Automating the Dependency Pair Method

Nao Hirokawa[1] and Aart Middeldorp[2⋆]

[1] Graduate School of Systems and Information Engineering
University of Tsukuba, Tsukuba 305-8573, Japan
nao@score.is.tsukuba.ac.jp

[2] Institute of Information Sciences and Electronics
University of Tsukuba, Tsukuba 305-8573, Japan
ami@is.tsukuba.ac.jp

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.

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 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 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 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.

⋆ Partially supported by the Grant-in-Aid for Scientific Research (C)(2) 13224006 of the Ministry of Education, Culture, Sports, Science and Technology of Japan.

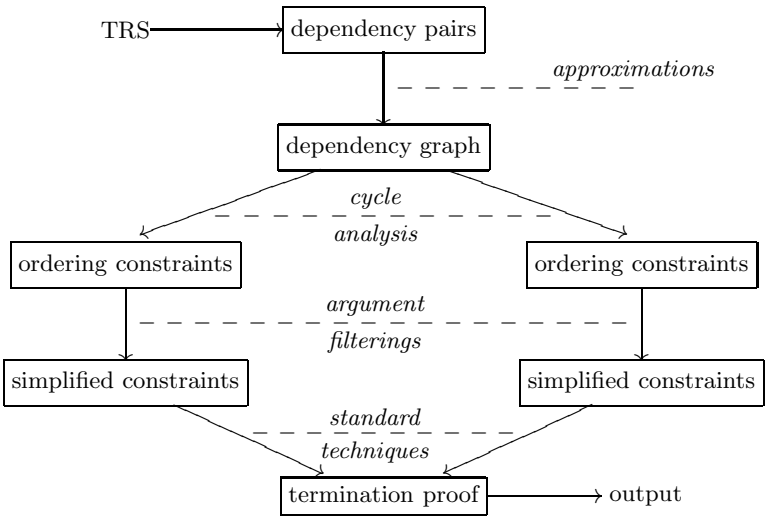


Fig. 1. The dependency pair method.

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 make some comments on dependency graph approximations. 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 [15] 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 [12] 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. We refer to [2, 9, 10] for motivations and additional refinements.[3] Let $\mathcal{R}$ be a term rewrite system (TRS for short) over a signature $\mathcal{F}$. 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

[3] The refinements (like narrowing and instantiation) transform dependency pairs with the aim of simplifying the resulting ordering constraints; they are orthogonal to the ideas we develop in this paper.

$t = f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with f defined, we write $t^\sharp$ for the term $f^\sharp(t_1, \ldots, t_n)$. 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 $\pi(f) = [i_1, \ldots, i_m]$. 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 (i.e., a transitive and reflexive relation on terms which is closed under contexts and substitutions) $\gtrsim$ 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 $(\succ^{=}_{\mathrm{lpo}}, \succ_{\mathrm{lpo}})$ for some strict precedence $\succ$. Here $\succ_{\mathrm{lpo}}$ denotes the lexicographic path order (LPO) induced by $\succ$.

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 exist substitutions σ and τ such that $t\sigma \to^*_{\mathcal{R}} u\tau$. 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 $\pi(\mathcal{R} \cup \mathcal{C}) \subseteq \gtrsim \cup >$ 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 $\mathsf{DP}(\mathcal{R})$. We write $\vDash_\exists \mathcal{R}, \mathcal{C}$ if there exist an argument filtering π and a reduction pair $(\gtrsim, >)$ such that*

$\pi(\mathcal{R} \cup \mathcal{C}) \subseteq \gtrsim \cup >$ *and* $\pi(\mathcal{C}) \cap > \neq \varnothing$. *We write* $(\gtrsim, >)_\pi \models_\exists \mathcal{R}, \mathcal{C}$ *if we want to indicate a combination of argument filtering and reduction pair that makes* $\models_\exists \mathcal{R}, \mathcal{C}$ *true.*

The existential quantifier in the notation indicates that *some* pair in $\mathcal{C}$ should be strictly decreasing. Theorem 2 can now be simply stated as "A TRS $\mathcal{R}$ is terminating if and only if $\models_\exists \mathcal{R}, \mathcal{C}$ for every cycle $\mathcal{C}$ in $\mathsf{DG}(\mathcal{R})$."

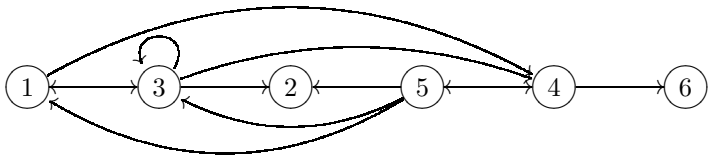
Example 4. Consider the following TRS (from [6]):

$$\begin{aligned}
\mathsf{ackin}(\mathsf{0}, x) &\to \mathsf{ackout}(\mathsf{s}(x)) & \mathsf{u}_{11}(\mathsf{ackout}(x)) &\to \mathsf{ackout}(x)\\
\mathsf{ackin}(\mathsf{s}(x), \mathsf{0}) &\to \mathsf{u}_{11}(\mathsf{ackin}(x, \mathsf{s}(\mathsf{0}))) & \mathsf{u}_{21}(\mathsf{ackout}(x), y) &\to \mathsf{u}_{22}(\mathsf{ackin}(y, x))\\
\mathsf{ackin}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{u}_{21}(\mathsf{ackin}(\mathsf{s}(x), y), x) & \mathsf{u}_{22}(\mathsf{ackout}(x)) &\to \mathsf{ackout}(x)
\end{aligned}$$

There are six dependency pairs:

$$\begin{aligned}
1: \quad \mathsf{ACKIN}(\mathsf{s}(x), \mathsf{0}) &\to \mathsf{ACKIN}(x, \mathsf{s}(\mathsf{0}))\\
2: \quad \mathsf{ACKIN}(\mathsf{s}(x), \mathsf{0}) &\to \mathsf{U}_{11}(\mathsf{ackin}(x, \mathsf{s}(\mathsf{0})))\\
3: \quad \mathsf{ACKIN}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{ACKIN}(\mathsf{s}(x), y)\\
4: \quad \mathsf{ACKIN}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{U}_{21}(\mathsf{ackin}(\mathsf{s}(x), y), x)\\
5: \quad \mathsf{U}_{21}(\mathsf{ackout}(x), y) &\to \mathsf{ACKIN}(y, x)\\
6: \quad \mathsf{U}_{21}(\mathsf{ackout}(x), y) &\to \mathsf{U}_{22}(\mathsf{ackin}(y, x))
\end{aligned}$$

The dependency graph



contains six cycles: $\{1,3,4,5\}$, $\{1,4,5\}$, $\{3,4,5\}$, $\{4,5\}$, $\{1,3\}$, and $\{3\}$. The constraints generated by Theorem 2 can be solved as follows.

- For cycles $\{1,3,4,5\}$, $\{1,4,5\}$, $\{3,4,5\}$, and $\{4,5\}$ we take the argument filtering π with $\pi(\mathsf{ACKIN}) = \pi(\mathsf{ackin}) = \pi(\mathsf{u}_{11}) = \pi(\mathsf{u}_{22}) = 1$, $\pi(\mathsf{U}_{21}) = [2]$, $\pi(\mathsf{ackout}) = [\,]$, $\pi(\mathsf{u}_{21}) = 2$ and LPO with precedence $0 \succ \mathsf{ackout}$ and $\mathsf{s} \succ \mathsf{U}_{21}$.
- For cycle $\{1,3\}$ we take the argument filtering π with $\pi(\mathsf{ACKIN}) = \pi(\mathsf{ackin}) = \pi(\mathsf{u}_{11}) = \pi(\mathsf{u}_{22}) = 1$, $\pi(\mathsf{ackout}) = [\,]$, $\pi(\mathsf{u}_{21}) = 2$ and LPO with precedence $0 \succ \mathsf{ackout}$.
- For cycle $\{3\}$ we take the argument filtering π with $\pi(\mathsf{ackin}) = \pi(\mathsf{u}_{11}) = \pi(\mathsf{u}_{22}) = 1$, $\pi(\mathsf{ACKIN}) = \pi(\mathsf{u}_{21}) = 2$, $\pi(\mathsf{ackout}) = [\,]$ and LPO with precedence $0 \succ \mathsf{ackout}$.

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 $t\sigma \to_{\mathcal{R}}^* u\tau$, the dependency graph cannot be computed in general. Hence, in order to mechanize the termination criterion of Theorem 2 one has to approximate the dependency graph. Arts and Giesl [2] proposed a simple approximation based on syntactic unification for this purpose.

Definition 5. *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.*

Middeldorp [13] showed that better approximations of the dependency graph are obtained by adopting tree automata techniques. These techniques are however computationally expensive. In a very recent paper Middeldorp [14] showed that the approximation of Arts and Giesl can be improved by symmetry considerations without incurring the overhead of tree automata techniques.

Definition 6. *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 $\mathcal{D}^{-1}$ by distinct fresh variables is denoted by $\mathsf{CAP}^{-1}(t)$. Here $\mathcal{D}^{-1} = \{\mathrm{root}(r) \mid l \to r \in \mathcal{R}\}$ if $\mathcal{R}$ is non-collapsing and $\mathcal{D}^{-1} = \mathcal{F}$ 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 $\mathsf{REN}(\mathsf{CAP}^{-1}(u))$ are unifiable.*

A comparison between the new estimation and the tree automata based approximations described in [13] can be found in [14]. 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 7. 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 [17] 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.

We refer to Section 6 for some statistics related to the two estimations.

4 Cycle Analysis

The use of Theorem 2 for ensuring termination requires that *all* cycles have to be considered (see [9] for concrete examples). 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 existing 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.

Corollary 8. *A TRS $\mathcal{R}$ is terminating if for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$ there exist an argument filtering π and a reduction pair $(\gtrsim, >)$ such that $\pi(\mathcal{R}) \subseteq \gtrsim \cup >$ and $\pi(\mathcal{S}) \subseteq >$.* □

We find it convenient to abbreviate the two conditions in Corollary 8 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.

The difference with Theorem 2 is that all pairs in an SCC must be strictly decreasing. This, however, makes the termination criterion of Corollary 8 strictly weaker than the one of Theorem 2, if we employ traditional (quasi-)simplification as reduction pairs. If we allow arbitrary reduction pairs then the termination criteria of Corollary 8 and Theorem 2 become equivalent, in other words, the reverse of Corollary 8 also holds. This, however, is only of theoretical interest.

Example 9. 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: $\{1, 3, 4, 5\}$. The constraints generated by Corollary 8 cannot be solved automatically.

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 $\pi(\mathcal{R} \cup \mathcal{S}) \subseteq \gtrsim \cup >$ and $\pi(\mathcal{S}) \cap > \neq \varnothing$. Then we compute the SCCs of the subgraph 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 and the efficiency of Corollary 8. The former is proved below and the latter is confirmed by extensive experiments (see Section 6) and explained in the paragraph following Example 12.

Definition 10. *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 11. *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 exists an argument filtering π and 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 11 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 8. However, from the proof of Theorem 11 we learn that a termination proof based on "$\vDash \mathcal{R}, \mathcal{S}$ for every SCC $\mathcal{S}$ in $\mathsf{DG}(\mathcal{R})$" can be directly transformed into a termination proof based on "$\vDash \mathcal{R}, \mathcal{C}$ 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 8.

Example 12. Consider the TRS of Example 4. If we take the argument filtering π with $\pi(\mathsf{ACKIN}) = \pi(\mathsf{ackin}) = \pi(\mathsf{u}_{11}) = \pi(\mathsf{u}_{22}) = 1$, $\pi(\mathsf{U}_{21}) = [2]$, $\pi(\mathsf{ackout}) = [\,]$ and $\pi(\mathsf{u}_{21}) = 2$ then the constraints for SCC $\{1, 3, 4, 5\}$ amount to

$$\begin{array}{cccc} 0 \gtrsim \mathsf{ackout} & \mathsf{ackout} \gtrsim \mathsf{ackout} & \mathsf{s}(x) > x & \mathsf{s}(x) > \mathsf{U}_{21}(x) \\ \mathsf{s}(x) \gtrsim x & y \gtrsim y & \mathsf{s}(x) > \mathsf{s}(x) & \mathsf{U}_{21}(y) > y \end{array}$$

LPO with precedence $0 \succ \mathsf{ackout}$ and $\mathsf{s} \succ \mathsf{U}_{21}$ satisfies all these constraints, except $\mathsf{s}(x) > \mathsf{s}(x)$. This latter constraint originates from dependency pair (3). Since the induced subgraph of this pair consists of a single arrow, there is one new SCC: $\{3\}$. By taking the argument filtering π with $\pi(\mathsf{ackin}) = \pi(\mathsf{u}_{11}) = \pi(\mathsf{u}_{22}) = 1$, $\pi(\mathsf{ACKIN}) = \pi(\mathsf{u}_{21}) = 2$ and $\pi(\mathsf{ackout}) = [\,]$, the resulting constraints for SCC $\{3\}$ are satisfied by LPO with precedence $0 \succ \mathsf{ackout}$.

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

termination according to Corollary 8 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 is at most 2^n-1. Example 13 below shows that this upper bound cannot be improved. It is easy to see that the new approach of Theorem 11 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 termination proof generated by the new algorithm involves (much) less work than the one generated by the approach based on Theorem 2.

Example 13. As an extreme example, consider the TRS $\mathcal{R}$ (Example 11 in [8]) consisting of the rules

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

The only defined symbol, D, occurs 12 times in the right-hand sides of the rules, so 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 some reduction pair (based on some quasi-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 14. Consider the following TRS (from [3]):

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

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

One can imagine several other heuristics, like computing all argument filterings for function symbols of arity $n \leqslant 2$ but only some for function symbols of higher arity. Needless to say, adopting any of these heuristics reduces the class of TRSs that can be proved terminating automatically. Nevertheless, the experiments reported in Section 6 reveal that the two heuristics described above are surprisingly effective.

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 definition explains how these minimal argument filterings can be computed.

Definition 15. *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 $\mathrm{dom}(\pi) \subseteq \mathrm{dom}(\pi')$.*

Definition 16. *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 fully computed, i.e., until $\mathsf{outer}(t,\pi) = \varnothing$. Next we present an equivalent non-recursive definition of $\mathsf{AF}(t,\pi)$.

Definition 17. *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 18. *For all terms t and argument filterings π, $\mathsf{AF}(t,\pi) = \mathsf{AF}'(t,\pi)$.*

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

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

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

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

Since a term t can be completely evaluated by an argument filtering π if and only if $\mathsf{outer}(t,\pi) = \varnothing$, the next result is an immediate consequence of Lemma 18.

Corollary 19. *$\mathsf{AF}(t,\epsilon)$ is the set of all minimal argument filterings π such that $\pi(t)$ can be completely evaluated.* □

Definition 16 is easily extended to rewrite rules.

Definition 20. *For a rewrite rule $l \to r$ we define $\mathsf{AF}(l \to r) = \bigcup\{\mathsf{AF}(r,\pi) \mid \pi \in \mathsf{AF}(l,\epsilon)\}$ and $\mathsf{AF}_{\mathrm{vc}}(l \to r) = \{\pi \in \mathsf{AF}(l \to r) \mid \mathcal{V}\mathsf{ar}(\pi(r)) \not\subseteq \mathcal{V}\mathsf{ar}(\pi(l))\}$.*

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

The idea is now to (1) compute all argument filterings (with respect to AF, $\mathsf{AF}_{\mathrm{vc}}$, or $\mathsf{AF}_{\mathrm{lpo}}$)) for each constraint *separately* and (2) subsequently *merge* them to obtain the argument filterings of the full set of constraints.

Definition 22. *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$\}$.*

The following lemma expresses the fact that merging preserves the minimality property. The easy proof is omitted. Similar statements hold for $\mathsf{AF}_{\mathrm{vc}}$ and $\mathsf{AF}_{\mathrm{lpo}}$.

Lemma 23. *If $l_1 \to r_1$ and $l_2 \to r_2$ are rewrite rules then $\mathsf{AF}(l_1 \to r_1) \otimes \mathsf{AF}(l_2 \to r_2)$ is the set of all minimal argument filterings π such that $\pi(l_1)$, $\pi(r_1)$, $\pi(l_2)$, and $\pi(r_2)$ can be completely evaluated.* □

We illustrate the divide and conquer approach on the TRS of Example 14.

Example 24. 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 last column shows the cumulative effect of the merge operation with respect to $\mathsf{AF}_{\mathrm{lpo}}$. For instance, merging the 5 argument filterings for rule 1 with the 96 for rule 2 produces 165 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.

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

Table 1. Divide and conquer example.

$l \to r$	$\mathsf{AF}(\mathcal{F}\mathsf{un}(l \to r))$	$\mathsf{AF}(l \to r)$	$\mathsf{AF}_{\mathrm{vc}}(l \to r)$	$\mathsf{AF}_{\mathrm{lpo}}(l \to r)$	conquer
1	6	6	6	5	
2	396	231	108	96	165
3	2376	981	327	281	104
4	396	216	102	97	10
5	6	6	6	5	50
6	396	216	102	97	281
7	2376	981	327	281	44
8	396	231	108	96	28
9	6	6	3	3	84
10	6	6	6	5	45
11	36	36	27	23	25
12	18	12	12	11	50
13	3	3	3	3	150
14	18	16	11	11	120
15	3888	513	282	151	40

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 16 by $\mathsf{AF}_h(\mathsf{outer}(t, \pi))$ where h is the heuristic. With respect to Example 24, the *some more* heuristic would produce 16 and the *some* heuristic just 9 suitable argument filterings.

6 Experiments

Our ideas have been implemented in the termination prover T$_{\mathrm{T}}$T (Tsukuba Termination Tool), which is described in [11] and available at

`http://www.score.is.tsukuba.ac.jp/ttt`

We tested 227 examples from three different sources:

- all 82 terminating examples (59 in Section 3 and 23 in Section 4) from Arts and Giesl [3],
- all 23 examples from Dershowitz [8],
- all 122 examples from Steinbach and Kühler [16, Sections 3 and 4].

Of these 227 examples, 225 are terminating (Examples 4.34 and 4.40 from [16] are not). All experiments were performed on a PC equipped with an 850 MHz Pentium III CPU and 512 MB memory. Our first experiment concerns the two estimations of the dependency graph mentioned in Section 3. Table 2 lists the 13 examples where the two estimations differ. Only for Example 4.50 in [16] (which happens to be the rule of Toyama that we encountered in Example 7) does the

Table 2. Dependency graph estimation (I).

TRS	#DPs	EDG #arrows	EDG* #arrows	EDG #SCCs	EDG* #SCCs	EDG #cycles	EDG* #cycles
[3]:3.23	2	4	2	1	1	3	1
[3]:3.44	4	4	0	2	0	2	0
[3]:3.45	4	5	3	3	2	3	2
[3]:3.48	6	17	12	2	2	8	4
[3]:4.20(a)	3	3	1	2	1	2	1
[3]:4.20(b)	4	7	5	2	1	4	3
[3]:4.21	6	12	8	2	2	6	4
[3]:4.37(b)	4	6	3	3	2	3	2
[16]:2.51	3	8	7	1	1	6	5
[16]:2.52	9	36	35	4	4	17	16
[16]:4.44	4	4	0	2	0	2	0
[16]:4.50	1	1	0	1	0	1	0
[16]:4.59	6	12	4	3	2	5	2

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 3, where we show the effect of both estimations in combination with the three approaches 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 3.) 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.

Table 4 shows for several examples the effect of the three approaches to cycle analysis in combination with the heuristics for reducing the number of argument filterings. Question marks denote a timeout of one hour. In all experiments we used EDG*, except for the columns labeled "none" where the termination criterion of Theorem 1 is used. The last two rows indicate how many of the 225 terminating TRSs could actually be proved terminating within, respectively,

Table 3. Dependency graph estimation (II).

TRS	cycle EDG	cycle EDG*	scc EDG	scc EDG*	new EDG	new EDG*
[3]:3.48	3.25	0.35	0.78	0.23	0.99	0.23
[16]:4.50	*0.00*	0.00	*0.00*	0.00	*0.00*	0.00
[16]:4.59	6.45	3.37	4.45	3.44	4.45	3.32

Table 4. Cycle analysis and heuristics for argument filtering.

	some				some more				all			
TRS	none	cycle	scc	new	none	cycle	scc	new	none	cycle	scc	new
[3]:3.10	?	*24.22*	*24.18*	*24.68*	?	*668.22*	*666.19*	*640.44*	?	?	?	?
[3]:3.11	?	10.75	*8.40*	4.97	?	123.35	*198.40*	48.34	?	?	?	?
[3]:3.13	?	*16.32*	*38.98*	*12.88*	?	*120.46*	*402.53*	*92.91*	?	?	?	?
[3]:3.38	*0.08*	*0.01*	*0.01*	*0.01*	*0.95*	*0.05*	*0.05*	*0.05*	5.41	4.13	0.81	0.86
[3]:3.55	?	34.68	*76.95*	18.01	?	538.19	?	256.51	?	?	?	?
[3]:4.35	*106.94*	*1.62*	*1.20*	*1.35*	*2663.14*	*13.73*	*13.08*	*13.47*	?	*774.51*	*750.45*	*755.81*
[8]:8	0.00	0.11	0.01	0.01	0.00	0.10	0.01	0.01	0.01	0.31	0.02	0.02
[8]:27	*0.00*	0.01	*0.00*	0.00	*0.01*	0.01	*0.01*	0.01	0.00	0.00	0.00	0.00
[16]:2.14	0.00	0.00	0.01	0.01	0.00	0.01	0.00	0.00	0.90	0.15	0.16	0.16
[16]:2.29	0.02	0.02	0.01	0.02	0.02	0.02	0.02	0.01	247.62	9.43	9.24	9.36
[16]:2.61	4.75	0.10	0.10	0.10	206.17	0.36	0.34	0.32	574.93	7.47	7.50	7.58
[16]:4.2	*0.03*	*10.77*	*0.04*	*0.05*	*0.13*	*11.06*	*0.06*	*0.06*	*0.62*	*10.91*	*0.12*	*0.13*
[16]:4.59	0.12	0.04	0.04	0.05	0.55	0.01	0.01	0.01	56.68	3.37	3.44	3.32
225	99	128	120	129	107	138	127	139	114	137	134	138
225	99	126	120	128	106	136	127	137	99	136	132	136

one hour and ten seconds, and with LPO with strict precedence as base order. Changing the base order will greatly affect these numbers.

Table 5 shows the effect of the divide and conquer approach. For some examples we observe a dramatic increase in performance whereas for other examples the required time increases significantly. One reason for the latter is that in the divide and conquer approach *all* suitable argument filterings are computed. In addition, for every suitable argument filtering we store the set of minimal LPO precedences that satisfy the resulting constraints. This avoids many additional calls to the LPO constraint solving procedure, but can have a negative impact on both space and time requirements. This point clearly requires further investigation. Furthermore, the order in which the solutions to individual constraints are merged together obviously influences the performance of the divide and conquer approach. Further research is needed to develop good strategies.

References

1. T. Arts. System description: The dependency pair method. In *Proc. 11th RTA*, volume 1833 of *LNCS*, pages 261–264, 2000.
2. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *TCS*, 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. Available from `http://aib.informatik.rwth-aachen.de/`.

Table 5. Divide and conquer experiments.

	some				some more				all			
TRS	none	cycle	scc	new	none	cycle	scc	new	none	cycle	scc	new
[3]:3.10	*23.31*	*22.25*	*22.43*	*22.59*	*1702.96*	*1665.68*	*1648.29*	*1667.55*	?	?	?	?
[3]:3.11	*1.92*	0.67	*0.23*	0.37	*30.03*	6.01	*1.76*	2.92	?	562.46	332.70	332.63
[3]:3.13	*2.70*	*1.17*	*0.73*	*0.72*	*197.45*	*8.40*	*5.75*	*5.79*	?	?	?	?
[3]:3.38	*0.01*	*0.01*	*0.01*	*0.01*	*0.94*	*0.04*	*0.05*	*0.05*	27.26	67.72	4.37	4.41
[3]:3.55	*8.35*	1.49	*0.45*	0.83	*186.40*	23.77	*5.20*	9.85	?	?	2082.20	2055.73
[3]:4.35	*0.12*	*0.35*	*0.18*	*0.18*	*0.40*	*0.59*	*0.42*	*0.43*	*247.20*	*243.34*	*240.37*	*245.95*
[8]:8	0.05	1.56	0.04	0.05	0.24	6.51	0.19	0.19	0.87	29.69	0.84	0.83
[8]:27	*0.00*	0.00	*0.00*	0.01	*0.01*	0.01	*0.01*	0.01	0.12	0.12	0.12	0.12
[16]:2.14	0.01	0.01	0.00	0.01	0.03	0.01	0.01	0.01	0.13	0.02	0.02	0.02
[16]:2.29	2.92	0.34	0.36	0.34	192.29	6.42	6.26	6.59	?	144.37	144.79	145.33
[16]:2.61	70.40	0.44	0.40	0.44	1640.27	3.11	3.23	3.23	?	61.13	60.87	62.29
[16]:4.2	*0.01*	*10.59*	*0.04*	*0.04*	*0.02*	*10.65*	*0.04*	*0.04*	*0.08*	*10.40*	*0.07*	*0.06*
[16]:4.59	1.75	0.05	0.05	0.05	79.85	0.60	0.59	0.59	?	20.28	19.76	19.91
225	99	128	120	129	107	138	127	139	109	137	136	139
225	96	128	120	129	99	135	124	136	94	126	124	128

4. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
5. F. Bellegarde and P. Lescanne. Termination by completion. *AAECC*, 1:79–96, 1990.
6. C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In *Proc. 17th CADE*, volume 1831 of *LNAI*, 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 *LNCS*, pages 16–26, 1995.
9. J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *AAECC*, 12(1,2):39–72, 2001.
10. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *JSC*, 34(1):21–58, 2002.
11. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proc. 14th RTA*, LNCS, 2003. To appear.
12. S. Kamin and J.J. Lévy. Two generalizations of the recursive path ordering. Unpublished manuscript, University of Illinois, 1980.
13. A. Middeldorp. Approximating dependency graphs using tree automata techniques. In *Proc. IJCAR*, volume 2083 of *LNAI*, pages 593–610, 2001.
14. A. Middeldorp. Approximations for strategies and termination. In *Proc. 2nd WRS*, volume 70(6) of *ENTCS*, 2002. (Invited paper.).
15. J. Steinbach. Automatic termination proofs with transformation orderings. In *Proc. 6th RTA*, volume 914 of *LNCS*, pages 11–25, 1995.
16. J. Steinbach and U. Kühler. Check your ordering – termination proofs and open problems. Technical Report SR-90-25, Universität Kaiserslautern, 1990.

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