $[\alpha]^l_{\mathbb{Z}}(t) \leqslant \max\{0, \alpha(P_1)\}$. Likewise,

$$[\alpha]^r_{\mathbb{Z}}(t) = \begin{cases} \alpha(n(P_2)) & \text{if } c(P_2) < 0 \\ \alpha(P_2) & \text{otherwise} \end{cases}$$

In the former case, $\alpha(n(P_2)) = \alpha(P_2) - c(P_2) > \alpha(P_2)$ and $\alpha(n(P_2)) \geqslant 0$. In the latter case $\alpha(P_2) \geqslant 0$. So in both cases we have $[\alpha]^r_{\mathbb{Z}}(t) \geqslant \max\{0, \alpha(P_2)\}$. Hence we obtain the desired inequalities. □

Corollary 5.7. *Let* $(\mathbb{Z}, \{f_{\mathbb{Z}}\}_{f\in\mathcal{F}})$ *be an* $\mathcal{F}$*-algebra such that every* $f_{\mathbb{Z}}$ *is a weakly monotone polynomial. Let* s *and* t *be terms. If* $P_{left}(s) - P_{right}(t) > 0$ *then* $s >_{\mathbb{N}} t$. *If* $P_{left}(s) - P_{right}(t) \geqslant 0$ *then* $s \geqslant_{\mathbb{N}} t$. □

Example 5.8. *Consider again the TRS of Example 5.4. By applying* P_{left} *to the left-hand sides and* P_{right} *to the right-hand sides of the rewrite rules and the dependency pairs, the following ordering constraints are obtained:*

$$1\colon\ 0 \geqslant 0 \qquad 2\colon\ 0 \geqslant 0 \qquad 3\colon\ x+1 \geqslant x+1 \qquad 7\colon\ x+2 > x+1$$

The only difference with the constraints in Example 5.4 is the interpretation of the term $\mathsf{s}(\mathsf{half}(x))$ *on the right-hand side of rule 3. We have* $P_{right}(\mathsf{half}(x)) = n(x-1) = x$ *and thus* $P_{right}(\mathsf{s}(\mathsf{half}(x))) = x+1$. *Although* $x+1$ *is less precise than* $\max\{0, x-1\} + 1$, *it is accurate enough to solve the ordering constraint resulting from rule 3.*

So once the interpretations $f_{\mathbb{Z}}$ are determined, we transform a rule $l \to r$ into the polynomial $P_{left}(l) - P_{right}(r)$. Standard techniques (cf. [34]) can then be used to test whether this polynomial is positive (or non-negative) for all values in $\mathbb{N}$ for the variables. The remaining question is how to find suitable interpretations for the function symbols. This problem will be discussed in Chapter 6.1.

5.2 Negative Coefficients

Let us start with an example which shows that negative coefficients in polynomial interpretations can be useful.

Example 5.9. *Consider the following variation of a TRS in [5]:*

$$\begin{array}{lll}
1: & 0 \leqslant y \to \mathsf{true} & 7: \quad x - 0 \to x \\
2: & \mathsf{s}(x) \leqslant 0 \to \mathsf{false} & 8: \quad \mathsf{s}(x) - \mathsf{s}(y) \to x - y \\
3: & \mathsf{s}(x) \leqslant \mathsf{s}(y) \to x \leqslant y & 9: \quad x - x \to 0 \\
4: & \mathsf{mod}(0, \mathsf{s}(y)) \to 0 & 10: \quad \mathsf{if}(\mathsf{true}, x, y) \to x \\
5: & \mathsf{mod}(\mathsf{s}(x), 0) \to 0 & 11: \quad \mathsf{if}(\mathsf{false}, x, y) \to y \\
6: & \mathsf{mod}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{if}(y \leqslant x, \mathsf{mod}(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(y)), \mathsf{s}(x)) &
\end{array}$$

There are six dependency pairs:

$$\begin{array}{ll}
12: & \mathsf{s}(x) \leqslant^\sharp \mathsf{s}(y) \to x \leqslant^\sharp y \\
13: & \mathsf{s}(x) -^\sharp \mathsf{s}(y) \to x -^\sharp y \\
14: & \mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{if}^\sharp(y \leqslant x, \mathsf{mod}(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(y)), \mathsf{s}(x)) \\
15: & \mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to y \leqslant^\sharp x \\
16: & \mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{mod}^\sharp(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(y)) \\
17: & \mathsf{mod}^\sharp(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{s}(x) -^\sharp \mathsf{s}(y)
\end{array}$$

The dependency graph contains three SCCs: $\{12\}$, $\{13\}$, *and* $\{16\}$. *The first two are handled by the subterm criterion (take* $\pi(\geqslant^\sharp) = 1$, *and* $\pi(-^\sharp) = 1$*). The SCC* $\{16\}$ *is problematic. The usable rules are* $\mathcal{U}(\{16\}) = \{7, 8, 9\}$. *We need to find a reduction triple* $(\gtrsim, \geqslant, >)$ *such that rules 7, 8, and 9 are weakly decreasing (i.e., compatible with* $\gtrsim$*) and dependency pair 16 is strictly decreasing (with respect to* $>$*). The only way to achieve the latter is by using the observation that* $\mathsf{s}(x)$ *is semantically greater than the syntactically larger term* $\mathsf{s}(x) - \mathsf{s}(y)$. *If we take the natural interpretation* $-_{\mathbb{Z}}(x, y) = x - y$, $\mathsf{s}_{\mathbb{Z}}(x) = x - 1$, *and* $0_{\mathbb{Z}} = 0$, *together with* $\mathsf{mod}^\sharp_{\mathbb{Z}}(x, y) = x$ *then we obtain the following induced ordering constraints over the natural numbers:*

$$\begin{array}{rc}
7: & x \geqslant x \\
8: & \max\{0, x - y\} \geqslant \max\{0, x - y\} \\
9: & 0 \geqslant 0 \\
16: & x + 1 > \max\{0, x - y\}
\end{array}$$

which are obviously satisfied. However, are we allowed to use an interpretation like $-_{\mathbb{Z}}(x, y) = x - y$ *in termination proofs?*

5.2.1 Theoretical Framework

The answer to the question in Example 5.9 appears to be negative because Lemma 5.2 no longer holds. Because the induced interpretation $-_{\mathbb{N}}(x,y) = \max\{0, x-y\}$ is not weakly monotone in its second argument, the order $\geqslant_{\mathbb{N}}$ of the induced algebra is *not* closed under contexts, so if $s \geqslant_{\mathbb{N}} t$ then it may happen that $C[s] \leqslant_{\mathbb{N}} C[t]$. Consequently, we do not obtain a reduction triple. However, if we have $s =_{\mathbb{N}} t$ rather than $s \geqslant_{\mathbb{N}} t$, closure under contexts is obtained for free. Here, for an algebra $\mathcal{A}$, we write $s =_{\mathcal{A}} t$ if $[\alpha]_{\mathcal{A}}(s) = [\alpha]_{\mathcal{A}}(t)$ for all assignments α. Note that the relation $=_{\mathbb{N}} \cup >_{\mathbb{N}}$ is properly contained in $\geqslant_{\mathbb{N}}$ and hence the reduction triple $(=_{\mathbb{N}}, \geqslant_{\mathbb{N}}, >_{\mathbb{N}})$ outperforms the reduction pair $(=_{\mathbb{N}}, >_{\mathbb{N}})$, i.e., the reduction triple $(=_{\mathbb{N}}, =_{\mathbb{N}} \cup >_{\mathbb{N}}, >_{\mathbb{N}})$. If we use the latter, all dependency pairs in a cycle have to be compatible with $=_{\mathbb{N}} \cup >_{\mathbb{N}}$, which is rather restrictive because dependency pairs that are transformed into a polynomial constraint of the form $x^2 \geqslant x$ or $x + 2y \geqslant x + y$ cannot be handled. The lemma and corollary below states the soundness of our approach in a more abstract setting.

Lemma 5.10. *Let $\mathcal{A}$ be an $\mathcal{F}$-algebra equipped with a well-founded order $>$. The triple $(=_{\mathcal{A}}, \geqslant_{\mathcal{A}}, >_{\mathcal{A}})$ is a reduction triple.* □

Corollary 5.11. *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in its dependency graph. If there exists an algebra $\mathcal{A}$ equipped with a well-founded order $>$ such that $\mathcal{R} \subseteq =_{\mathcal{A}}$, $\mathcal{C} \subseteq \geqslant_{\mathcal{A}}$, and $\mathcal{C} \cap >_{\mathcal{A}} \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.* □

The constraint $\mathcal{R} \subseteq =_{\mathcal{A}}$ in Corollary 5.11 means that $\mathcal{A}$ is a model of $\mathcal{R}$. It cannot be weakened to $\mathcal{U}(\mathcal{C}) \subseteq =_{\mathcal{A}}$. The reason is that $\mathcal{C}_{\mathcal{E}}$ does not admit any nontrivial models; in any model $\mathcal{A}$ of $\mathcal{C}_{\mathcal{E}}$ we have $x = \mathsf{cons}_{\mathcal{A}}(x,y) = y$ for all x, y in the carrier of $\mathcal{A}$. This is a problem for the TRS in Example 5.9. There we have $\mathcal{U}(\{16\}) \subseteq =_{\mathbb{N}}$ but one easily checks that $\mathcal{R} \subseteq =_{\mathbb{N}}$ implies $\mathsf{mod}_{\mathbb{N}}(x,y) = x \bmod y$, which cannot be represented with polynomials of finite degree. The following example shows that replacing $\mathcal{R} \subseteq =_{\mathcal{A}}$ by $\mathcal{U}(\mathcal{C}) \subseteq =_{\mathcal{A}}$ is actually unsound.

Example 5.12. *Consider the following non-terminating TRS $\mathcal{R}_1 \cup \mathcal{R}_2$ of Example 1.2. The only dependency pair $\mathsf{f}^\sharp(\mathsf{a},\mathsf{b},x) \to \mathsf{f}^\sharp(x,x,x)$ forms a cycle in the dependency graph. There are no usable rules. If we take the polynomial interpretation $\mathsf{a}_{\mathbb{Z}} = 1$, $\mathsf{b}_{\mathbb{Z}} = 0$, and $\mathsf{f}^\sharp_{\mathbb{Z}}(x,y,z) = x - y$ then the dependency pair is transformed into $1 - 0 = 1 > 0 = x - x$.*

In order to conclude the absence of $\mathcal{C}$-minimal *innermost* rewrite sequences we are allowed to replace $\mathcal{R}$ by $\mathcal{U}(\mathcal{C})$. This follows from Theorem 3.28.

Corollary 5.13. *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in its innermost dependency graph. If there exists an algebra $\mathcal{A}$ equipped with a well-founded order $>$ such that $\mathcal{U}(\mathcal{C}) \subseteq =_{\mathcal{A}}$, $\mathcal{C} \subseteq \geqslant_{\mathcal{A}}$, and $\mathcal{C} \cap >_{\mathcal{A}} \neq \varnothing$ then there are no $\mathcal{C}$-minimal innermost rewrite sequences.* □

With this result the innermost termination of the TRS in Example 5.9 is proved. Since the TRS in question is a locally confluent overlay system, by a result of Gramlich [28] this suffices to conclude termination. However, the usual syntactic condition to ensure local confluence, viz. non-overlappingness (cf. Theorem 2.22), is not fulfilled.

In the following we prove that also for termination we may replace $\mathcal{R} \subseteq =_{\mathcal{A}}$ by $\mathcal{U}(\mathcal{C}) \subseteq =_{\mathcal{A}}$, provided $>$ is a well-order and, more importantly, $\mathcal{A}$ regards $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ as right-linear. The latter concept is defined as follows.

Definition 5.14. *A linear term s is called a* linearization *of a term t if $s = t\sigma$ for some variable substitution σ. Let $\mathcal{A}$ be an algebra. A TRS $\mathcal{R}$ is* $\mathcal{A}$-right-linear *if for every rule $l \to r \in \mathcal{R}$ there exists a linearization r' of r such that $r =_{\mathcal{A}} r'$.*

The following definition introduces a new algebraic construction that is used to prove the desired result.

Definition 5.15. *Let $\mathcal{F}$ be a signature and let $\mathcal{A} = (A, \{f_{\mathcal{A}}\}_{f \in \mathcal{F}})$ be an $\mathcal{F}$-algebra equipped with a well-order $>$. Let $\mathbb{F}$ be the set of all nonempty finite subsets of A. We define the* set extension *of $\mathcal{A}$ as the $(\mathcal{F} \cup \{\mathsf{cons}\})$-algebra with carrier $\mathbb{F}$ and interpretations $\mathsf{cons}_{\mathbb{F}}(X, Y) = X \cup Y$ and*

$$f_{\mathbb{F}}(X_1, \dots, X_n) = \{f_{\mathcal{A}}(x_1, \dots, x_n) \mid x_1 \in X_1, \dots, x_n \in X_n\}$$

for all $f \in \mathcal{F}$. The relations $\supseteq_{\mathbb{F}}$, $\geqslant_{\mathbb{F}}$, and $>_{\mathbb{F}}$ are defined on terms as follows:

$$\begin{aligned} s \supseteq_{\mathbb{F}} t \quad & \text{if } [\alpha]_{\mathbb{F}}(s) \supseteq [\alpha]_{\mathbb{F}}(t) \\ s \geqslant_{\mathbb{F}} t \quad & \text{if } \max([\alpha]_{\mathbb{F}}(s)) \geqslant \max([\alpha]_{\mathbb{F}}(t)) \\ s >_{\mathbb{F}} t \quad & \text{if } \max([\alpha]_{\mathbb{F}}(s)) > \max([\alpha]_{\mathbb{F}}(t)) \end{aligned}$$

for all assignments $\alpha \colon \mathcal{V} \to \mathbb{F}$.

Note that since $[\alpha]_{\mathbb{F}}(u)$ is a finite nonempty set for every term u and $>$ is a well-order, the relations $\geqslant_{\mathbb{F}}$ and $>_{\mathbb{F}}$ are well-defined.

Lemma 5.16. *The triple $(\supseteq_{\mathbb{F}}, \geqslant_{\mathbb{F}}, >_{\mathbb{F}})$ is a $\mathcal{C}_{\mathcal{E}}$-compatible reduction triple.*

Proof. The relations $\supseteq_{\mathbb{F}}$ and $\geqslant_{\mathbb{F}}$ are clearly preorders. Closure under contexts of $\supseteq_{\mathbb{F}}$ follows because all interpretations in $\mathbb{F}$ are weakly monotone with respect to set inclusion. We show that $\supseteq_{\mathbb{F}}$ is closed under substitutions. Suppose that $s \supseteq_{\mathbb{F}} t$ and let σ be a substitution. Let $\alpha \colon \mathcal{V} \to \mathbb{F}$ be an arbitrary assignment. We have to show that $[\alpha]_{\mathbb{F}}(s\sigma) \supseteq [\alpha]_{\mathbb{F}}(t\sigma)$. Define the assignment β as $\beta(x) = [\alpha]_{\mathbb{F}}(x\sigma)$ for all $x \in \mathcal{V}$. It is not difficult to show that $[\alpha]_{\mathbb{F}}(s\sigma) = [\beta]_{\mathbb{F}}(s)$ and $[\alpha]_{\mathbb{F}}(t\sigma) = [\beta]_{\mathbb{F}}(t)$. The assumption $s \supseteq_{\mathbb{F}} t$ yields $[\beta]_{\mathbb{F}}(s) \supseteq [\beta]_{\mathbb{F}}(t)$. Closure under substitutions of $\geqslant_{\mathbb{F}}$ and $>_{\mathbb{F}}$ follows in the same way. The relation $>_{\mathbb{F}}$ is a proper order. It inherits well-foundedness from $>$. Since $\supseteq_{\mathbb{F}} \subseteq \geqslant_{\mathbb{F}}$ and $\geqslant_{\mathbb{F}} \cdot >_{\mathbb{F}} = >_{\mathbb{F}}$, compatibility holds. We have $\mathcal{C}_{\mathcal{E}} \subseteq \supseteq_{\mathbb{F}}$ by the definition of $\mathsf{cons}_{\mathbb{F}}$. □

The next example illustrates the difference between $>_{\mathbb{N}}$ and $>_{\mathbb{F}}$.

Example 5.17. *Consider again the TRS and the interpretation of Example 5.12. If we take an assignment α with $\alpha(x) = \{0, 1\}$ then $[\alpha]_{\mathbb{F}}(\mathsf{f}^\sharp(\mathsf{a}, \mathsf{b}, x)) = \{1\}$ and $[\alpha]_{\mathbb{F}}(\mathsf{f}^\sharp(x, x, x)) = \{0, 1\}$. Hence $\mathsf{f}^\sharp(\mathsf{a}, \mathsf{b}, x) >_{\mathbb{F}} \mathsf{f}^\sharp(x, x, x)$ does not hold.*

In the following lemma $\beta \in \alpha$ abbreviates "$\beta(x) \in \alpha(x)$ for all $x \in \mathcal{V}$".

Lemma 5.18. *Let t be a term. For every assignment $\alpha \colon \mathcal{V} \to \mathbb{F}$ we have*

$$[\alpha]_{\mathbb{F}}(t) \supseteq \{[\beta]_{\mathcal{A}}(t) \mid \beta \in \alpha\}.$$

Moreover, if t is linear then the reverse inclusion also holds.

Proof. We show the inclusion by induction on t. Let $\beta \in \alpha$. If t is a variable then $[\beta]_{\mathcal{A}}(t) = \beta(t) \in \alpha(t) = [\alpha]_{\mathbb{F}}(t)$. Suppose $t = f(t_1, \ldots, t_n)$. The induction hypothesis yields $[\beta]_{\mathcal{A}}(t_i) \in [\alpha]_{\mathbb{F}}(t_i)$ for all $i \in \{1, \ldots, n\}$. Hence, by the definition of $f_{\mathbb{F}}$,

$$[\beta]_{\mathcal{A}}(t) = f_{\mathcal{A}}([\beta]_{\mathcal{A}}(t_1), \ldots, [\beta]_{\mathcal{A}}(t_n)) \in f_{\mathbb{F}}([\alpha]_{\mathbb{F}}(t_1), \ldots, [\alpha]_{\mathbb{F}}(t_n)) = [\alpha]_{\mathbb{F}}(t).$$

Now suppose that t is linear. We show the reverse inclusion

$$[\alpha]_{\mathbb{F}}(t) \subseteq \{[\beta]_{\mathcal{A}}(t) \mid \beta \in \alpha\}$$

by induction on t. If t is a variable then

$$[\alpha]_{\mathbb{F}}(t) = \alpha(t) = \{\beta(t) \mid \beta \in \alpha\} = \{[\beta]_{\mathcal{A}}(t) \mid \beta \in \alpha\}.$$

Suppose $t = f(t_1, \ldots, t_n)$. The induction hypothesis yields

$$[\alpha]_{\mathbb{F}}(t_i) \subseteq \{[\beta]_{\mathcal{A}}(t_i) \mid \beta \in \alpha\}$$

for all $i \in \{1, \ldots, n\}$. Because t is linear, the variables in $t_1, \ldots, t_n$ are pairwise disjoint and hence $[\alpha]_{\mathbb{F}}(t_1) \times \cdots \times [\alpha]_{\mathbb{F}}(t_n) \subseteq \{([\beta]_{\mathcal{A}}(t_1), \ldots, [\beta]_{\mathcal{A}}(t_n)) \mid \beta \in \alpha\}$. Consequently,

$$\begin{aligned}[\alpha]_{\mathbb{F}}(t) &= \{f_{\mathcal{A}}(a_1, \ldots, a_n) \mid (a_1, \ldots, a_n) \in [\alpha]_{\mathbb{F}}(t_1) \times \cdots \times [\alpha]_{\mathbb{F}}(t_n)\} \\ &\subseteq \{f_{\mathcal{A}}([\beta]_{\mathcal{A}}(t_1), \ldots, [\beta]_{\mathcal{A}}(t_n)) \mid \beta \in \alpha\} \\ &= \{[\beta]_{\mathcal{A}}(t) \mid \beta \in \alpha\}\end{aligned}$$

□

As can be seen from Example 5.17, set equality in the above lemma is not guaranteed without the linearity of t.

The following lemma relates interpretations in $\mathcal{A}$ to interpretations in $\mathbb{F}$.

Lemma 5.19. *Let $l \to r$ be an $\mathcal{A}$-right-linear rewrite rule.*

- *If $l =_{\mathcal{A}} r$ then $l \supseteq_{\mathbb{F}} r$.*
- *If $l \geqslant_{\mathcal{A}} r$ then $l \geqslant_{\mathbb{F}} r$.*
- *If $l >_{\mathcal{A}} r$ then $l >_{\mathbb{F}} r$.*

Proof. Let r' be a linearization of r such that $r' =_{\mathcal{A}} r$ and let σ be a substitution such that $r'\sigma = r$. We may assume that σ only affects the (fresh) variables in $\mathcal{V}\mathsf{ar}(r') \setminus \mathcal{V}\mathsf{ar}(r)$. In particular, $l = l\sigma$. Since the relations $\supseteq_{\mathbb{F}}$, $\geqslant_{\mathbb{F}}$, and $>_{\mathbb{F}}$ are closed under substitutions it is sufficient to show $l \supseteq_{\mathbb{F}} r'$, $l \geqslant_{\mathbb{F}} r'$, and $l >_{\mathbb{F}} r'$ under the stated conditions. We have

$$[\alpha]_{\mathbb{F}}(l) \supseteq \{[\beta]_{\mathcal{A}}(l) \mid \beta \in \alpha\}$$

and

$$[\alpha]_{\mathbb{F}}(r') = \{[\beta]_{\mathcal{A}}(r') \mid \beta \in \alpha\} = \{[\beta]_{\mathcal{A}}(r) \mid \beta \in \alpha\}$$

according to Lemma 5.18. Now, if $l =_{\mathcal{A}} r$ then $[\beta]_{\mathcal{A}}(l) = [\beta]_{\mathcal{A}}(r)$ for all assignments β and thus $[\alpha]_{\mathbb{F}}(l) \supseteq [\alpha]_{\mathbb{F}}(r')$. Hence $l \supseteq_{\mathbb{F}} r'$ by definition. Next, if $l \geqslant_{\mathcal{A}} r$ then $[\beta]_{\mathcal{A}}(l) \geqslant [\beta]_{\mathcal{A}}(r)$ for all assignments β. Hence

$$\begin{aligned}\max([\alpha]_{\mathbb{F}}(l)) &\geqslant \max\{[\beta]_{\mathcal{A}}(l) \mid \beta \in \alpha\}\\ &\geqslant \max\{[\beta]_{\mathcal{A}}(r) \mid \beta \in \alpha\}\\ &= \max([\alpha]_{\mathbb{F}}(r'))\end{aligned}$$

and thus $l \geqslant_{\mathbb{F}} r'$ by definition. The proof that $l >_{\mathbb{F}} r$ whenever $l >_{\mathcal{A}} r$ follows in exactly the same way. □

We are now ready for the usable rules criterion announced earlier.

Theorem 5.20. *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a cycle in its dependency graph. If there exists an algebra $\mathcal{A}$ equipped with a well-order $>$ such that $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ is $\mathcal{A}$-right-linear, $\mathcal{U}(\mathcal{C}) \subseteq =_{\mathcal{A}}$, $\mathcal{C} \subseteq \geqslant_{\mathcal{A}}$, and $\mathcal{C} \cap >_{\mathcal{A}} \neq \varnothing$ then there are no $\mathcal{C}$-minimal rewrite sequences.*

Proof. From Lemma 5.19 we obtain $\mathcal{U}(\mathcal{C}) \subseteq \supseteq_{\mathbb{F}}$, $\mathcal{C} \subseteq \geqslant_{\mathbb{F}}$, and $\mathcal{C} \cap >_{\mathbb{F}} \neq \varnothing$. Lemma 5.16 states that $(\supseteq_{\mathbb{F}}, \geqslant_{\mathbb{F}}, >_{\mathbb{F}})$ is a reduction triple and $\mathcal{C}_{\mathcal{E}} \subseteq \supseteq_{\mathbb{F}}$. Hence the conditions of Theorem 3.36 are satisfied (by taking the trivial argument filtering) and the result follows. □

Note that the set extension is only used in the proof of Theorem 5.20.

Example 5.21. *Consider again the problematic SCC $\mathcal{C} = \{16\}$ of Example 5.9. We claim that the induced algebra over $\mathbb{N}$ regards $\mathcal{U}(\mathcal{C}) \cup \mathcal{C}$ as right-linear. For the rules in $\mathcal{U}(\mathcal{C}) = \{7, 8, 9\}$ this is clear as they are right-linear. Because of the interpretation $\mathsf{mod}^\sharp_{\mathbb{Z}}(x, y) = x$, we have*

$$\mathsf{mod}^\sharp(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(y)) =_{\mathbb{N}} \mathsf{mod}^\sharp(\mathsf{s}(x) - \mathsf{s}(y), \mathsf{s}(z))$$

and thus the single rule in $\mathcal{C}$ is regarded as right-linear. Hence Theorem 5.20 is applicable and we can finally conclude the termination of the TRS in Example 5.9.

5.2.2 Towards Automation

How do we verify a constraint like $x+1 > \max\{0, x-y\}$? The approach that we developed in Section 5.1.2 for dealing with negative constants is not applicable because Lemma 5.6 relies essentially on weak monotonicity of the polynomial interpretations.

Let $\mathcal{P}_{\geqslant 0}$ be a subset of the set of polynomials P with integral coefficients such that $\alpha(P) \geqslant 0$ for all $\alpha\colon \mathcal{V} \to \mathbb{N}$ for which membership is decidable. For instance, $\mathcal{P}_{\geqslant 0}$ could be the set of polynomials without negative coefficients. We define $\mathcal{P}_{<0}$ in the same way.

Definition 5.22. *Let $(\mathbb{Z}, \{f_{\mathbb{Z}}\}_{f \in \mathcal{F}})$ be an algebra. With every term t we associate a polynomial $Q(t)$ as follows:*

$$Q(t) = \begin{cases} t & \text{if } t \text{ is a variable} \\ P & \text{if } t = f(t_1, \ldots, t_n) \text{ and } P \in \mathcal{P}_{\geqslant 0} \\ 0 & \text{if } t = f(t_1, \ldots, t_n) \text{ and } P \in \mathcal{P}_{<0} \\ v(P) & \text{otherwise} \end{cases}$$

where $P = f_{\mathbb{Z}}(Q(t_1), \ldots, Q(t_n))$. *In the last clause* $v(P)$ *denotes a fresh* abstract *variable that we uniquely associate with* P.

There are two kinds of indeterminates in $Q(t)$: ordinary variables occurring in t and abstract variables. The intuitive meaning of an abstract variable $v(P)$ is $\max\{0, P\}$. The latter quantity is always non-negative, like an ordinary variable ranging over the natural numbers, but from $v(P)$ we can extract the original polynomial P and this information may be crucial for a comparison between two polynomial expressions to succeed. Note that the polynomial P associated with an abstract variable $v(P)$ may contain other abstract variables. However, because $v(P)$ is different from previously selected abstract variables, there are no spurious loops like $P_1 = v(x - v(P_2))$ and $P_2 = v(x - v(P_1))$.

The reason for using $\mathcal{P}_{\geqslant 0}$ and $\mathcal{P}_{<0}$ in the above definition is to make our approach independent of the particular method that is used to test non-negativeness or negativeness of polynomials.

Definition 5.23. *With every assignment* $\alpha\colon \mathcal{V} \to \mathbb{N}$ *we associate an assignment* $\alpha^*\colon \mathcal{V} \to \mathbb{N}$ *defined as follows:*

$$\alpha^*(x) = \begin{cases} \max\{0, \alpha^*(P)\} & \text{if } x \text{ is an abstract variable } v(P) \\ \alpha(x) & \text{otherwise} \end{cases}$$

The above definition is recursive because P may contain abstract variables. However, since $v(P)$ is different from previously selected abstract variables, the recursion terminates and it follows that α^* is well-defined.

Theorem 5.24. *Let* $(\mathbb{Z}, \{f_{\mathbb{Z}}\}_{f \in \mathcal{F}})$ *be an algebra such that every* $f_{\mathbb{Z}}$ *is a polynomial. Let* t *be a term. For every assignment* α *we have* $[\alpha]_{\mathbb{N}}(t) = \alpha^*(Q(t))$.

Proof. We show that $[\alpha]_{\mathbb{N}}(t) = \alpha^*(Q(t))$ by induction on t. If t is a variable then $[\alpha]_{\mathbb{N}}(t) = \alpha(t) = \alpha^*(t) = \alpha^*(Q(t))$. Suppose $t = f(t_1, \ldots, t_n)$. Let $P = f_{\mathbb{Z}}(Q(t_1), \ldots, Q(t_n))$. The induction hypothesis yields $[\alpha]_{\mathbb{N}}(t_i) = \alpha^*(Q(t_i))$ for all i and thus

$$\begin{aligned} [\alpha]_{\mathbb{N}}(t) &= f_{\mathbb{N}}(\alpha^*(Q(t_1)), \ldots, \alpha^*(Q(t_n))) \\ &= \max\{0, f_{\mathbb{Z}}(\alpha^*(Q(t_1)), \ldots, \alpha^*(Q(t_n)))\} = \max\{0, \alpha^*(P)\} \end{aligned}$$

We distinguish three cases, corresponding to the definition of $Q(t)$.

- First suppose that $P \in \mathcal{P}_{\geqslant 0}$. This implies that $\alpha^*(P) \geqslant 0$ and thus we have $\max\{0, \alpha^*(P)\} = \alpha^*(P)$. Hence $[\alpha]_{\mathbb{N}}(t) = \alpha^*(P) = \alpha^*(Q(t))$.
- Next suppose that $P \in \mathcal{P}_{<0}$. So $\alpha^*(P) < 0$ and thus $\max\{0, \alpha^*(P)\} = 0$. Hence $[\alpha]_{\mathbb{N}}(t) = 0 = \alpha^*(Q(t))$.
- In the remaining case we do not know the status of P. We have $Q(t) = v(P)$ and thus $\alpha^*(Q(t)) = \max\{0, \alpha^*(P)\}$ which immediately yields the desired identity $[\alpha]_{\mathbb{N}}(t) = \alpha^*(Q(t))$.

□

Corollary 5.25. *Let* $(\mathbb{Z}, \{f_{\mathbb{Z}}\}_{f \in \mathcal{F}})$ *be an* $\mathcal{F}$*-algebra such that every* $f_{\mathbb{Z}}$ *is a polynomial. Let* s *and* t *be terms. If* $Q(s) = Q(t)$ *then* $s =_{\mathbb{N}} t$. *If* $\alpha^*(Q(s) - Q(t)) > 0$ *for all assignments* $\alpha\colon \mathcal{V} \to \mathbb{N}$ *then* $s >_{\mathbb{N}} t$. *If* $\alpha^*(Q(s) - Q(t)) \geqslant 0$ *for all assignments* $\alpha\colon \mathcal{V} \to \mathbb{N}$ *then* $s \geqslant_{\mathbb{N}} t$. □

Example 5.26. *Consider again dependency pair 16 from Example 5.9:*

$$\mathsf{mod}^\sharp(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{mod}^\sharp(\mathsf{s}(x)-\mathsf{s}(y),\mathsf{s}(y))$$

We have $Q(\mathsf{mod}^\sharp(\mathsf{s}(x),\mathsf{s}(y))) = x+1$ *and* $Q(\mathsf{mod}^\sharp(\mathsf{s}(x)-\mathsf{s}(y),\mathsf{s}(y))) = v(x-y)$. *Since* $x+1-v(x-y)$ *may be negative (when interpreting* $v(x-y)$ *as a variable), the above corollary cannot be used to conclude that 16 is strictly decreasing. However, if we estimate* $v(x-y)$ *by* x*, the non-negative part of* $x-y$*, then we obtain* $x+1-x=1$ *which is clearly positive.*

Given a polynomial P with coefficients in $\mathbb{Z}$, we denote the non-negative part of P by $N(P)$.

Lemma 5.27. *Let* Q *be a polynomial with integral coefficients. Suppose* $v(P)$ *is an abstract variable that occurs in* Q *but not in* $N(Q)$*. If* Q' *is the polynomial obtained from* Q *by replacing* $v(P)$ *with* $N(P)$ *then* $\alpha^*(Q) \geqslant \alpha^*(Q')$ *for all assignments* $\alpha\colon \mathcal{V} \to \mathbb{N}$.

Proof. Let $\alpha\colon \mathcal{V} \to \mathbb{N}$ be an arbitrary assignment. In $\alpha^*(Q)$ every occurrence of $v(P)$ is assigned the value $\alpha^*(v(P)) = \max\{0, \alpha^*(P)\}$. We have $\alpha^*(N(P)) \geqslant \alpha^*(P) \geqslant \alpha^*(v(P))$. By assumption, $v(P)$ occurs only in the negative part of Q. Hence Q is (strictly) anti-monotone in $v(P)$ and therefore $\alpha^*(Q) \geqslant \alpha^*(Q')$. □

In order to determine whether $s \geqslant_{\mathbb{N}} t$ (or $s >_{\mathbb{N}} t$) holds, the idea now is to first use standard techniques to test the non-negativeness of $Q = Q(s) - Q(t)$ (i.e., we determine whether $\alpha(Q) \geqslant 0$ for all assignments α by checking whether $Q \in \mathcal{P}_{\geqslant 0}$). If Q is non-negative then we certainly have $\alpha^*(Q) \geqslant 0$ for all assignments α and thus $s \geqslant_{\mathbb{N}} t$ follows from Corollary 5.25. If non-negativeness cannot be shown then we apply the previous lemma to replace an abstract variable that occurs only in the negative part of Q. The resulting polynomial Q' is tested for non-negativeness. If the test succeeds then for all assignments α we have $\alpha^*(Q') \geqslant 0$ and thus also $\alpha^*(Q) \geqslant 0$ by the previous lemma. According to Corollary 5.25 this is sufficient to conclude $s \geqslant_{\mathbb{N}} t$. Otherwise we repeat the above process with Q'. The process terminates when there are no more abstract variables left that appear only in the negative part of the current polynomial.

We conclude this chapter by mentioning the work of Lucas [43, 44], in which polynomials with *real* coefficients for automatically proving termination of (context-sensitive) rewriting systems are considered. He solves the problem of well-foundedness by replacing the standard order on $\mathbb{R}$ with $>_\delta$ for some fixed positive $\delta \in \mathbb{R}$: $x >_\delta y$ if and only if $x - y \geqslant \delta$. In addition, he demands that interpretations are uniformly bounded from below (i.e., there exists an $m \in \mathbb{R}$ such that $f_{\mathbb{R}}(x_1,\ldots,x_n) \geqslant m$ for all function symbols f and $x_1,\ldots,x_n \geqslant m$). While this method allows one to use finer positive polynomial like $x^2 - \frac{1}{2}x + 1$, the latter requirement entails that interpretations like $x-1$ or $x-y+1$ cannot be handled. This contrasts our approach in which a given algebra (on $\mathbb{Z}$) is replaced by an induced algebra (on $\mathbb{N}$). We anticipate that by combining both approaches, unrestricted polynomial interpretations with real coefficients like $x - \frac{1}{2}y$ can be used for termination proofs.

Chapter 6

Tyrolean Termination Tool

The Tyrolean Termination Tool (T$_T$T) is a tool for automatically proving termination and innermost termination of term rewrite systems and simply typed applicative rewrite systems. The techniques introduced in the preceding chapters have been implemented in T$_T$T. T$_T$T is described in Section 6.1. In Section 6.2 we provide experimental data for all new techniques presented in preceding chapters. In the final section we compare our tool with other systems.

6.1 Tyrolean Termination Tool

The Tyrolean Termination Tool is the successor of the Tsukuba Termination Tool [29], which was a tool for automatically proving termination of term rewrite systems. T$_T$T produces high-quality output and has a convenient web interface. The tool is available at

`http://cl2-informatik.uibk.ac.at/ttt`

This section is organized as follows: In Section 6.1.1 we explain the web interface of T$_T$T. In Section 6.1.2 we explain how to use T$_T$T to obtain termination proofs of simply-typed applicative systems. In Section 6.1.3 we describe how to input a collection of rewrite systems and how to interpret the resulting output. Some implementation details are given in Section 6.1.4.

6.1.1 Web interface

The web interface provides a fully automatic mode and a semi-automatic mode.

Fully Automatic Mode. Figure 6.3 shows the web interface for the fully automatic mode. In this mode T$_T$T uses a simple strategy to (recursively) solve the ordering constraints for each SCC of the approximated dependency graph. The strategy is based on the new features described in the previous chapters and uses LPO (both with strict and quasi-precedence) with *some* argument filterings in Section 4.3.1 and mostly linear polynomial interpretations with coefficients from $\{-1, 0, 1\}$ as base orders.

After computing the SCCs of the approximated (innermost) dependency graph, the strategy subjects each SCC to the following algorithm:

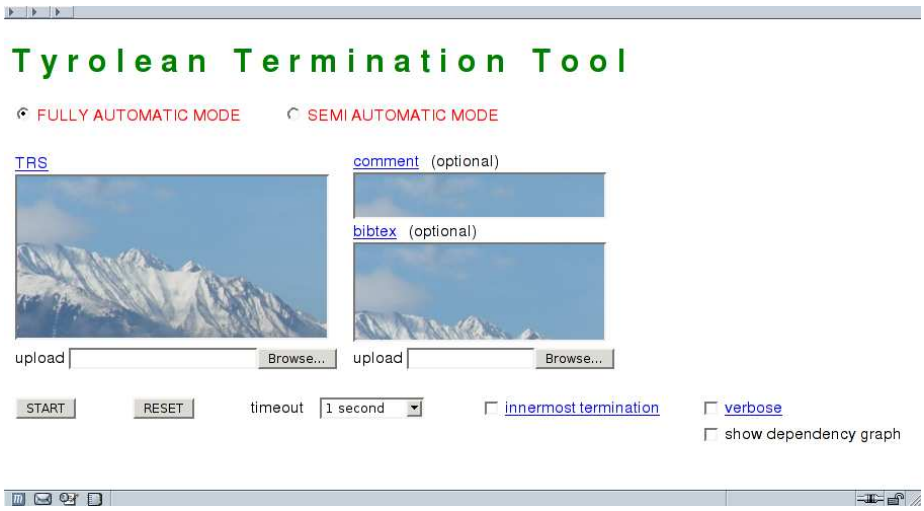


Figure 6.1: A screen shot of the fully automatic mode of T$_{\mathsf{T}}$T.

1. First we check whether the *subterm criterion* is applicable.

2. If the subterm criterion was unsuccessful, we compute the *usable rules*.

3. The resulting (usable rules and dependency pairs) constraints are subjected to the *natural* (see below) polynomial interpretation.

4. If the constraints could not be solved in step 3, we employ the *divide and conquer* algorithm for computing suitable argument filterings with respect to the *some* heuristic in Section 4.3.1 and the lexicographic path order (LPO) with *strict* precedence.

5. If the previous step was unsuccessful, we repeat step 3 with *arbitrary* linear polynomial interpretations with coefficients from $\{0, 1\}$.

6. Next we repeat step 4 with the variant of LPO based on *quasi-precedences* and a small increase in the search space for argument filterings (see below).

7. If the constraints could still not be solved, we try polynomial interpretations with negative constants.

8. As a last resort, we use polynomial interpretations with coefficients from $\{-1, 0, 1\}$ in connection with Corollary 5.13 (for innermost termination) or Corollary 5.20 (for termination). If the latter fails, due to the $\mathcal{A}$-right-linearity restriction, we give Corollary 5.11 a try.

If only part of an SCC could be handled, we subject the resulting new SCCs recursively to the same algorithm.

Taking the following polynomial interpretations for certain function symbols that appear in many example TRSs is what we call *natural* (for other function

symbols we take linear interpretations with coefficients from $\{0, 1\}$):

$$\begin{array}{cccc} \mathsf{0}_{\mathbb{Z}} = 0 & \mathsf{1}_{\mathbb{Z}} = 1 & \mathsf{2}_{\mathbb{Z}} = 2 & \cdots \\ \mathsf{s}_{\mathbb{Z}}(x) = x + 1 & \mathsf{p}_{\mathbb{Z}}(x) = x - 1 & +_{\mathbb{Z}}(x, y) = x + y & \times_{\mathbb{Z}}(x, y) = xy \end{array}$$

If the current set of constraints can be solved in step 4 or 5, then they can also be solved in step 6 or 7, respectively, but the reverse is not true. The sole reason for adopting LPO and polynomial interpretations in alternating layers is efficiency; the search space in steps 3 and 4 is significantly smaller than in steps 5 and 6. Needless to say, the search space in step 5 is much smaller than in step 7 which in turn is much smaller than in step 8. The reason for putting the subterm criterion first is that with this criterion many SCCs can be eliminated very quickly, cf. the third paragraph of Section 6.1.4. The extension of the search space for argument filterings mentioned in step 6 is obtained by also considering the full *reverse* argument filtering $[n, \ldots, 1]$ for every n-ary function symbol. The advantage of this extension is that there is no need for a specialized version of LPO with right-to-left status.

The effectiveness of the automatic strategy can be seen from the data presented in Figure 6.2, which were obtained by running T$_{\mathsf{T}}$T in fully automatic mode on the 89 terminating TRSs (66 in Section 3 and 23 in Section 4) of [5]. An explanation of the data is given in Section 6.1.3.

An empirical evaluation of the automatic strategy can be found in Section 6.2.

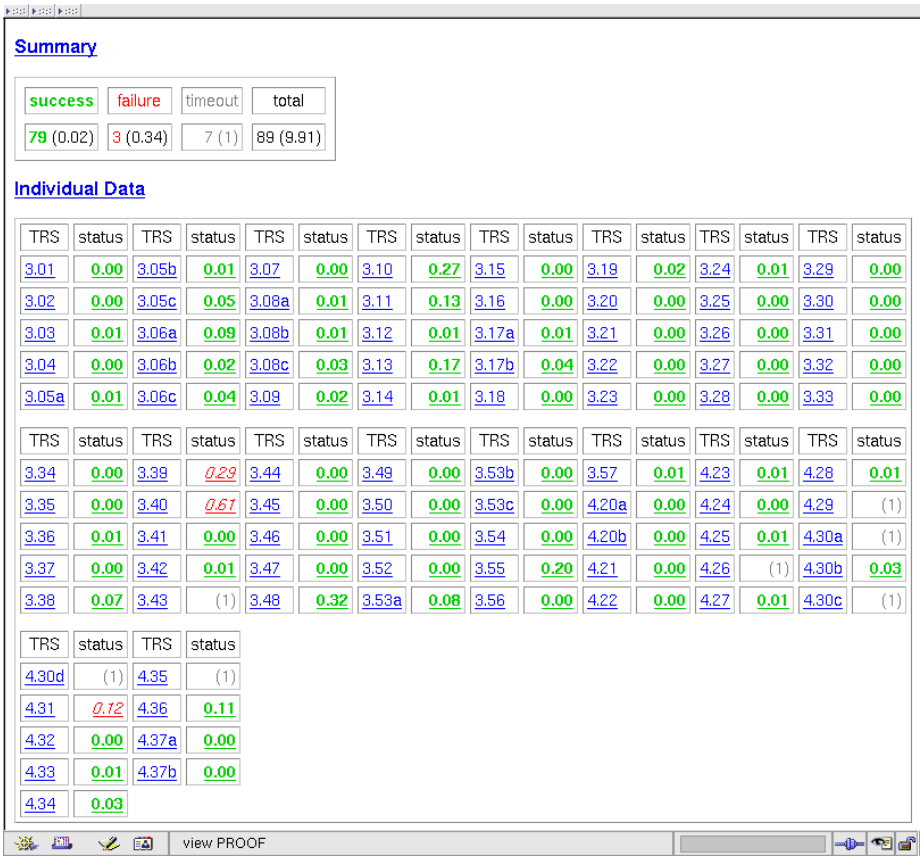
Semi Automatic Mode. Figure 6.3 shows the web interface for the semi-automatic mode. Although the fully automatic mode usually derives most power of T$_{\mathsf{T}}$T, sometimes the user needs to set options of T$_{\mathsf{T}}$T properly: For instance, this is the case for the TRS of Example 4.24.

Most of the boxes and buttons are self-explanatory. Many correspond to settings of the fully automatic mode. There are three features that are not covered by the fully automatic mode. First of all, the user can select the Knuth-Bendix order as base order of a reduction triple. Secondly, the user can choose the (non-)heuristics for argument filterings described in Section 4.3.1. Finally, by clicking the *enumerate* box, the user can use the enumeration method, which tries to find a suitable argument filtering by enumerating possible all argument filterings. For most examples the divide and conquer method of Section 4.3 is more efficient than the straightforward enumeration method, but still, there are TRSs where enumeration is more effective (cf. Section 6.2), so the user has the option to change the search strategy.

6.1.2 Simply-Typed Applicative Rewrite Systems

Besides ordinary first-order TRSs, T$_{\mathsf{T}}$T accepts simply-typed applicative rewrite systems [1]. Applicative terms are built from variables, constants, and a single binary operator $\cdot$, called application. Constants and variables are equipped with a simple type such that the rewrite rules typecheck.

Definition 6.1. *Let B be a set of constants and $\to$ a binary function symbol. A term in $\mathcal{T}(B \cup \{\to\}, \varnothing)$ is called a* simple type*, and we write* ST *for $\mathcal{T}(B \cup$*



Summary

success	failure	timeout	total
79 (0.02)	3 (0.34)	7 (1)	89 (9.91)

Individual Data

TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status
3.01	0.00	3.05b	0.01	3.07	0.00	3.10	0.27	3.15	0.00	3.19	0.02	3.24	0.01	3.29	0.00
3.02	0.00	3.05c	0.05	3.08a	0.01	3.11	0.13	3.16	0.00	3.20	0.00	3.25	0.00	3.30	0.00
3.03	0.01	3.06a	0.09	3.08b	0.01	3.12	0.01	3.17a	0.01	3.21	0.00	3.26	0.00	3.31	0.00
3.04	0.00	3.06b	0.02	3.08c	0.03	3.13	0.17	3.17b	0.04	3.22	0.00	3.27	0.00	3.32	0.00
3.05a	0.01	3.06c	0.04	3.09	0.02	3.14	0.01	3.18	0.00	3.23	0.00	3.28	0.00	3.33	0.00

TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status	TRS	status
3.34	0.00	3.39	0.29	3.44	0.00	3.49	0.00	3.53b	0.00	3.57	0.01	4.23	0.01	4.28	0.01
3.35	0.00	3.40	0.61	3.45	0.00	3.50	0.00	3.53c	0.00	4.20a	0.00	4.24	0.00	4.29	(1)
3.36	0.01	3.41	0.00	3.46	0.00	3.51	0.00	3.54	0.00	4.20b	0.00	4.25	0.01	4.30a	(1)
3.37	0.00	3.42	0.01	3.47	0.00	3.52	0.00	3.55	0.20	4.21	0.00	4.26	(1)	4.30b	0.03
3.38	0.07	3.43	(1)	3.48	0.32	3.53a	0.08	3.56	0.00	4.22	0.00	4.27	0.01	4.30c	(1)

TRS	status	TRS	status
4.30d	(1)	4.35	(1)
4.31	0.12	4.36	0.11
4.32	0.00	4.37a	0.00
4.33	0.01	4.37b	0.00
4.34	0.03		

Figure 6.2: Output produced by TTT.

$\{\to\}, \varnothing)$. *Let* $\mathcal{F}$ *be a set of constants and* $\mathcal{V}$ *a set of variables. Let* Γ *be a function from* $\mathcal{F} \cup \mathcal{V}$ *to* ST. *Consider the following typing rules.*

$$\frac{t \in \mathcal{F} \cup \mathcal{V} \quad \Gamma(t) = \tau}{\Gamma \vdash t : \tau} \qquad \frac{\Gamma \vdash t_1 : \tau_1 \to \tau_2 \quad t_2 : \tau_1}{\Gamma \vdash t_1 \cdot t_2 : \tau_2}$$

If $\Gamma \vdash t : \tau$ *can be derived then* t *is called a* simply-typed applicative term *and we write* $\mathrm{type}(t) = \tau$. *Furthermore, we write* $\mathcal{ST}$ *for all simply-typed applicative terms.*

Note that if $t \in \mathcal{ST}$ then $\mathrm{type}(t)$ is uniquely determined. If $t \in \mathcal{ST}$ then t can be written in the form $a \cdot t_1 \cdot \cdots \cdot t_n$, where $a \in \mathcal{F} \cup \mathcal{V}$. Moreover, $\mathrm{type}(\tau)$ can be written in the form $\tau_1 \to \cdots \to \tau_n \to \tau$.

Definition 6.2. *A* simply-typed applicative rewrite system *(*STARS*)* $\mathcal{R}$ *is a set of rewrite rules* $l \to r$ *between terms in* $\mathcal{ST}$ *such that* $\mathrm{type}(l) = \mathrm{type}(r)$ *and* l *is of the form* $c \cdot t_1 \cdot \cdots \cdot t_n$ *with a constant* c.

A typical example is provided by the following rules for the map function

$$(\mathsf{map} \cdot f) \cdot \mathsf{nil} \to \mathsf{nil}$$
$$(\mathsf{map} \cdot f) \cdot ((\mathsf{cons} \cdot x) \cdot y) \to (\mathsf{cons} \cdot (f \cdot x)) \cdot ((\mathsf{map} \cdot f) \cdot y)$$

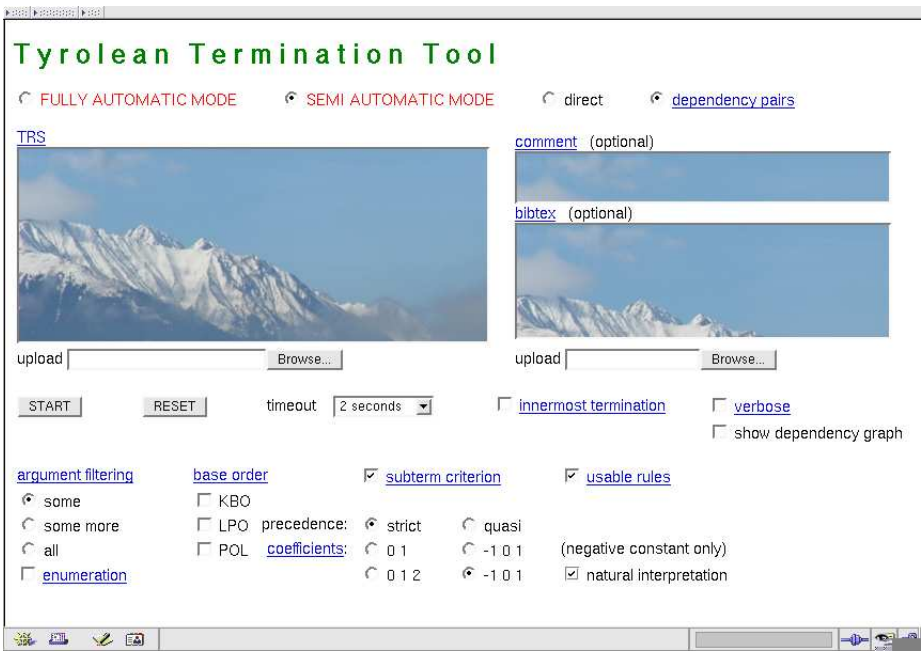


Figure 6.3: A screen shot of the semi automatic mode of T$_{\text{T}}$T.

with the type declaration $\mathsf{nil}\colon \alpha$, $\mathsf{cons}\colon \beta \to \alpha \to \alpha$, $\mathsf{map}\colon (\beta \to \beta) \to \alpha \to \alpha$, $f\colon \beta \to \beta$, $x\colon \beta$, and $y\colon \alpha$. Here α is the list type and β the type of elements of lists. STARSs are useful to model higher-order functions in a first-order setting. As usual, the application operator $\cdot$ is suppressed in the notation and parentheses are removed under the "association to the left" rule. The above rules then become

$$\mathsf{map}\ f\ \mathsf{nil} \to \mathsf{nil}$$
$$\mathsf{map}\ f\ (\mathsf{cons}\ x\ y) \to \mathsf{cons}\ (f\ x)\ (\mathsf{map}\ f\ y)$$

This corresponds to the syntax of STARSs in T$_{\text{T}}$T. The types of constants must be declared by the keyword `TYPES`. The types of variables is automatically inferred when typechecking the rules, which follow the `RULES` keyword. So the above STARS would be inputted to T$_{\text{T}}$T as

```
TYPES
 nil : a                                  ;
cons : b => (a => a)                      ;
 map : (b => b) => a => a                 ;

RULES
map f nil         -> nil                  ;
map f (cons x y) -> cons (f x) (map f y)  ;
```

In order to prove termination of STARSs, T$_{\text{T}}$T uses the two-phase transformation developed by Aoto and Yamada [2]. In the first phase all head variables

(e.g. `f` in the subterm `f x` of the second right-hand side) are removed by the *head variable instantiation* technique.

Definition 6.3. *The set of all head variables in a term t, denoted by $\mathsf{HV}(t)$, is defined as follows:*

$$\mathsf{HV}(t) = \begin{cases} \varnothing & \text{if } t \in \mathcal{F} \cup \mathcal{V} \\ \mathcal{V}\mathsf{ar}(t_1) \cup \mathsf{HV}(t_2) & \text{if } t = t_1 \cdot t_2 \text{ and } t_1 \in \mathcal{F} \cup \mathcal{V} \\ \mathsf{HV}(t_1 \cdot t_2) \cup \mathsf{HV}(t_3) & \text{if } t = (t_1 \cdot t_2) \cdot t_3 \end{cases}$$

Definition 6.4. *Let $\mathcal{R}$ be an STARS. For each $l \to r \in \mathcal{R}$, we define $\psi_{l\to r}$ as follows: If $x \in \mathsf{HV}(l) \cup \mathsf{HV}(r)$ then*

$$\psi_{l\to r}(x) = \{f \cdot x_1 \cdot \cdots \cdot x_n \mid f \in \mathcal{F} \text{ and } \mathrm{type}(f) = \tau_1 \to \cdots \to \tau_n \to \mathrm{type}(x)\}$$

Otherwise $\psi_{l\to r}(x) = \{x\}$. Here $x_1, \ldots, x_n$ are pairwise distinct fresh variables. The head variable instantiation $\mathsf{HVI}(\mathcal{R})$ *of $\mathcal{R}$ is*

$$\bigcup_{l\to r\in\mathcal{R}} \{l\sigma \to r\sigma \mid \forall x \in \mathcal{V}\mathsf{ar}(l) \;\; \sigma(x) \in \psi_{l\to r}(x)\}$$

The soundness of this phase relies on (1) the *ground term existence condition* and (2) η-saturation. Basically, the former states that all simple types are inhabited by at least one ground term. The latter is defined as follows.

Definition 6.5. *Let $\mathcal{R}$ be an STARS. The* η-saturation $\mathcal{R}_\eta$ *of $\mathcal{R}$ is the smallest set $\mathcal{S}$ containing $\mathcal{R}$ such that if $l \to r \in \mathcal{S}$ and l is not of basic type then $l \cdot x \to r \cdot x \in \mathcal{S}$ for some variable x.*

Users need not be concerned about these technicalities as T$_\mathsf{T}$T automatically adds fresh constants of the appropriate types to the signature so that the ground term existence condition is satisfied. Moreover, T$_\mathsf{T}$T performs η-saturation automatically.

After the first phase an ordinary TRS is obtained in which the application symbol is the only non-constant symbol. Such TRSs are not easily proved terminating since the root symbol of every term that has at least two symbols is the application symbol and thus provides no information which could be put to good use. In the second phase applicative terms are transformed into ordinary terms by the *translation to functional form* technique. This technique removes all occurrences of the application symbol.

Definition 6.6. *The map FF is defined as follows:*

$$\mathsf{FF}(f \cdot t_1 \cdot \ldots \cdot t_n) = \begin{cases} f_n(\mathsf{FF}(t_1), \ldots, \mathsf{FF}(t_n)) & \text{if } f \text{ is a constant} \\ f & \text{if } f \text{ is a variable} \end{cases}$$

for all applicative terms $f \cdot t_1 \cdot \ldots \cdot t_n$. Here, f_0 stands for f, and f_n is an n-ary function symbol for $1 \leqslant i \leqslant n$. We write $\mathsf{FF}(\mathcal{R})$ for $\{\mathsf{FF}(l) \to \mathsf{FF}(r) \mid l \to r \in \mathcal{R}\}$.

Summing up, for a STARS $\mathcal{R}$ given as input, T$_\mathsf{T}$T tries to prove termination of the TRS $\mathsf{FF}(\mathsf{HVI}(\mathcal{R}_\eta))$.

Theorem 6.7 ([2]). *Let $\mathcal{R}$ be an STARS satisfying the ground term existence condition. $\mathcal{R}$ is terminating if the TRS* $\mathsf{FF}(\mathsf{HVI}(\mathcal{R}_\eta))$ *is terminating.* □

We contend ourselves with showing the Postscript output (in Fig. 6.4) produced by TTT on the following variation of combinatory logic (inspired by a question posted on the TYPES Forum by Jeremy Dawson[1]):

```
TYPES
I : o => o                                  ;
W : (o => o => o) => o => o                 ;
S : (o => o => o) => (o => o) => o => o ;

RULES
I x     -> x          ;
W f x   -> f x x      ;
S x y z -> x z (y z) ;
```

Note that the types are crucial for termination; the untyped version admits the cyclic rewrite step W W W → W W W.

6.1.3 A Collection of Rewrite Systems as Input

Single TRSs (or STARSs) are inputted by typing (the type declarations and) the rules into the upper left text area or by uploading a file via the browse button. Besides the original TTT syntax (which is obtained by clicking the TRS link), TTT supports the official format[2] of the Termination Problems Data Base. The user can also upload a zip archive. All files ending in `.trs` are extracted from the archive and the termination prover runs on each of these files in turn. The results are collected and presented in two tables. The first table lists for each TRS the execution time in seconds together with the status: **bold green** indicates success, *red italics* indicates failure, and gray indicates timeout. By clicking **green** (*red*) entries the user can view the termination proof (attempt) in HTML or high-quality Postscript format. The second table gives the number of successes and failures, both with the average time spent on each TRS, the number of timeouts, and the total number of TRSs extracted from the zip archive together with the total execution time. Figure 6.2 shows the two tables for the 89 terminating TRSs in Sections 3 and 4 of [5]. Here we used TTT's fully automatic mode with a timeout of 1 second (for each TRS). The experiment was performed on a PC equipped with a 2.80 GHz Intel Pentium Processor 4 and 512 KB of memory, using native-compiled code for Linux/Debian.

6.1.4 Some Implementation Details

The web interface of TTT is written in Ruby[3] and the termination prover underlying TTT is written in Objective Caml (OCaml),[4] using the third-party libraries[5] `findlib`, `extlib`, and `pcre-ocaml`.

[1] Posted on August 3, 2004.
[2] `http://www.lri.fr/~marche/tpdb/format.html`
[3] `http://www.ruby-lang.org/`
[4] `http://www.ocaml.org/`
[5] `http://caml.inria.fr/humps/`

Termination Proof Script[a]

Consider the simply-typed applicative TRS

$$\begin{aligned} \mathsf{I}\ x &\to x \\ \mathsf{W}\ f\ x &\to f\ x\ x \\ \mathsf{S}\ x\ y\ z &\to x\ z\ (y\ z) \end{aligned}$$

over the signature $\mathsf{I}\colon \mathsf{o} \Rightarrow \mathsf{o}$, $\mathsf{W}\colon (\mathsf{o} \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}) \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}$, and $\mathsf{S}\colon (\mathsf{o} \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}) \Rightarrow (\mathsf{o} \Rightarrow \mathsf{o}) \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}$. In order to satisfy the ground term existence condition we extend the signature by $\mathsf{c}\colon \mathsf{o} \Rightarrow \mathsf{o} \Rightarrow \mathsf{o}$ and $\mathsf{c}'\colon \mathsf{o}$. Instantiating all head variables yields the following rules:

$$\begin{aligned} \mathsf{I}\ x &\to x \\ \mathsf{W}\ \mathsf{c}\ x &\to \mathsf{c}\ x\ x \\ \mathsf{S}\ \mathsf{c}\ \mathsf{I}\ z &\to \mathsf{c}\ z\ (\mathsf{I}\ z) \\ \mathsf{S}\ \mathsf{c}\ (\mathsf{W}\ w)\ z &\to \mathsf{c}\ z\ (\mathsf{W}\ w\ z) \\ \mathsf{S}\ \mathsf{c}\ (\mathsf{S}\ w\ v)\ z &\to \mathsf{c}\ z\ (\mathsf{S}\ w\ v\ z) \\ \mathsf{S}\ \mathsf{c}\ (\mathsf{c}\ w)\ z &\to \mathsf{c}\ z\ (\mathsf{c}\ w\ z) \end{aligned}$$

By transforming terms into functional form the TRS

$$\begin{aligned} 1:&\quad \mathsf{I}_1(x) &&\to x \\ 2:&\quad \mathsf{W}_2(\mathsf{c}, x) &&\to \mathsf{c}_2(x, x) \\ 3:&\quad \mathsf{S}_3(\mathsf{c}, \mathsf{I}, z) &&\to \mathsf{c}_2(z, \mathsf{I}_1(z)) \\ 4:&\quad \mathsf{S}_3(\mathsf{c}, \mathsf{W}_1(w), z) &&\to \mathsf{c}_2(z, \mathsf{W}_2(w, z)) \\ 5:&\quad \mathsf{S}_3(\mathsf{c}, \mathsf{S}_2(w, v), z) &&\to \mathsf{c}_2(z, \mathsf{S}_3(w, v, z)) \\ 6:&\quad \mathsf{S}_3(\mathsf{c}, \mathsf{c}_1(w), z) &&\to \mathsf{c}_2(z, \mathsf{c}_2(w, z)) \end{aligned}$$

is obtained. There are 3 dependency pairs:

$$\begin{aligned} 7:&\quad \mathsf{S}^\sharp_3(\mathsf{c}, \mathsf{I}, z) &&\to \mathsf{I}^\sharp_1(z) \\ 8:&\quad \mathsf{S}^\sharp_3(\mathsf{c}, \mathsf{W}_1(w), z) &&\to \mathsf{W}^\sharp_2(w, z) \\ 9:&\quad \mathsf{S}^\sharp_3(\mathsf{c}, \mathsf{S}_2(w, v), z) &&\to \mathsf{S}^\sharp_3(w, v, z) \end{aligned}$$

The approximated dependency graph contains one SCC: $\{9\}$.

- Consider the SCC $\{9\}$. By taking the simple projection π with $\pi(\mathsf{S}^\sharp_3) = 2$, the dependency pair simplifies to

  $$9:\quad \mathsf{S}_2(w, v) \to v$$

  and is compatible with the proper subterm relation.

Hence the TRS is terminating.

[a] Tyrolean Termination Tool (0.01 seconds) — September 21, 2005

Figure 6.4: Example output.

The termination prover consists of about 13,000 lines of OCaml code. About 20% is used for the manipulation of terms and rules. Another 15% is devoted to graph manipulations. This part of the code is not only used to compute dependency graph approximations, but also for precedences in KBO and LPO, and for the dependency relation which is used to compute the usable rules. The various termination methods that are provided by T$_{\text{T}}$T account for less than 10% each. Most of the remaining code deals with I/O: parsing the input and producing HTML and Postscript output. For the official Termination Problems Data Base format we use parsers written in OCaml by Claude Marché. A rich OCaml library for the manipulation of terms (or rose trees) and graphs would have made our task much easier!

It is interesting to note that two of the original techniques that make T$_{\text{T}}$T fast, the recursive SCC algorithm and the subterm criterion, account for just 13 and 20 or 11 lines, respectively. Actually, we implemented the subterm criterion twice. The former count refers to a straightforward encoding that generates all simple projections until a suitable one is found. This encoding works fine on all examples we tested (see Section 6.2), with one exception (`AProVE/AAECC-ring`). The reason is that the dependency graph of that TRS contains an SCC consisting of nine 7-ary and one 6-ary dependency pair symbols, amounting to $7^9 \times 6 = 242121642$ simple projections. The latter count refers to the specialization of the divide and conquer algorithm developed in Section 3.19 for arbitrary argument filterings. (The generic divide and conquer algorithm is implemented in 209 lines of OCaml code.) This implementation is the one used in the experiments described in Section 6.2 and the only one available from the web interface.

Concerning the implementation of simply-typed applicative rewrite systems, we use the Damas-Milner type reconstruction algorithm (see e.g. [48]) to infer the types of variables.

We conclude this section with some remarks on the implementation of base orders in T$_{\text{T}}$T. The implementation of LPO follows [29] but we first check whether the current pair of terms can be oriented by the embedding order in every recursive call to LPO. This improves the efficiency by about 20%. The implementation of KBO is based on [40]. We use the "method for complete description" [16] to compute a suitable weight function. The implementation of polynomial interpretations with coefficients from $\{0, 1\}$ is based on [11, Figure 1] together with the simplification rules described in Section 4.4.1 of the same paper. The current implementation of polynomial interpretations with coefficients from $\{-1, 0, 1\}$ in T$_{\text{T}}$T is rather naive. We anticipate that the recent techniques of [11] can be extended to handle negative coefficients.

6.2 Experiments

In this chapter we assess the techniques introduced in the preceding chapters on the 773 TRSs (at least 94 of which are non-terminating) in the Termination Problem Data Base (version 2.0).[6]

We list the number of successful termination attempts, the number of failures (which means that no termination proof was found while fully exploring the search space implied by the options), and the number of timeouts, whose values are given in parentheses. The figures below the number of successes and failures

[6] `http://www.lri.fr/~marche/tpdb/`

Table 6.1: Dependency graph approximations.

	termination		*innermost termination*	
	EDG	EDG*	EIDG	EIDG*
success	43	51	73	80
	0.00	0.01	0.01	0.01
failure	730	722	700	693
	0.01	0.01	0.10	0.21
timeout (*30* s)	0	0	0	0
total time	10	11	69	145
total arrows	52788	51642	48618	47575

denote the average time in seconds. Some tables contain execution time of individual TRSs. There, we use the following convention to indicate the status: **bold green** indicates success, *red italics* indicates failure, and gray ∞ indicates timeout.

Dependency Graph Approximations. Our first experiment concerns the new estimations of the (innermost) dependency graph mentioned in Section 4.1. Remark that (innermost) termination of TRS is concluded when the approximated (innermost) dependency graph contains no SCCs (cf. Chapter 4.2). Table 6.1 shows that EDG* and EIDG* outperform EDG and EIDG and that the overhead of the additional unification in EDG* is negligible. As we see later, the computational cost of EIDG* is very small compared to the methods that are applied afterwards, except for the subterm criterion. Actually, there are a few exceptional cases. Table 6.2 lists all TRSs for which the computation of EIDG* takes longer than 5 seconds. Since EIDG* uses unification twice as often as EIDG, it is natural to expect that EIDG* takes twice as long as EIDG. However, computing EIDG* for `Cime/mucrl1` takes much longer than expected. The reason is that it consists of a very large number of rewrite rules, and therefore the time to compute usable rules is not negligible. The inefficiency of `currying/AG01_3.13` is due to large term structures. We remark that the current version of T$_\forall$T uses a simple exponential algorithm for unification. The use of an almost-linear time unification algorithm would significantly reduce the computation time for the last TRS.

Cycle Analysis and Subterm Criterion. Looking at Table 6.3 the following claim is clearly verified: "the recursive SCC algorithm ($\models$) has the power of the standard cycle criterion ($\models_\exists$) and the efficiency of the SCC approach ($\models_\forall$)." The subterm criterion is extremely fast. It is interesting to note that it could handle 1052 of the 1589 generated SCCs, resulting in termination proofs for 206 of the 773 TRSs. Clearly, the subterm criterion should be the method of first choice in any termination prover incorporating the dependency pair technique.

Table 6.2: Innermost dependency graph approximations: individual TRSs.

			EIDG		EIDG*	
$\mathcal{R}$	rules	DPs	time	(arrows)	time	(arrows)
Cime/mucrl1	377	61	*1.91*	(136)	*8.17*	(132)
TRCSR/Ex1_2_AEL03_C	74	113	*3.60*	(1631)	*7.69*	(1631)
TRCSR/Ex6_15_AEL02_C	90	131	*5.48*	(2092)	*11.74*	(2089)
TRCSR/ExAppendixB_AEL03_C	80	124	*4.86*	(1930)	*10.40*	(1930)
currying/AG01_3.13	14	36	*3.91*	(822)	*7.75*	(822)

Table 6.3: Cycle analysis and the subterm criterion.

	$\models_\exists$	$\models_\forall$	$\models$
success	205	175	206
	0.22	0.01	0.01
failure	521	598	567
	0.44	0.02	0.02
timeout (*30* s)	47	0	0
total time	1684	11	12

Techniques for Argument filterings. Tables 6.4 and 6.5 compare enumeration (E) with the dynamic programming method described in Section 4.3.3 ($\mathtt{DP}_1$), in combination with the heuristics of Section 4.3.1. Column $\mathtt{DP}_2$ is similar to $\mathtt{DP}_1$ but it uses the approach described in the last paragraph of Section 4.3.2. We adopted the usable rule criterion.

Comparing the E and $\mathtt{DP}_2$ columns, the effectiveness of the divide and conquer approach of Section 4.3.3 is clear, especially if one keeps in mind that all possible partial argument filterings that solve the constraints are computed. In contrast, enumeration terminates as soon as the first successful argument filtering is generated. So the average time in case of failure is probably more significant (since it implies that the search space is fully explored), but then the advantage of the divide and conquer approach over enumeration is even more pronounced. Another interesting conclusion is that the combination of KBO, *some* heuristic and $\mathtt{DP}_2$ is surprisingly efficient.

Usable Rules. In Table 6.6 the combination of the subterm criterion and the usable rule criterion in connection with traditional simplification orders is examined. In all experiments we used the EDG* approximation of the dependency graph and the EIDG* approximation of the innermost dependency graph. Moreover, we adopted the recursive SCC algorithm of Section 4.2 for handling cycles in the graph. When the lexicographic path order or Knuth-Bendix order is used, the divide and conquer technique with dynamic programming described in Section 4.3.3 is used to search for suitable argument filterings. The experiments were conducted in the same environment as described in Section 6.1.3.

We tested individual methods and combinations of them. The results are summarized in Tables 6.6–6.8. The letters in the column headings have the

Table 6.4: Argument filterings.

	some			*some more*			*all*		
LPO	E	DP_1	DP_2	E	DP_1	DP_2	E	DP_1	DP_2
success	246	243	247	259	259	261	267	266	267
	0.15	0.13	0.08	0.14	0.09	0.20	0.47	0.15	0.19
failure	485	502	510	430	470	474	407	441	442
	0.74	0.40	0.31	0.83	0.63	0.55	0.96	0.99	0.90
timeout (*30* s)	42	28	16	84	44	38	99	66	64
total time	1656	1071	658	2915	1640	1450	3485	2454	2369
	some			*some more*			*all*		
KBO	E	DP_1	DP_2	E	DP_1	DP_2	E	DP_1	DP_2
success	266	261	268	287	279	281	288	276	293
	0.18	0.22	0.06	0.33	0.48	0.21	0.49	0.69	0.43
failure	468	458	501	387	392	410	366	366	376
	0.75	0.63	0.25	0.53	0.65	0.74	0.41	0.72	0.42
timeout (*30* s)	39	54	4	99	102	82	119	131	104
total time	1567	1965	260	3270	3450	2822	3863	4383	3401

following meaning:

- `s` the subterm criterion of Section 3.2,
- `u` the usable rule criterion of Section 3.4 (for Tables 6.6 and 6.7) and 5.2 (for Table 6.8),
- `l` lexicographic path order in combination with *some* argument filterings,
- `k` Knuth-Bendix order in combination with *some* argument filterings,
- `p` polynomial interpretation restricted to linear polynomials with coefficients and constants indicated in the table headings.

Polynomial Interpretations with Negative Coefficients Table 6.7 shows the effect of the usable rule criterion in combination with linear polynomial interpretations possible with negative coefficients. Enlarging the coefficient domain $\{0, 1\}$ with the value 2 gives very few additional examples (when using the usable rule criterion). The effect of allowing -1 as constant is more apparent. An interesting remark is that there is no overlap between the additional TRSs that can be proved terminating by allowing 2 and by allowing -1.

In Table 6.8 we use the negative coefficient method developed in Chapter 5.2. Theorem 5.20 (`up`) is much more powerful and considerably faster than Theorem 5.11 (`p`). As we discuss below, the $\mathcal{A}$-right-linearity condition in Theorem 5.20 is not restrictive for most ordinary TRSs. Nevertheless, there are five TRSs that can be handled by `p` but not by `up`. Three of them are variations of Toyama's example, the remaining two are the one-rule system $x \times ((-y) \times y) \to (-(y \times y)) \times x$ and the two-rule system consisting of $\mathsf{f}(x, x) \to \mathsf{f}(\mathsf{g}(x), x)$ and $\mathsf{g}(x) \to \mathsf{s}(x)$. In parentheses we provide the results

Table 6.5: Argument filterings: individual TRSs.

	some			*some more*			*all*		
LPO	E	DP_1	DP_2	E	DP_1	DP_2	E	DP_1	DP_2
AG01/3.10	*3.03*	*0.10*	*0.08*	*85.98*	*0.19*	*0.19*	*6784.15*	*1.63*	*3.47*
AG01/3.11	**0.09**	**0.03**	**0.04**	**0.51**	**0.05**	**0.06**	**11.73**	**0.52**	**0.63**
AG01/3.13	*0.22*	*0.03*	*0.03*	*1.12*	*0.05*	*0.04*	*594.37*	*17.67*	*33.74*
AG01/3.55	**0.10**	**0.04**	**0.03**	**0.53**	**0.05**	**0.06**	**12.14**	**0.53**	**0.70**
D33/11	**0.01**	**0.02**	**0.02**	**0.01**	**0.02**	**0.03**	**1.73**	**0.03**	**0.02**
	some			*some more*			*all*		
KBO	E	DP_1	DP_2	E	DP_1	DP_2	E	DP_1	DP_2
AG01/3.10	*3.16*	*1.93*	*0.15*	*100.46*	*25.97*	*11.96*	*7872.95*	*301.04*	*1125.30*
AG01/3.11	**0.10**	**0.23**	**0.06**	**0.10**	**1.61**	**0.99**	**95.11**	**929.74**	**1.90**
AG01/3.13	*0.27*	*0.27*	*0.06*	*25.90*	*6.78*	*0.74*	*654.91*	*180.67*	*59.19*
AG01/3.55	**0.11**	**0.24**	**0.06**	**0.11**	**1.62**	**0.99**	**92.43**	**925.17**	**1.92**
D33/11	**0.01**	**0.03**	**0.02**	**0.01**	**0.03**	**0.02**	**1.70**	**0.03**	**0.02**

Table 6.6: Subterm criterion and usable rules: LPO and KBO.

	s	l	ul	sul	k	uk	suk
success	206	206	244	260	162	262	290
	0.01	0.06	0.14	0.14	0.29	0.29	0.24
failure	567	540	504	488	548	465	437
	0.02	0.57	0.43	0.43	0.96	0.75	0.79
timeout (*30* s)	0	27	25	25	63	46	46
total time	12	1132	999	996	2464	1807	1796

of the following refinement: If the TRS in question is non-overlapping then we use the innermost version of Theorem 5.11 (i.e., usable rules without $\mathcal{C}_\mathcal{E}$ and with EIDG^* instead of EDG^* to estimate the dependency graph), otherwise we use Theorem 5.11. There are two additional TRSs that are handled by this strategy but not by Theorem 5.20. One is a variant of Toyama's example. The other one (AG01/3.6) clearly illustrates the limitation of the $\mathcal{A}$-right-linearity condition; in order to handle the term $\mathsf{gcd}^\sharp(x - y, \mathsf{s}(y))$ which appears on the right-hand side of a dependency pair, one must use an interpretation $\mathsf{gcd}^\sharp_\mathbb{Z}(x, y)$ that depends on both arguments, contradicting $\mathcal{A}$-right-linearity.

Fully automatic mode. Table 6.9 shows the power of the fully automatic mode of TTT. A remarkable 94% of the successful termination proofs obtained with a 30 seconds timeout are also obtained when setting the timeout to 1 second. Most of the remaining 6% are due to polynomial interpretations. The figures in parentheses refer to a version in which the conditions of Lemma 3.41 are not activated.

Table 6.7: Polynomial interpretation.

coefficients		$0,1$			$0,1,2$			$0,1$	
constants		$0,1$			$0,1,2$			$-1,0,1$	
	p	up	sup	p	up	sup	p	up	sup
success	248	340	364	263	348	369	256	355	380
	0.36	0.29	0.27	0.38	0.29	0.26	0.29	0.29	0.20
failure	458	395	371	372	347	326	348	326	300
	1.10	0.77	0.82	0.96	0.60	0.63	1.61	1.25	1.17
timeout (*30* s)	67	38	38	138	78	78	169	92	93
total time	2602	1545	1542	4597	2648	2643	5704	3267	3219

Table 6.8: Polynomial interpretations with negative coefficients from $\{-1,0,1\}$.

	p	up		sup	
success	151	264	(192)	303	(286)
	0.46	0.36	(0.37)	0.30	(0.20)
failure	398	356	(386)	324	(325)
	2.29	1.36	(1.99)	1.29	(1.74)
timeout (*30* s)	224	153	(195)	146	(162)
total time	7702	5169	(6687)	4889	(5484)

6.3 Comparison

Needless to say, T$_{\text{T}}$T is not the only available tool for proving termination of rewrite systems. In this final section we compare our tool with the other tools that participated in the TRS category of the 2005 termination competition.[7]

- We start our discussion with CiME [10], the very first tool for automatically proving termination of rewrite systems that is still available. CiME is a tool with powerful techniques for finding termination proofs based on polynomial interpretations in connection with the dependency pair method. Since CiME does not support (yet) the most recent insights in the dependency pair method, it is less powerful than AProVE (described below) or T$_{\text{T}}$T. In contrast to T$_{\text{T}}$T, CiME can handle rewrite systems with AC operators. As a matter of fact, termination is only a side-issue in CiME. Its main strength lies in completing equational theories modulo theories like AC and C.

- Matchbox [57] is a tool that is entirely based on methods from formal language theory. These methods are especially useful for proving termination

[7] We downloaded the tools from `http://www.lri.fr/~marche/termination-competition/2005`

Table 6.9: Fully automatic mode.

t	*1*	*2*	*10*	*30*	
success	387	394	403	411	(411)
	0.03	0.06	0.17	0.55	(0.55)
failure	180	200	223	239	(234)
	0.20	0.32	0.72	1.75	(2.31)
timeout (t s)	206	179	147	128	(123)
total time	254	445	1696	4331	(4604)

of string rewrite systems. Matchbox tries to establish termination or non-termination by using recent results on match-bounded rewriting [18, 19].

- TEPARLA [56] is a new tool based on semantic labelling [60] with a two-valued domain and polynomial interpretations. By applying labelling repeatedly, the limitation of having only two labels is (partly) overcome. As a preprocessing step, rewrite systems are transformed into equivalent ones (as far as non-termination is concerned) over a signature containing only function symbols of arity at most two.

- TPA [38] is another recent tool based on semantic labelling. Besides a two-valued domain it also uses natural numbers as labels [39], which is surprisingly powerful. Polynomial interpretations and recursive path orders are available as basic techniques. Both TEPARLA and TPA can prove relative termination (Geser [17]). Because of semantic labelling, both tools are capable of proving termination of rewrite systems that are not handled by any current tool based on dependency pairs (cf. Table 6.3).

- Last but not least, AProVE [24], a very powerful tool for proving termination and non-termination of ordinary rewrite systems (possibly modulo AC), logic programs, conditional rewrite systems, context-sensitive rewrite systems. Of all existing tools, AProVE supports the most base orders and even offers several different algorithms implementing these. AProVE has several methods that are not available in any other tool. We mention here the size-change principle [52], transformations for dependency pairs like narrowing and instantiation, and a modular refinement where the set of usable rules is determined after a suitable argument filtering has been computed. These are integrated by the dependency pair framework [25], which encompasses the recursive SCC algorithm.

Table 6.3 shows the performance of these tools on the important examples in this paper. The left column for T$_{\text{T}}$T refers to the 2005 termination competition version, the right column to the version described in Section 6.1. The difference (Example 5.9) is due to Theorem 5.20. Example 3.38 shows the usefulness of semantic labelling. AProVE handles the leading example of Chapter 3 (`dnf`) by the size-change principle; for the leading example of Section 5.1 (`identity`) it uses a polynomial interpretation with negative constants. TPA solves the latter example by semantic labelling with natural numbers.

Table 6.10: Termination tools. (3600 seconds timeout)

	AProVE	CiME	Matchbox	TEPARLA	TPA	TTT	
Example 1.3	**2.54**	**0.81**	∞	*77.21*	∞	**0.32**	**0.06**
Example 3.15	**3.76**	**374.74**	∞	*85.22*	∞	**0.24**	**0.02**
Example 3.16	**0.93**	**0.03**	*29.02*	*61.60*	**2.90**	**0.01**	**0.00**
Example 3.38	*61.11*	*0.04*	*96.66*	**0.89**	**1.33**	*4.71*	*6.43*
Example 4.17	**5.10**	**1.80**	**2.70**	**0.10**	**0.17**	**0.08**	**0.00**
Example 4.23	**1.88**	*0.04*	*810.56*	**0.02**	**0.02**	**0.02**	**0.11**
Example 4.24	∞	∞	∞	*1730.37*	∞	*225.03*	∞
Example 4.25	**9.96**	**504.14**	∞	**169.60**	**1674**	**0.16**	**0.08**
Example 4.41	**1.34**	*568.95*	*1133.09*	*2708.99*	∞	**0.11**	**0.01**
Example 5.4	**2.27**	*0.09*	*7.09*	*3.46*	**5.41**	**0.01**	**0.01**
Example 5.9	*65.05*	*858.00*	∞	*12.64*	∞	*3.80*	**0.02**
`dnf`	**1.40**	*0.07*	∞	*47.26*	∞	**0.01**	**0.00**
`identity`	**2.23**	*0.22*	*8.43*	*5.28*	**0.01**	**0.01**	**0.00**

Bibliography

[1] T. Aoto and T. Yamada. Termination of simply typed term rewriting by translation and labelling. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 380–394, 2003.

[2] T. Aoto and T. Yamada. Termination of simply-typed applicative term rewriting systems. In *Proceedings of the 2nd International Workshop on Higher-Order Rewriting*, Technical Report AIB-2004-03, RWTH Aachen, pages 61–65, 2004.

[3] T. Arts. System description: The dependency pair method. In *Proceedings of the 11th International Conference on Rewriting Techniques and Applications*, volume 1833 of *Lecture Notes in Computer Science*, pages 261–264, 2000.

[4] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[5] T. Arts and J. Giesl. A collection of examples for termination of term rewriting using dependency pairs. Technical Report AIB-2001-09, RWTH Aachen, 2001.

[6] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[7] L. Bachmair and N. Dershowitz. Commutation, transformation, and termination. In *Proceedings of the 8th International Conference on Automated Deduction*, volume 230 of *Lecture Notes in Computer Science*, pages 5–20, 1986.

[8] C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In *Proceedings of the 17th International Conference on Automated Deduction*, volume 1831 of *Lecture Notes in Artificial Intelligence*, pages 346–364, 2000.

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

[10] E. Contejean, C. Marché, B. Monate, and X. Urbain. C*i*ME version 2, 2000. Available at `http://cime.lri.fr/`.

[11] E. Contejean, C. Marché, A.-P. Tomás, and X. Urbain. Mechanically proving termination using polynomial interpretations. *Journal of Automated Reasoning*, 2006. To appear.

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

[13] N. Dershowitz. 33 Examples of termination. In *French Spring School of Theoretical Computer Science*, volume 909 of *Lecture Notes in Computer Science*, pages 16–26, 1995.

[14] N. Dershowitz. Termination by abstraction. In *Proceedings of the 20th International Conference on Logic Programming*, volume 3132 of *Lecture Notes in Computer Science*, pages 1–18, 2004.

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

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

[17] A. Geser. *Relative Termination*. PhD thesis, University of Passau, 1990.

[18] A. Geser, D. Hofbauer, and J. Waldmann. Match-bounded string rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 15:149–171, 2004.

[19] A. Geser, D. Hofbauer, J. Waldmann, and H. Zantema. On tree automata that certify termination of left-linear term rewriting systems. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3467, pages 353–267, 2005.

[20] J. Giesl. Generating polynomial orderings for termination proofs. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Computer Science*, pages 426–431, 1995.

[21] J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1,2):39–72, 2001.

[22] J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.

[23] J. Giesl and A. Middeldorp. Transformation techniques for context-sensitive rewrite systems. *Journal of Functional Programming*, 14:329–427, 2004.

[24] J. Giesl, P. Schneider-Kamp, and R. Thiemann. Aprove 1.2: Automatic termination proofs in the dependency pair framework. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, 2006. To appear.

[25] J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 3452 of *Lecture Notes in Artificial Intelligence*, pages 301–331, 2005.

[26] J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 2850 of *Lecture Notes in Artificial Intelligence*, pages 165–179, 2003.

[27] B. Gramlich. Generalized sufficient conditions for modular termination of rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 5:131–158, 1994.

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

[29] N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proceedings of the 14th International Conference on Rewriting Techniques and Applications*, volume 2706 of *Lecture Notes in Computer Science*, pages 311–320, 2003.

[30] N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 249–268, 2004.

[31] N. Hirokawa and A. Middeldorp. Polynomial interpretations with negative coefficients. In *Proceedings of the 7th International Conference on Artificial Intelligence and Symbolic Computation*, volume 3249 of *Lecture Notes in Artificial Intelligence*, pages 185–198, 2004.

[32] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199(1,2):172–199, 2005. A preliminary version appeared in the proceedings of the 19th International Conference on Automated Deduction, volume 2741 of *LNAI*, pages 32–46, 2003.

[33] N. Hirokawa and A. Middeldorp. Tyrolean termination tool. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3467 of *Lecture Notes in Computer Science*, pages 175–184, Nara, 2005. Springer-Verlag.

[34] H. Hong and D. Jakuš. Testing positiveness of polynomials. *Journal of Automated Reasoning*, 21:23–28, 1998.

[35] G. Huet and D. Lankford. On the uniform halting problem for term rewriting systems. Technical Report 283, INRIA, 1980.

[36] S. Kamin and J.J. Lévy. Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois, 1980.

[37] D.E. Knuth and P.B. Bendix. Simple word problems in universal algebra. In *Computational Problems in Abstract Algebra*, Pergamon Press, pages 263–297, 1970.

[38] A. Koprowski. TPA: Termination proved automatically. In *Proceedings of the 17th International Conference on Rewriting Techniques and Applications*, Lecture Notes in Computer Science, 2006. To appear.

[39] A. Koprowski and H. Zantema. Recursive path ordering for infinite labelled rewrite systems. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, Lecture Notes in Artificial Intelligence, 2006. To appear.

[40] K. Korovin and A. Voronkov. Orienting rewrite rules with the Knuth-Bendix order. *Information and Computation*, 183:165–186, 2003.

[41] D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

[42] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.

[43] S. Lucas. Polynomials for proving termination of context-sensitive rewriting. In *Proceedings of the 7th International Conference on Foundations of Software Science and Computation Structures*, volume 2987 of *Lecture Notes in Computer Science*, pages 318–332, 2004.

[44] S. Lucas. Polynomials over the reals in proof of termination. *RAIRO Theoretical Informatics and Applications*, 39:547–586, 2005.

[45] A. Middeldorp. Approximating dependency graphs using tree automata techniques. In *Proceedings of the International Joint Conference on Automated Reasoning*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 593–610, 2001.

[46] A. Middeldorp. Approximations for strategies and termination. In *Proceedings of the 2nd International Workshop on Reduction Strategies in Rewriting and Programming*, volume 70(6) of *Electronic Notes in Theoretical Computer Science*, 2002.

[47] A. Middeldorp and H. Ohsaki. Type introduction for equational rewriting. *Acta Informatica*, 36(12):1007–1029, 2000.

[48] B.C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

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

[50] J. Steinbach and U. Kühler. Check your ordering – termination proofs and open problems. Technical Report SR-90-25, Universität Kaiserslautern, 1990.

[51] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003.

[52] R. Thiemann and J. Giesl. The size-change principle and dependency pairs for termination of term rewriting. *Applicable Algebra in Engineering, Communication and Computing*, 16(4):229–270, 2005.

[53] R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proceedings of the 2nd International Joint Conference on Automated Reasoning*, volume 3097 of *Lecture Notes in Artificial Intelligence*, pages 75–90, 2004.

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

[55] X. Urbain. Modular & incremental automated termination proofs. *Journal of Automated Reasoning*, 32:315–355, 2004.

[56] J. van der Wulp. Proving termination of term rewriting automatically. Technical Report 1075, Technische Universiteit Eindhoven, 2005. Master's thesis.

[57] J. Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications*, volume 3091 of *Lecture Notes in Computer Science*, pages 85–94, 2004.

[58] H. Xi. Towards automated termination proofs through "freezing". In *Proceedings of the 9th International Conference on Rewriting Techniques and Applications*, volume 1379 of *Lecture Notes in Computer Science*, pages 271–285, 1998.

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

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

Appendix A

Example

We use T$_\text{T}$T to prove termination of the TRS in Example 1.3. The input looks like this:

```
[x, y, n, xs, ys]

   x - 0    -> x;
s(x) - s(y) -> x - y;

   0 <= y    -> true;
s(x) <= 0    -> false;
s(x) <= s(y) -> x <= y;

if(true,  x, y) -> x;
if(false, x, y) -> y;

   x | 0    -> true;
s(x) | s(y) -> if(x <= y, s(x) | (y-x), false);

filter(x, nil)    -> nil;
filter(x, y : ys) -> if_filter(x | y, x, y : ys);
if_filter(true,  x, y : ys) -> filter(x, ys);
if_filter(false, x, y : ys) -> y : filter(x, ys);

sieve(nil)    -> nil;
sieve(x : xs) -> x : sieve(filter(x, xs));
```

The next three pages contain the termination proof generated by T$_\text{T}$T.

Termination Proof Script[1]

Consider the TRS $\mathcal{R}$ consisting of the rewrite rules

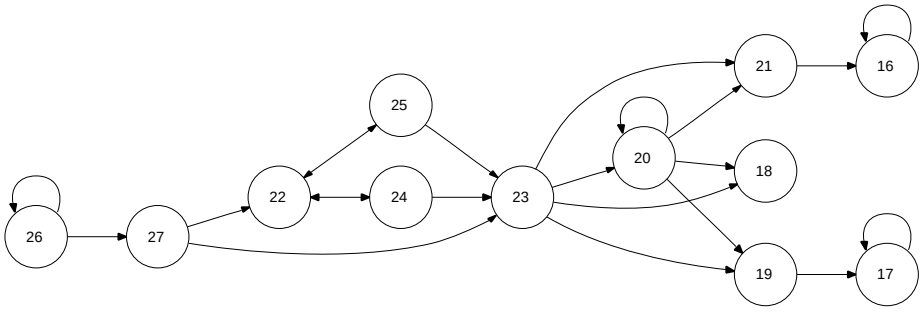
$$
\begin{aligned}
1:&\quad x - 0 &&\to x\\
2:&\quad \mathsf{s}(x) - \mathsf{s}(y) &&\to x - y\\
3:&\quad 0 \leqslant y &&\to \mathsf{true}\\
4:&\quad \mathsf{s}(x) \leqslant 0 &&\to \mathsf{false}\\
5:&\quad \mathsf{s}(x) \leqslant \mathsf{s}(y) &&\to x \leqslant y\\
6:&\quad \mathsf{if}(\mathsf{true}, x, y) &&\to x\\
7:&\quad \mathsf{if}(\mathsf{false}, x, y) &&\to y\\
8:&\quad x \mid 0 &&\to \mathsf{true}\\
9:&\quad \mathsf{s}(x) \mid \mathsf{s}(y) &&\to \mathsf{if}(x \leqslant y, \mathsf{s}(x) \mid (y - x), \mathsf{false})\\
10:&\quad \mathsf{filter}(x, \mathsf{nil}) &&\to \mathsf{nil}\\
11:&\quad \mathsf{filter}(x, y : ys) &&\to \mathsf{if}_{\mathsf{filter}}(x \mid y, x, y : ys)\\
12:&\quad \mathsf{if}_{\mathsf{filter}}(\mathsf{true}, x, y : ys) &&\to \mathsf{filter}(x, ys)\\
13:&\quad \mathsf{if}_{\mathsf{filter}}(\mathsf{false}, x, y : ys) &&\to y : \mathsf{filter}(x, ys)\\
14:&\quad \mathsf{sieve}(\mathsf{nil}) &&\to \mathsf{nil}\\
15:&\quad \mathsf{sieve}(x : xs) &&\to x : \mathsf{sieve}(\mathsf{filter}(x, xs))
\end{aligned}
$$

There are 12 dependency pairs:

$$
\begin{aligned}
16:&\quad \mathsf{s}(x) -^{\sharp} \mathsf{s}(y) &&\to x -^{\sharp} y\\
17:&\quad \mathsf{s}(x) \leqslant^{\sharp} \mathsf{s}(y) &&\to x \leqslant^{\sharp} y\\
18:&\quad \mathsf{s}(x) \mid^{\sharp} \mathsf{s}(y) &&\to \mathsf{IF}(x \leqslant y, \mathsf{s}(x) \mid (y - x), \mathsf{false})\\
19:&\quad \mathsf{s}(x) \mid^{\sharp} \mathsf{s}(y) &&\to x \leqslant^{\sharp} y\\
20:&\quad \mathsf{s}(x) \mid^{\sharp} \mathsf{s}(y) &&\to \mathsf{s}(x) \mid^{\sharp} (y - x)\\
21:&\quad \mathsf{s}(x) \mid^{\sharp} \mathsf{s}(y) &&\to y -^{\sharp} x\\
22:&\quad \mathsf{FILTER}(x, y : ys) &&\to \mathsf{IF}_{\mathsf{FILTER}}(x \mid y, x, y : ys)\\
23:&\quad \mathsf{FILTER}(x, y : ys) &&\to x \mid^{\sharp} y\\
24:&\quad \mathsf{IF}_{\mathsf{FILTER}}(\mathsf{true}, x, y : ys) &&\to \mathsf{FILTER}(x, ys)\\
25:&\quad \mathsf{IF}_{\mathsf{FILTER}}(\mathsf{false}, x, y : ys) &&\to \mathsf{FILTER}(x, ys)\\
26:&\quad \mathsf{SIEVE}(x : xs) &&\to \mathsf{SIEVE}(\mathsf{filter}(x, xs))\\
27:&\quad \mathsf{SIEVE}(x : xs) &&\to \mathsf{FILTER}(x, xs)
\end{aligned}
$$

The approximated dependency graph

[1] Tyrolean Termination Tool (0.03 seconds) — May 21, 2006



contains 5 SCCs: $\{16\}$, $\{17\}$, $\{20\}$, $\{22, 24, 25\}$ and $\{26\}$.

- Consider the SCC $\{16\}$. By taking the simple projection π with $\pi(-^\sharp) = 1$, the dependency pair simplifies to

$$16: \quad \mathsf{s}(x) \;\to\; x$$

  and is compatible with the proper subterm relation.

- Consider the SCC $\{17\}$. By taking the simple projection π with $\pi(\leqslant^\sharp) = 1$, the dependency pair simplifies to

$$17: \quad \mathsf{s}(x) \;\to\; x$$

  and is compatible with the proper subterm relation.

- Consider the SCC $\{20\}$. By taking the polynomial interpretation $\mathsf{true}_{\mathbb{N}} = \mathsf{sieve}_{\mathbb{N}}(x) = \mathsf{nil}_{\mathbb{N}} = \mathsf{false}_{\mathbb{N}} = \mathsf{0}_{\mathbb{N}} = 1$, $-_{\mathbb{N}}(x, y) = x$, $\mathsf{s}_{\mathbb{N}}(x) = x + 1$, $:_{\mathbb{N}}(x, y) = y$, $|^\sharp_{\mathbb{N}}(x, y) = |_{\mathbb{N}}(x, y) = \mathsf{filter}_{\mathbb{N}}(x, y) = \leqslant_{\mathbb{N}}(x, y) = y + x + 1$, $\mathsf{if}_{\mathbb{N}}(x, y, z) = z + y$ and $\mathsf{if}_{\mathsf{filter}\mathbb{N}}(x, y, z) = z + y + 1$, the involved rules reduce to the following inequalities:

$$\begin{aligned}
1: &\quad x &&\gtrsim x\\
2: &\quad x+1 &&> x\\
3: &\quad y+2 &&> 1\\
4: &\quad x+3 &&> 1\\
5: &\quad x+y+3 &&> x+y+1\\
6: &\quad x+y &&\gtrsim x\\
7: &\quad x+y &&\gtrsim y\\
8: &\quad x+2 &&> 1\\
9: &\quad x+y+3 &&\gtrsim x+y+3\\
10: &\quad x+2 &&> 1\\
11: &\quad x+ys+1 &&\gtrsim x+ys+1\\
12: &\quad x+ys+1 &&\gtrsim x+ys+1\\
13: &\quad x+ys+1 &&\gtrsim x+ys+1\\
14: &\quad 1 &&\gtrsim 1\\
15: &\quad 1 &&\gtrsim 1\\
20: &\quad x+y+3 &&> x+y+2
\end{aligned}$$

- Consider the SCC $\{22, 24, 25\}$. By taking the simple projection π with $\pi(\mathsf{FILTER}) = 2$ and $\pi(\mathsf{IF}_{\mathsf{FILTER}}) = 3$, the dependency pairs simplify to

$$\begin{aligned} 22: &\quad y : ys \;\to\; y : ys \\ 24: &\quad y : ys \;\to\; ys \\ 25: &\quad y : ys \;\to\; ys \end{aligned}$$

  and are compatible with the subterm relation: rule 22 is weakly decreasing and the rules in $\{24, 25\}$ are strictly decreasing.

- Consider the SCC $\{26\}$. By taking the polynomial interpretation $\mathsf{true}_{\mathbb{N}} = \mathsf{nil}_{\mathbb{N}} = \mathsf{false}_{\mathbb{N}} = \mathsf{0}_{\mathbb{N}} = 1$, $-_{\mathbb{N}}(x, y) = x$, $\mathsf{SIEVE}_{\mathbb{N}}(x) = \mathsf{sieve}_{\mathbb{N}}(x) = \mathsf{s}_{\mathbb{N}}(x) = x + 1$, $\mathsf{filter}_{\mathbb{N}}(x, y) = y$, $|_{\mathbb{N}}(x, y) = \leqslant_{\mathbb{N}}(x, y) = :_{\mathbb{N}}(x, y) = y + x + 1$, $\mathsf{if}_{\mathsf{filter}\,\mathbb{N}}(x, y, z) = z$ and $\mathsf{if}_{\mathbb{N}}(x, y, z) = z + y$, the involved rules reduce to the following inequalities:

$$\begin{aligned} 1: &\quad x \gtrsim x \\ 2: &\quad x + 1 > x \\ 3: &\quad y + 2 > 1 \\ 4: &\quad x + 3 > 1 \\ 5: &\quad x + y + 3 > x + y + 1 \\ 6: &\quad x + y \gtrsim x \\ 7: &\quad x + y \gtrsim y \\ 8: &\quad x + 2 > 1 \\ 9: &\quad x + y + 3 \gtrsim x + y + 3 \\ 10: &\quad 1 \gtrsim 1 \\ 11: &\quad ys + y + 1 \gtrsim ys + y + 1 \\ 12: &\quad ys + y + 1 > ys \\ 13: &\quad ys + y + 1 \gtrsim ys + y + 1 \\ 14: &\quad 2 > 1 \\ 15: &\quad xs + x + 2 \gtrsim xs + x + 2 \\ 26: &\quad xs + x + 2 > xs + 1 \end{aligned}$$

Hence the TRS is terminating.

Index