Automating the Dependency Pair Method

Nao Hirokawa   Aart Middeldorp

Institute of Computer Science
University of Innsbruck
6020 Innsbruck, Austria

Abstract

Developing automatable methods for proving termination of term rewrite systems that resist traditional techniques based on simplification orders has become an active research area in the past few years. The dependency pair method of Arts and Giesl is one of the most popular such methods. However, there are several obstacles that hamper its automation. In this paper we present new ideas to overcome these obstacles. We provide ample numerical data supporting our ideas.

Key words: Term Rewriting, Termination, Dependency Pair Method

1 Introduction

Proving termination of term rewrite systems has been an active research area for several decades. In recent years the emphasis has shifted towards the development of powerful methods for automatically proving termination. The traditional methods for automated termination proofs of rewrite systems are simplification orders like the recursive path order, the Knuth-Bendix order, and (most) polynomial orders. The termination proving power of these methods has been significantly extended by the *dependency pair method* of Arts and Giesl [2]. In this method, depicted in Fig. 1, a rewrite system is transformed into groups of ordering constraints such that termination of the system is equivalent to the (separate) solvability of these groups. The number and size of these groups is determined by the approximation used to estimate the dependency graph and, more importantly, by the cycle analysis algorithm that

Email addresses: `nao.hirokawa@uibk.ac.at` (Nao Hirokawa), `aart.middeldorp@uibk.ac.at` (Aart Middeldorp).

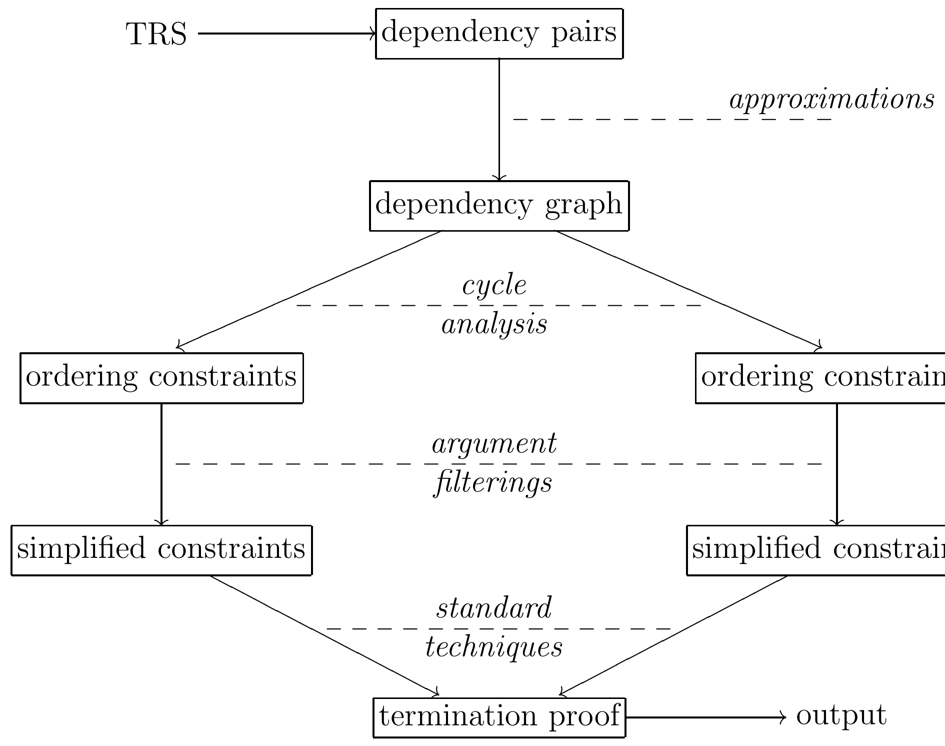

Fig. 1. The dependency pair method.

is used to extract the groups from the approximated dependency graph. 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.

The dependency pair method can also be used for automatically proving *innermost* termination. Innermost termination is easier to prove than termination for the following two reasons: (1) the innermost dependency graph is typically much smaller than the dependency graph and (2) each group of ordering constraints for innermost termination is (often strictly) contained in a group of ordering constraints for termination.

We present several new ideas which help to tackle the argument filtering problem in Section 5. In Section 4 we present a new algorithm for cycle analysis and in Section 3 we present new approximations of the (innermost) dependency graph. A brief introduction to the dependency pair method is given in the next section. In Section 6 we report on the numerous experiments that we performed to assess the viability of our ideas.

It goes without saying that the dependency pair method is not the only automatable method for proving termination of rewrite systems that cannot be handled by traditional simplification orders. We mention here the pioneering work of Steinbach [22] on automating the transformation order of Bellegarde and Lescanne [5] and the more recent work of Borralleras *et al.* [6] on transforming the semantic path order of Kamin and Lévy [17] into a monotonic version that is amenable to automation. We believe that an implementation of the monotonic semantic path order of [6] may benefit from the ideas presented in this paper.

2 Dependency Pairs

We assume familiarity with the basics of term rewriting ([4]). In this section we recall the basic notions and results of the dependency pair method. Throughout this paper we deal with finite term rewrite systems over finite signatures. We refer to [2,9,10,12,21,26] for motivations and additional refinements. [1] Let $\mathcal{R}$ be a term rewrite system (TRS for short) over a signature $\mathcal{F}$. The set of function symbols appearing in a term t is denoted by $\mathcal{F}\mathsf{un}(t)$. The root symbol of a term t is denoted by $\mathrm{root}(t)$; if t is a variable then $\mathrm{root}(t) = t$. A function symbol $f \in \mathcal{F}$ is defined if $f = \mathrm{root}(l)$ for some rewrite rule $l \to r \in \mathcal{R}$. Let $\mathcal{F}^\sharp$ denote the union of $\mathcal{F}$ and $\{f^\sharp \mid f \text{ is a defined symbol of } \mathcal{R}\}$ where $f^\sharp$ has the same arity as f. Given a term $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with f defined, we write $t^\sharp$ for the term $f^\sharp(t_1, \ldots, t_n)$. (Here $\mathcal{V}$ denotes the infinite set of variables at our disposal.) If $l \to r \in \mathcal{R}$ and t is a subterm of r with defined root symbol then the rewrite rule $l^\sharp \to t^\sharp$ is a *dependency pair* of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted by $\mathsf{DP}(\mathcal{R})$. In examples we often write F for $\mathsf{f}^\sharp$. 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: $\pi(t) = t$ if t is a variable, $\pi(t) = \pi(t_i)$ if $t = f(t_1, \ldots, t_n)$ and $\pi(f) = i$, and $\pi(t) = f(\pi(t_{i_1}), \ldots, \pi(t_{i_m}))$ if $t = f(t_1, \ldots, t_n)$ and $\pi(f) = [i_1, \ldots, i_m]$.

[1] Some of the refinements (like narrowing and instantiation) transform dependency pairs with the aim of simplifying the resulting ordering constraints, other refinements use the modular structure of the TRS to reduce the number of constraints; most refinements are orthogonal to the ideas we develop in this paper.

Thus, an argument filtering is used to replace function symbols by one of their arguments or to eliminate certain arguments of function symbols. In Section 5 we consider argument filterings that are partially defined.

A *reduction pair* consists of a rewrite preorder $\gtrsim$ (i.e., a transitive and reflexive relation on terms which is closed under contexts and substitutions) and a compatible well-founded order $>$ which is closed under substitutions. Compatibility means that the inclusion $\gtrsim \cdot > \subseteq >$ or the inclusion $> \cdot \gtrsim \subseteq >$ holds. Reduction pairs are used to solve groups of simplified ordering constraints and hence are typically based on traditional simplification orders. In all our examples and experiments we use the pair $(\succ_{\text{lpo}}^{=}, \succ_{\text{lpo}})$ for some strict precedence $\succ$. Here $\succ_{\text{lpo}}$ denotes the lexicographic path order (LPO) induced by $\succ$ and $\succ_{\text{lpo}}^{=}$ denotes the reflexive closure of $\succ_{\text{lpo}}$.

Theorem 1 (Arts and Giesl [2]) *A TRS $\mathcal{R}$ over a signature $\mathcal{F}$ is terminating if and only if there exist an argument filtering π for $\mathcal{F}^\sharp$ and a reduction pair $(\gtrsim, >)$ such that $\pi(l) \gtrsim \pi(r)$ for every rewrite rule $l \to r \in \mathcal{R}$ and $\pi(l) > \pi(r)$ for every dependency pair $l \to r \in \mathsf{DP}(\mathcal{R})$.* □

We abbreviate the two conditions in the above theorem to $\pi(\mathcal{R}) \subseteq \gtrsim$ and $\pi(\mathsf{DP}(\mathcal{R})) \subseteq >$. Rather than considering all dependency pairs at the same time, like in the above theorem, it is advantageous to treat groups of dependency pairs separately. These groups are extracted from the *dependency graph* $\mathsf{DG}(\mathcal{R})$ of $\mathcal{R}$. The nodes of $\mathsf{DG}(\mathcal{R})$ 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 exists a substitution σ such that $t\sigma \to_{\mathcal{R}}^{*} u\sigma$.[2] A *cycle* is a non-empty 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 non-empty path in $\mathcal{C}$ from $s \to t$ to $u \to v$.

Theorem 2 (Giesl, Arts, and Ohlebusch [10]) *A TRS $\mathcal{R}$ is terminating if and only if for every cycle $\mathcal{C}$ in $\mathsf{DG}(\mathcal{R})$ there exist an argument filtering π and a reduction pair $(\gtrsim, >)$ such that $\pi(\mathcal{R}) \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \gtrsim \cup >$, and $\pi(\mathcal{C}) \cap > \neq \varnothing$.* □

The last condition in Theorem 2 denotes the situation that $\pi(s) > \pi(t)$ for at least one dependency pair $s \to t \in \mathcal{C}$.

Definition 3 *Let $\mathcal{R}$ be a TRS and let $\mathcal{C}$ be a subset of $\mathsf{DP}(\mathcal{R})$. For $\mathcal{R}' \subseteq \mathcal{R}$ we write $\vDash_\exists \mathcal{R}', \mathcal{C}$ if there exist an argument filtering π and a reduction pair $(\gtrsim, >)$ such that $\pi(\mathcal{R}') \subseteq \gtrsim$, $\pi(\mathcal{C}) \subseteq \gtrsim \cup >$, and $\pi(\mathcal{C}) \cap > \neq \varnothing$. We write $(\gtrsim, >)_\pi \vDash_\exists \mathcal{R}', \mathcal{C}$ if we want to indicate a combination of argument filtering and reduction pair that makes $\vDash_\exists \mathcal{R}', \mathcal{C}$ true.*

The existential quantifier in the notation indicates that *some* pair in $\mathcal{C}$ should

[2] The terms t and u do not share variables since different occurrences of dependency pairs are always assumed to be variable disjoint.

be strictly decreasing. Theorem 2 can now be simply stated as "A TRS $\mathcal{R}$ is terminating if and only if $\vDash_\exists \mathcal{R}, \mathcal{C}$ for every cycle $\mathcal{C}$ in $\mathsf{DG}(\mathcal{R})$."

Example 4 *Consider the following TRS (from [3]):*

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

There are three dependency pairs:

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

The dependency graph

$$\begin{tikzcd} 1 & 3 \arrow[l] \arrow[r, leftrightarrow] & 2 \end{tikzcd}$$

contains one cycle: $\{2, 3\}$*. The constraints generated by Theorem 2 can be solved by taking the argument filtering* π *with* $\pi(\mathsf{EVENODD}) = 1$, $\pi(\mathsf{evenodd}) = 2$, $\pi(\mathsf{not}) = [\,]$ *and LPO with precedence* $\mathsf{0} \succ \mathsf{not} \succ \mathsf{false}$ *and* $\mathsf{not} \succ \mathsf{true}$.

We conclude this introductory section by stating the following variant of Theorem 2 for *innermost* termination. Here $\mathsf{IDG}(\mathcal{R})$ denotes the innermost dependency graph of $\mathcal{R}$, whose nodes 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 in normal form and $t\sigma \xrightarrow{\mathrm{i}}{}^{*}_{\mathcal{R}} u\tau$, where $\xrightarrow{\mathrm{i}}_{\mathcal{R}}$ denotes innermost rewriting. Furthermore, $\mathcal{U}(\mathcal{C})$ denotes the set of *usable* rules of $\mathcal{R}$ for the function symbols in the right-hand sides of the dependency pairs in $\mathcal{C}$. These are defined as follows. 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 $g \in \mathcal{F}\mathsf{un}(r)$. For a set $\mathcal{G}$ of function symbols we denote by $\mathcal{R}{\restriction}\mathcal{G}$ the set of all 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 $\mathcal{U}(\mathcal{C}) = \bigcup_{l \to r \in \mathcal{C}} \mathcal{U}(r)$.

Theorem 5 ([10]) *A TRS* $\mathcal{R}$ *is innermost terminating if* $\vDash_\exists \mathcal{U}(\mathcal{C}), \mathcal{C}$ *for every cycle* $\mathcal{C}$ *in* $\mathsf{IDG}(\mathcal{R})$. $\square$

The constraints of Theorem 5 are easier to satisfy than those of Theorem 2. Moreover, several conditions are known which guarantee that innermost termination implies termination, the most important of which is the absence of critical pairs ([13]).

Example 6 *We show that the following TRS $\mathcal{R}$ (from [3]) is terminating:*

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

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

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

The innermost dependency graph

$$\begin{tikzcd}
6 \arrow[loop above] & 7 \arrow[l] & 9 \arrow[l] \arrow[r, leftrightarrow] \arrow[loop above] & 8
\end{tikzcd}$$

contains three cycles: $\{6\}$, $\{9\}$, and $\{8, 9\}$. However, the computable approximations of the innermost dependency graph presented in the next section contain additional arrows from 8 to 7 and from 8 to 8, resulting in the additional cycle $\{8\}$. The constraints generated by Theorem 5 can be solved as follows.

- *For cycle $\{6\}$ there are no usable rules. LPO with empty precedence orients rule 6 from left to right.*
- *For cycle $\{8\}$ the usable rules are $\{1, 2\}$. If we take the argument filtering π with $\pi(+) = [2]$, $\pi(\mathsf{s}) = [\,]$, and $\pi(\mathsf{QUOT}) = 2$, the resulting constraints are satisfied by LPO with precedence $\mathsf{0} \succ + \succ \mathsf{s}$.*
- *For cycle $\{9\}$ there are no usable rules. We take LPO with empty precedence.*
- *For cycle $\{8, 9\}$ the usable rules are $\{1, 2\}$. We take the argument filtering π with $\pi(\mathsf{QUOT}) = 1$ and LPO with precedence $+ \succ \mathsf{s}$.*

In the next three sections we address the various problems that arise when automating the dependency pair technique.

3 Dependency Graph Approximations

Since it is undecidable whether there exist substitutions σ, τ such that $t\sigma \to_{\mathcal{R}}^* u\tau$ ($t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}}^* u\tau$), the (innermost) dependency graph cannot be computed in general. Hence, in order to mechanize the termination criterion of Theorem 2 (5) one has to approximate the (innermost) dependency graph. Arts and Giesl [2] proposed simple approximations based on syntactic unification for this purpose.

Definition 7 *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 8 ([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 [19] showed that better approximations of the dependency graph are obtained by adopting tree automata techniques. These techniques are however computationally expensive. In a recent paper Middeldorp [20] showed that the approximation of Definition 7 can be improved by symmetry considerations without incurring the overhead of tree automata techniques.

Definition 9 *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 [19] can be found in [20]. 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 10 *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 [25] 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 $\mathsf{F}(x, x, x)$ rewrites to an instance of $\mathsf{F}(\mathsf{a}, \mathsf{b}, x)$. On the other hand, the estimated dependency graph contains a cycle and the constraints resulting from Theorem 2 cannot be solved by any quasi-simplification order.*

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

Definition 11 *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 $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))$ 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 12 *Consider the non-terminating TRS $\mathcal{R}$ consisting of the rules*

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

There are two dependency pairs:

$$1\colon \mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},x)) \to \mathsf{H}(\mathsf{f}(x,x,x)) \qquad 2\colon \mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},x)) \to \mathsf{F}(x,x,x)$$

Because there are no terms s and t such that $\mathsf{H}(\mathsf{f}(s,s,s)) \xrightarrow{\mathsf{i}}{}^{*} \mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},t))$, the innermost dependency graph $\mathsf{IDG}(\mathcal{R})$ contains no arrows. Hence $\mathcal{R}$ is innermost terminating. We have $\mathcal{U}(\mathsf{H}(\mathsf{f}(x,x,x))) = \{\mathsf{f}(x,x,x) \to \mathsf{c}\}$ and thus $\mathsf{H}(\mathsf{f}(x,x,x))$ and $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(\mathsf{H}(\mathsf{f}(x,x,x)))}(\mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},x')))) = \mathsf{REN}(\mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},x'))) = \mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},x''))$ 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 $\mathsf{CAP}_{\mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},x))}(\mathsf{H}(\mathsf{f}(x,x,x))) = \mathsf{H}(y)$ unifies with $\mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},x'))$. 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 $\pi(\mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},x))) > \pi(\mathsf{H}(\mathsf{f}(x,x,x)))$ and $\pi(\mathsf{f}(x,x,x)) \gtrsim \pi(\mathsf{c})$. The first condition can only be satisfied if $\pi(\mathsf{H}) \in \{1,[1]\}$ 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 $\pi(\mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},t))) > \pi(\mathsf{H}(\mathsf{f}(t,t,t)))$ as $>$ is closed under substitutions. Closure under contexts of $\gtrsim$ yields $\pi(\mathsf{H}(\mathsf{f}(t,t,t))) \gtrsim \pi(\mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},t)))$. Hence $\pi(\mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},t))) > \pi(\mathsf{H}(\mathsf{f}(\mathsf{a},\mathsf{b},t)))$ by the compatibility of $>$ and $\gtrsim$, contradicting the well-foundedness of $>$.

The following lemma states the soundness of the new approximation.

Lemma 13 *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 $u = f(u_1, \ldots, u_n)$ with $f \notin \mathcal{D}^{-1}_{\mathcal{U}(t)}$ then $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u)) = f(\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u_1)), \ldots, \mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u_n)))$. 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 $f(t_1\sigma, \ldots, t_n\sigma)$ and we have $u_i\sigma \to^*_{\mathcal{U}(t)^{-1}} t_i\sigma$ for each $i \in \{1, \ldots, n\}$. 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 $\mu = \mu_1 \uplus \cdots \uplus \mu_n$ is well-defined and clearly satisfies $t\sigma = \mathsf{REN}(\mathsf{CAP}^{-1}_{\mathcal{U}(t)}(u))\mu$. $\square$

The following 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 13, which is essential for inferring innermost termination.

Example 14 *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:

$$1\colon \mathsf{F}(x, x) \to \mathsf{F}(\mathsf{g}(x), x) \qquad\qquad 2\colon \mathsf{F}(x, x) \to \mathsf{G}(x)$$

Since $\mathsf{F}(\mathsf{g}(\mathsf{h}(x)), \mathsf{h}(x)) \xrightarrow{\mathsf{i}} \mathsf{F}(\mathsf{h}(x), \mathsf{h}(x))$, $\mathsf{IDG}(\mathcal{R})$ *contains arrows from* 1 *to* 1 *and* 2. *However,* $\mathsf{CAP}^{-1}_{\mathcal{U}(\mathsf{F}(\mathsf{g}(x),x))}(\mathsf{F}(x', x')) = \mathsf{CAP}^{-1}_{\{\mathsf{g}(\mathsf{h}(x)) \to \mathsf{h}(x)\}}(\mathsf{F}(x', x')) = \mathsf{F}(x', x')$ *does not unify with* $\mathsf{F}(\mathsf{g}(x), x)$. *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 a 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 11 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 15 *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:* $\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{b})$. *Because of* $\mathsf{F}(\mathsf{b}) \xrightarrow{\mathsf{i}} \mathsf{F}(\mathsf{a})$, $\mathsf{IDG}(\mathcal{R})$ *contains a cycle. This cycle would not be detected if* CAP^{-1} *is replaced by* CAP^{-1}_s: $\mathsf{REN}(\mathsf{CAP}^{-1}_{\mathsf{F}(\mathsf{a})}(\mathsf{F}(\mathsf{a}))) = \mathsf{F}(\mathsf{a})$ *does not unify with* $\mathsf{F}(\mathsf{b})$.

The following theorem summarizes the relationships between the various approximations. The only non-trivial inclusions are $\mathsf{DG}(\mathcal{R}) \subseteq \mathsf{EDG}^*(\mathcal{R})$ ([20]) and $\mathsf{IDG}(\mathcal{R}) \subseteq \mathsf{EIDG}^*(\mathcal{R})$ (Lemma 13).

Theorem 16 *For any TRS $\mathcal{R}$, the following inclusions hold:*

$$\begin{array}{ccccc}
\mathsf{DG}(\mathcal{R}) & \subseteq & \mathsf{EDG}^*(\mathcal{R}) & \subseteq & \mathsf{EDG}(\mathcal{R}) \\
\rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} & & \rotatebox{90}{$\subseteq$} \\
\mathsf{IDG}(\mathcal{R}) & \subseteq & \mathsf{EIDG}^*(\mathcal{R}) & \subseteq & \mathsf{EIDG}(\mathcal{R})
\end{array}$$

□

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 11. This can be seen from Example 12.

4 Cycle Analysis

The use of Theorem 2 (5) for ensuring (innermost) termination requires that *all* cycles have to be considered.

Example 17 *Consider the TRS from [9] 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

$$1\colon \mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(\mathsf{s}(x)) \qquad 2\colon \mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(x)$$

and the dependency graph

$$\begin{tikzcd}
1 \arrow[loop above] \arrow[r, leftrightarrow] & 2 \arrow[loop above]
\end{tikzcd}$$

has three cycles. The constraints (generated by Theorems 2 and 5) for cycles $\{2\}$ and $\{1,2\}$ are readily satisfied, 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 ([1,7]) 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.

The next two statements are immediate consequences of Theorems 2 and 5.

Corollary 18 *A TRS $\mathcal{R}$ is terminating if and only if for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$ there exist an argument filtering π and a reduction pair $(\gtrsim, >)$ such that $\pi(\mathcal{R}) \subseteq \gtrsim$ and $\pi(\mathcal{S}) \subseteq >$.* $\square$

We find it convenient to abbreviate the two conditions in Corollary 18 to $(\gtrsim, >)_\pi \vDash_\forall \mathcal{R}, \mathcal{S}$. We write $\vDash_\forall \mathcal{R}, \mathcal{S}$ if there exist an argument filtering π and a reduction pair $(\gtrsim, >)$ such that $(\gtrsim, >)_\pi \vDash_\forall \mathcal{R}, \mathcal{S}$. The universal quantifier in the notation indicates that *all* pairs in $\mathcal{S}$ should be strictly decreasing.

Corollary 19 *A TRS $\mathcal{R}$ is innermost terminating if $\vDash_\forall \mathcal{U}(\mathcal{S}), \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{IDG}(\mathcal{R})$.* $\square$

The difference with Theorems 2 and 5 is that all pairs in an SCC must be strictly decreasing. This, however, makes the (innermost) termination criterion of Corollary 18 (19) strictly weaker than the one of Theorem 2 (5), if we employ reduction pairs based on (quasi-)simplification orders.

Example 20 *Consider again the TRS of Example 4. The dependency graph (which can be computed with the estimations mentioned in the preceding section) contains one SCC: $\{2, 3\}$. The constraints generated by Corollary 18 cannot be solved by a combination of an argument filtering π and a reduction pair $(\gtrsim, >)$ based on a quasi-simplification order $\gtrsim$. To see this, suppose that both $\pi(\mathsf{EVENODD}(x, \mathsf{0})) > \pi(\mathsf{EVENODD}(x, \mathsf{s}(\mathsf{0})))$ and $\pi(\mathsf{EVENODD}(\mathsf{s}(x), \mathsf{s}(\mathsf{0}))) > \pi(\mathsf{EVENODD}(x, \mathsf{0}))$. Since every quasi-simplification order satisfies $\mathsf{s}(\mathsf{0}) \gtrsim \mathsf{0}$, the first constraint requires $\pi(\mathsf{s}) = [\,]$, $\pi(\mathsf{EVENODD}) \in \{[1, 2], [2], 2\}$, and $\mathsf{0} > \mathsf{s}$. So the second constraint reduces to $\pi(\mathsf{EVENODD}(\mathsf{s}, \mathsf{s})) > \pi(\mathsf{EVENODD}(x, \mathsf{0}))$ and this latter constraint can only be satisfied if $\pi(\mathsf{EVENODD}) \in \{[2], 2\}$ and $\mathsf{s} > \mathsf{0}$. This is clearly impossible.*

Also the TRS of Example 6 cannot be proved (innermost) terminating if we use a quasi-simplification order in combination with Corollary 18 (19); one easily shows that there is no argument filtering π and reduction pair $(\gtrsim, >)$ based on a quasi-simplification order $\gtrsim$ such that both $\pi(\mathsf{QUOT}(x, \mathsf{0}, \mathsf{s}(z))) > \pi(\mathsf{QUOT}(x, z + \mathsf{s}(\mathsf{0}), \mathsf{s}(z)))$ and $\pi(\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y), z)) > \pi(\mathsf{QUOT}(x, y, z))$.

In order to cope with this problem, we propose a new recursive approach to compute and solve SCCs. More precisely, if $\mathcal{S}$ is the current SCC then we first compute (see the next section) an argument filtering π and a reduction pair $(\gtrsim, >)$ such that $\pi(\mathcal{R}) \subseteq \gtrsim$, $\pi(\mathcal{S}) \subseteq \gtrsim \cup >$, and $\pi(\mathcal{S}) \cap > \neq \varnothing$. Then we compute the SCCs of the subgraph[3] of $\mathsf{DG}(\mathcal{R})$ induced by the pairs $l \to r$ of $\mathcal{S}$ that are not strictly decreasing. 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 Theorem 2 (5) and the efficiency of Corollary 18 (19). The former is proved below and the latter is confirmed by extensive experiments (see Section 6) and explained in the paragraph following Theorem 24.

Definition 21 *Let $\mathcal{R}$ be a TRS and $\mathcal{S}$ a subset of the dependency pairs in $\mathsf{DP}(\mathcal{R})$. We write $\vDash \mathcal{R}, \mathcal{S}$ if there exist an argument filtering π and a reduction pair $(\gtrsim, >)$ such that $(\gtrsim, >)_\pi \vDash_\exists \mathcal{R}, \mathcal{S}$ and $\vDash \mathcal{R}, \mathcal{S}'$ for all SCCs $\mathcal{S}'$ of the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by the pairs $l \to r \in \mathcal{S}$ such that $\pi(l) \not> \pi(r)$.*

Theorem 22 *Let $\mathcal{R}$ be a TRS. The following conditions are equivalent:*

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

PROOF. First suppose $\vDash \mathcal{R}, \mathcal{S}$ 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 $\vDash_\exists \mathcal{R}, \mathcal{C}$. Let $\mathcal{S}$ be the SCC that contains $\mathcal{C}$. We use induction on the size of $\mathcal{S}$. We have $\vDash \mathcal{R}, \mathcal{S}$ by assumption. So there exist an argument filtering π and a reduction pair $(\gtrsim, >)$ such that $(\gtrsim, >)_\pi \vDash_\exists \mathcal{R}, \mathcal{S}$ and $\vDash \mathcal{R}, \mathcal{S}'$ for all SCCs $\mathcal{S}'$ of the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by the pairs $l \to r \in \mathcal{S}$ such that $\pi(l) \not> \pi(r)$. Let us denote the set of these pairs by $\bar{\mathcal{S}}$. If $\pi(\mathcal{C}) \cap > \neq \varnothing$ then $(\gtrsim, >)_\pi \vDash_\exists \mathcal{R}, \mathcal{C}$. Otherwise, all pairs in $\mathcal{C}$ belong to $\bar{\mathcal{S}}$ and thus $\mathcal{C}$ is a cycle in the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by $\bar{\mathcal{S}}$. Hence $\mathcal{C}$ is contained in an SCC $\mathcal{S}'$ of this subgraph. We have $\vDash \mathcal{R}, \mathcal{S}'$ by assumption. Since $|\mathcal{S}'| < |\mathcal{S}|$ we can apply the induction hypothesis to obtain the desired $\vDash_\exists \mathcal{R}, \mathcal{C}$.

Next we suppose that $\vDash_\exists \mathcal{R}, \mathcal{C}$ 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 $\vDash \mathcal{R}, \mathcal{S}$. We use induction on the size of $\mathcal{S}$. Since $\mathcal{S}$ is also a cycle, $(\gtrsim, >)_\pi \vDash_\exists \mathcal{R}, \mathcal{S}$ for some argument filtering π and reduction pair $(\gtrsim, >)$. Let $\bar{\mathcal{S}} = \{l \to r \in \mathcal{S} \mid \pi(l) \not> \pi(r)\}$. Since $\pi(\mathcal{S}) \cap > \neq \varnothing$, $\bar{\mathcal{S}}$ is a proper subset of $\mathcal{S}$. Hence every SCC $\mathcal{S}'$ in the subgraph of $\mathsf{DG}(\mathcal{R})$ induced by $\bar{\mathcal{S}}$ is smaller than $\mathcal{S}$, and thus $\vDash \mathcal{R}, \mathcal{S}'$ by the induction hypothesis. Consequently, $\vDash \mathcal{R}, \mathcal{S}$. □

The above proof provides quite a bit more information than the statement of Theorem 22 suggests. As a matter of fact, both conditions are equivalent to termination of $\mathcal{R}$, and also equivalent to the criterion "$\vDash_\forall \mathcal{R}, \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$" of Corollary 18. However, from the proof of Theorem 22 we learn that a termination proof based on "$\vDash \mathcal{R}, \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$" can

[3] In other words, we restrict $\mathsf{DG}(\mathcal{R})$ to the nodes in $\{l \to r \in \mathcal{S} \mid \pi(l) \not> \pi(r)\}$.

be directly transformed into a termination proof based on "$\models_\exists \mathcal{R}, \mathcal{C}$ 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 18.

Theorem 22 and the discussion following it easily generalize to the innermost case.

Definition 23 *Let $\mathcal{R}$ be a TRS and $\mathcal{S}$ a subset of the dependency pairs in $\mathsf{DP}(\mathcal{R})$. We write $\models_{\mathsf{i}} \mathcal{U}(\mathcal{S}), \mathcal{S}$ if there exist an argument filtering π and a reduction pair $(\gtrsim, >)$ such that $(\gtrsim, >)_\pi \models_\exists \mathcal{U}(\mathcal{S}), \mathcal{S}$ and $\models_{\mathsf{i}} \mathcal{U}(\mathcal{S}'), \mathcal{S}'$ for all SCCs $\mathcal{S}'$ of the subgraph of $\mathsf{IDG}(\mathcal{R})$ induced by the pairs $l \to r \in \mathcal{S}$ such that $\pi(l) \not> \pi(r)$.*

Theorem 24 *Let $\mathcal{R}$ be a TRS. The following conditions are equivalent:*

(1) $\models_{\mathsf{i}} \mathcal{U}(\mathcal{S}), \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{IDG}(\mathcal{R})$,
(2) $\models_\exists \mathcal{U}(\mathcal{C}), \mathcal{C}$ 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 18 (19) is bounded by n. We already remarked that the number of cycles and hence the number of groups generated by the cycle approach of Theorem 2 (5) is at most $2^n - 1$. Example 25 below shows that this upper bound cannot be improved. It is easy to see that the new approach of Theorem 22 (24) 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 Theorem 2 (5).

Example 25 *As an extreme example, consider the TRS $\mathcal{R}$ (Example 11 in [8]) consisting of the rules*

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

The only defined symbol, D, occurs 12 times in the right-hand sides of the rules, so there are 12 dependency pairs. All these dependency pairs have a right-hand side $\mathsf{D}^\sharp(t)$ with t a variable. It follows that the dependency graph is a complete graph. Consequently, there are $2^{12} - 1 = 4095$ cycles but just 1

SCC. Since $\mathcal{R}$ is compatible with LPO, all groups of ordering constraints are easily solved.

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

5 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 section we present two new ideas to reduce the number of computed argument filterings.

5.1 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 26 *Consider the following TRS (from [3]), encoding the quicksort* *algorithm:*

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

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 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 supports the situation where different recursive calls of the same function depend on a decrease of different arguments.

5.2 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 27 *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 28 *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 29 *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 30 *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 $\pi'' \in \mathsf{AF}'(t,\pi')$. $\square$

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 30.

Corollary 31 $\mathsf{AF}(t,\epsilon)$ *is the set of all minimal argument filterings* π *such that* $\pi(t)$ *can be completely evaluated.* $\square$

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

Definition 32 *Let* $\mathcal{T}$ *be a set of terms. We denote by* $\mathsf{AF}(\mathcal{T})$ *the set of all minimal argument filterings that completely evaluate each term in* $\mathcal{T}$*. In particular, we define* $\mathsf{AF}(\varnothing) = \{\epsilon\}$*.*

Definition 33 *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 34 *If* $\mathcal{T}_1, \mathcal{T}_2$ *are sets of terms then* $\mathsf{AF}(\mathcal{T}_1 \cup \mathcal{T}_2) = \mathsf{AF}(\mathcal{T}_1) \otimes \mathsf{AF}(\mathcal{T}_2)$.

PROOF. First we show that $\mathsf{AF}(\mathcal{T}_1 \cup \mathcal{T}_2) \subseteq \mathsf{AF}(\mathcal{T}_1) \otimes \mathsf{AF}(\mathcal{T}_2)$. Let $\pi \in \mathsf{AF}(\mathcal{T}_1 \cup \mathcal{T}_2)$. Let π_1 and π_2 be the minimum restrictions of π that completely evaluate every term in $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively. We have $\pi_1 \in \mathsf{AF}(\mathcal{T}_1)$ and $\pi_2 \in \mathsf{AF}(\mathcal{T}_2)$ by definition. Since π_1 and π_2 are compatible, $\pi_1 \cup \pi_2 \in \mathsf{AF}(\mathcal{T}_1) \otimes \mathsf{AF}(\mathcal{T}_2)$. Since $\pi_1 \cup \pi_2$ completely evaluates every term in $\mathcal{T}_1 \cup \mathcal{T}_2$, we obtain $\pi = \pi_1 \cup \pi_2$ from the minimality of π.

Next we show that $\mathsf{AF}(\mathcal{T}_1) \otimes \mathsf{AF}(\mathcal{T}_2) \subseteq \mathsf{AF}(\mathcal{T}_1 \cup \mathcal{T}_2)$. Let $\pi \in \mathsf{AF}(\mathcal{T}_1) \otimes \mathsf{AF}(\mathcal{T}_2)$. So there exist compatible $\pi_1 \in \mathsf{AF}(\mathcal{T}_1)$ and $\pi_2 \in \mathsf{AF}(\mathcal{T}_2)$ such that $\pi = \pi_1 \cup \pi_2$. Since π completely evaluates every term in $\mathcal{T}_1 \cup \mathcal{T}_2$, there must be a $\pi' \in \mathsf{AF}(\mathcal{T}_1 \cup \mathcal{T}_2)$ such that $\pi' \subseteq \pi$. Because π' completely evaluates every term in $\mathcal{T}_1$ and $\mathcal{T}_2$, 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 31 and Lemma 34 yields that $\mathsf{AF}(\mathcal{T})$ can be computed as $\bigotimes\{\mathsf{AF}(t, \epsilon) \mid t \in \mathcal{T}\}$.

Definition 28 (for $\pi = \epsilon$) is easily extended to rewrite rules.

Definition 35 *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 36 *Let $l \to r$ be a rewrite rule. We define $\mathsf{AF}_{\mathrm{lpo}}(l \to r) = \{\pi \in \mathsf{AF}(l \to r) \mid \pi(l) \succ_{\mathrm{lpo}}^{=} \pi(r)$ for some precedence $\succ\}$.*

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

Example 37 *Table 1 shows for each rule $l \to r$ the number of argument filterings in $\mathsf{AF}(\mathcal{F}\mathsf{un}(l \to r))$, $\mathsf{AF}(l \to r)$, $\mathsf{AF}_{\mathrm{vc}}(l \to r)$, 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 38 *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 $A_{\mathrm{lpo}(\mathcal{R})} = \{\pi \in A \mid \pi(\mathcal{R}) \subseteq \succ_{\mathrm{lpo}}^{=}$ for some precedence $\succ\}$.*

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

Table 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

single 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 26. Here we use the merge order corresponding to the numbering of the rewrite rules.

Example 39 *Table 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.*

Table 2
Divide and conquer: quicksort (II).

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

Table 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

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 28 by $\mathsf{AF}_h(\mathsf{outer}(t,\pi))$ where h is the heuristic. With respect to Example 39, the *some more* heuristic would produce 16 and the *some* heuristic just 9 suitable argument filterings. This can be inferred from Table 2.

An additional advantage of the divide and conquer approach is that the argument filterings for the rewrite rule constraints, which in the case of termination are part of every group of ordering constraints, need to be computed only once.

5.3 Dynamic Programming

The effectiveness of the divide and conquer approach depends very much on the merge order. Table 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 40 *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 $\mathrm{root}(\mathcal{R}_1) =$

root($\mathcal{R}_2$). In general, the size of $A_1 \otimes A_2$ increases with the size of 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 33) is not satisfied or if the orientability condition of the employed base order is not satisfied (cf. Definition 38). 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 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.

Suppose we want to compute $\mathsf{AF}(\mathcal{R})_{\text{lpo}(\mathcal{R})}$. 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 $\mathsf{AF}(\mathcal{R}')_{\text{lpo}(\mathcal{R}')}$. The table is initialized as follows:

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

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 root($\mathcal{R}_1 \cup \mathcal{R}_2$) is minimal and we add the following entry to the table:

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

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

Example 41 *For the fifteen rewrite rules of the TRS $\mathcal{R}$ of Example 26, after initializing the table, it turns out that $|\text{root}(\{14, 15\})| = |\{\leq, \mathsf{false}\}| = 2$ is minimal, so we add*

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

to the table. Next the pair of $\{2\}$ and $\{10\}$ is selected. Continuing in this fashion, the data in Table 4 is computed (left to right, top to bottom).

When using the condition "$\models \mathcal{R}, \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$" of Theorem 22 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 $\mathrm{A}(\mathcal{R} \cup \mathcal{S}) = (\mathrm{A}(\mathcal{R}) \otimes \mathrm{A}(\mathcal{S}))_{\text{lpo}(\mathcal{R} \cup \mathcal{S})}$. 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.

In the case of innermost termination, different SCCs (may) have different usable rules and some rewrite rules may not be usable at all. So it does not

Table 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

make sense to compute $\mathrm{A}(\mathcal{R})$. Rather, we compute $\mathrm{A}(\mathcal{R}')$ for suitable subsets of $\mathcal{R}$ on demand. This is illustrated in the following example.

Example 42 *Consider the TRS $\mathcal{R}$ of Example 6. The* EIDG *or* EIDG^* *approximated innermost dependency graph contains two SCCs: $\mathcal{S}_1 = \{6\}$, $\mathcal{S}_2 = \{8,9\}$.*

- *For SCC $\mathcal{S}_1$, we have to prove "$\vDash_{\mathrm{i}} \mathcal{U}(\mathcal{S}_1), \mathcal{S}_1$". Since $\mathcal{U}(\mathcal{S}_1) = \varnothing$ and $\mathcal{S}_1$ has just one element, initializing the table* A *will produce the answer:*

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

  Since $\mathrm{A}(\{6\})$ *contains a suitable argument filtering, we are done.*
- *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:*

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

  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 mix-

ture 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* $\mathrm{A}(\{8,9\})$*:*

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

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 (cf. the last item in Example 6). By 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* $\vDash_{\mathsf{i}} \mathcal{U}(\mathcal{S}_3), \mathcal{S}_3$ *and thus also* $\vDash_{\mathsf{i}} \mathcal{U}(\mathcal{S}_2), \mathcal{S}_2$.

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 [12] 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. 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.

6 Experiments

Our ideas have been implemented in the termination prover TT, which is described in [14] and available at

`http://colo2-c703.uibk.ac.at/`

We tested examples from three different sources:

- all 109 examples (66 in Section 3 and 43 in Section 4) from Arts and Giesl [3],
- all 23 examples from Dershowitz [8],
- all 122 examples from Steinbach and Kühler [23, Sections 3 and 4].

Seven of these examples appear in more than one collection, so the total number is 247.

Of these 247 examples, 241 are innermost terminating (Examples 2.5, 4.6, 4.34, 4.40, 4.49, and 4.54 from [23] are not) and 221 are terminating. All experiments were performed on a PC equipped with a 2.20 GHz Mobile Intel Pentium 4 Processor - M and 512 MB of memory.

6.1 Dependency Graph

Our first experiment concerns the new estimations of the (innermost) dependency graph mentioned in Section 3. Table 5 lists the 13 examples where the estimations differ. Only for Example 4.50 in [23] (which happens to be the rule of Toyama that we encountered in Example 10) does the estimation influence the ability to prove termination automatically, although termination is proved faster with the EDG^* approximation—the overhead of using EDG^* instead of EDG is negligible. This can be seen from Table 6, where we show the effect of both estimations in combination with the new algorithm for cycle analysis. In these and all subsequent experiments, LPO with strict precedence is used as base order. (The ideas described in Section 5 were not used for Table 6.) The numbers denote execution time in seconds. *Italics* indicate that termination could not be proved within the given time, while fully exploring the search space implied by the options.

6.2 Cycle Analysis

Tables 7, 8, and 9 show the effect of the three approaches to cycle analysis in combination with the heuristics for reducing the number of argument fil-

Table 5
Dependency graph estimation (I).

		EDG | EDG*						EIDG | EIDG*					
TRS	DPs	arrows		SCCs		cycles		arrows		SCCs		cycles	
[3]:3.23	2	4	2	1	1	3	1	4	2	1	1	3	1
[3]:3.44	4	4	0	2	0	2	0	4	0	2	0	2	0
[3]:3.45	4	5	3	3	2	3	2	5	3	3	2	3	2
[3]:4.20a	3	3	1	2	1	2	1	2	0	1	0	1	0
[3]:4.20b	4	7	5	2	1	4	3	5	3	2	1	2	1
[3]:4.21	6	12	8	2	2	6	4	6	2	2	0	2	0
[3]:4.37b	4	6	3	3	2	3	2	2	2	2	2	2	2
[23]:2.8	8	24	24	3	3	7	7	19	18	3	3	3	3
[23]:2.51	3	8	7	1	1	6	5	8	7	1	1	6	5
[23]:2.52	9	36	35	4	4	17	16	36	35	4	4	17	16
[23]:4.31	3	4	4	2	2	2	2	4	2	2	1	2	1
[23]:4.50	1	1	0	1	0	1	0	0	0	0	0	0	0
[23]:4.59	6	12	4	3	2	5	2	12	4	3	2	5	2

terings. In all experiments we used $\mathsf{E(I)DG}^*$ to approximate the (innermost) dependency graph and enumeration to search for suitable argument filterings.

From the data in Tables 7, 8, and 9 one might get the impression that the advantage of the new recursive SCC method for cycle analysis is not that significant. This is simply due to the fact that the (innermost) dependency graphs of most of the TRSs in [3,8,23] contain relatively few cycles. We refer to the appendix for a larger example where the use of the recursive SCC algorithm is crucial for obtaining a termination proof within a reasonable amount of time.

6.3 Divide and Conquer

Tables 10 and 11 compare enumeration (E) with the divide and conquer approach to find appropriate argument filterings, where we consider both the naive (linear) method described in Section 5.2 (DC) as well as the more involved dynamic programming method described in Section 5.3 (DP), in combination with the heuristics of Section 5.1. We used a timeout of 60 seconds.

Table 6
Dependency graph estimation (II).

	termination		innermost termination	
TRS	EDG	EDG*	EIDG	EIDG*
[3]:3.23	0.00	0.01	0.01	0.01
[3]:3.44	0.01	0.00	0.00	0.00
[3]:3.45	0.02	0.01	0.00	0.01
[3]:4.20a	0.00	0.00	0.02	0.01
[3]:4.20b	0.01	0.01	0.00	0.00
[3]:4.21	0.01	0.01	0.01	0.01
[3]:4.37b	0.00	0.00	0.02	0.00
[23]:2.8	0.02	0.01	0.02	0.02
[23]:2.51	0.00	0.00	0.01	0.02
[23]:2.52	0.04	0.02	0.05	0.08
[23]:4.31	0.09	0.09	0.01	0.01
[23]:4.50	*0.01*	0.01	0.01	0.01
[23]:4.59	2.20	1.64	0.09	0.01
Example 4	0.01	0.01	0.01	0.01
Example 6	*0.02*	*0.02*	0.01	0.01
Example 25	6.97	6.92	1.57	1.64
Example 26	3683.49	3683.59	1.92	1.91
total time	3692.90	3692.35	3.76	3.76

Comparing the E and DP columns, the effectiveness of the divide and conquer approach of Section 5.3 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 that can be drawn from the two tables is that the *some more* heuristic is surprisingly powerful. Moreover, if termination cannot be proved within 1 second then it is unlikely that a termination proof (with respect to the same parameters) will be produced at

Table 7
Cycle analysis: *some* argument filterings.

	termination			innermost termination		
	cycle	SCC	new	cycle	SCC	new
success	129	117	129	168	152	168
average time	0.27	0.01	0.10	0.03	0.02	0.01
failure	118	130	118	79	95	79
average time	0.18	0.61	0.20	0.12	0.05	0.09
timeout	0	0	0	0	0	0
total time	56.20	80.08	35.55	13.63	6.66	9.33

Table 8
Cycle analysis: *some more* argument filterings.

	termination			innermost termination		
	cycle	SCC	new	cycle	SCC	new
success	138	125	138	173	157	173
average time	0.53	0.01	0.20	0.03	0.01	0.02
failure	107	118	107	73	90	73
average time	0.99	0.60	1.09	0.22	0.11	0.25
timeout	2	4	2	1	0	1
total time	299.29	311.85	264.02	80.78	12.32	81.55

all.

In the final table of this paper we present the individual timings for those examples in the collection of Arts and Giesl [3] for which the computation of at least one of the E and DP data in Table 10 exceeds 5 seconds, together with [8, Example 11] (the differentiation TRS of Example 25) and [23, Examples 2.29 and 4.29]. The reason for including the latter is that these are the only examples (of the 247 tested) where enumeration outperforms the divide and conquer approach. This is because the TRSs that can be proved terminating by LPO (and so without using dependency pairs) and the number of argument filterings for which the dependency pair constraints can be solved is staggering.

Table 9
Cycle analysis: *all* argument filterings.

	termination			innermost termination		
	cycle	SCC	new	cycle	SCC	new
success	137	135	138	175	170	176
average time	0.91	0.85	0.85	0.06	0.05	0.05
failure	102	105	102	68	74	68
average time	0.79	0.73	0.77	0.10	0.13	0.13
timeout	8	7	7	4	3	3
total time	685.26	611.42	615.70	256.80	199.06	198.14

Table 10
Divide and conquer: termination.

	some			some more			all		
	E	DC	DP	E	DC	DP	E	DC	DP
success	129	128	129	138	138	138	138	128	137
average time	0.10	0.05	0.07	0.20	0.81	0.22	0.85	0.88	0.33
success in 1s	127	127	128	137	126	136	127	119	130
failure	118	118	118	107	107	108	102	103	106
average time	0.20	0.11	0.01	1.09	0.32	0.03	0.77	0.76	0.39
timeout	0	1	0	2	2	1	7	16	4
total time	35.55	78.43	10.52	264.02	265.92	93.26	615.70	1257.75	326.22

7 Conclusion

We conclude by stating that the techniques presented in this paper are very useful for obtaining termination proofs of TRSs efficiently and automatically. Nevertheless, as can be inferred from Tables 10 and 11, there are numerous (innermost) terminating TRSs that cannot be handled by using only our techniques. Taking a different base order (LPO with quasi-precedence, polynomial interpretations, Knuth-Bendix order) will cover some of these. Dependency pair transformations [9,12] like narrowing and instantiation or taking polynomial interpretations with negative coefficients [16] (which do not produce

Table 11
Divide and conquer: innermost termination.

	some			some more			all		
	E	DC	DP	E	DC	DP	E	DC	DP
success	168	168	168	173	173	173	176	172	177
average time	0.01	0.02	0.02	0.02	0.04	0.02	0.05	0.41	0.10
success in 1s	168	168	168	173	172	173	173	165	175
failure	79	79	79	73	73	73	74	67	66
average time	0.09	0.08	0.04	0.25	0.44	0.05	0.13	1.87	0.34
timeout	0	0	0	1	1	0	3	8	0
total time	9.33	9.12	5.55	81.55	99.14	6.38	198.14	732.36	41.91

quasi-simplification orders) as base order cover many more. Furthermore, recent modular refinements [15,24,26] have a very positive effect on the termination proving power as well as the efficiency of the dependency pair method. Since termination is an undecidable property, it goes without saying that there always remain terminating TRSs that are beyond the scope of any automatable method for proving termination. As a particular challenge we mention Example 33 in [8] which encodes the battle of Hydra and Hercules.

Acknowledgements

The suggestions by the referees helped to improve the paper.

References

[1] 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.

[2] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.

[3] 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.

[4] F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

Table 12
Divide and conquer: individual TRSs.

	some		some more		all	
TRS	E	DP	E	DP	E	DP
[3]:3.5b	0.09	0.02	0.44	0.05	8.11	0.67
[3]:3.5c	0.13	0.03	0.92	0.11	59.04	2.59
[3]:3.6c	*0.08*	*0.01*	*0.32*	*0.08*	*7.89*	*1.92*
[3]:3.8c	0.07	0.02	0.44	0.55	9.99	7.58
[3]:3.10	*11.86*	*0.04*	*326.81*	*0.10*	*43537.44*	*0.80*
[3]:3.11	2.31	0.08	23.32	0.12	3683.59	0.87
[3]:3.13	*6.07*	*0.12*	*44.69*	*0.19*	*24292.03*	*451.25*
[3]:3.53a	*1.22*	*0.02*	*32.38*	*0.16*	*893.74*	*5.11*
[3]:3.55	8.56	0.17	120.91	0.63	54020.70	7.97
[3]:3.57	*0.39*	*0.04*	*9.73*	*0.41*	*39.30*	*4.12*
[3]:4.30c	*0.07*	*0.01*	*0.34*	*0.03*	*7.36*	*0.14*
[3]:4.35	*0.62*	*0.12*	*6.75*	*0.30*	*383.37*	*4.20*
[3]:4.36	*1.49*	*0.05*	*19.45*	*0.57*	*1161.64*	*21.02*
[8]:11	0.01	0.30	0.01	19.77	7.04	134.92
[23]:2.29	0.01	0.42	0.02	3.94	4.38	61.14
[23]:4.28	0.17	6.13	0.16	80.23	0.17	1080.96

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

[6] 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.

[7] E. Contejean, C. Marché, B. Monate, and X. Urbain. C*i*ME version 2, 2000. Available at `http://cime.lri.fr/`.

[8] 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.

[9] 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.

[10] J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.

[11] J. Giesl and A. Middeldorp. Transformation techniques for context-sensitive rewrite systems. *Journal of Functional Programming*, 14:329–427, 2004.

[12] 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.

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

[14] 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.

[15] 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.

[16] N. Hirokawa and A. Middeldorp. Polynomial interpretations with negative coefficients. In *Proceedings of the 7th International Conference on Artificial Intelligence and Symbolic Computation*, Lecture Notes in Artificial Intelligence, 2004. To appear.

[17] S. Kamin and J.J. Lévy. Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois, 1980.

[18] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):294–343, 2002.

[19] 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.

[20] 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.

[21] E. Ohlebusch. Hierarchical termination revisited. *Information Processing Letters*, 84(4):207–214, 2002.

[22] J. Steinbach. Automatic termination proofs with transformation orderings. In *Proceedings of the 6th International Conference on Rewriting Techniques and Applications*, volume 914 of *Lecture Notes in Computer Science*, pages 11–25, 1995.

[23] J. Steinbach and U. Kühler. Check your ordering – termination proofs and open problems. Technical Report SR-90-25, Universität Kaiserslautern, 1990.

[24] 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.

[25] Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.

[26] X. Urbain. Modular & incremental automated termination proofs. *Journal of Automated Reasoning*, 2004. To appear.

A A Larger Example

We conclude this paper with a relatively small TRS that contains very many cycles. The TRS is obtained by applying the first transformation of Giesl and Middeldorp [11, Definition 12] to a context-sensitive rewrite system that approximates the infinite sequence $\frac{1}{1}$, $\frac{1}{4}$, $\frac{1}{9}$, …, $\frac{1}{n^2}$ whose partial sums converge to $\frac{\pi^2}{6}$ (Lucas [18, Example 2]). In the termination proof we use LPO with quasi-precedence and linear polynomial interpretations with coefficients in $\{0, 1\}$ as base orders. With respect to the former, we use the *some* heuristic described in Section 5.1 and the divide and conquer technique with dynamic programming described in Section 5.3.

Example 43 *Consider the following TRS $\mathcal{R}$:*

$$\begin{array}{rrcl}
1: & \mathsf{terms_a}(x) & \to & \mathsf{recip}(\mathsf{sqr_a}(\mathsf{m}(x))) : \mathsf{terms}(\mathsf{s}(x)) \\
2: & \mathsf{sqr_a}(\mathsf{0}) & \to & \mathsf{0} \\
3: & \mathsf{sqr_a}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{sqr_a}(\mathsf{m}(x)) +_\mathsf{a} \mathsf{dbl_a}(\mathsf{m}(x))) \\
4: & \mathsf{dbl_a}(\mathsf{0}) & \to & \mathsf{0} \\
5: & \mathsf{dbl_a}(\mathsf{s}(x)) & \to & \mathsf{s}(\mathsf{s}(\mathsf{dbl_a}(\mathsf{m}(x)))) \\
6: & \mathsf{0} +_\mathsf{a} y & \to & \mathsf{m}(y) \\
7: & \mathsf{s}(x) +_\mathsf{a} y & \to & \mathsf{s}(\mathsf{m}(x) +_\mathsf{a} \mathsf{m}(y)) \\
8: & \mathsf{first_a}(\mathsf{0}, z) & \to & \mathsf{nil} \\
9: & \mathsf{first_a}(\mathsf{s}(x), y : z) & \to & \mathsf{m}(y) : \mathsf{first}(x, z) \\
10: & \mathsf{half_a}(\mathsf{0}) & \to & \mathsf{0} \\
11: & \mathsf{half_a}(\mathsf{s}(\mathsf{0})) & \to & \mathsf{0} \\
12: & \mathsf{half_a}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{s}(\mathsf{half_a}(\mathsf{m}(x))) \\
13: & \mathsf{half_a}(\mathsf{dbl}(x)) & \to & \mathsf{m}(x)
\end{array}$$

$$\begin{array}{rrcl@{\qquad}rrcl}
14: & \mathsf{m}(\mathsf{terms}(x)) & \to & \mathsf{terms_a}(\mathsf{m}(x)) & 25: & \mathsf{terms_a}(x) & \to & \mathsf{terms}(x) \\
15: & \mathsf{m}(\mathsf{sqr}(x)) & \to & \mathsf{sqr_a}(\mathsf{m}(x)) & 26: & \mathsf{sqr_a}(x) & \to & \mathsf{sqr}(x) \\
16: & \mathsf{m}(x + y) & \to & \mathsf{m}(x) +_\mathsf{a} \mathsf{m}(y) & 27: & x +_\mathsf{a} y & \to & x + y \\
17: & \mathsf{m}(\mathsf{dbl}(x)) & \to & \mathsf{dbl_a}(\mathsf{m}(x)) & 28: & \mathsf{dbl_a}(x) & \to & \mathsf{dbl}(x)
\end{array}$$

18:	$\mathsf{m}(\mathsf{first}(x,y)) \to \mathsf{first_a}(\mathsf{m}(x),\mathsf{m}(y))$	29:	$\mathsf{first_a}(x,y) \to \mathsf{first}(x,y)$
19:	$\mathsf{m}(\mathsf{half}(x)) \to \mathsf{half_a}(\mathsf{m}(x))$	30:	$\mathsf{half_a}(x) \to \mathsf{half}(x)$
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:	$\mathsf{m}(\mathsf{0}) \to \mathsf{0}$		
24:	$\mathsf{m}(\mathsf{nil}) \to \mathsf{nil}$		

There are 33 dependency pairs:

31: $\mathsf{TERMS_a}(x) \to \mathsf{SQR_a}(\mathsf{m}(x))$
32: $\mathsf{TERMS_a}(x) \to \mathsf{M}(x)$
33: $\mathsf{SQR_a}(\mathsf{s}(x)) \to \mathsf{sqr_a}(\mathsf{m}(x)) +^{\sharp}_{\mathsf{a}} \mathsf{dbl_a}(\mathsf{m}(x))$
34: $\mathsf{SQR_a}(\mathsf{s}(x)) \to \mathsf{SQR_a}(\mathsf{m}(x))$
35: $\mathsf{SQR_a}(\mathsf{s}(x)) \to \mathsf{M}(x)$
36: $\mathsf{SQR_a}(\mathsf{s}(x)) \to \mathsf{DBL_a}(\mathsf{m}(x))$
37: $\mathsf{DBL_a}(\mathsf{s}(x)) \to \mathsf{DBL_a}(\mathsf{m}(x))$
38: $\mathsf{DBL_a}(\mathsf{s}(x)) \to \mathsf{M}(x)$
39: $\mathsf{0} +^{\sharp}_{\mathsf{a}} y \to \mathsf{M}(y)$
40: $\mathsf{s}(x) +^{\sharp}_{\mathsf{a}} y \to \mathsf{m}(x) +^{\sharp} \mathsf{m}(y)$
41: $\mathsf{s}(x) +^{\sharp}_{\mathsf{a}} y \to \mathsf{M}(x)$
42: $\mathsf{s}(x) +^{\sharp}_{\mathsf{a}} y \to \mathsf{M}(y)$
43: $\mathsf{FIRST_a}(\mathsf{s}(x), y : z) \to \mathsf{M}(y)$
44: $\mathsf{HALF_a}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{HALF_a}(\mathsf{m}(x))$
45: $\mathsf{HALF_a}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{M}(x)$
46: $\mathsf{HALF_a}(\mathsf{dbl}(x)) \to \mathsf{M}(x)$
47: $\mathsf{M}(\mathsf{terms}(x)) \to \mathsf{TERMS_a}(\mathsf{m}(x))$
48: $\mathsf{M}(\mathsf{terms}(x)) \to \mathsf{M}(x)$
49: $\mathsf{M}(\mathsf{sqr}(x)) \to \mathsf{SQR_a}(\mathsf{m}(x))$
50: $\mathsf{M}(\mathsf{sqr}(x)) \to \mathsf{M}(x)$
51: $\mathsf{M}(x + y) \to \mathsf{m}(x) +^{\sharp}_{\mathsf{a}} \mathsf{m}(y)$
52: $\mathsf{M}(x + y) \to \mathsf{M}(x)$
53: $\mathsf{M}(x + y) \to \mathsf{M}(y)$
54: $\mathsf{M}(\mathsf{dbl}(x)) \to \mathsf{DBL_a}(\mathsf{m}(x))$
55: $\mathsf{M}(\mathsf{dbl}(x)) \to \mathsf{M}(x)$
56: $\mathsf{M}(\mathsf{first}(x,y)) \to \mathsf{FIRST_a}(\mathsf{m}(x),\mathsf{m}(y))$
57: $\mathsf{M}(\mathsf{first}(x,y)) \to \mathsf{M}(x)$
58: $\mathsf{M}(\mathsf{first}(x,y)) \to \mathsf{M}(y)$
59: $\mathsf{M}(\mathsf{half}(x)) \to \mathsf{HALF_a}(\mathsf{m}(x))$
60: $\mathsf{M}(\mathsf{half}(x)) \to \mathsf{M}(x)$

$$\begin{aligned} 61: \quad & \mathsf{M}(x : y) \to \mathsf{M}(x) \\ 62: \quad & \mathsf{M}(\mathsf{recip}(x)) \to \mathsf{M}(x) \\ 63: \quad & \mathsf{M}(\mathsf{s}(x)) \to \mathsf{M}(x) \end{aligned}$$

The dependency graph contains a single SCC that consists of all dependency pairs. By taking the argument filtering π *with* $\pi(\mathsf{DBL_a}) = \pi(\mathsf{HALF_a}) = \pi(\mathsf{M}) = \pi(\mathsf{half_a}) = \pi(:) = \pi(\mathsf{half}) = \pi(\mathsf{m}) = \pi(\mathsf{recip}) = 1$ *and* $\pi(\mathsf{FIRST_a}) = 2$ *together with LPO with quasi-precedence* $\mathsf{0} \succ \mathsf{nil}$, $\mathsf{terms_a} \approx \mathsf{terms} \succ \mathsf{TERMS_a} \succ \mathsf{SQR_a} \approx \mathsf{sqr_a} \approx \mathsf{sqr} \succ + \approx +_{\mathsf{a}} \succ +^{\sharp}_{\mathsf{a}}$, $\mathsf{sqr} \succ \mathsf{dbl} \approx \mathsf{dbl_a} \succ \mathsf{s}$, $+ \succ \mathsf{s}$, *and* $\mathsf{first_a} \approx \mathsf{first}$, *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 argument filtering* π *with* $\pi(\mathsf{m}) = \pi(\mathsf{dbl_a}) = \pi(\mathsf{sqr_a}) = \pi(\mathsf{terms_a}) = \pi(:) = \pi(\mathsf{dbl}) = \pi(\mathsf{M}) = \pi(\mathsf{recip}) = \pi(\mathsf{s}) = \pi(\mathsf{sqr}) = \pi(\mathsf{terms}) = 1$ *and* $\pi(+_{\mathsf{a}}) = \pi(+) = 2$ *together with LPO with precedence* $\mathsf{0} \succ \mathsf{nil}$, $\mathsf{first_a} \approx \mathsf{first}$*, and* $\mathsf{half_a} \approx \mathsf{half}$*, all rewrite rules are (weakly) decreasing, dependency pair* 60 *is strictly decreasing, and the remaining dependency pairs* 61 *and* 62 *are weakly decreasing.*
  - *There is one new SCC:* $\{61, 62\}$*. By taking the argument filtering* π *with* $\pi(\mathsf{m}) = \pi(\mathsf{dbl_a}) = \pi(\mathsf{half_a}) = \pi(\mathsf{sqr_a}) = \pi(:) = \pi(\mathsf{dbl}) = \pi(\mathsf{half}) = \pi(\mathsf{M}) = \pi(\mathsf{s}) = \pi(\mathsf{sqr}) = 1$ *and* $\pi(+^{\sharp}_{\mathsf{a}}) = \pi(+) = 2$ *together with LPO with precedence* $\mathsf{0} \succ \mathsf{nil}$, $\mathsf{terms} \succ \mathsf{recip}$, $\mathsf{first_a} \approx \mathsf{first}$*, and* $\mathsf{terms_a} \approx \mathsf{terms}$*, all rewrite rules are (weakly) decreasing, dependency pair* 62 *is strictly decreasing, and dependency pairs* 61 *is weakly decreasing.*

    There is one new SCC: $\{61\}$*. By taking the polynomial interpretation* $[\mathsf{0}] = [\mathsf{sqr_a}](x) = [\mathsf{nil}] = [\mathsf{s}](x) = [\mathsf{sqr}](x) = 0$*,* $[\mathsf{terms_a}](x) = [\mathsf{terms}](x) = 1$*,* $[\mathsf{dbl_a}](x) = [\mathsf{half_a}](x) = [\mathsf{dbl}](x) = [\mathsf{half}](x) = [\mathsf{M}](x) = [\mathsf{recip}](x) = x$*,* $[\mathsf{m}](x) = [:](x, y) = x+1$*, and* $[+_{\mathsf{a}}](x, y) = [\mathsf{first_a}](x, y) = [+](x, y) = [\mathsf{first}](x, y) = y$*, all rewrite rules are weakly decreasing and dependency pair* 61 *is strictly decreasing. Hence* $\vDash \mathcal{R}, \{61\}$*.*

    We obtain $\vDash \mathcal{R}, \{61, 62\}$*.*

  We obtain $\vDash \mathcal{R}, \{60, 61, 62\}$*.*

Finally, $\vDash \mathcal{R}, \mathsf{DP}(\mathcal{R})$*. Hence the termination of* $\mathcal{R}$ *is proved. The proof took 133.10 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.*