RWTH Aachen

Department of Computer Science

Technical Report

Improving Dependency Pairs

Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, and Stephan Falke

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2003-4

RWTH Aachen · Department of Computer Science · July 2003 (revised version)

Improving Dependency Pairs*

Jürgen Giesl, René Thiemann, Peter Schneider-Kamp, Stephan Falke

LuFG Informatik II, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany
{giesl|thiemann}@informatik.rwth-aachen.de
{nowonder|spf}@i2.informatik.rwth-aachen.de

Abstract. The dependency pair approach [2, 12, 13] is one of the most powerful techniques for termination and innermost termination proofs of term rewrite systems (TRSs). For any TRS, it generates inequality constraints that have to be satisfied by weakly monotonic well-founded orders. We improve the dependency pair approach by considerably reducing the number of constraints produced for (innermost) termination proofs.
Moreover, we extend transformation techniques to manipulate dependency pairs which simplify (innermost) termination proofs significantly. In order to fully automate the dependency pair approach, we show how transformation techniques and the search for suitable orders can be mechanized efficiently. We implemented our results in the automated termination prover AProVE and evaluated them on large collections of examples.

1 Introduction

Termination is an essential property of term rewrite systems. Most traditional methods to prove termination of TRSs (automatically) use *simplification orders* [8, 28], where a term is greater than its proper subterms (*subterm property*). Examples for simplification orders include lexicographic or recursive path orders [7, 19], the Knuth-Bendix order [20], and (most) polynomial orders [22]. However, there are numerous important TRSs which are not *simply terminating*, i.e., their termination cannot be shown by simplification orders. Therefore, the *dependency pair* approach [2, 12, 13] was developed which allows the application of simplification orders to non-simply terminating TRSs. In this way, the class of systems where termination is provable mechanically increases significantly.

Example 1 The following TRS from [2] is not simply terminating, since in the last quot-rule, the left-hand side is embedded in the right-hand side if y is instantiated with $\mathsf{s}(x)$. Thus, classical approaches for automated termination proofs fail on this example, while it is easy to handle with dependency pairs.

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

* Extended version of a paper from the *Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '03)*, Almaty, Kazakhstan, LNAI, Springer-Verlag, 2003.

In Sect. 2, we recapitulate the dependency pair approach for termination and innermost termination proofs (i.e., one tries to show that all reductions are finite, where in the innermost termination case, one only considers reductions of innermost redexes). Then we present new results which show that the approach can be improved significantly by reducing the constraints for termination (Sect. 3) and innermost termination (Sect. 4). Sect. 5 introduces new conditions for transforming dependency pairs by narrowing, rewriting, and instantiation in order to simplify (innermost) termination proofs further.

For automated (innermost) termination proofs, the constraints generated by the dependency pair approach are pre-processed by an *argument filtering* and afterwards, one tries to solve them by standard simplification orders. We present an algorithm to generate argument filterings in our improved dependency pair approach (Sect. 6) and discuss heuristics to increase efficiency in Sect. 7.

Our improvements and algorithms are implemented in our termination prover AProVE. We give empirical results which show that they are extremely successful in practice. Details on our experiments can be found in the appendix. Thus, the contributions of this paper are also very helpful for other tools based on dependency pairs (e.g., [1], CiME [6], TTT [17]). Moreover, we conjecture that they can also be used in other recent approaches for termination [5, 11] which have several aspects in common with dependency pairs. Finally, dependency pairs can be combined with other termination techniques (e.g., in [29] we integrated dependency pairs and the *size-change principle* from termination analysis of functional [23] and logic programs [10]). Moreover, the system TALP [26] uses dependency pairs for termination proofs of logic programs. Thus, improving dependency pairs is also useful for termination analysis of other kinds of programming languages.

2 Dependency Pairs

We briefly present the *dependency pair* approach of Arts and Giesl and refer to [2, 12, 13] for refinements and motivations. We assume familiarity with term rewriting (see, e.g., [4]). For a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined symbols* $\mathcal{D}$ are the root symbols of the left-hand sides of rules and the *constructors* are $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. We restrict ourselves to finite signatures and TRSs. Let $\mathcal{F}^\sharp = \{f^\sharp \mid f \in \mathcal{D}\}$ be a set of *tuple symbols*, where $f^\sharp$ has the same arity as f and we often write F for $f^\sharp$, etc. If $t = g(t_1, \ldots, t_m)$ with $g \in \mathcal{D}$, we write $t^\sharp$ for $g^\sharp(t_1, \ldots, t_m)$.

Definition 2 (Dependency Pair) *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 called a* dependency pair *of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted by $DP(\mathcal{R})$.*

So the dependency pairs of the TRS in Ex. 1 are

$$\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y) \tag{1}$$

$$\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y) \tag{2}$$

$$\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y)) \tag{3}$$

To use dependency pairs for (innermost) termination proofs, we need the notion of (innermost) *chains*. We always assume that different occurrences of dependency pairs are variable disjoint and we always consider substitutions whose domains may be infinite. Here, $\xrightarrow{\text{i}}_{\mathcal{R}}$ denotes innermost reductions.

Definition 3 ($\mathcal{R}$-Chain) *A sequence of dependency pairs $s_1 \to t_1$, $s_2 \to t_2, \dots$ is an* $\mathcal{R}$-chain *if there exists a substitution σ such that $t_j\sigma \to_{\mathcal{R}}^* s_{j+1}\sigma$ for every two consecutive pairs $s_j \to t_j$ and $s_{j+1} \to t_{j+1}$ in the sequence. Such a chain is an* innermost $\mathcal{R}$-chain *if $t_j\sigma \xrightarrow{\text{i}}_{\mathcal{R}}^* \sigma$ and if $s_j\sigma$ is a normal form for all j.*

Now we obtain the following sufficient and necessary criterion for termination and innermost termination.

Theorem 4 (Termination Criterion [2]) *$\mathcal{R}$ terminates iff there is no infinite chain. $\mathcal{R}$ is innermost terminating iff there is no infinite innermost chain.*

To estimate which dependency pairs may occur consecutively in (innermost) chains, one builds a so-called (innermost) *dependency graph* whose nodes are the dependency pairs and there is an arc from $v \to w$ to $s \to t$ iff $v \to w$, $s \to t$ is an (innermost) chain. In our example, the dependency graph and the innermost dependency graph have the arcs (1) $\Rightarrow$ (1), (2) $\Rightarrow$ (1), (3) $\Rightarrow$ (2), and (3) $\Rightarrow$ (3).

Since it is undecidable whether two dependency pairs form an (innermost) chain, we construct *estimated* graphs such that all cycles in the real graph are also cycles in the estimated graph. Let $\textsc{cap}(t)$ result from replacing all variables and all subterms of t that have a defined root symbol by different fresh variables. Here, multiple occurrences of the same variable are replaced by the same fresh variable, but multiple occurrences of the same subterm with defined root are replaced by pairwise different fresh variables. Let $\textsc{ren}(t)$ result from replacing all occurrences of variables in t by different fresh variables (i.e., $\textsc{ren}(t)$ is a linear term). For instance, $\textsc{cap}(\mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y))) = \mathsf{QUOT}(z, \mathsf{s}(y_1))$, $\textsc{cap}(\mathsf{QUOT}(x, x)) = \mathsf{QUOT}(x_1, x_1)$, and $\textsc{ren}(\mathsf{QUOT}(x, x)) = \mathsf{QUOT}(x_1, x_2)$. We define $\textsc{cap}_v$ like $\textsc{cap}$ except that subterms with defined root that already occur in v are not replaced by new variables.

Definition 5 (Estimated (innermost) dependency graph) *The* estimated dependency graph *of a TRS $\mathcal{R}$ is the directed graph whose nodes are the dependency pairs and there is an arc from $v \to w$ to $s \to t$ iff $\textsc{ren}(\textsc{cap}(w))$ and s are unifiable. In the* estimated innermost dependency graph *there is an arc from $v \to w$ to $s \to t$ iff $\textsc{cap}_v(w)$ and s are unifiable by a most general unifier (mgu) μ such that $v\mu$ and $s\mu$ are in normal form.*

In Ex. 1, the estimated dependency graph and the estimated innermost dependency graph are identical to the real dependency graph. Alternative approximations of dependency graphs can be found in [16, 24].

A set $\mathcal{P} \neq \varnothing$ of dependency pairs is called a *cycle* if for any two pairs $v \to w$ and $s \to t$ in $\mathcal{P}$ there is a non-empty path from $v \to w$ to $s \to t$ in the graph which only traverses pairs from $\mathcal{P}$. In our example, we have the cycles $\mathcal{P}_1 = \{(1)\}$ and $\mathcal{P}_2 = \{(3)\}$. Since we only regard finite TRSs, any infinite (innermost) chain of dependency pairs corresponds to a cycle in the (innermost) dependency graph.

To show (innermost) termination of TRSs, one proves absence of infinite (innermost) chains separately for every cycle. To this end, one generates sets of constraints which should be satisfied by some *reduction pair* $(\succsim, \succ)$ [21] consisting of a quasi-rewrite order $\succsim$ (i.e., $\succsim$ is reflexive, transitive, monotonic (closed under contexts), stable (closed under substitutions)) and a stable well-founded order $\succ$ which is compatible with $\succsim$ (i.e., $\succsim \circ \succ \subseteq \succ$ and $\succ \circ \succsim \subseteq \succ$). Note that $\succ$ need not be monotonic. Essentially, the constraints for termination of a cycle $\mathcal{P}$ ensure that all rewrite rules and all dependency pairs in $\mathcal{P}$ are weakly decreasing (w.r.t. $\succsim$) and at least one dependency pair in $\mathcal{P}$ is strictly decreasing (w.r.t. $\succ$). For innermost termination, only the *usable rules* have to be weakly decreasing. In Ex. 1, the usable rules for $\mathcal{P}_1$ are empty and the usable rules for $\mathcal{P}_2$ are the minus-rules.

Definition 6 (Usable Rules) *For $f \in \mathcal{F}$, let $Rls(f) = \{l \to r \in \mathcal{R} \mid root(l) = f\}$. For any term, the* usable rules *are the smallest set of rules such that*

- $\mathcal{U}(x) = \varnothing$ *for* $x \in \mathcal{V}$ *and*
- $\mathcal{U}(f(t_1, \ldots, t_n)) = Rls(f) \;\cup\; \bigcup_{l \to r \in Rls(f)} \mathcal{U}(r) \;\cup\; \bigcup_{j=1}^{n} \mathcal{U}(t_j)$.

Moreover, for any set $\mathcal{P}$ of dependency pairs, we define $\mathcal{U}(\mathcal{P}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}(t)$.

We want to use standard techniques to synthesize reduction pairs satisfying the constraints of the dependency pair approach. Most existing techniques generate monotonic *orders* $\succ$. However, for the dependency pair approach we only need a monotonic *quasi-order* $\succsim$, whereas $\succ$ does not have to be monotonic. (This is often called "*weak* monotonicity".) For that reason, before synthesizing a suitable order, some of the arguments of function symbols can be eliminated. To perform this elimination of arguments resp. of function symbols the concept of argument filtering was introduced in [2] (here we use the notation of [21]).

Definition 7 (Argument Filtering) *An* argument filtering *π for a signature $\mathcal{F}$ maps every n-ary function symbol to an argument position $i \in \{1, \ldots, n\}$ or to a (possibly empty) list $[i_1, \ldots, i_m]$ of argument positions with $1 \leq i_1 < \ldots < i_m \leq n$. The signature $\mathcal{F}_\pi$ consists of all function symbols f such that $\pi(f) = [i_1, \ldots, i_m]$, 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, which is defined as:*

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

An argument filtering with $\pi(f) = i$ for some $f \in \mathcal{F}$ is called collapsing.

Now the technique of automating dependency pairs can be formulated as follows. Here, we always use argument filterings for the signature $\mathcal{F} \cup \mathcal{F}^\sharp$.

Theorem 8 (Automating Dependency Pairs [2, 13]) *A TRS $\mathcal{R}$ is terminating iff for any cycle $\mathcal{P}$ of the (estimated) dependency graph, there is a reduction pair $(\succsim, \succ)$ and an argument filtering π such that both*

(a) $\pi(s) \succ \pi(t)$ for one dependency pair $s \to t$ from $\mathcal{P}$ and
$\pi(s) \succsim \pi(t)$ or $\pi(s) \succ \pi(t)$ for all other dependency pairs $s \to t$ from $\mathcal{P}$
(b) $\pi(l) \succsim \pi(r)$ for all $l \to r \in \mathcal{R}$

$\mathcal{R}$ is innermost terminating if for any cycle $\mathcal{P}$ of the (estimated) innermost dependency graph, there is a reduction pair $(\succsim, \succ)$ and an argument filtering π with both

(c) $\pi(s) \succ \pi(t)$ for one dependency pair $s \to t$ from $\mathcal{P}$ and
$\pi(s) \succsim \pi(t)$ or $\pi(s) \succ \pi(t)$ for all other dependency pairs $s \to t$ from $\mathcal{P}$
(d) $\pi(l) \succsim \pi(r)$ for all $l \to r \in \mathcal{U}(\mathcal{P})$

So in Ex. 1, we obtain the following constraints for termination. Here, $(\succsim_i, \succ_i)$ is the reduction pair and π_i is the argument filtering for cycle $\mathcal{P}_i$, where $i \in \{1, 2\}$.

$$\pi_1(\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y))) \succ_1 \pi_1(\mathsf{MINUS}(x, y)) \tag{4}$$
$$\pi_2(\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y))) \succ_2 \pi_2(\mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y))) \tag{5}$$
$$\pi_i(\mathsf{minus}(x, \mathsf{0})) \succsim_i \pi_i(x) \tag{6}$$
$$\pi_i(\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y))) \succsim_i \pi_i(\mathsf{minus}(x, y)) \tag{7}$$
$$\pi_i(\mathsf{quot}(\mathsf{0}, \mathsf{s}(y))) \succsim_i \pi_i(\mathsf{0}) \tag{8}$$
$$\pi_i(\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y))) \succsim_i \pi_i(\mathsf{s}(\mathsf{quot}(\mathsf{minus}(x, y), \mathsf{s}(y)))) \tag{9}$$

The filtering $\pi_i(\mathsf{minus}) = [1]$ replaces all terms $\mathsf{minus}(t_1, t_2)$ by $\mathsf{minus}(t_1)$. With this filtering, (4)–(9) are satisfied by the lexicographic path order (LPO) with the precedence $\mathsf{quot} > \mathsf{s} > \mathsf{minus}$. Thus, termination of this TRS is proved.

For innermost termination, we only obtain the constraint (4) for the cycle $\mathcal{P}_1$, since it has no usable rules. For $\mathcal{P}_2$, the constraints (8) and (9) are not necessary, since the quot-rules are not usable for any right-hand side of a dependency pair. In general, the constraints for innermost termination are always a subset of the constraints for termination. Thus, for classes of TRSs where innermost termination already implies termination (e.g., non-overlapping TRSs) [14], one should always use the approach for innermost termination when attempting termination proofs.

As shown in [16], to implement Thm. 8, one should not compute all cycles, but only maximal cycles (*strongly connected components (SCCs)*) that are not contained in other cycles. When solving the constraints of Thm. 8 for an SCC, the strict constraint $\pi(s) \succ \pi(t)$ may be satisfied for *several* dependency pairs $s \to t$ in the SCC. Thus, subcycles of the SCC containing such a strictly decreasing dependency pair do not have to be considered anymore. So after solving the constraints for the initial SCCs, all strictly decreasing dependency pairs are removed and one now builds SCCs from the remaining dependency pairs, etc.

3 Improved Termination Proofs

Now the technique of Thm. 8 for automated termination proofs is improved. For automation, one usually uses a *quasi-simplification order* $\succsim$ (i.e., a monotonic, stable quasi-order with $f(\ldots t \ldots) \succsim t$ for any term t and symbol f). As observed in [25], then the constraints (a) and (b) of Thm. 8 even imply C_ε-termination of $\mathcal{R}$. A TRS $\mathcal{R}$ is C_ε*-terminating* iff $\mathcal{R} \cup \{c(x,y) \to x, c(x,y) \to y\}$ is terminating where c is a fresh function symbol not occurring in $\mathcal{R}$. Urbain showed in [31] how to use dependency pairs for modular termination proofs of hierarchical combinations of C_ε-terminating TRSs. However in the results of [31], he did not integrate the consideration of cycles in (estimated) dependency graphs and required all dependency pairs to be strictly decreasing. Thm. 9 extends his modularity results by combining them with cycles. In this way, one obtains an improvement for termination proofs with dependency pairs which can be used for TRSs in general. The advantage is that the set of constraints (b) in Thm. 8 is reduced significantly.

The crucial idea of [31] is to consider the recursion hierarchy of function symbols. A function symbol f *depends on* the symbol h (denoted $f \geq_d h$) if $f = h$ or if there exists a symbol g such that g occurs in an f-rule and g depends on h. We define $>_d = \geq_d \setminus \leq_d$ and $\sim_d = \geq_d \cap \leq_d$. So $f \sim_d g$ means that f and g are mutually recursive. If $\mathcal{R} = \mathcal{R}_1 \uplus \ldots \uplus \mathcal{R}_n$ and $f \sim_d g$ iff $\mathit{Rls}(f) \cup \mathit{Rls}(g) \subseteq \mathcal{R}_i$, then we call $\mathcal{R}_1, \ldots, \mathcal{R}_n$ a *separation* of $\mathcal{R}$. Moreover, we extend $\geq_d$ to the sets $\mathcal{R}_i$ by defining $\mathcal{R}_i \geq_d \mathcal{R}_j$ iff $f \geq_d g$ for all f, g with $\mathit{Rls}(f) \subseteq \mathcal{R}_i$ and $\mathit{Rls}(g) \subseteq \mathcal{R}_j$. For any i, let $\mathcal{R}'_i$ denote the rules that $\mathcal{R}_i$ depends on, i.e., $\mathcal{R}'_i = \bigcup_{\mathcal{R}_i \geq_d \mathcal{R}_j} \mathcal{R}_j$.

It is clear that any cycle can only consist of dependency pairs from one $\mathcal{R}_i$. Thus, in Thm. 8 we only have to regard cycles $\mathcal{P}$ with dependency pairs from $DP(\mathcal{R}_i)$. However, to detect the cycles $\mathcal{P}$, we still have to regard the dependency graph of the whole TRS $\mathcal{R}$. The reason is that we have to consider $\mathcal{R}$-chains, not just $\mathcal{R}_i$- or $\mathcal{R}'_i$-chains.[1]

Thm. 9 states that instead of requiring $\pi(l) \succsim \pi(r)$ for all rules $l \to r$ of $\mathcal{R}$, it suffices to demand it only for rules that $\mathcal{R}_i$ depends on, i.e., for rules from $\mathcal{R}'_i$. So in the termination proof of Ex. 1, $\pi(l) \succsim \pi(r)$ does not have to be required for the quot-rules when regarding the cycle $\mathcal{P}_1 = \{\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x,y)\}$. However, this improvement is sound only if $\succsim$ is a quasi-simplification order.[2]

Theorem 9 (Improved Termination Proofs with DPs) *Let $\mathcal{R}_1, \ldots, \mathcal{R}_n$ be a* separation *of $\mathcal{R}$. The TRS $\mathcal{R}$ is terminating if for all $1 \leq i \leq n$ and any cycle $\mathcal{P}$ of the (estimated) dependency graph of $\mathcal{R}$ with $\mathcal{P} \subseteq DP(\mathcal{R}_i)$, there is a reduction pair $(\succsim, \succ)$ where $\succsim$ is a quasi-simplification order and an argument filtering π such that both*

[1] To see this, consider Toyama's TRS [30] where $\mathcal{R}_1 = \mathcal{R}'_1 = \{\mathsf{f}(0,1,x) \to \mathsf{f}(x,x,x)\}$ and $\mathcal{R}_2 = \mathcal{R}'_2 = \{\mathsf{g}(x,y) \to x, \mathsf{g}(x,y) \to y\}$. $\mathcal{R}'_1$'s and $\mathcal{R}'_2$'s dependency graphs are empty, whereas the dependency graph of $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$ has a cycle. Hence, if one only considers the graphs of $\mathcal{R}'_1$ and $\mathcal{R}'_2$, one could falsely prove termination.

[2] It suffices if $\succsim$ is extendable to $c(x,y) \succsim x, c(x,y) \succsim y$ and $(\succsim, \succ)$ is still a reduction pair.

(a) $\pi(s) \succ \pi(t)$ *for one dependency pair* $s \to t$ *from* $\mathcal{P}$ *and*
$\pi(s) \succsim \pi(t)$ *or* $\pi(s) \succ \pi(t)$ *for all other dependency pairs* $s \to t$ *from* $\mathcal{P}$
(b) $\pi(l) \succsim \pi(r)$ *for all* $l \to r \in \mathcal{R}'_i$

Proof. We prove that (a) and (b) imply termination of $\mathcal{R}$. If $\mathcal{R}$ is not terminating, then by Thm. 4 there exists an infinite chain $s_1 \to t_1, s_2 \to t_2, \ldots$ of dependency pairs where $t_j\sigma \to^*_{\mathcal{R}} s_{j+1}\sigma$ for all j. As can be seen from the proof of Thm. 4 in [2, Thm. 6], the infinite chain and the substitution can be chosen in such a way that all $s_j\sigma$ and $t_j\sigma$ are terminating.

Without loss of generality, these dependency pairs come from a cycle $\mathcal{P}$ of the dependency graph of $\mathcal{R}$ where $\mathcal{P} \subseteq DP(\mathcal{R}_i)$. Let $\mathcal{R}''_i = \mathcal{R} \setminus \mathcal{R}'_i$. Then $\mathcal{R}'_i$ and $\mathcal{R}''_i$ form a hierarchical combination (thus, defined symbols of $\mathcal{R}'_i$ may occur as constructors in $\mathcal{R}''_i$, but not vice versa). By [31, Lemma 2] there exists a substitution σ' such that $t_j\sigma' \to^*_{\mathcal{R}'_{i\varepsilon}} s_{j+1}\sigma'$, where $\mathcal{R}'_{i\varepsilon} = \mathcal{R}'_i \cup \{c(x,y) \to x,\ c(x,y) \to y\}$. Since $\succsim$ is a quasi-simplification order with $\pi(l) \succsim \pi(r)$ for all $l \to r \in \mathcal{R}'_i$ by Constraint (b), we obtain $\pi(t_j\sigma') = \pi(t_j)\sigma'_\pi \succsim \pi(s_{j+1})\sigma'_\pi = \pi(s_{j+1}\sigma')$ where $\sigma'_\pi(x) = \pi(\sigma'(x))$ for all $x \in \mathcal{V}$. Note that here we also need $c(x,y) \succsim x$ and $c(x,y) \succsim y$, which holds for all quasi-simplification orders. Constraint (a) implies $\pi(s_j)\sigma'_\pi \succ \pi(t_j)\sigma'_\pi$ for infinitely many j and $\pi(s_j)\sigma'_\pi \succsim \pi(t_j)\sigma'_\pi$ for all remaining j. Thus, by compatibility of $\succsim$ and $\succ$ we obtain a contradiction to the well-foundedness of $\succ$. □

Example 10 This TRS of [27] shows that Thm. 9 not only increases efficiency, but also leads to a more powerful method. Here, $\mathsf{int}(\mathsf{s}^n(\mathsf{0}), \mathsf{s}^m(\mathsf{0}))$ computes $[\mathsf{s}^n(\mathsf{0}), \mathsf{s}^{n+1}(\mathsf{0}), \ldots, \mathsf{s}^m(\mathsf{0})]$, nil is the empty list, and cons represents list insertion.

$$\mathsf{intlist}(\mathsf{nil}) \to \mathsf{nil} \tag{10}$$
$$\mathsf{intlist}(\mathsf{cons}(x,y)) \to \mathsf{cons}(\mathsf{s}(x), \mathsf{intlist}(y)) \tag{11}$$
$$\mathsf{int}(\mathsf{0},\mathsf{0}) \to \mathsf{cons}(\mathsf{0}, \mathsf{nil}) \tag{12}$$
$$\mathsf{int}(\mathsf{0},\mathsf{s}(y)) \to \mathsf{cons}(\mathsf{0}, \mathsf{int}(\mathsf{s}(\mathsf{0}), \mathsf{s}(y))) \tag{13}$$
$$\mathsf{int}(\mathsf{s}(x),\mathsf{0}) \to \mathsf{nil} \tag{14}$$
$$\mathsf{int}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{intlist}(\mathsf{int}(x,y)) \tag{15}$$

The TRS is separated into the $\mathsf{intlist}$-rules $\mathcal{R}_1$ and the int-rules $\mathcal{R}_2 >_d \mathcal{R}_1$. The constraints of Thm. 8 for termination of $\mathcal{P} = \{\mathsf{INTLIST}(\mathsf{cons}(x,y)) \to \mathsf{INTLIST}(y)\}$ cannot be solved with reduction pairs based on simplification orders:

We must satisfy $\pi(\mathsf{INTLIST}(\mathsf{cons}(x,y))) \succ \pi(\mathsf{INTLIST}(y))$ $(*)$. We distinguish three cases. First, let $\pi(\mathsf{s}) \neq [\,]$ or $\pi(\mathsf{int}) = [\,]$. Then we have $\pi(\mathsf{int}(\mathsf{0},\mathsf{s}(y))) \succsim \pi(\mathsf{cons}(\mathsf{0},\mathsf{int}(\mathsf{s}(\mathsf{0}),\mathsf{s}(y)))) \succsim \pi(\mathsf{cons}(\mathsf{0},\mathsf{int}(\mathsf{0},\mathsf{s}(y))))$ by weak decreasingness of Rule (13) and the subterm property. When substituting x by $\mathsf{0}$ and y by $\mathsf{int}(\mathsf{0},\mathsf{s}(y))$ in $(*)$, we obtain a contradiction to well-foundedness of $\succ$.

Next, let $\pi(\mathsf{intlist}) = [\,]$. Rule (11) implies $\mathsf{intlist} \succsim \pi(\mathsf{cons}(\mathsf{s}(x), \mathsf{intlist}(\ldots)))$ which gives a similar contradiction when substituting x by $\mathsf{s}(x)$ and y by $\mathsf{intlist}(...)$ in $(*)$.

Finally, let $\pi(\mathsf{s}) = [\,]$, $\pi(\mathsf{int}) \neq [\,]$, and $\pi(\mathsf{intlist}) \neq [\,]$. Now we obtain a contradiction, since the filtered rule (15) cannot be weakly decreasing. The reason is that x or y occur on its right-hand side, but not on its left-hand side.

In contrast, when using Thm. 9, only $\mathcal{R}'_1 = \mathcal{R}_1$ must be weakly decreasing when examining $\mathcal{P}$. These constraints are satisfied by the embedding order using an argument filtering with $\pi(\mathsf{cons}) = [2]$, $\pi(\mathsf{intlist}) = \pi(\mathsf{INTLIST}) = 1$, $\pi(s) = [1]$.

The constraints from $\mathcal{R}_2$'s cycle and rules from $\mathcal{R}'_2 = \mathcal{R}_1 \cup \mathcal{R}_2$ can also be oriented (by LPO and a filtering with $\pi(\mathsf{cons}) = 1$ and $\pi(\mathsf{INT}) = 2$). However, this part of the proof requires the consideration of cycles of the (estimated) dependency graph. The reason is that there is no argument filtering and simplification order such that both dependency pairs of $\mathcal{R}_2$ are strictly decreasing: $\pi(\mathsf{INT}(\mathsf{s}(x), \mathsf{s}(y))) \succ \pi(\mathsf{INT}(x, y))$ implies $\pi(\mathsf{s}) = [1]$. But then $\pi(\mathsf{INT}(\mathsf{0}, \mathsf{s}(y)))$ is embedded in $\pi(\mathsf{INT}(\mathsf{s}(\mathsf{0}), \mathsf{s}(y)))$. Hence, we have $\pi(\mathsf{INT}(\mathsf{s}(\mathsf{0}), \mathsf{s}(y))) \not\succ \pi(\mathsf{INT}(\mathsf{0}, \mathsf{s}(y)))$ for any simplification order $\succ$.

So if one only considers cycles or if one only uses Urbain's modularity result [31], then Ex. 10 fails with simplification orders. Instead, both refinements should be combined as in Thm. 9.

4 Improved Innermost Termination Proofs

Proving innermost termination with dependency pairs is easier than proving termination for two reasons: the innermost dependency graph has less arcs than the dependency graph and we only require $l \succsim r$ for the *usable* rules instead of all rules. In Sect. 3 we showed that for termination, it suffices to require $l \succsim r$ only for the rules of $\mathcal{R}'_i$ if the current cycle consists of $\mathcal{R}_i$-dependency pairs. Still, $\mathcal{R}'_i$ is always a superset of the usable rules. Now we present a new improvement of Thm. 8 for innermost termination in order to reduce the set of usable rules.

The idea is to apply the argument filtering first and to determine the usable rules afterwards. However, for collapsing argument filterings this destroys the soundness of the technique. Consider the non-innermost terminating TRS

$$\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(\mathsf{double}(x)) \qquad \mathsf{double}(\mathsf{0}) \to \mathsf{0} \qquad \mathsf{double}(\mathsf{s}(x)) \to \mathsf{s}(\mathsf{s}(\mathsf{double}(x)))$$

In the cycle $\{\mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(\mathsf{double}(x))\}$, we could use the argument filtering $\pi(\mathsf{double}) = 1$ which results in $\{\mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(x)\}$. Since the filtered dependency pair contains no defined symbols, we would conclude that the cycle has no usable rules. Then, we could easily orient the only resulting constraint $\mathsf{F}(\mathsf{s}(x)) \succ \mathsf{F}(x)$ for this cycle and falsely prove innermost termination. Note that the elimination of double in the term $\mathsf{F}(\mathsf{double}(x))$ is not due to the outer function symbol F, but due to a collapsing argument filtering for double itself. For that reason a defined symbol like double may only be ignored if all its occurrences are in positions which are filtered away by the function symbols *above* them. Moreover, similar to the refinement of $\textsc{cap}_v$, we build usable rules only from those subterms of right-hand sides of dependency pairs that do not occur in the corresponding left-hand side of the dependency pair.

Definition 11 (Usable Rules w.r.t. Argument Filtering) *Let π be an argument filtering. For an n-ary symbol f, the set $\mathcal{R}egPos_\pi(f)$ of* regarded positions *is $\{i\}$, if $\pi(f) = i$, and it is $\{i_1, \ldots, i_m\}$, if $\pi(f) = [i_1, \ldots, i_m]$. For a term, the* usable rules w.r.t. π *are the smallest set of rules such that*

- $\mathcal{U}(x, \pi) = \varnothing$ *for $x \in \mathcal{V}$ and*
- $\mathcal{U}(f(t_1, \ldots, t_n), \pi) = Rls(f) \,\cup\, \bigcup_{l \to r \in Rls(f)} \mathcal{U}(r, \pi) \,\cup\, \bigcup_{j \in \mathcal{R}egPos_\pi(f)} \mathcal{U}(t_j, \pi)$.

For a term s with $\mathcal{V}(t) \subseteq \mathcal{V}(s)$, we define $\mathcal{U}_s(t, \pi) = \varnothing$ if t is a subterm of s. Otherwise, the definition is similar to $\mathcal{U}(t, \pi)$, i.e.,

$$\mathcal{U}_s(f(t_1, \ldots, t_n), \pi) = Rls(f) \,\cup\, \bigcup_{l \to r \in Rls(f)} \mathcal{U}(r, \pi) \,\cup\, \bigcup_{j \in \mathcal{R}egPos_\pi(f)} \mathcal{U}_s(t_j, \pi).$$

Moreover, for any set $\mathcal{P}$ of dependency pairs, let $\mathcal{U}(\mathcal{P}, \pi) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}_s(t, \pi)$.

To prove the soundness of our refinement for innermost termination proofs, we need the following lemma. For a reduction pair $(\succsim, \succ)$, the pair $(\succsim_\pi, \succ_\pi)$ results from applying an argument filtering, where $t \succsim_\pi u$ iff $\pi(t) \succsim \pi(u)$ and $t \succ_\pi u$ iff $\pi(t) \succ \pi(u)$. In [2] it was shown that $(\succsim_\pi, \succ_\pi)$ is indeed a reduction pair as well.

Lemma 12 (Properties of Usable Rules) *Let $\mathcal{R}$ be a TRS, let π be an argument filtering, let $(\succsim, \succ)$ be a reduction pair, and let s and t be terms with $\mathcal{V}(t) \subseteq \mathcal{V}(s)$, where s is in normal form. Then we have*

(i) $\mathcal{U}_{s\sigma}(t\sigma, \pi) \subseteq \mathcal{U}_s(t, \pi) \subseteq \mathcal{U}(t, \pi)$ *for all substitutions σ*
(ii) $t \to_\mathcal{R} u$ *implies* $\mathcal{U}_s(t, \pi) \supseteq \mathcal{U}_s(u, \pi)$
(iii) If $l \succsim_\pi r$ holds for all $l \to r \in \mathcal{U}_s(t, \pi)$ and $t \to_\mathcal{R} u$, then $t \succsim_\pi u$.
(iv) If $l \succsim_\pi r$ holds for all $l \to r \in \mathcal{U}_s(t, \pi)$ and $t \to_\mathcal{R}^* u$, then $t \succsim_\pi u$.

Proof.

(i) We use structural induction on t. If t is a variable, then t is a subterm of s and thus, $\mathcal{U}_{s\sigma}(t\sigma, \pi) = \mathcal{U}_s(t, \pi) = \varnothing$. Otherwise, t has the form $f(t_1, \ldots, t_n)$. Then

$$\begin{aligned}
\mathcal{U}_{s\sigma}(t\sigma, \pi) &= Rls(f) \cup \textstyle\bigcup_{l \to r \in Rls(f)} \mathcal{U}(r, \pi) \cup \bigcup_{j \in \mathcal{R}egPos_\pi(f)} \mathcal{U}_{s\sigma}(t_j\sigma, \pi) \\
&\overset{(ind.)}{\subseteq} Rls(f) \cup \textstyle\bigcup_{l \to r \in Rls(f)} \mathcal{U}(r, \pi) \cup \bigcup_{j \in \mathcal{R}egPos_\pi(f)} \mathcal{U}_s(t_j, \pi) = \mathcal{U}_s(t, \pi) \\
&\overset{(ind.)}{\subseteq} Rls(f) \cup \textstyle\bigcup_{l \to r \in Rls(f)} \mathcal{U}(r, \pi) \cup \bigcup_{j \in \mathcal{R}egPos_\pi(f)} \mathcal{U}(t_j, \pi) = \mathcal{U}(t, \pi)
\end{aligned}$$

(ii) Let $t \to_\mathcal{R} u$ using the rule $l \to r \in \mathcal{R}$. We perform structural induction on the position p of the redex. If $p = \epsilon$ then $t = l\sigma \to_\mathcal{R} r\sigma = u$ for a substitution σ. As t can be reduced, it is no normal form and thus, no subterm of s. Hence, $\mathcal{U}_s(t, \pi) \supseteq \mathcal{U}(r, \pi) \supseteq \mathcal{U}_s(r\sigma, \pi) = \mathcal{U}_s(u, \pi)$, by (i).
Otherwise $p = jp', t = f(t_1 \ldots t_j \ldots t_n)$, $u = f(t_1 \ldots u_j \ldots t_n)$, and $t_j \to_\mathcal{R} u_j$. If $j \notin \mathcal{R}egPos_\pi(f)$, then $\mathcal{U}_s(t, \pi) = \mathcal{U}_s(u, \pi)$. If $j \in \mathcal{R}egPos_\pi(f)$, then

$$\begin{aligned}
\mathcal{U}_s(t, \pi) &= Rls(f) \cup \textstyle\bigcup_{l \to r \in Rls(f)} \mathcal{U}(r, \pi) \cup ... \cup \mathcal{U}_s(t_j, \pi) \cup ... \\
&\overset{(ind.)}{\supseteq} Rls(f) \cup \textstyle\bigcup_{l \to r \in Rls(f)} \mathcal{U}(r, \pi) \cup ... \cup \mathcal{U}_s(u_j, \pi) \cup ... = \mathcal{U}_s(u, \pi).
\end{aligned}$$

(iii) We use induction on the position p of the redex. If $p = \epsilon$ then $t = l\sigma \to_{\mathcal{R}} r\sigma = u$. Again, t is not a normal form and therefore, no subterm of s. Hence, $l \to r \in \mathcal{U}_s(t, \pi)$, so $l \succsim_\pi r$ and $t \succsim_\pi u$, since $\succsim_\pi$ is stable.
If $p = jp'$, we have $t = f(t_1 \ldots t_j \ldots t_n)$, $u = f(t_1 \ldots u_j \ldots t_n)$, and $t_j \to_{\mathcal{R}} u_j$. If $j \notin \mathcal{R}egPos_\pi(f)$, then $\pi(t) = \pi(u)$ and thus, $t \succsim_\pi u$. Otherwise, $j \in \mathcal{R}egPos_\pi(f)$ and hence, $\mathcal{U}_s(t, \pi) \supseteq \mathcal{U}_s(t_j, \pi)$. So we can apply the induction hypothesis and conclude $t_j \succsim_\pi u_j$. Monotonicity of $\succsim_\pi$ implies $t \succsim_\pi u$.

(iv) This follows from (ii) and (iii) by induction on the number of reduction steps. □

Now we can refine the innermost termination technique of Thm. 8 (c) and (d) to the following one where the set of usable rules is reduced significantly.

Theorem 13 (Improved Innermost Termination with DPs) *$\mathcal{R}$ is innermost terminating if for any cycle $\mathcal{P}$ of the (estimated) innermost dependency graph, there is a reduction pair $(\succsim, \succ)$ and an argument filtering π such that both*

(c) $\pi(s) \succ \pi(t)$ for one dependency pair $s \to t$ from $\mathcal{P}$ and
$\pi(s) \succsim \pi(t)$ or $\pi(s) \succ \pi(t)$ for all other dependency pairs $s \to t$ from $\mathcal{P}$
(d) $\pi(l) \succsim \pi(r)$ for all $l \to r \in \mathcal{U}(\mathcal{P}, \pi)$

Proof. By Thm. 4, we have to show absence of infinite innermost chains. Let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite innermost chain from the cycle $\mathcal{P}$. So there is a substitution σ with $t_j\sigma \xrightarrow{\text{i}}_{\mathcal{R}}^* s_{j+1}\sigma$ for all j, where all s_j and $s_j\sigma$ are in normal form. From (d) we get $l \succsim_\pi r$ for all $l \to r \in \mathcal{U}_{s_j}(t_j, \pi)$. Since $\mathcal{U}_{s_j\sigma}(t_j\sigma, \pi) \subseteq \mathcal{U}_{s_j}(t_j, \pi)$ by Lemma 12 (i), we also have $l \succsim_\pi r$ for all $l \to r \in \mathcal{U}_{s_j\sigma}(t_j\sigma, \pi)$. Hence, we can use Lemma 12 (iv) to obtain $t_j\sigma \succsim_\pi s_{j+1}\sigma$. By (c) and closure of $\succ_\pi$ under substitutions, we obtain $s_1\sigma \succsim_\pi t_1\sigma \succsim_\pi s_2\sigma \succsim_\pi \ldots$ where $s_j\sigma \succ_\pi t_j\sigma$ holds for infinitely many j in contradiction to the well-foundedness of $\succ_\pi$. □

Example 14 This TRS of [18] for list reversal shows the advantages of Thm. 13.

$$\mathsf{rev}(\mathsf{nil}) \to \mathsf{nil} \tag{16}$$
$$\mathsf{rev}(\mathsf{cons}(x, l)) \to \mathsf{cons}(\mathsf{rev1}(x, l), \mathsf{rev2}(x, l)) \tag{17}$$
$$\mathsf{rev1}(x, \mathsf{nil}) \to x \tag{18}$$
$$\mathsf{rev1}(x, \mathsf{cons}(y, l)) \to \mathsf{rev1}(y, l) \tag{19}$$
$$\mathsf{rev2}(x, \mathsf{nil}) \to \mathsf{nil} \tag{20}$$
$$\mathsf{rev2}(x, \mathsf{cons}(y, l)) \to \mathsf{rev}(\mathsf{cons}(x, \mathsf{rev}(\mathsf{rev2}(y, l)))) \tag{21}$$

When proving innermost termination with Thm. 8, for the cycle of the REV- and $\mathsf{REV2}$-dependency pairs, we would obtain inequalities from the dependency pairs and $\pi(l) \succsim \pi(r)$ for all rules $l \to r$, since all rules are usable. But with standard reduction pairs which are based on lexicographic or recursive path orders possibly with status (RPOS), Knuth-Bendix orders (KBO), or polynomial orders, these constraints are not satisfiable for any argument filtering.

To prove this for LPO, RPO(S), and KBO, we first show that if an argument position is eliminated by an argument filtering π, then the constraints cannot be satisfied. From (18) we obtain $1 \in \mathcal{R}egPos_\pi(\mathsf{rev1})$ which leads to $2 \in \mathcal{R}egPos_\pi(\mathsf{rev1})$ and $1, 2 \in \mathcal{R}egPos_\pi(\mathsf{cons})$ by using (19) twice, so $\pi(\mathsf{rev1}) = \pi(\mathsf{cons}) = [1, 2]$. Using (17) we obtain $1 \in \mathcal{R}egPos_\pi(\mathsf{rev})$. Now we can conclude $\pi(\mathsf{rev2}) = [1, 2]$ from (21). If we have $\pi(\mathsf{rev}) = 1$, then (17) yields a contradiction to the subterm property. Hence, $\pi(\mathsf{rev}) = [1]$. Thus, if we search for a simplification order such that the rules are weakly decreasing, then we are not allowed to drop any argument or function symbol in the filtering. Hence, it is sufficient to examine whether the orders above are able to make the unfiltered rules weakly decreasing.

There is no KBO satisfying these constraints since (17) is duplicating. If we want to orient the constraints by some lexicographic or recursive path order, we need a precedence with $\mathsf{rev2} > \mathsf{rev}$ due to (21). But this precedence cannot be extended further such that (17) can be oriented.

There is also no polynomial order such that the rules are weakly decreasing. A polynomial interpretation has the following form.

$$\begin{array}{lll} \mathcal{P}ol(\mathsf{rev}(l)) & = p_1(l), & \text{where } \ p_1(l) = p'_1 \cdot l^{n_1} + p''_1(l) \\ \mathcal{P}ol(\mathsf{rev1}(x, l)) & = p_2(x, l), & \text{where } p_2(x, l) = p'_2(x) \cdot l^{n_2} + p''_2(x, l) \\ \mathcal{P}ol(\mathsf{rev2}(x, l)) & = p_3(x, l), & \text{where } p_3(x, l) = p'_3(x) \cdot l^{n_3} + p''_3(x, l) \\ \mathcal{P}ol(\mathsf{cons}(x, l)) & = p_4(x, l), & \text{where } p_4(x, l) = p'_4(x) \cdot l^{n_4} + p''_4(x, l) \\ \mathcal{P}ol(\mathsf{nil}) & = p_5 & \end{array}$$

Here, n_1, n_2, n_3, n_4 denote the highest exponents used for l in the respective polynomials, where p'_i and p''_i are polynomials with coefficients from $\mathbb{N}$. So in $p''_1(l), p''_2(x, l), p''_3(x, l), p''_4(x, l)$, the variable l occurs only with exponents smaller than the corresponding n_i. Similar to the argumentation above, where we showed that with simplification orders one may not filter away any arguments, it is easy to show that $\mathcal{P}ol(\mathsf{rev1}(x, l))$, $\mathcal{P}ol(\mathsf{rev2}(x, l))$, and $\mathcal{P}ol(\mathsf{cons}(x, l))$ must depend on x and l and $\mathcal{P}ol(\mathsf{rev}(l))$ must depend on l. Hence, all values n_i must be at least 1 and the polynomials p'_i are not the number 0.

From the constraints of (17) and (21) we obtain

$$\mathcal{P}ol(\mathsf{rev}(\mathsf{cons}(x, l))) \geq \mathcal{P}ol(\mathsf{cons}(\mathsf{rev1}(x, l), \mathsf{rev2}(x, l)))$$
$$\mathcal{P}ol(\mathsf{rev2}(x, \mathsf{cons}(x, l))) \geq \mathcal{P}ol(\mathsf{rev}(\mathsf{cons}(x, \mathsf{rev}(\mathsf{rev2}(x, l))))).$$

We now examine those parts of the polynomials which have the largest exponent for l. So for large enough instantiations of l (and instantiations of x where the p'_i are non-zero) we must have

$$p'_1 \cdot p'_4(x)^{n_1} \cdot l^{n_1 \cdot n_4} \geq p'_4(p'_2(x) \cdot l^{n_2}) \cdot p'_3(x)^{n_4} \cdot l^{n_3 \cdot n_4} \tag{22}$$
$$p'_3(x) \cdot p'_4(x)^{n_3} \cdot l^{n_3 \cdot n_4} \geq p'_4(x)^{n_1} \cdot {p'_1}^{n_1 \cdot n_4 + 1} \cdot p'_3(x)^{n_1^2 \cdot n_4} \cdot l^{n_1^2 \cdot n_3 \cdot n_4} \tag{23}$$

Comparison of the highest exponents of l yields $n_1 \cdot n_4 \geq n_3 \cdot n_4 \geq n_1^2 \cdot n_3 \cdot n_4$ and thus, $n_1 = n_3 = 1$. Moreover, $p'_4(x)$ may not depend on x, since otherwise

(22) would imply $n_1 \cdot n_4 \geq n_3 \cdot n_4 + n_2$. Now (22) and (23) simplify to

$$p'_1 \geq p'_3(x)^{n_4} \tag{24}$$

$$p'_3(x) \geq {p'_1}^{n_4+1} \cdot p'_3(x)^{n_4} \tag{25}$$

From (24) and (25) we can conclude that $p'_3(x)$ does not depend on x and $p'_3 = p'_1 = 1$. Hence, our polynomial interpretation is as follows:

$$\begin{aligned}
\mathcal{P}ol(\mathsf{rev}(l)) &= l + p''_1\\
\mathcal{P}ol(\mathsf{rev1}(x,l)) &= p'_2(x) \cdot l^{n_2} + p''_2(x,l)\\
\mathcal{P}ol(\mathsf{rev2}(x,l)) &= l + p''_3(x)\\
\mathcal{P}ol(\mathsf{cons}(x,l)) &= p'_4 \cdot l^{n_4} + p''_4(x,l)\\
\mathcal{P}ol(\mathsf{nil}) &= p_5
\end{aligned}$$

Now we obtain

$$p'_4 \cdot p_5^{n_4} + p''_4(x,p_5) + p''_1 = \tag{26}$$

$$\mathcal{P}ol(\mathsf{rev}(\mathsf{cons}(x,\mathsf{nil}))) \geq \tag{27}$$

$$\mathcal{P}ol(\mathsf{cons}(\mathsf{rev1}(x,\mathsf{nil}),\mathsf{rev2}(x,\mathsf{nil}))) \geq \tag{28}$$

$$\mathcal{P}ol(\mathsf{cons}(x,\mathsf{rev2}(x,\mathsf{nil}))) = \tag{29}$$

$$p'_4 \cdot (p_5 + p''_3(x))^{n_4} + p''_4(x, p_5 + p''_3(x)) \geq \tag{30}$$

$$p'_4 \cdot p_5^{n_4} + p''_4(x,p_5) + p'_4 \cdot p''_3(x)^{n_4} \tag{31}$$

The step from (27) to (28) is due to the weak decreasingness of Rule (17) and the step from (28) to (29) follows from monotonicity and Rule (18). Note that these inequalities give a contradiction if one instantiates x with a large enough value like $p''_1 + 1$, since $\mathcal{P}ol(\mathsf{rev2}(x,l))$ and hence $p''_3(x)$ must depend on x.

So the most common orders that are amenable to automation fail when trying to prove termination according to Thm. 8. In contrast, when using Thm. 13 and a filtering with $\pi(\mathsf{cons}) = [2]$, $\pi(\mathsf{REV}) = \pi(\mathsf{rev}) = 1$, and $\pi(\mathsf{REV2}) = \pi(\mathsf{rev2}) = 2$, we do not obtain any constraints from the rev1-rules and all filtered constraints can be oriented by the embedding order.

Our experiments with the system AProVE show that Thm. 9 and 13 indeed improve upon Thm. 8 in practice by increasing power (in particular if reduction pairs are based on simple fast orders like the embedding order) and by reducing runtimes (in particular if reduction pairs are based on more complex orders). More details are given in the appendix.

5 Transforming Dependency Pairs

To increase the power of the dependency pair technique, a dependency pair may be transformed into several new pairs by *narrowing*, *rewriting*, and *instantiation* [2, 12]. A term t' is an *$\mathcal{R}$-narrowing of t with the mgu μ*, if a non-variable subterm $t|_p$ of t unifies with the left-hand side of a (variable-renamed) rule $l \to r \in \mathcal{R}$ with mgu μ, and $t' = t[r]_p\,\mu$. To distinguish the variants for termination and innermost termination, we speak of *t*- and *i-narrowing* resp. *-instantiation*.

Definition 15 (Transformations) *For a TRS $\mathcal{R}$ and a set $\mathcal{P}$ of pairs of terms*

- $\mathcal{P} \uplus \{s \to t\}$ t-narrows *to* $\mathcal{P} \uplus \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\}$ *iff* $t_1, \ldots, t_n$ *are all* $\mathcal{R}$*-narrowings of* t *with the mgu's* $\mu_1, \ldots, \mu_n$ *and* t *does not unify with (variable-renamed) left-hand sides of pairs in* $\mathcal{P}$*. Moreover,* t *must be linear.*
- $\mathcal{P} \uplus \{s \to t\}$ i-narrows *to* $\mathcal{P} \uplus \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\}$ *iff* $t_1, \ldots, t_n$ *are all* $\mathcal{R}$*-narrowings of* t *with the mgu's* $\mu_1, \ldots, \mu_n$ *such that* $s\mu_i$ *is in normal form. Moreover, for all* $v \to w \in \mathcal{P}$ *where* t *unifies with the (variable-renamed) left-hand side* v *by a mgu* μ*, one of the terms* $s\mu$ *or* $v\mu$ *must not be in normal form.*
- $\mathcal{P} \uplus \{s \to t\}$ rewrites *to* $\mathcal{P} \uplus \{s \to t'\}$ *iff* $\mathcal{U}(t|_p)$ *is non-overlapping and* $t \to_{\mathcal{R}} t'$*, where* p *is the position of the redex.*
- $\mathcal{P} \uplus \{s \to t\}$ *is* t-instantiated *to* $\mathcal{P} \uplus \{s\mu \to t\mu \mid \mu = mgu(\textsc{ren}(\textsc{cap}(w)), s),\ v \to w \in \mathcal{P}\}$.
- $\mathcal{P} \uplus \{s \to t\}$ *is* i-instantiated *to* $\mathcal{P} \uplus \{s\mu \to t\mu \mid \mu = mgu(\textsc{cap}_v(w), s), v \to w \in \mathcal{P}, s\mu, v\mu \text{ are normal forms}\}$.

For innermost termination, Def. 15 extends the transformations of [2, 12] by permitting their application for a larger set of TRSs. In [12], narrowing a pair $s \to t$ was not permitted if t unifies with the left-hand side of some dependency pair, whereas now this is possible under certain conditions. Rewriting dependency pairs was only allowed if all usable rules for the current cycle were non-overlapping, whereas now this is only required for the usable rules of the redex to be rewritten. Finally, when instantiating dependency pairs, in contrast to [12] one can now use $\textsc{cap}_v$. Moreover, for both instantiation and narrowing of dependency pairs, now one only has to consider instantiations which turn left-hand sides of dependency pairs into normal forms.

The following theorem states that in the techniques for termination and innermost termination proofs (Thm. 9 and 13), instead of the original dependency pairs one may regard pairs that are transformed according to Def. 15. Of course, then Thm. 9 and 13 have to be updated accordingly (e.g., in Thm. 9, instead of $\mathcal{P} \subseteq DP(\mathcal{R}_i)$ we now permit that $\mathcal{P}$ results from dependency pairs of $DP(\mathcal{R}_i)$ by transformations).

Theorem 16 (Narrowing, Rewriting, Instantiation) *Let $DP(\mathcal{R})'$ result from $DP(\mathcal{R})$ by t-narrowing and t-instantiation (for termination) resp. by i-narrowing, rewriting, and i-instantiation (for innermost termination). If the dependency pair constraints for (innermost) termination are satisfiable using $DP(\mathcal{R})'$, then $\mathcal{R}$ is (innermost) terminating. Moreover, if certain reduction pairs and argument filterings satisfy the constraints for $DP(\mathcal{R})$, then the* same *reduction pairs and argument filterings satisfy the constraints for $DP(\mathcal{R})'$.*[3]

[3] Of course, the constraints depend on the approximation of the (innermost) dependency graph. Here, we use the estimation of Def. 5.

Proof. For *soundness* of the transformations, we prove that if there is an infinite (innermost) chain of pairs from $DP(\mathcal{R})$, then there is also an infinite (innermost) chain of pairs from $DP(\mathcal{R})'$.[4] Hence, if the dependency pair constraints using $DP(\mathcal{R})'$ are satisfiable, then by the soundness of the dependency pair approach, the TRS is (innermost) terminating. For *completeness*, we show that if the dependency pair constraints for $DP(\mathcal{R})$ using the estimated (innermost) dependency graph of Def. 5 are satisfied by some reduction pairs and argument filterings, then the same reduction pairs and argument filterings satisfy the constraints for $DP(\mathcal{R})'$.

- *narrowing*: Soundness of t-narrowing is proved in [2, Thm. 27] and soundness of i-narrowing is proved in [12, Thm. 12] (the soundness of the refined version of i-narrowing in Def. 15 follows from the fact that in innermost chains, one regards a substitution such that the instantiated left components of all dependency pair are in normal form).
  For completeness of t-narrowing, we assume that $DP(\mathcal{R})'$ is the result of t-narrowing a dependency pair $s \to t$ from $DP(\mathcal{R})$. In this transformation, $s \to t$ was replaced by its narrowings $s\mu_1 \to t_1, \ldots, s\mu_n \to t_n$.
  We first show that if $\mathcal{P}'$ is a cycle of the estimated dependency graph of $DP(\mathcal{R})'$ containing some pair $s\mu_i \to t_i$, then $\mathcal{P} = \mathcal{P}' \setminus \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\} \cup \{s \to t\}$ is a cycle in the estimated dependency graph of $DP(\mathcal{R})$. Assume there is an arc from $v \to w$ to $s\mu_i \to t_i$ in the estimated dependency graph of $DP(\mathcal{R})'$, i.e., $\textsc{ren}(\textsc{cap}(w))\sigma = s\mu_i\sigma$ for some substitution σ. Then there is also an arc from $v \to w$ to $s \to t$ in the estimated dependency graph of $DP(\mathcal{R})$ using the same substitution σ on the variables of $v \to w$ by extending it to behave like $\mu_i\sigma$ on the variables of s and t. (Recall that we may assume that the variables in $s \to t$ are disjoint from all other variables.) Similarly, if there is an arc from $s\mu_i \to t_i$ to $v \to w$ (i.e., $\textsc{ren}(\textsc{cap}(t_i))\sigma = v\sigma$), then there is also an arc from $s \to t$ to $v \to w$ in the estimated dependency graph of $DP(\mathcal{R})$ using a similar substitution σ as above. The reason is that $t\mu_i \to_{\mathcal{R}} t_i$, and hence, $\textsc{ren}(\textsc{cap}(t_i))$ is an instance of $\textsc{ren}(\textsc{cap}(t\mu_i))$ which in turn is an instance of $\textsc{ren}(\textsc{cap}(t))$. Thus, by extending σ to the variables in $\textsc{ren}(\textsc{cap}(t))$ in an appropriate way, we also obtain $\textsc{ren}(\textsc{cap}(t))\sigma = v\sigma$.
  Now we show that if a reduction pair $(\succsim, \succ)$ and an argument filtering π satisfy all constraints for a cycle $\mathcal{P} = \mathcal{P}' \setminus \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\} \cup \{s \to t\}$, then they also satisfy the constraints for the cycle $\mathcal{P}'$. These constraints only differ in that $s \succsim_\pi t$ resp. $s \succ_\pi t$ is replaced by $s\mu_i \succsim_\pi t_i$ resp. $s\mu_i \succ_\pi t_i$. The constraints of Type (b) are the same. Note that if $\mathcal{P}$ and $\mathcal{P}'$ contain a pair $F(\ldots) \to \ldots$, then for every function symbol g occurring below the root of right-hand sides in pairs of $\mathcal{P}$ or $\mathcal{P}'$, we have $g \leq_d f$. Then $s \succsim_\pi t$ implies $s\mu_i \succsim_\pi t\mu_i \succsim_\pi t_i$ by stability of $\succsim_\pi$ and by the fact that $t\mu_i$ rewrites to t_i

[4] The converse direction (i.e., if there is an infinite (innermost) chain of pairs from $DP(\mathcal{R})'$ then there is also an infinite (innermost) chain of pairs from $DP(\mathcal{R})$) holds as well for rewriting, instantiation, and t-narrowing. For i-narrowing, this direction only holds if the usable rules are non-overlapping (cf. [2, Ex. 43] and [12, Thm. 17]).

using a g-rule for a function symbol $g \leq_d f$. Hence, the constraints of Type (b) imply that all g-rules are weakly decreasing. Similarly, $s \succ_\pi t$ implies $s\mu_i \succ_\pi t\mu_i \succsim_\pi t_i$.

Now we prove completeness of i-narrowing. As for t-narrowing, we first show that if $\mathcal{P}'$ is a cycle of the estimated innermost dependency graph of $DP(\mathcal{R})'$ containing some pair $s\mu_i \to t_i$, then $\mathcal{P} = \mathcal{P}' \setminus \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\} \cup \{s \to t\}$ is a cycle in the estimated innermost dependency graph of $DP(\mathcal{R})$. As in the termination case, one can show that arcs from $v \to w$ to $s\mu_i \to t_i$ correspond to arcs from $v \to w$ to $s \to t$ in the estimated innermost dependency graph of $DP(\mathcal{R})$. Similarly, if there is an arc from $s\mu_i \to t_i$ to $v \to w$ (i.e., $\text{CAP}_{s\mu_i}(t_i)\sigma = v\sigma$), then there is also an arc from $s \to t$ to $v \to w$ in the estimated dependency graph of $DP(\mathcal{R})$ using a similar substitution σ. The reason is again that $\text{CAP}_{s\mu_i}(t_i)$ is an instance of $\text{CAP}_{s\mu_i}(t\mu_i)$ which in turn is an instance of $\text{CAP}_s(t)$. To see this, recall that $t\mu_i$ rewrites to t_i. Thus, the subterm of $t\mu_i$ that is the redex in this reduction cannot occur in $s\mu_i$, since $s\mu_i$ must be a normal form. Hence, in $\text{CAP}_{s\mu_i}(t\mu_i)$, this subterm (or a subterm containing this redex) is replaced by a fresh variable and thus, $\text{CAP}_{s\mu_i}(t_i)$ is an instance of $\text{CAP}_{s\mu_i}(t\mu_i)$. If a subterm of t occurs also in s, then the corresponding subterm of $t\mu_i$ also occurs in $s\mu_i$. In contrast, there may subterms of $t\mu_i$ that occur in $s\mu_i$, whereas no corresponding subterm of t occurs in s. This indicates that $\text{CAP}_{s\mu_i}(t\mu_i)$ is an instance of $\text{CAP}_s(t)$.

Next we show that if a reduction pair $(\succsim, \succ)$ and an argument filtering π satisfy all constraints for a cycle $\mathcal{P} = \mathcal{P}' \setminus \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\} \cup \{s \to t\}$, then they also satisfy the constraints for the cycle $\mathcal{P}'$. One difference between these constraints is that $s \succsim_\pi t$ resp. $s \succ_\pi t$ is replaced by $s\mu_i \succsim_\pi t_i$ resp. $s\mu_i \succ_\pi t_i$. Note that $s \succsim_\pi t$ again implies $s\mu_i \succsim_\pi t\mu_i \succsim_\pi t_i$ by stability of $\succsim_\pi$ and by the fact that $t\mu_i$ rewrites to t_i. Here, $t\mu_i \succsim t_i$ follows by Lemma 12 (iii), since all rules in $\mathcal{U}_{s\mu_i}(t\mu_i, \pi) \subseteq \mathcal{U}_s(t, \pi)$ are weakly decreasing (cf. Lemma 12 (i)). Similarly, $s \succ_\pi t$ implies $s\mu_i \succ_\pi t_i$.

The other difference is in the set of usable rules. But we have $\mathcal{U}_{s\mu_i}(t_i, \pi) \subseteq \mathcal{U}_{s\mu_i}(t\mu_i, \pi) \subseteq \mathcal{U}_s(t, \pi)$ by Lemma 12 (ii) and (i). Therefore, we obtain $\mathcal{U}(\mathcal{P}', \pi) \subseteq \mathcal{U}(\mathcal{P}, \pi)$.

- *rewriting*: We assume that $DP(\mathcal{R})'$ is the result of rewriting a dependency pair $s \to t$ from $DP(\mathcal{R})$ to $s \to t'$ (i.e., t rewrites to t' at some position p). For soundness of our refined version of rewriting we adapt the proof of [12, Thm. 18]. Let $\ldots, s \to t, v \to w, \ldots$ be an innermost chain of pairs from $DP(\mathcal{R})$. Hence, there exists a substitution σ with $t\sigma \xrightarrow{\text{i}}^*_{\mathcal{R}} v\sigma$ and $s\sigma, v\sigma$ are normal forms. Thus, $t\sigma$ is weakly innermost terminating. Due to the innermost reduction strategy, we can split up the reduction of $t\sigma$ into two parts. First, we reduce only on positions on or below p until $t|_p\sigma$ is a normal form u. Afterwards we perform the remaining reduction steps from $t\sigma[u]_p$ to $v\sigma$. The only rules applicable to $t|_p\sigma$ are $\mathcal{U}(t|_p\sigma)$ and as $\mathcal{U}(t|_p\sigma)$ is non-overlapping, by [15, Thm. 3.2.11 (1a) and (4a)], $t|_p\sigma$ is confluent and terminating. With

$t|_p \to_{\mathcal{R}} t'|_p$ we obtain $t|_p\sigma \to_{\mathcal{R}} t'|_p\sigma$. Hence, $t'|_p\sigma$ is terminating as well and thus, it also reduces innermost to the same normal form u using the confluence of $t|_p\sigma$. So we have $t'\sigma = t\sigma[t'|_p\sigma]_p \xrightarrow{\mathsf{i}}^*_{\mathcal{R}} t\sigma[u]_p$. Afterwards, we can apply the same remaining steps as above that lead from $t\sigma[u]_p$ to $v\sigma$. Therefore $\ldots, s \to t', v \to w, \ldots$ is an innermost chain as well. The proof for the completeness of rewriting is analogous to the proof of completeness of i-narrowing.

- *instantiation*: Soundness of instantiation is proved in [12, Thm. 20] (the soundness of our refined version of i-instantiation is again due to the fact that in innermost chains one only regards substitutions which instantiate all left-hand sides of dependency pairs to normal forms). The completeness proofs are analogous to the completeness proofs for t-narrowing and i-narrowing, respectively. □

By Thm. 16, these transformations never complicate (innermost) termination proofs (but they may increase the number of constraints by producing similar constraints that can be solved by the same argument filterings and reduction pairs). So sometimes the runtime is increased by these transformations. On the other hand, the transformations are often crucial for the success of the proof.

Example 17 In the following TRS [3], the minus-rules of Ex. 1 are extended with

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

When trying to prove innermost termination, no simplification order satisfies the constraints of Thm. 13 for the following cycle.

$$\mathsf{QUOT}(x, \mathsf{s}(y)) \to \mathsf{IF}(\mathsf{le}(\mathsf{s}(y), x), x, \mathsf{s}(y)) \tag{32}$$

$$\mathsf{IF}(\mathsf{true}, x, y) \to \mathsf{QUOT}(\mathsf{minus}(x, y), y) \tag{33}$$

The reason is that from the dependency pair constraints of this cycle we obtain

$$\begin{aligned}
\pi(\mathsf{IF}(\mathsf{true}, x, \mathsf{s}(y))) &\succsim\\
\pi(\mathsf{QUOT}(\mathsf{minus}(x, \mathsf{s}(y)), \mathsf{s}(y))) &\succsim\\
\pi(\mathsf{IF}(\mathsf{le}(\mathsf{s}(y), \mathsf{minus}(x, \mathsf{s}(y))), \mathsf{minus}(x, \mathsf{s}(y)), \mathsf{s}(y)))&
\end{aligned}$$

where one of the constraints has to be strict. Hence, we have

$$\pi(\mathsf{IF}(\mathsf{true}, x, \mathsf{s}(y))) \succ \pi(\mathsf{IF}(\mathsf{le}(\mathsf{s}(y), \mathsf{minus}(x, \mathsf{s}(y))), \mathsf{minus}(x, \mathsf{s}(y)), \mathsf{s}(y)))$$

From the minus-rules we see that an argument filtering π must not drop the first argument of minus. Hence, by the subterm property we get $\pi(\mathsf{minus}(x,y)) \succsim \pi(x)$. This leads to

$$\pi(\mathsf{IF}(\mathsf{true}, x, \mathsf{s}(y))) \succ \pi(\mathsf{IF}(\mathsf{le}(\mathsf{s}(y), x), x, \mathsf{s}(y))). \tag{34}$$

In order to obtain a contradiction we first show the following property.

$$\pi(\mathsf{le}(\mathsf{s}(\mathsf{true}), \mathsf{s}(\mathsf{true}))) \succsim \pi(\mathsf{true}) \tag{35}$$

If $\pi(\mathsf{le}) = [\,]$, then using the first le-rule we can directly conclude that (35) holds. Otherwise, by the last le-rule we get $\pi(\mathsf{le}(\mathsf{s}(\mathsf{true}), \mathsf{s}(\mathsf{true}))) \succsim \pi(\mathsf{le}(\mathsf{true}, \mathsf{true}))$ and $\pi(\mathsf{le}(\mathsf{true}, \mathsf{true})) \succsim \pi(\mathsf{true})$ by the subterm property.

Now, using (34), (35), and the substitution $\{x/\mathsf{s}(\mathsf{true}), y/\mathsf{true}\}$ we obtain the desired contradiction.

$$\begin{aligned} \pi(\mathsf{IF}(\mathsf{true}, \mathsf{s}(\mathsf{true}), \mathsf{s}(\mathsf{true}))) &\succ \\ \pi(\mathsf{IF}(\mathsf{le}(\mathsf{s}(\mathsf{true}), \mathsf{s}(\mathsf{true})), \mathsf{s}(\mathsf{true}), \mathsf{s}(\mathsf{true}))) &\succsim \\ \pi(\mathsf{IF}(\mathsf{true}, \mathsf{s}(\mathsf{true}), \mathsf{s}(\mathsf{true}))).& \end{aligned}$$

On the other hand, when transforming the dependency pairs, the resulting constraints can easily be satisfied by simplification orders. Intuitively, $x \succ \mathsf{minus}(x,y)$ only has to be satisfied if $\mathsf{le}(\mathsf{s}(y), x)$ reduces to true. This argumentation can be simulated using the transformations of Def. 15. By i-narrowing, we perform a case analysis on how the le-term in (32) can be evaluated. In the first narrowing, x is instantiated by $\mathsf{0}$. This results in a pair $\mathsf{QUOT}(\mathsf{0}, \mathsf{s}(y)) \to \mathsf{IF}(\mathsf{false}, \mathsf{0}, \mathsf{s}(y))$ which is not in a cycle. The other narrowing is

$$\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{IF}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \tag{36}$$

which forms a new cycle with (33). Now we perform i-instantiation of (33) and see that x and y must be of the form $\mathsf{s}(\ldots)$. So (33) is replaced by the new pair

$$\mathsf{IF}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{QUOT}(\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y)) \tag{37}$$

that forms a cycle with (36). Finally, we do a rewrite step on (37) and obtain

$$\mathsf{IF}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y)) \tag{38}$$

The constraints from the resulting cycle $\{(36), (38)\}$ (and from all other cycles) can be solved by $\pi(\mathsf{minus}) = \pi(\mathsf{QUOT}) = 1$, $\pi(\mathsf{IF}) = 2$, and the embedding order.

The crucial problem with the refinement of Def. 15 is that these transformations may be applied infinitely many times. Therefore, we have developed restricted *safe* transformations which are guaranteed to terminate. Our experiments on the collections of examples from [3, 9, 27] show that whenever the proof succeeds using narrowing, rewriting, and instantiation, then applying these safe transformations is sufficient.

A narrowing or instantiation step is *safe* if it reduces the number of pairs in cycles of the estimated (innermost) dependency graph. For a set of pairs $\mathcal{P}$, $\mathrm{SCC}(\mathcal{P})$ denotes the set of maximal cycles built from pairs of $\mathcal{P}$. Then, the transformation is safe if $\Sigma_{\mathcal{S}\in \mathrm{SCC}(\mathcal{P})}|\mathcal{S}|$ decreases. Moreover, it is also considered safe if by the transformation step, all descendants of some original dependency pair disappear from cycles. For every pair $s \to t$, $o(s \to t)$ denotes the original dependency pair whose repeated transformation led to $s \to t$. Now a transformation is also safe if $\{o(s \to t) \mid s \to t \in \bigcup_{\mathcal{S}\in \mathrm{SCC}(\mathcal{P})} \mathcal{S}\}$ decreases.

As an example, consider $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{g}(\mathsf{b}), \mathsf{g}(x) \to \mathsf{f}(x)\}$. The estimated dependency graph has the cycle $\{\mathsf{F}(\mathsf{a}) \to \mathsf{G}(\mathsf{b}), \mathsf{G}(x) \to \mathsf{F}(x)\}$. Instantiation transforms the second pair into $\mathsf{G}(\mathsf{b}) \to \mathsf{F}(\mathsf{b})$. Now there is no cycle anymore, since $\mathcal{F}(\mathsf{b})$ does not unify with $\mathcal{F}(\mathsf{a})$. Thus, this instantiation step is safe. Finally for each pair, one single narrowing and instantiation step which does not satisfy the above requirements is also considered safe. Hence, the narrowing and instantiation steps in Ex. 17 were safe as well.

As for termination, in innermost termination proofs we also benefit from considering the recursion hierarchy. So if $\mathcal{R}_1, \ldots, \mathcal{R}_n$ is a separation of the TRS $\mathcal{R}$ and $\mathcal{R}_i >_d \mathcal{R}_j$, then we show absence of innermost $\mathcal{R}$-chains built from $DP(\mathcal{R}_j)$ before dealing with $DP(\mathcal{R}_i)$. Now innermost rewriting a dependency pair $F(\ldots) \to \ldots$ is *safe* if it is performed with rules that do not depend on f (i.e., with g-rules where $g <_d f$). The reason is that innermost termination of g is already verified when proving innermost termination of f. So in Ex. 17, when proving innermost termination of the QUOT-cycle, we may assume innermost termination of minus and thus, the rewrite step from (37) to (38) was safe.

Definition 18 (Safe Transformations) *Let $\mathcal{Q}$ result from a set $\mathcal{P}$ of pairs of terms by transforming $s \to t \in \mathcal{P}$ as in Def. 15. The transformation is* safe *if*

(1) $s \to t$ was transformed by narrowing or instantiation and

- $\Sigma_{\mathcal{S}\in \mathrm{SCC}(\mathcal{P})}|\mathcal{S}| > \Sigma_{\mathcal{S}\in \mathrm{SCC}(\mathcal{Q})}|\mathcal{S}|$*, or*
- $\{o(s \to t) \mid s \to t \in \bigcup_{\mathcal{S}\in \mathrm{SCC}(\mathcal{P})} \mathcal{S}\} \supsetneq \{o(s \to t) \mid s \to t \in \bigcup_{\mathcal{S}\in \mathrm{SCC}(\mathcal{Q})} \mathcal{S}\}$

(2) $s \to t$ was transformed by innermost rewriting with the rule $l \to r$ and $\mathrm{root}(l) <_d f$ where $f^\sharp = \mathrm{root}(s)$

(3) $s \to t$ was transformed by narrowing and all previous steps which transformed $o(s \to t)$ to $s \to t$ were not narrowing steps

(4) $s \to t$ was transformed by instantiation and all previous steps which transformed $o(s \to t)$ to $s \to t$ were not instantiation steps

The following theorem proves that the repeated application of safe transformations is indeed terminating.

Theorem 19 (Termination) *Let $\mathcal{R}$ have the separation $\mathcal{R}_1, \ldots, \mathcal{R}_n$ and $\mathcal{P} \subseteq DP(\mathcal{R}_i)$. If there are no infinite innermost $\mathcal{R}$-chains from $DP(\mathcal{R}_j)$ for all $\mathcal{R}_j <_d \mathcal{R}_i$, then any repeated application of safe transformations on $\mathcal{P}$ terminates.*

Proof. We define a measure on sets of pairs $\mathcal{P}$ consisting of four components:

$$(a)\ \ |\{\, o(s \to t) \mid s \to t \in \bigcup_{\mathcal{S} \in \mathrm{SCC}(\mathcal{P})} \mathcal{S}\}| \qquad\qquad (c)\ \ |\mathcal{P}|$$
$$(b)\ \ \Sigma_{\mathcal{S} \in \mathrm{SCC}(\mathcal{P})} |\mathcal{S}| \qquad\qquad (d)\ \ \mathcal{P}$$

These 4-tuples are compared lexicographically by the usual order on naturals for components (a)-(c). For (d), we use the (multi)set extension of the innermost rewrite relation of $\bigcup_{\mathcal{R}_j <_d \mathcal{R}_i} \mathcal{R}_j$. Thus, we obtain a well-founded relation $\succ$ where $\mathcal{P}_1 \succ \mathcal{P}_2$ iff $\mathcal{P}_1$'s measure is greater than the measure of $\mathcal{P}_2$. Due to (a), (b), and (d), any safe transformation of $\mathcal{P}$ with steps (1) or (2) decreases the measure of $\mathcal{P}$.

For a set of pairs $\mathcal{P}$, let $w(\mathcal{P}) = \langle \mathcal{P}_{\neg n,\neg i}, \mathcal{P}_{n,\neg i}, \mathcal{P}_{\neg n,i}, \mathcal{P}_{n,i}\rangle$. $\mathcal{P}_{\neg n,\neg i}$ consists of those $s \to t \in \mathcal{P}$ where no (n)arrowing or (i)nstantiation was used to transform $o(s \to t)$ to $s \to t$. $\mathcal{P}_{n,\neg i}$ are the pairs where narrowing, but no instantiation was used, etc. Every safe transformation step decreases $w(\mathcal{P})$ lexicographically w.r.t. $\succ$: Transformations with (1) or (2) decrease one component of $w(\mathcal{P})$ w.r.t. $\succ$ and do not modify the others. Transformations with (3) or (4) reduce the size of one component of $w(\mathcal{P})$ (so the component decreases w.r.t. $\succ$ according to (c)) and increase the size of some component on its right-hand side. $\square$

After each transformation, the current cycle or SCC of the estimated (innermost) dependency graph is re-computed. For this re-computation, one only has to regard the former neighbors of the transformed pair in the old graph. Only former neighbors may have arcs to or from the new pairs resulting from the transformation. Regarding neighbors in the graphs also suffices when performing the unifications required for narrowing and instantiation. In this way, the transformations can be performed efficiently. Recall that one should always regard SCCs first and afterwards, one builds new SCCs from the remaining pairs which were not strictly decreasing (Sect. 2) [16]. Of course, these pairs may already have been transformed during the (innermost) termination proof of the SCC. So this approach has the advantage that one never repeats transformations for the same dependency pairs.

6 Computing Argument Filterings

In the dependency pair approach we may apply an argument filtering π to a set of constraints before starting an orientation attempt with a reduction pair. However, the number of possible argument filterings is exponential in the arities of the function symbols. We now show how to search for suitable argument filterings efficiently in the improved dependency pair approach of Thm. 9 and Thm. 13. More precisely, for every cycle $\mathcal{P}$, we show how to compute small subsets $\Pi^t(\mathcal{P})$ and $\Pi^i(\mathcal{P})$ of argument filterings which contain all filterings which could possibly satisfy the constraints for termination or innermost termination, respectively. A corresponding algorithm was presented in [16] for termination proofs w.r.t. Thm. 8. However, we now develop such an algorithm for the improved versions of the dependency pair approach from Thm. 9 and Thm. 13. In particular for innermost termination (Thm. 13), the algorithm is considerably more involved since the set of constraints depends on the argument filtering used. Moreover, instead of treating constraints separately as in [16], we process them according to an efficient depth-first strategy.

Let $\mathcal{RP}$ be a class of reduction pairs describing the particular base order used (e.g., $\mathcal{RP}$ may contain all LPOs with arbitrary precedences or all recursive path orders with status, etc.). For any set of dependency pairs $\mathcal{P}$, $\Pi(\mathcal{P})$ denotes the set of all argument filterings where at least one dependency pair in $\mathcal{P}$ is strictly decreasing and the remaining ones are weakly decreasing w.r.t. some reduction pair in $\mathcal{RP}$. When referring to "dependency pairs", we also permit pairs resulting from dependency pairs by narrowing, rewriting, or instantiation.

We use the approach of [16] to consider partial argument filterings, i.e., filterings which are only defined on a subset of the signature. For example, in a term $f(g(x), y)$, if $\pi(f) = [2]$, then we do not have to determine $\pi(g)$, since all occurrences of g are filtered away. Thus, we leave argument filterings as undefined as possible and permit the application of π to a term t whenever π is *sufficiently defined* for t. More precisely, any partial argument filtering π is sufficiently defined for a variable x. So the the domain of π may even be empty, i.e., $DOM(\pi) = \varnothing$. An argument filtering π is sufficiently defined for a term $f(t_1, \ldots, t_n)$ iff $f \in DOM(\pi)$ and π is sufficiently defined for all t_i with $i \in \mathcal{R}egPos_\pi(f)$. An argument filtering is sufficiently defined for a set of terms T iff it is sufficiently defined for all terms in T. To compare argument filterings which only differ in their domain DOM, we introduce a relation "$\sqsubseteq$". Then $\Pi(\mathcal{P})$ should only contain $\sqsubseteq$-minimal elements, i.e., if $\pi' \in \Pi(\mathcal{P})$, then $\Pi(\mathcal{P})$ does not contain any $\pi \sqsubset \pi'$. Of course, all argument filterings in $\Pi(\mathcal{P})$ must be sufficiently defined for the terms in the dependency pairs of $\mathcal{P}$.

Definition 20 ($\sqsubseteq$ and $\Pi(\mathcal{P})$) *For two (partial) argument filterings, we define $\pi \sqsubseteq \pi'$ iff $DOM(\pi) \subseteq DOM(\pi')$ and $\pi(f) = \pi'(f)$ for all $f \in DOM(\pi)$. For a set $\mathcal{P}$ of dependency pairs, let $\Pi(\mathcal{P})$ consist of all $\sqsubseteq$-minimal elements of $\{\pi \mid$ there is a $(\succsim, \succ) \in \mathcal{RP}$ such that $\pi(s) \succ \pi(t)$ for at least one $s \to t \in \mathcal{P}$ and $\pi(s) \succsim \pi(t)$ for all other $s \to t \in \mathcal{P}\}$.*

We now define a superset $\Pi^t(\mathcal{P})$ of all argument filterings where the constraints (a) and (b) for termination of the cycle $\mathcal{P}$ are satisfied by some reduction pair of $\mathcal{RP}$. So only these argument filterings have to be regarded when automating Thm. 9. To this end, we have to *extend* partial argument filterings.

Definition 21 (Ex_f, $\Pi^t(\mathcal{P})$) *For a partial argument filtering π and $f \in \mathcal{D}$, $Ex_f(\pi)$ consists of all $\sqsubseteq$-minimal argument filterings π' such that $\pi \sqsubseteq \pi'$ and such that there is a $(\succsim, \succ) \in \mathcal{RP}$ with $\pi'(l) \succsim \pi'(r)$ for all $l \to r \in Rls(f)$. For a set Π of filterings, let $Ex_f(\Pi) = \bigcup_{\pi \in \Pi} Ex_f(\pi)$. If $\mathcal{P}$ originates from $DP(\mathcal{R}_i)$ by t-narrowing and t-instantiation and $\{f_1, ..., f_k\}$ are $\mathcal{R}'_i$'s defined symbols, then $\Pi^t(\mathcal{P}) = Ex_{f_k}(...Ex_{f_1}(\Pi(\mathcal{P}))...)$.*

We compute $\Pi^t(\mathcal{P})$ by depth-first search. So we start with some $\pi \in \Pi(\mathcal{P})$ and extend it to a minimal π' such that the f_1-rules are weakly decreasing. Then π' is extended such that the f_2-rules are weakly decreasing, etc. Here, f_1 is considered before f_2 if $f_1 >_d f_2$. When we have $\Pi^t(\mathcal{P})$'s first element π_1, we check whether Constraints (a) and (b) of Thm. 9 are satisfiable with π_1. In case of success, we do not compute further elements of $\Pi^t(\mathcal{P})$. Only if the constraints are not satisfiable with π_1, we determine $\Pi^t(\mathcal{P})$'s next element, etc. The advantage of this approach is that $\Pi(\mathcal{P})$ is usually rather small, since it only contains argument filterings that satisfy a *strict* inequality.

For innermost termination, the set of constraints to be satisfied depends on the argument filtering used. If $f \geq_d g$, then when orienting the rules of f, we do not necessarily have to orient the rules of g as well, since all occurrences of g in f-rules may have been deleted by the argument filtering, cf. Thm. 13. To formalize this, we define a relation "$\vdash_{\mathcal{P}}$" on sets of argument filterings. Let us extend $\mathcal{R}egPos_\pi$ to *partial* argument filterings by defining $\mathcal{R}egPos_\pi(f) = \varnothing$ for all $f \notin DOM(\pi)$. Now $\mathcal{U}(\mathcal{P}, \pi)$ is also defined for partial filterings by simply disregarding all subterms of function symbols where π is not defined.

For a partial argument filtering π, whenever $Rls(f)$ is included in the usable rules $\mathcal{U}(\mathcal{P}, \pi)$ for the cycle $\mathcal{P}$, then the relation "$\vdash_{\mathcal{P}}$" can extend π in order to make the f-rules weakly decreasing. We label each argument filtering by the set of those function symbols whose rules are already guaranteed to be weakly decreasing.

Definition 22 ($\vdash_{\mathcal{P}}$) *Each argument filtering π is labelled with a set $\mathcal{G} \subseteq \mathcal{D}$ and we denote a labelled argument filtering by $\pi_{\mathcal{G}}$. For sets of labelled argument filterings, we define the relation "$\vdash_{\mathcal{P}}$": $\Pi \uplus \{\pi_{\mathcal{G}}\} \vdash_{\mathcal{P}} \Pi \cup \{\pi'_{\mathcal{G}\cup\{f\}} \mid \pi' \in Ex_f(\pi)\}$, if $f \in \mathcal{D} \setminus \mathcal{G}$ and $Rls(f) \subseteq \mathcal{U}(\mathcal{P}, \pi)$.*

When proving innermost termination, we will only regard argument filterings that result from $\Pi(\mathcal{P})$ by applying $\vdash_{\mathcal{P}}$-reductions as long as possible. In order to prove that normal forms w.r.t. $\vdash_{\mathcal{P}}$ are unique, we need the following lemma. It states that $Ex_f(\pi)$ always consists of pairwise incompatible argument filterings. Here, two argument filterings π_1 and π_2 are *compatible* if $\pi_1(f) = \pi_2(f)$ for all $f \in DOM(\pi_1) \cap DOM(\pi_2)$, cf. [16].

Lemma 23 (Incompatibility) *Let T be a finite set of terms.*

(a) Let π, π_1, π_2 be (partial) argument filterings. Let π_1, π_2 be elements of $\{\pi' \mid \pi \sqsubseteq \pi'$ and π' is sufficiently defined for $T\}$, where π_1 is a $\sqsubseteq$-minimal element of this set. If π_1 and π_2 are compatible, then $\pi_1 \sqsubseteq \pi_2$.
(b) If $\pi_1, \pi_2 \in Ex_f(\pi)$ with $\pi_1 \neq \pi_2$, then π_1 and π_2 are incompatible, i.e., $Ex_f(\pi)$ consists of pairwise incompatible argument filterings.
(c) If Π consists of pairwise incompatible argument filterings, then $Ex_f(\Pi)$ consists of pairwise incompatible argument filterings as well.

Proof. (a) We perform induction on T using the multiset version of the proper subterm relation. If $T = \varnothing$, then the only minimal extension of π that is sufficiently defined for T is $\pi_1 = \pi$. Hence, $\pi_1 = \pi \sqsubseteq \pi_2$.

Next let $T = T' \uplus \{x\}$ for a variable x. Clearly, both π_1 and π_2 are also sufficiently defined for T' and moreover, π_1 is a minimal extension of π that is sufficiently defined for T'. Thus, the claim follows from the induction hypothesis.

If $T = T' \uplus \{f(t_1, \ldots, t_n)\}$, then $f \in \mathit{DOM}(\pi_1)$. Let $T'' = T' \cup \{t_i \,|\, i \in \mathcal{R}egPos_{\pi_1}(f)\}$. Both π_1 and π_2 are sufficiently defined for T'' (for π_2 this follows from $\pi_2(f) = \pi_1(f)$ by compatibility of π_1 and π_2). If π_1 is a minimal extension of π that is sufficiently defined for T'', then the claim is implied by the induction hypothesis. Otherwise, we have $f \notin \mathit{DOM}(\pi)$ and we obtain the following minimal extension π_1' of π that is sufficiently defined for T'': $\mathit{DOM}(\pi_1') = \mathit{DOM}(\pi_1) \setminus \{f\}$ and $\pi_1'(g) = \pi_1(g)$ for all $g \in \mathit{DOM}(\pi_1')$. Then the induction hypothesis implies $\pi_1' \sqsubseteq \pi_2$. Since π_1 only differs from π_1' on the function symbol f and since $\pi_1(f) = \pi_2(f)$, we obtain $\pi_1 \sqsubseteq \pi_2$.

(b) Let $\pi_1, \pi_2 \in \mathit{Ex}_f(\pi)$ be compatible. As both filterings are minimal extensions of π that are sufficiently defined for the terms on left- or right-hand sides of rules from $\mathit{Rls}(f)$, we use (a) to conclude both $\pi_1 \sqsubseteq \pi_2$ and $\pi_2 \sqsubseteq \pi_1$, which implies $\pi_1 = \pi_2$.

(c) Let $\pi_1 \in \mathit{Ex}_f(\pi_1')$ and $\pi_2 \in \mathit{Ex}_f(\pi_2')$, where $\pi_1', \pi_2' \in \mathit{\Pi}$. If $\pi_1' = \pi_2'$, then π_1 and π_2 are incompatible by (b). Otherwise $\pi_1' \neq \pi_2'$, and by the assumption about $\mathit{\Pi}$ we obtain that π_1' and π_2' are incompatible. As $\pi_1' \sqsubseteq \pi_1$ and $\pi_2' \sqsubseteq \pi_2$, this implies that π_1 and π_2 are incompatible as well. $\square$

The next theorem shows the desired properties of the relation $\vdash_{\mathcal{P}}$.

Theorem 24 *$\vdash_{\mathcal{P}}$ is terminating and confluent.*

Proof. The termination of $\vdash_{\mathcal{P}}$ is obvious as the labellings increase in every $\vdash_{\mathcal{P}}$-step. Hence for confluence, it suffices to show local confluence. The only crucial non-determinism in the definition of $\vdash_{\mathcal{P}}$ is the choice of f. Suppose that $f_0, f_1 \in \mathcal{D} \setminus \mathcal{G}$ with $f_0 \neq f_1$ and $\mathit{Rls}(f_0) \cup \mathit{Rls}(f_1) \subseteq \mathcal{U}(\mathcal{P}, \pi)$. This leads to two possible reduction steps

$$\mathit{\Pi} \uplus \{\pi_{\mathcal{G}}\} \vdash_{\mathcal{P}} \mathit{\Pi} \cup \mathit{\Pi}_0, \quad \text{where } \mathit{\Pi}_0 = \{\pi^0_{\mathcal{G} \cup \{f_0\}} \mid \pi^0 \in \mathit{Ex}_{f_0}(\pi)\}$$
$$\mathit{\Pi} \uplus \{\pi_{\mathcal{G}}\} \vdash_{\mathcal{P}} \mathit{\Pi} \cup \mathit{\Pi}_1, \quad \text{where } \mathit{\Pi}_1 = \{\pi^1_{\mathcal{G} \cup \{f_1\}} \mid \pi^1 \in \mathit{Ex}_{f_1}(\pi)\}$$

Note that $\mathcal{U}(\mathcal{P}, \pi) \subseteq \mathcal{U}(\mathcal{P}, \pi^i)$ holds for all $\pi^i \in \mathit{Ex}_{f_i}(\pi)$. Thus for all filterings $\pi^i_{\mathcal{G} \cup \{f_i\}} \in \mathit{\Pi}_i$, we have $f_{1-i} \in \mathcal{D} \setminus (\mathcal{G} \cup \{f_i\})$ and $\mathit{Rls}(f_{1-i}) \subseteq \mathcal{U}(\mathcal{P}, \pi^i)$. Hence, we can build the following reductions (where we also allow the application of Ex_f

to labelled argument filterings by simply ignoring their labels).

$$
\begin{array}{llll}
\Pi \cup \Pi_0 & \overset{|\Pi_0|}{\vdash}_{\mathcal{P}} & (\Pi \setminus \Pi_0) & \cup \left\{ \pi'_{\mathcal{G}\cup\{f_0,f_1\}} \mid \pi' \in Ex_{f_1}(Ex_{f_0}(\pi)) \right\} \\
 & \overset{|\Pi\cap\Pi_1|}{\vdash}_{\mathcal{P}} & (\Pi \setminus (\Pi_0 \cup \Pi_1)) & \cup \left\{ \pi'_{\mathcal{G}\cup\{f_0,f_1\}} \mid \pi' \in Ex_{f_1}(Ex_{f_0}(\pi)) \right\} \\
 & & & \cup \left\{ \pi'_{\mathcal{G}\cup\{f_1,f_0\}} \mid \pi' \in Ex_{f_0}(\Pi \cap \Pi_1) \right\}
\end{array}
$$

and

$$
\begin{array}{llll}
\Pi \cup \Pi_1 & \overset{|\Pi_1|}{\vdash}_{\mathcal{P}} & (\Pi \setminus \Pi_1) & \cup \left\{ \pi'_{\mathcal{G}\cup\{f_1,f_0\}} \mid \pi' \in Ex_{f_0}(Ex_{f_1}(\pi)) \right\} \\
 & \overset{|\Pi\cap\Pi_0|}{\vdash}_{\mathcal{P}} & (\Pi \setminus (\Pi_1 \cup \Pi_0)) & \cup \left\{ \pi'_{\mathcal{G}\cup\{f_1,f_0\}} \mid \pi' \in Ex_{f_0}(Ex_{f_1}(\pi)) \right\} \\
 & & & \cup \left\{ \pi'_{\mathcal{G}\cup\{f_0,f_1\}} \mid \pi' \in Ex_{f_1}(\Pi \cap \Pi_0) \right\}
\end{array}
$$

where $Ex_{f_0}(\Pi\cap\Pi_1) \subseteq Ex_{f_0}(Ex_{f_1}(\pi))$ and $Ex_{f_1}(\Pi\cap\Pi_0) \subseteq Ex_{f_1}(Ex_{f_0}(\pi))$. Hence, the missing step to finish this proof is to show $Ex_{f_0}(Ex_{f_1}(\pi)) = Ex_{f_1}(Ex_{f_0}(\pi))$. Because of symmetry, it suffices to prove $Ex_{f_0}(Ex_{f_1}(\pi)) \subseteq Ex_{f_1}(Ex_{f_0}(\pi))$. To this end, we only have to show that for every $\pi_{01} \in Ex_{f_0}(Ex_{f_1}(\pi))$ there exists a $\pi_{10} \in Ex_{f_1}(Ex_{f_0}(\pi))$ with $\pi_{10} \sqsubseteq \pi_{01}$. The reason is that in an analogous way one can show that for π_{10} there also exists a $\pi'_{01} \in Ex_{f_0}(Ex_{f_1}(\pi))$ with $\pi'_{01} \sqsubseteq \pi_{10}$. Hence, we have $\pi'_{01} \sqsubseteq \pi_{10} \sqsubseteq \pi_{01}$ and by Lemma 23 (b) and (c), this implies $\pi'_{01} = \pi_{10} = \pi_{01}$.

Let $\pi_{01} \in Ex_{f_0}(Ex_{f_1}(\pi))$. By the definition of Ex, there must be a $\pi_1 \in Ex_{f_1}(\pi)$ and a reduction pair $(\succsim, \succ) \in \mathcal{RP}$ with $\pi_1 \sqsubseteq \pi_{01}$ and $\pi_{01}(l) \succsim \pi_{01}(r)$ for all $l \to r \in Rls(f_0)$. As $\pi_1 \in Ex_{f_1}(\pi)$, we may conclude in the same way that $\pi \sqsubseteq \pi_1$ and $\pi_1(l) \succsim' \pi_1(r)$ for all f_1-rules and some reduction pair $(\succsim', \succ') \in \mathcal{RP}$. Since $\pi \sqsubseteq \pi_{01}$ and since the f_0-rules can be oriented in a weakly decreasing way using π_{01}, there exists a $\pi_0 \in Ex_{f_0}(\pi)$ with $\pi \sqsubseteq \pi_0 \sqsubseteq \pi_{01}$ such that the f_0-rules can also be oriented using π_0. Since $\pi_0 \sqsubseteq \pi_{01}$ and since the f_1-rules can be oriented with π_{01}, there is a $\pi_{10} \in Ex_{f_1}(\pi_0)$ with $\pi_0 \sqsubseteq \pi_{10} \sqsubseteq \pi_{01}$ such that π_{10} also permits an orientation of the f_1-rules. As explained above, this suffices to prove $Ex_{f_0}(Ex_{f_1}(\pi)) \subseteq Ex_{f_1}(Ex_{f_0}(\pi))$. □

Now we can define the set of argument filterings that are regarded for innermost termination proofs.

Definition 25 ($\Pi^i(\mathcal{P})$) *Let $Nf_{\vdash_{\mathcal{P}}}(\Pi)$ denote the normal form of Π w.r.t. $\vdash_{\mathcal{P}}$. Then we define $\Pi^i(\mathcal{P}) = Nf_{\vdash_{\mathcal{P}}}(\{\pi_{\varnothing} \mid \pi \in \Pi(\mathcal{P})\})$.*

To compute $\Pi^i(\mathcal{P})$, we again start with some $\pi \in \Pi(\mathcal{P})$. Now π only has to be extended in order to make the rules for a symbol f weakly decreasing if the f-rules are contained in $\mathcal{U}(\mathcal{P}, \pi)$. If by this extension, the rules for some new symbol g become usable, then a subsequent extension with Ex_g is also necessary, etc.

Thm. 26 states that by $\Pi^t(\mathcal{P})$ (resp. $\Pi^i(\mathcal{P})$), one indeed obtains all argument filterings which could possibly solve the dependency pair constraints. Here, $\mathcal{P}$ may also result from narrowing, rewriting, and instantiating dependency pairs. In this way the set of argument filterings is reduced dramatically and thus, efficiency is increased. For example, for a TRS from [3, Ex. 3.11] computing quicksort, $\Pi^t(\mathcal{P})$ reduces the number of argument filterings from more than 26 million to 3734 and with $\Pi^i(\mathcal{P})$ we obtain a reduction from more than 1.4 million to 783.

Theorem 26 *Let $\mathcal{P}$ be a cycle. If the constraints (a) and (b) of Thm. 9 for termination are satisfied for some reduction pair from $\mathcal{RP}$ and argument filtering π, then $\pi' \sqsubseteq \pi$ for some $\pi' \in \Pi^t(\mathcal{P})$. If the constraints (c) and (d) of Thm. 13 for innermost termination are satisfied for some reduction pair from $\mathcal{RP}$ and argument filtering π, then $\pi' \sqsubseteq \pi$ for some $\pi' \in \Pi^i(\mathcal{P})$.*

Proof. Let π be an argument filtering and let $(\succsim, \succ) \in \mathcal{R}P$ be a reduction pair that solve the constraints (a) and (b) from Thm. 9 or the constraints (c) and (d) from Thm. 13 for a cycle $\mathcal{P}$, respectively.

We first consider the termination case. There must be a minimal argument filtering $\pi_0 \in \Pi(\mathcal{P})$ with $\pi_0 \sqsubseteq \pi$ that solves the constraints in (a) using $(\succsim, \succ)$. Let $\mathcal{P}$ originate from $DP(\mathcal{R}_i)$, where $\mathcal{R}'_i$ has the defined symbols $\{f_1, \ldots, f_k\}$. As $\pi_0 \sqsubseteq \pi$ and $\pi(l) \succsim \pi(r)$ for all f_1-rules $l \to r$, there must be a filtering $\pi_1 \in Ex_{f_1}(\pi_0)$ with $\pi_1 \sqsubseteq \pi$. We continue in this way and obtain an argument filtering $\pi_k \in Ex_{f_k}(\ldots Ex_{f_1}(\Pi(\mathcal{P}))\ldots) = \Pi^t(\mathcal{P})$ with $\pi_k \sqsubseteq \pi$.

In the innermost case, let $\Pi(\mathcal{P}) = \Pi_0 \vdash_{\mathcal{P}} \Pi_1 \vdash_{\mathcal{P}} \ldots \vdash_{\mathcal{P}} \Pi_n = \Pi^i(\mathcal{P})$ be a $\vdash_{\mathcal{P}}$-reduction to normal form. We show that for all $0 \leq j \leq n$ there is a $\pi_j \in \Pi_j$ with $\pi_j \sqsubseteq \pi$ by induction on j. For $j = 0$, since π solves the constraints in (c), by definition there is again a minimal argument filtering $\pi_0 \in \Pi(\mathcal{P})$ with $\pi_0 \sqsubseteq \pi$. For $j > 0$, we assume that there is a $\pi_{j-1} \in \Pi_{j-1}$ with $\pi_{j-1} \sqsubseteq \pi$. Thus, we either have $\pi_{j-1} \in \Pi_j$ as well or else, Π_j results from Π_{j-1} by replacing π_{j-1} by all elements of $Ex_f(\pi_{j-1})$ for some f with $Rls(f) \subseteq \mathcal{U}(\mathcal{P}, \pi_{j-1})$. Since $\pi_{j-1} \sqsubseteq \pi$, we have $\mathcal{U}(\mathcal{P}, \pi_{j-1}) \subseteq \mathcal{U}(\mathcal{P}, \pi)$ and thus, π also makes the f-rules weakly decreasing. This implies that there must be a $\pi_j \in Ex_f(\pi_{j-1}) \subseteq \Pi_j$ with $\pi_j \sqsubseteq \pi$. $\square$

The converse directions of this theorem do not hold, since in the computation of $\Pi^t(\mathcal{P})$ and $\Pi^i(\mathcal{P})$, when extending argument filterings, one does not take the orders into account. So even if $Ex_f(Ex_g(\ldots)) \neq \varnothing$, it could be that there is no reduction pair such that both f- and g-rules are weakly decreasing w.r.t. the *same* reduction pair.

The technique of this section can be extended by storing both argument filterings and corresponding parameters of the order in the sets $\Pi(\mathcal{P})$ and $Ex_f(\ldots)$. For example, if $\mathcal{RP}$ is the set of all LPOs, then $\Pi(\mathcal{P})$ would now contain all (minimal) pairs of argument filterings π and precedences such that $\pi(s) \succ_{lpo} \pi(t)$ resp. $\pi(s) \succsim_{lpo} \pi(t)$ holds for $s \to t \in \mathcal{P}$. When extending argument filterings, one would also have to extend the corresponding precedence. Of course, such an extension is only permitted if the extended precedence is still irreflexive (and hence, well founded). Then, $\Pi^t(\mathcal{P})$ (resp. $\Pi^i(\mathcal{P})$) is non-empty iff the constraints for (innermost) termination are satisfiable for $\mathcal{P}$. Thus, after computing $\Pi^t(\mathcal{P})$ resp. $\Pi^i(\mathcal{P})$, no further checking of orders and constraints is necessary anymore. This variant is particularly suitable for orders with few parameters like LPO.

7 Heuristics

Now we present heuristics to improve the efficiency of the approach. They concern the search for argument filterings (Sect. 7.1) and for base orders (Sect. 7.2 and 7.3). In contrast to the improvements of the preceding sections, these heuristics affect the power of the method, i.e., there exist examples whose (innermost) termination can no longer be proved when following the heuristics.

7.1 Type Inference for Argument Filterings

In Sect. 6, we have shown how to reduce the set of possible argument filterings by removing filterings which cannot satisfy the constraints for (innermost) termination using dependency pairs. However, since the resulting sets $\Pi^t(\mathcal{P})$ and $\Pi^i(\mathcal{P})$ can still be large, it is often advantageous to reduce them even further. To this end, we have developed the following heuristic based on *type inference*.

In natural examples, termination of a function is usually due to the decrease of arguments of the *same type*. Of course, this type may be different for the different functions in a TRS. So we use a (monomorphic) type inference algorithm to transform a TRS into a sorted TRS (i.e., a TRS with rules $l \to r$ where l and r are well-typed terms of the same type). As a good heuristic to reduce the set of possible argument filterings further, one can require that for every symbol f, either no argument position is eliminated or all non-eliminated argument positions are of the same type. In other words, if f is n-ary, then $\pi(f) = [1, \ldots, n]$, $\pi(f) \in \{1, \ldots, n\}$, or $\pi(f) = [i_1, \ldots, i_k]$ where the argument positions $i_1, \ldots, i_k$ all have the same type. Our experiments show that all examples in the collections of [3, 9, 27] that can be solved using LPO as a base order can still be solved when using this heuristic.

7.2 Embedding Order for Dependency Pairs

To increase efficiency in our depth-first algorithm of Sect. 6, a successful heuristic is to only use the embedding order when orienting the constraints $\pi(s) \succ \pi(t)$ and $\pi(s) \succsim \pi(t)$ for dependency pairs $s \to t$. Only for constraints of the form $\pi(l) \succsim \pi(r)$ for rules $l \to r$, one may apply more complicated quasi-orders e.g., LPO, RPO(S), or polynomial orders. The advantage of this approach is that now $\Pi(\mathcal{P})$ is much smaller than when using more powerful orders. Thus, the depth-first search starting with $\Pi(\mathcal{P})$ can be performed very quickly. Our experiments show that due to the improvements in Sect. 3 and 4, this heuristic succeeds for more than 96 % of those examples from [3, 9, 27] where a full LPO was successful, while reducing runtimes by at least 58 %.

7.3 Bottom-Up Heuristic

To determine argument filterings in Sect. 6, we start with the dependency pairs and treat the constraints for rules afterwards, where f-rules are considered before g-rules if $f >_d g$. In contrast, now we suggest a bottom-up approach which starts with determining an argument filtering for constructors and then moves upwards through the recursion hierarchy where g is treated before f if $f >_d g$. While in Sect. 6, we determined *sets* of argument filterings, now we only determine one single argument filtering, even if several ones are possible. To obtain an efficient technique, no backtracking takes place, i.e., if at some point one selects the "wrong" argument filtering, then the proof can fail.

More precisely, we first guess an argument filtering π which is only defined for constructors. For every n-ary constructor c we define $\pi(c) = [1, \ldots, n]$ or we let π filter away all argument of c that do not have the same type as c's result. Afterwards, for every function symbol f, we try to extend π on f such that $\pi(l) \succsim \pi(r)$ for all f-rules $l \to r$. We consider functions according to the recursion hierarchy $>_d$. So when extending π on f, π is already defined on all $g <_d f$. Among the extensions of π which permit an orientation of the f-rules, we choose $\pi(f)$ such that it eliminates as many arguments of f as possible. Of course, this is just one of the potential argument filterings for f. If we have chosen the "wrong" argument filtering for f, the (innermost) termination proof might fail, although there would have been a solution with a different argument filtering. If we are not able to orient the rules of f, then we mark f as not orientable. Finally, after having treated all rules, the filtering is extended to the tuple symbols by trying to orient the dependency pairs as well (where at least one dependency pair must be strictly decreasing). Of course, this extension is done separately for every SCC or cycle, respectively.

In termination proofs, if $f \in \mathcal{R}_j$ is not orientable, then all symbols in $\mathcal{R}_i \geq_d \mathcal{R}_j$ as well as all dependency pairs resulting from $\mathcal{R}_i \geq_d \mathcal{R}_j$ are also not orientable. In innermost termination proofs, if f is not orientable, then a symbol that depends on f can still be orientable if one can extend the argument filtering in such a way that all occurrences of f in its rules are eliminated. Similarly, dependency pairs can still be orientable if the argument filtering eliminates all occurrences of f. Thus, here the bottom-up approach has the advantage that we already know that certain argument positions must be eliminated when extending the argument filtering to new function symbols.

This algorithm can also be modified by determining both the argument filtering and the reduction pair step by step. For example, a successful option is to use linear polynomial orders with coefficients 0 and 1. By permitting the coefficient 0, polynomial orders can also perform argument filtering, i.e., one does not have to use any extra argument filterings anymore. Again we consider two possibilities for the interpretation of constructors. One possibility is to map every n-ary constructor $c(x_1, \ldots, x_n)$ to the polynomial $1 + x_1 + \ldots + x_n$ if $n > 0$ and to 0, otherwise. The other possibility is to map every constructor $c(x_1, \ldots, x_n)$ to the polynomial $1 + x_{i_1} + \ldots + x_{i_k}$ if $i_1, \ldots, i_k$ are the argument positions with the same type as c's result and $k > 0$. Otherwise, c is mapped to 0. When extending the polynomial interpretation to a function f, we have already determined the polynomial interpretation for all symbols $g <_d f$. Then, we try to find a *minimal* polynomial for f such that the f-rules are weakly decreasing (i.e., as many coefficients as possible should be 0). The combination of the bottom-up algorithm with other orders (e.g., LPO) works in a similar way. Here, one always determines a minimal precedence which may be extended when proceeding to the next function symbol in the recursion hierarchy.

The bottom-up algorithm reduces the search space enormously. The number of TRSs from [3, 9, 27] where the bottom-up algorithm succeeds is 94 % of the number achieved by the full dependency pair approach with LPO, but runtime is reduced to less than 18 %.

8 Conclusion and Implementation in the System AProVE

We have presented improvements of the dependency pair approach which significantly reduce the sets of constraints $\pi(l) \succsim \pi(r)$ for both termination and innermost termination proofs. Moreover, we extended the applicability of dependency pair transformations and developed a criterion to ensure that their application is terminating without compromising the power of the approach in almost all examples. To implement the approach, we have given an algorithm for computing argument filterings which is tailored to the improvements presented before. Finally, we have developed heuristics to increase efficiency which proved successful in large case studies.

We implemented these results in the system AProVE (Automated Program Verification Environment), which is available at `http://www-i2.informatik.rwth-aachen.de/AProVE`. The tool is written in Java and proofs can be performed both in a fully automated or in an interactive mode via a graphical user interface. To combine the heuristics of Sect. 7, for every SCC $\mathcal{P}$, AProVE offers the following combination algorithm which uses the heuristics as a pre-processing step and only calls the full dependency pair approach for cycles where the heuristics fail:

1. Safe transformations with Cases (1) and (2) of Def. 18
2. Bottom-up heuristic of Sect. 7.3
3. Heuristics of Sect. 7.1 and Sect. 7.2 with LPO as base order
4. Remaining safe transformations according to Def. 18.
   If at least one transformation was applied, go back to 1.
5. Full dependency pair approach with RPO as base order

When the constraints for the SCC are solved, the algorithm is called recursively with the SCCs of those remaining pairs which were only weakly decreasing. We tested the combination algorithm on the collections of [3, 9, 27] (108 TRSs for termination, 151 TRSs for innermost termination). Our system succeeded on 96.6 % of the innermost termination examples (including all of [3]) and on

93.5 % of the examples for termination. The automated proof for the whole collection took 80 seconds for innermost termination and 27 seconds for termination. These results indicate that the contributions of the paper are indeed very useful in practice.

References

1. T. Arts. System description: The dependency pair method. In *Proc. 11th RTA*, LNCS 1833, 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. Available from `http://aib.informatik.rwth-aachen.de`.
4. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
5. C. Borralleras, M. Ferreira, and A. Rubio. Complete monotonic semantic path orderings. In *Proc. 17th CADE*, LNAI 1831, pages 346–364, 2000.
6. E. Contejean, C. Marché, B. Monate, and X. Urbain. Cime version 2, 2000. Available from `http://cime.lri.fr`.
7. N. Dershowitz. Orderings for term-rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.
8. N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3:69–116, 1987.
9. N. Dershowitz. 33 examples of termination. In *Proc. French Spring School of Theoretical Computer Science*, LNCS 909, pages 16–26, 1995.
10. N. Dershowitz, N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. A general framework for automatic termination analysis of logic programs. *Applicable Algebra in Engineering, Communication and Computing*, 12(1,2):117–156, 2001.
11. O. Fissore, I. Gnaedig, and H. Kirchner. Cariboo: An induction based proof tool for termination with strategies. In *Proc. 4th PPDP*, pages 62–73. ACM, 2002.
12. 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.
13. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
14. B. Gramlich. On proving termination by innermost termination. In *Proc. 7th RTA*, LNCS 1103, pages 97–107, 1996.
15. B. Gramlich. *Termination and Confluence Properties of Structured Rewrite Systems.* PhD thesis, Universität Kaiserslautern, Germany, 1996.
16. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. In *Proc. 19th CADE*, LNAI 2741, 2003.
17. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proc. 14th RTA*, LNCS 2706, pages 311–320, 2003.
18. G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25:239–299, 1982.
19. S. Kamin and J. J. Lévy. Two generalizations of the recursive path ordering. Unpublished Manuscript, University of Illinois, IL, USA, 1980.
20. D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon, 1970.
21. K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proc. 1st PPDP*, LNCS 1702, pages 48–62, 1999.
22. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.

23. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. In *Proc. POPL '01*, pages 81–92, 2001.
24. A. Middeldorp. Approximating dependency graphs using tree automata techniques. In *Proc. IJCAR 2001*, LNAI 2083, pages 593–610, 2001.
25. E. Ohlebusch. Hierarchical termination revisited. *Information Processing Letters*, 84(4):207–214, 2002.
26. E. Ohlebusch, C. Claves, and C. Marché. TALP: A tool for the termination analysis of logic programs. In *Proc. 11th RTA*, LNCS 1833, pages 270–273, 2000.
27. J. Steinbach. Automatic termination proofs with transformation orderings. In *Proc. 6th RTA*, LNCS, pages 11–25, 1995. Full version appeared as Technical Report SR-92-23, Universität Kaiserslautern, Germany.
28. J. Steinbach. Simplification orderings: History of results. *Fundamenta Informaticae*, 24:47–87, 1995.
29. R. Thiemann and J. Giesl. Size-change termination for term rewriting. In *Proc. 14th RTA*, LNCS 2706, pages 264–278, 2003.
30. Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.
31. X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In *Proc. IJCAR 2001*, LNAI 2083, pages 485–498, 2001.

A Empirical Results

We have implemented the results of this paper in our system AProVE and the following tables show the success rate and runtimes for the different techniques and heuristics. We have tested the system on the examples of [3, 9, 27] (108 TRSs for termination, 151 TRSs for innermost termination). In the tables, "S" stands for examples of Steinbach's collection [27] and "D" denotes examples of Dershowitz [9]. The remaining examples are taken from Chapter 3 and 4 of the collection of Arts and Giesl [3]. However for termination proofs, we did not regard the TRSs from Chapter 4 of [3], since many of them are only innermost terminating, but not terminating. Since some TRSs occur in several of these collections, we only consider each TRS once (for that reason, examples of [9] and [27] are not reconsidered if they also appear in [3]).

We have used AProVE in the following different settings:

- *Normal* is the method of Sect. 3 – 6 with reduction pairs based on LPO or the embedding order. For the LPO we allow different symbols to be equal in the precedence. Moreover, when computing the sets $\Pi^t(\mathcal{P})$ and $\Pi^i(\mathcal{P})$, we determine both the argument filterings and the precedences of the LPO, as illustrated at the end of Sect. 6. However, in this setting, we do not apply the heuristics of Sect. 7.
- *Old* is like *Normal*, but it uses Thm. 8 instead of Thm. 9 and 13.
- *Type* is like *Normal*, but it uses the type inference heuristic of Sect. 7.1.
- *Emb* is like *Normal*, but it applies the heuristic to use the embedding order for dependency pairs (Sect. 7.2).
- *Bottom-Up* uses the bottom-up heuristic of Sect. 7.3 where we determine both the argument filtering and the reduction pair step by step. Here, for every SCC or cycle, the bottom-up algorithm may be applied several times. For

example, one may use linear polynomial orders with coefficients 0 and 1, but if the constraints are not satisfied with the first possibility for interpreting constructors, then one can try the other possibility as illustrated in Sect. 7.3. In a similar way, one can use LPO instead of polynomial orders and regard the two possibilities for filtering constructors (no elimination of arguments or a filtering determined by type inference). Finally, it is also possible to combine both kinds of reduction pairs by first checking the two possibilities with polynomial orders and then checking the two possibilities with LPO, if the constraints for this cycle have not yet been solved. In this way, we obtained 3 different versions of the bottom-up algorithm where in the last version, the bottom-up algorithm may be applied at most four times for each cycle or SCC, respectively.

– *Combi* is the combination algorithm of Sect. 8 where we used the last version of the bottom-up algorithm described above.

The next two tables summarize the results of our experiments. In the "Power" column we give the number of examples where the proof attempt was successful. In square brackets we indicate the percentage of these examples compared to the number of all examples in the collection. In the "Time" column we give the overall runtime for running the system on all examples of the collection (also on the ones where the proof attempt failed). For each example we used a time-out of 30 seconds. The average time required for each example is given in square brackets. The detailed results of our experiments can be found at the end of this appendix.

Algorithm	Order	Power	Time
Termination			
Old	EMB	52 [48.1 %]	110.8 s [1.0 s]
Normal	EMB	64 [59.2 %]	113.1 s [1.0 s]
Old	LPO	81 [75.0 %]	320.0 s [2.9 s]
Normal	LPO	85 [78.7 %]	246.3 s [2.2 s]
Innermost Termination			
Old	EMB	100 [66.2 %]	213.5 s [1.4 s]
Normal	EMB	121 [80.1 %]	152.4 s [1.0 s]
Old	LPO	128 [84.7 %]	412.1 s [2.7 s]
Normal	LPO	131 [86.7 %]	353.9 s [2.3 s]

The above table illustrates the usefulness of the results from Sect. 3 and 4 and shows that Thm. 9 and 13 indeed improve upon Thm. 8 in practice. If one uses simple reduction pairs like the embedding order where orientability can be checked very efficiently, then compared to Thm. 8, Thm. 9 and 13 increase power by more than 20 % on the examples from [3, 9, 27]. For innermost termination, runtimes are decreased by about 28 %, while for termination one keeps approximately the same runtimes. If the reduction pairs are more complex (i.e., LPO), then Thm. 9 and 13 significantly reduce runtime (by about 23 % for termination and about 14 % for innermost termination), while power is increased moderately.

Algorithm	Orders	Power	Time
Termination			
Normal	LPO	85 [78.7 %]	246.3 s [2.2 s]
Type	LPO	85 [78.7 %]	226.1 s [2.0 s]
Emb	LPO	82 [75.9 %]	104.1 s [0.9 s]
Bottom-Up	LPO	66 [61.1 %]	76.9 s [0.7 s]
Bottom-Up	Polo	75 [69.4 %]	25.4 s [0.2 s]
Bottom-Up	Polo, LPO	80 [74.0 %]	27.2 s [0.2 s]
Combi		101 [93.5 %]	28.4 s [0.2 s]
Innermost Termination			
Normal	LPO	131 [86.7 %]	353.9 s [2.3 s]
Type	LPO	131 [86.7 %]	316.3 s [2.0 s]
Emb	LPO	128 [84.7 %]	137.2 s [0.9 s]
Bottom-Up	LPO	113 [74.8 %]	115.7 s [0.7 s]
Bottom-Up	Polo	125 [82.7 %]	60.6 s [0.4 s]
Bottom-Up	Polo, LPO	126 [83.4 %]	63.2 s [0.4 s]
Combi		146 [96.6 %]	86.1 s [0.5 s]

This table illustrates the usefulness of the heuristics of Sect. 7 and the combination algorithm described in Sect. 8. The results show that the type inference heuristic on its own does not improve the performance very much, but it also does not reduce the set of examples where the method is successful. With the embedding order heuristic from Sect. 7.2 we only lose a few examples in comparison to the full algorithm, but we need less than half of the time. Using the bottom-up heuristics, there exist several examples where we are no longer able to prove (innermost) termination, but we are at least three times faster than with the full approach. We also see that using simple polynomial orders for the bottom-up heuristic is fast and successful. Finally, with the combined algorithm, we obtain the best of all methods. The algorithm is almost as fast as the bottom-up algorithm and we get a success rate of over 93 % for the examples from the collections of [3, 9, 27].

The following tables give the detailed runtimes (in seconds, where "∞" denotes a time-out after 30 seconds) and results for the examples (where "OK" means that the proof succeeded and "-" means that the proof failed).

Table 1. Termination

Algorithm Orders	Old EMB	Normal EMB	Old LPO	Normal LPO
3.1	0.5 OK	0.6 OK	0.1 OK	0.6 OK
3.2	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.3	0.8 -	0.7 -	0.6 OK	0.5 OK
3.4	0.0 -	0.0 -	0.5 OK	0.1 OK
3.5	1.6 -	0.5 OK	1.8 OK	1.7 OK
3.5a	1.9 -	0.6 OK	2.1 OK	2.0 OK
3.5b	0.6 -	1.6 -	6.2 OK	5.1 OK
3.6	3.7 -	3.7 -	∞ [-]	∞ [-]
3.6a	2.2 -	2.3 -	∞ [-]	27.0 -
3.6b	0.6 -	1.9 -	∞ [-]	∞ [-]
3.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.8	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.8a	0.2 OK	0.2 OK	0.2 OK	0.2 OK
3.8b	0.6 -	0.7 -	1.5 OK	0.7 OK
3.9	2.2 -	0.4 OK	1.1 OK	0.6 OK
3.10	2.9 -	12.4 -	∞ [-]	∞ [-]
3.11	1.5 -	1.0 OK	3.2 OK	2.2 OK
3.12	0.3 -	0.3 -	1.2 -	1.1 -
3.13	∞ [-]	∞ [-]	∞ [-]	∞ [-]
3.14	1.7 -	1.7 -	2.3 OK	1.6 OK
3.15	0.0 -	0.0 -	0.0 -	0.0 -
3.16	0.0 -	0.0 -	0.1 OK	0.1 OK
3.17	0.1 -	0.0 -	2.4 OK	1.6 OK
3.17a	0.5 -	0.0 -	∞ [-]	∞ [-]
3.18	0.0 -	0.1 -	0.3 OK	0.1 OK
3.19	0.1 -	0.0 -	0.4 OK	0.2 OK
3.20	0.1 OK	0.1 OK	0.3 OK	0.3 OK
3.21	0.1 OK	0.1 OK	0.7 OK	0.6 OK
3.22	0.1 -	0.0 -	0.4 -	3.7 -
3.23	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.24	0.2 -	0.2 -	0.2 -	0.2 -
3.25	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.26	0.1 -	0.1 -	0.0 OK	0.0 OK
3.27	0.1 -	0.1 -	0.0 OK	0.0 OK
3.28	0.1 -	0.1 -	0.2 OK	0.2 OK
3.29	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.30	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.31	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.32	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.33	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.34	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.35	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.36	0.7 -	0.5 OK	0.4 OK	0.4 OK
3.37	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.38	0.2 -	0.2 OK	0.8 OK	0.7 OK
3.39	0.1 -	0.4 -	0.6 -	0.8 -
3.40	0.1 -	1.3 -	0.7 -	3.4 -

Algorithm Orders	Old EMB	Normal EMB	Old LPO	Normal LPO
3.41	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.42	0.2 OK	0.1 OK	0.2 OK	0.2 OK
3.43	0.0 OK	0.0 OK	0.1 OK	0.1 OK
3.44	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.45	0.0 OK	0.0 OK	0.1 OK	0.0 OK
3.46	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.47	0.0 OK	0.0 OK	0.1 OK	0.1 OK
3.48	0.0 -	0.0 -	4.2 OK	2.2 OK
3.49	0.1 -	0.1 -	0.2 -	0.2 -
3.50	0.0 -	0.0 OK	0.0 OK	0.0 OK
3.51	0.0 -	0.0 -	0.5 OK	0.1 OK
3.52	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.53	2.0 -	2.1 -	3.1 -	3.1 -
3.53a	0.0 -	0.1 -	0.0 -	0.0 -
3.53b	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.54	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.55	2.0 -	1.2 OK	5.5 OK	4.5 OK
3.56	0.1 -	0.1 OK	0.2 -	0.1 OK
3.57	0.1 -	0.0 -	∞ [-]	1.3 OK
S.1	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.2	0.1 OK	0.1 OK	6.0 OK	5.7 OK
S.3	0.1 OK	0.1 OK	0.0 OK	0.0 OK
S.4	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.5	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.6	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.10	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.11	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.12	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.14	0.1 -	0.1 -	1.5 -	1.5 -
S.15	0.1 -	0.1 -	0.1 -	0.1 -
S.17	0.1 -	0.1 -	0.2 -	0.2 -
S.18	0.0 -	0.0 -	0.0 -	0.0 -
S.22	0.1 OK	0.1 OK	0.2 OK	0.2 OK
S.24	1.3 -	0.3 OK	28.7 -	3.3 OK
S.25	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.26	4.1 -	0.5 OK	12.0 -	1.1 OK
S.27	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.28	0.0 -	0.0 -	0.2 -	0.1 -
S.29	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.30	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.31	2.2 OK	2.2 OK	3.8 OK	3.8 OK
D.1	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.2	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.3	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.6	0.0 -	0.0 -	0.0 OK	0.0 OK
D.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.8	0.1 OK	0.1 OK	0.1 OK	0.1 OK

Algorithm Orders	Old EMB		Normal EMB		Old LPO		Normal LPO	
D.9	0.0	-	0.0	-	0.0	OK	0.0	OK
D.11	0.6	OK	0.6	OK	0.6	OK	0.6	OK
D.12	0.0	-	0.0	-	0.0	-	0.0	-
D.13	0.0	-	0.0	-	0.1	-	0.1	-
D.17	0.0	-	0.0	-	0.1	OK	0.1	OK
D.18	0.0	OK	0.0	OK	0.0	OK	0.0	OK
D.20	∞	[-]	∞	[-]	1.1	OK	1.0	OK
D.21	0.1	OK	0.1	OK	0.1	OK	0.1	OK
D.28	0.1	-	0.0	-	0.0	OK	0.0	OK
D.29	0.5	-	0.1	OK	0.1	OK	0.1	OK
D.30	7.4	-	7.4	-	0.5	OK	0.5	OK
D.32	0.4	OK	0.4	OK	0.8	OK	1.1	OK
D.33	0.2	-	0.3	-	6.9	-	4.2	-
Sum:	110	52	113	64	320	81	246	85
Avg/%:	1.0	48.1	1.0	59.2	2.9	75.0	2.2	78.7

Table 2. Innermost Termination

Algorithm Orders	Old EMB	Normal EMB	Old LPO	Normal LPO
3.1	0.5 OK	0.5 OK	0.6 OK	0.6 OK
3.2	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.3	0.8 -	0.7 -	0.6 OK	0.4 OK
3.4	0.8 -	0.2 OK	0.2 OK	0.2 OK
3.5	1.5 -	0.5 OK	1.8 OK	1.7 OK
3.5a	1.9 -	0.6 OK	2.1 OK	2.1 OK
3.5b	0.6 -	1.6 -	6.0 OK	5.1 OK
3.6	3.7 -	3.7 -	∞ [-]	∞ [-]
3.6a	2.2 -	2.3 -	28.9 -	27.0 -
3.6b	0.7 -	1.9 -	∞ [-]	∞ [-]
3.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.8	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.8a	0.2 OK	0.2 OK	0.2 OK	0.2 OK
3.8b	0.6 -	0.7 -	1.5 OK	0.7 OK
3.9	2.2 -	0.4 OK	1.1 OK	0.6 OK
3.10	2.7 -	12.5 -	∞ [-]	∞ [-]
3.11	1.5 -	1.0 OK	3.3 OK	2.3 OK
3.12	0.3 -	0.3 -	1.1 -	1.1 -
3.13	∞ [-]	∞ [-]	∞ [-]	∞ [-]
3.14	1.7 -	1.7 -	1.7 OK	1.6 OK
3.15	0.0 -	0.0 -	0.0 -	0.0 -
3.16	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.17	1.0 -	0.9 -	1.8 OK	1.8 OK
3.17a	2.7 -	2.6 -	∞ [-]	∞ [-]
3.18	1.3 -	0.2 OK	0.2 OK	0.2 OK
3.19	1.4 -	0.3 OK	0.2 OK	0.2 OK
3.20	0.1 OK	0.1 OK	0.3 OK	0.3 OK
3.21	0.1 OK	0.1 OK	0.6 OK	0.6 OK
3.22	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.23	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.24	0.2 -	0.2 -	0.2 -	0.2 -
3.25	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.26	0.2 -	0.3 -	0.0 OK	0.0 OK
3.27	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.28	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.29	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.30	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.31	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.32	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.33	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.34	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.35	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.36	0.7 -	0.5 OK	0.4 OK	0.4 OK
3.37	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.38	0.2 -	0.1 OK	0.9 OK	0.7 OK
3.39	0.1 OK	0.1 OK	0.2 OK	0.1 OK
3.40	0.2 OK	0.2 OK	0.2 OK	0.2 OK

Algorithm Orders	Old EMB	Normal EMB	Old LPO	Normal LPO
3.41	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.42	0.1 OK	0.1 OK	0.2 OK	0.2 OK
3.43	0.0 OK	0.0 OK	0.1 OK	0.1 OK
3.44	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.45	0.0 OK	0.0 OK	0.1 OK	0.0 OK
3.46	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.47	0.1 OK	0.0 OK	0.0 OK	0.1 OK
3.48	6.3 -	6.0 -	2.8 OK	1.4 OK
3.49	0.1 -	0.1 -	0.2 -	0.3 -
3.50	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.51	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.52	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.53	2.0 -	2.0 -	3.0 -	3.1 -
3.53a	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.53b	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.54	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.55	2.1 -	1.2 OK	5.5 OK	4.5 OK
3.56	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.57	1.4 -	1.2 -	2.0 OK	2.1 OK
4.1	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.2	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.3	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.4	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.4a	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.5	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.6	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.8	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.9	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.10	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.11	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.12	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.12a	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.13	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.14	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.15	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.16	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.17	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.18	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.19	0.1 OK	0.1 OK	0.1 OK	0.1 OK
4.20	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.20a	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.21	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.22	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.23	0.2 -	0.1 OK	0.1 OK	0.2 OK
4.25	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.26	0.8 -	0.8 OK	1.9 OK	1.8 OK
4.27	0.1 OK	0.1 OK	0.1 OK	0.1 OK

Algorithm Orders	Old EMB	Normal EMB	Old LPO	Normal LPO
4.28	0.1 OK	0.1 OK	0.1 OK	0.1 OK
4.29	1.0 -	1.0 OK	8.5 OK	7.1 OK
4.30	1.3 -	1.1 OK	4.8 OK	4.7 OK
4.30a	0.1 OK	0.1 OK	0.1 OK	0.2 OK
4.30b	1.3 -	1.2 OK	5.1 OK	6.3 OK
4.30c	2.9 -	2.9 -	23.4 -	29.4 -
4.31	0.2 OK	0.2 OK	2.2 OK	2.2 OK
4.32	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.33	0.2 OK	0.2 OK	0.2 OK	0.2 OK
4.34	0.7 -	0.7 -	0.6 OK	0.3 OK
4.35	∞ [-]	8.2 OK	∞ [-]	∞ [-]
4.36	2.7 -	3.0 OK	∞ [-]	9.6 OK
4.37	0.1 OK	0.0 OK	0.0 OK	0.0 OK
4.37a	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.1	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.2	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.3	0.0 OK	0.0 OK	0.0 OK	0.1 OK
S.4	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.5	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.6	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.10	0.1 OK	0.0 OK	0.0 OK	0.0 OK
S.11	0.1 OK	0.0 OK	0.0 OK	0.0 OK
S.12	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.14	0.4 OK	0.2 OK	0.1 OK	0.2 OK
S.15	0.1 -	0.1 -	0.2 -	0.1 -
S.17	0.7 -	0.5 -	0.8 -	0.8 -
S.18	0.1 -	0.0 -	0.0 -	0.0 -
S.22	0.2 OK	0.1 OK	0.1 OK	0.2 OK
S.24	2.1 -	0.2 OK	28.1 -	3.1 OK
S.25	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.26	9.4 -	0.6 OK	12.0 -	1.2 OK
S.27	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.28	0.1 -	0.1 -	0.2 -	0.2 -
S.29	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.30	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.31	∞ [-]	2.2 OK	3.7 OK	3.8 OK
D.1	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.2	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.3	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.6	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.8	0.1 OK	0.1 OK	0.1 OK	0.1 OK
D.9	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.11	1.0 OK	0.9 OK	0.5 OK	0.6 OK
D.12	0.1 -	0.1 -	0.1 -	0.1 -
D.13	0.8 -	0.6 -	0.9 -	0.9 -
D.17	2.2 OK	0.1 OK	0.1 OK	0.1 OK

Algorithm Orders	Old EMB		Normal EMB		Old LPO		Normal LPO	
D.18	0.0	OK	0.0	OK	0.0	OK	0.0	OK
D.20	∞	[-]	∞	[-]	1.9	OK	2.0	OK
D.21	0.1	OK	0.1	OK	0.1	OK	0.1	OK
D.28	0.0	OK	0.0	OK	0.0	OK	0.0	OK
D.29	0.5	-	0.1	OK	0.1	OK	0.1	OK
D.30	7.4	-	7.4	-	0.5	OK	0.5	OK
D.32	0.4	OK	0.4	OK	0.8	OK	1.1	OK
D.33	7.2	-	7.1	-	∞	[-]	∞	[-]
Sum:	213	100	152	121	412	128	353	131
Avg/%:	1.4	66.2	1.0	80.1	2.7	84.7	2.3	86.7

Table 3. Termination

Algorithm Orders	Normal LPO	Type LPO	Emb LPO	Bottom-Up LPO	Bottom-Up Polo	Bottom-Up Polo, LPO	Combi
3.1	0.6 OK	0.6 OK	0.6 OK	0.6 OK	0.5 OK	0.6 OK	0.5 OK
3.2	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.0 OK
3.3	0.5 OK	0.4 OK	0.3 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK
3.4	0.1 OK	0.1 OK	0.2 OK	0.0 -	0.0 -	0.1 -	0.1 OK
3.5	1.7 OK	1.6 OK	0.6 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK
3.5a	2.0 OK	1.4 OK	0.7 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK
3.5b	5.1 OK	2.3 OK	0.9 OK	0.3 OK	0.3 OK	0.3 OK	0.2 OK
3.6	∞ [-]	∞ [-]	4.1 -	2.9 -	0.3 OK	0.3 OK	0.2 OK
3.6a	27.0 -	17.6 -	2.6 -	2.2 -	0.2 OK	0.3 OK	0.3 OK
3.6b	∞ [-]	∞ [-]	15.8 -	3.2 -	0.4 OK	0.4 OK	0.3 OK
3.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.8	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.8a	0.2 OK	0.2 OK	0.2 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.8b	0.7 OK	0.6 OK	0.5 OK	0.3 OK	0.3 OK	0.3 OK	0.2 OK
3.9	0.6 OK	0.6 OK	0.4 OK	0.3 OK	0.3 OK	0.3 OK	0.2 OK
3.10	∞ [-]	∞ [-]	13.2 -	10.5 -	1.5 OK	1.5 OK	1.4 OK
3.11	2.2 OK	2.0 OK	1.2 OK	0.7 OK	0.6 OK	0.7 OK	0.5 OK
3.12	1.1 -	1.1 -	0.5 -	0.2 -	0.1 OK	0.1 OK	0.0 OK
3.13	∞ [-]	∞ [-]	∞ [-]	∞ [-]	0.8 OK	0.8 OK	0.8 OK
3.14	1.6 OK	1.6 OK	1.7 -	1.1 OK	0.1 OK	0.1 OK	0.1 OK
3.15	0.0 -	0.0 -	0.0 -	0.0 -	0.0 OK	0.0 OK	0.0 OK
3.16	0.1 OK	0.1 OK	0.1 OK	0.0 OK	0.0 -	0.1 OK	0.0 OK
3.17	1.6 OK	1.7 OK	0.7 OK	0.4 -	0.1 OK	0.1 OK	0.1 OK
3.17a	∞ [-]	∞ [-]	4.8 -	0.6 -	0.1 OK	0.1 OK	0.1 OK
3.18	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 -	0.1 OK	0.1 OK
3.19	0.2 OK	0.2 OK	0.2 OK	0.1 OK	0.0 -	0.2 OK	0.2 OK
3.20	0.3 OK	0.3 OK	0.1 OK	0.1 OK	0.0 OK	0.0 OK	0.0 OK
3.21	0.6 OK	0.6 OK	0.1 OK	0.1 -	0.1 -	0.1 -	0.3 OK
3.22	3.7 -	3.8 -	0.4 -	0.1 -	0.1 -	0.2 -	3.3 -
3.23	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.24	0.2 -	0.2 -	0.2 -	0.2 -	0.0 OK	0.0 OK	0.0 OK
3.25	0.0 OK	0.0 OK	0.0 OK	0.1 -	0.1 -	0.1 -	0.1 OK
3.26	0.0 OK	0.0 OK	0.1 -	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.27	0.0 OK	0.0 OK	0.0 OK	0.1 -	0.1 -	0.1 -	0.1 OK
3.28	0.2 OK	0.1 OK	0.2 OK	0.0 -	0.0 -	0.1 -	0.6 OK
3.29	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.30	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.31	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.32	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.33	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.34	0.0 OK	0.0 OK	0.0 OK	0.0 -	0.0 -	0.0 -	0.0 OK
3.35	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.36	0.4 OK	0.4 OK	0.5 OK	0.5 OK	0.1 OK	0.1 OK	0.1 OK
3.37	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.38	0.7 OK	0.7 OK	0.2 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.39	0.8 -	0.8 -	0.5 -	0.3 -	0.2 OK	0.3 OK	0.3 OK
3.40	3.4 -	3.4 -	1.4 -	1.1 -	0.4 OK	0.4 OK	0.5 OK

Algorithm Orders	Normal LPO	Type LPO	Emb LPO	Bottom-Up LPO	Bottom-Up Polo	Bottom-Up Polo, LPO	Combi
3.41	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.1 OK	0.0 OK	0.0 OK
3.42	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.2 OK
3.43	0.1 OK	0.1 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.44	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.45	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.1 OK
3.46	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.47	0.1 OK	0.1 OK	0.2 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.48	2.2 OK	1.6 OK	1.1 -	0.0 -	0.0 -	0.1 -	2.3 OK
3.49	0.2 -	0.1 -	0.1 -	0.0 -	0.1 -	0.1 -	0.1 OK
3.50	0.0 OK	0.0 OK	0.0 OK	0.0 -	0.0 -	0.0 -	0.0 OK
3.51	0.1 OK	0.1 OK	0.1 OK	0.0 -	0.0 -	0.1 -	0.1 OK
3.52	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.53	3.1 -	3.2 -	2.1 -	1.9 -	0.4 OK	0.4 OK	0.3 OK
3.53a	0.0 -	0.0 -	0.0 -	0.1 -	0.0 OK	0.0 OK	0.0 OK
3.53b	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.54	0.0 OK	0.0 OK	0.0 OK	0.1 -	0.1 -	0.1 -	0.0 OK
3.55	4.5 OK	3.2 OK	1.4 OK	0.9 OK	0.8 OK	0.8 OK	0.7 OK
3.56	0.1 OK	0.1 OK	0.1 OK	0.0 -	0.0 -	0.0 -	0.1 OK
3.57	1.3 OK	1.5 OK	0.9 OK	0.3 -	0.1 -	0.4 -	0.2 OK
S.1	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.2	5.7 OK	3.3 OK	1.5 OK	0.0 -	0.0 -	0.0 -	0.7 OK
S.3	0.0 OK	0.1 OK	0.0 OK	0.1 OK	0.1 OK	0.1 OK	0.0 OK
S.4	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.5	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.6	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.10	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.11	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.12	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.14	1.5 -	1.5 -	0.1 -	0.1 -	0.0 -	0.1 -	0.8 -
S.15	0.1 -	0.1 -	0.1 -	0.1 -	0.1 -	0.1 -	0.2 -
S.17	0.2 -	0.1 -	0.1 -	0.0 -	0.0 -	0.0 -	0.2 -
S.18	0.0 -	0.0 -	0.0 -	0.1 -	0.1 -	0.1 -	0.1 OK
S.22	0.2 OK	0.2 OK	0.1 OK	0.2 OK	0.1 OK	0.2 OK	0.2 OK
S.24	3.3 OK	2.7 OK	0.4 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK
S.25	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.26	1.1 OK	1.2 OK	0.6 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.27	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.28	0.1 -	0.1 -	0.0 -	0.0 -	0.0 -	0.0 -	0.5 -
S.29	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.0 OK	0.1 OK
S.30	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.31	3.8 OK	2.8 OK	2.6 OK	0.9 -	0.8 -	0.9 -	1.6 OK
D.1	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.2	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.3	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.6	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 -	0.0 OK	0.0 OK
D.7	0.0 OK	0.0 OK	0.0 OK	0.0 -	0.0 -	0.0 -	0.0 OK
D.8	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK

Algorithm Orders	Normal LPO		Type LPO		Emb LPO		Bottom-Up LPO		Bottom-Up Polo		Bottom-Up Polo, LPO		Combi	
D.9	0.0	OK	0.0	OK	0.0	OK	0.0	-	0.0	-	0.0	-	0.0	OK
D.11	0.6	OK	0.6	OK	0.6	OK	1.0	OK	0.5	OK	0.5	OK	0.5	OK
D.12	0.0	-	0.0	-	0.0	-	0.0	-	0.0	-	0.1	-	0.4	OK
D.13	0.1	-	0.1	-	0.1	-	0.0	-	0.0	-	0.1	-	0.3	-
D.17	0.1	OK	0.1	OK	0.0	OK	0.0	OK	0.0	-	0.1	OK	0.0	OK
D.18	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
D.20	1.0	OK	1.0	OK	0.9	OK	0.9	OK	0.9	OK	0.9	OK	0.9	OK
D.21	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.2	OK	0.1	OK	0.0	OK
D.28	0.0	OK	0.0	OK	0.0	OK	0.0	-	0.0	-	0.0	-	0.1	OK
D.29	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.0	OK
D.30	0.5	OK	0.5	OK	0.3	OK	7.3	-	7.3	-	7.4	-	0.3	OK
D.32	1.1	OK	0.7	OK	0.7	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK
D.33	4.2	-	3.3	-	1.1	-	0.1	-	0.1	-	0.1	-	2.0	-
Sum:	246	85	226	85	104	82	76	66	25	75	27	80	28	101
Avg/%:	2.2	78.7	2.0	78.7	0.9	75.9	0.7	61.1	0.2	69.4	0.2	74.0	0.2	93.5

Table 4. Innermost Termination

Algorithm Orders	Normal LPO	Type LPO	Emb LPO	Bottom-Up LPO	Bottom-Up Polo	Bottom-Up Polo, LPO	Combi
3.1	0.6 OK	0.7 OK	0.6 OK	0.6 OK	0.5 OK	0.6 OK	0.5 OK
3.2	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.0 OK
3.3	0.4 OK	0.4 OK	0.3 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK
3.4	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK	0.2 OK	0.1 OK
3.5	1.7 OK	1.6 OK	0.6 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK
3.5a	2.1 OK	1.4 OK	0.7 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK
3.5b	5.1 OK	2.3 OK	0.9 OK	0.3 OK	0.3 OK	0.3 OK	0.2 OK
3.6	∞ [-]	∞ [-]	4.1 -	2.8 -	0.3 OK	0.3 OK	0.2 OK
3.6a	27.0 -	17.6 -	2.5 -	2.2 -	0.4 OK	0.3 OK	0.3 OK
3.6b	∞ [-]	∞ [-]	15.3 -	3.2 -	0.4 OK	0.4 OK	0.3 OK
3.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.8	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.8a	0.2 OK	0.2 OK	0.2 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.8b	0.7 OK	0.6 OK	0.6 OK	0.3 OK	0.3 OK	0.3 OK	0.2 OK
3.9	0.6 OK	0.6 OK	0.5 OK	0.3 OK	0.3 OK	0.2 OK	0.2 OK
3.10	∞ [-]	∞ [-]	12.9 -	10.6 -	1.5 OK	1.5 OK	1.3 OK
3.11	2.3 OK	1.9 OK	1.2 OK	0.7 OK	0.6 OK	0.7 OK	0.5 OK
3.12	1.1 -	1.1 -	0.4 -	0.2 -	0.1 OK	0.1 OK	0.0 OK
3.13	∞ [-]	∞ [-]	∞ [-]	∞ [-]	0.8 OK	0.8 OK	0.7 OK
3.14	1.6 OK	1.6 OK	1.7 -	1.1 OK	0.1 OK	0.2 OK	0.1 OK
3.15	0.0 -	0.0 -	0.0 -	0.0 -	0.0 OK	0.0 OK	0.0 OK
3.16	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.1 OK
3.17	1.8 OK	1.7 OK	0.8 OK	0.6 -	0.1 OK	0.1 OK	0.1 OK
3.17a	∞ [-]	∞ [-]	3.8 -	1.3 -	0.3 OK	0.3 OK	0.3 OK
3.18	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK
3.19	0.2 OK	0.2 OK	0.2 OK	0.3 OK	0.2 OK	0.2 OK	0.2 OK
3.20	0.3 OK	0.3 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.0 OK
3.21	0.6 OK	0.6 OK	0.1 OK	0.1 -	0.1 -	0.1 -	0.3 OK
3.22	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.23	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.24	0.2 -	0.2 -	0.2 -	0.2 -	0.0 OK	0.0 OK	0.0 OK
3.25	0.0 OK	0.1 OK	0.0 OK	0.1 -	0.1 -	0.1 -	0.1 OK
3.26	0.0 OK	0.0 OK	0.3 -	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.27	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.28	0.0 OK	0.1 OK	0.1 OK	0.1 -	0.1 -	0.1 -	0.2 OK
3.29	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.30	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.31	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.32	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.33	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.34	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.35	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.36	0.4 OK	0.4 OK	0.6 OK	0.5 OK	0.1 OK	0.1 OK	0.1 OK
3.37	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
3.38	0.7 OK	0.7 OK	0.2 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.39	0.1 OK	0.1 OK	0.2 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
3.40	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK

Algorithm Orders	Normal LPO		Type LPO		Emb LPO		Bottom-Up LPO		Bottom-Up Polo		Bottom-Up Polo, LPO		Combi	
3.41	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
3.42	0.2	OK	0.2	OK	0.2	OK	0.2	OK	0.2	OK	0.3	OK	0.2	OK
3.43	0.1	OK	0.1	OK	0.0	OK	0.0	OK	0.0	OK	0.1	OK	0.1	OK
3.44	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
3.45	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.1	OK	0.0	OK	0.1	OK
3.46	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
3.47	0.1	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
3.48	1.4	OK	0.8	OK	6.5	-	5.2	-	5.4	-	5.4	-	20.9	OK
3.49	0.3	-	0.2	-	0.1	-	0.1	-	0.1	-	0.1	-	0.1	OK
3.50	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
3.51	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
3.52	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
3.53	3.1	-	3.1	-	2.2	-	1.9	-	0.4	OK	0.4	OK	0.4	OK
3.53a	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
3.53b	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
3.54	0.0	OK	0.0	OK	0.0	OK	0.1	-	0.1	-	0.1	-	0.1	OK
3.55	4.5	OK	3.2	OK	1.4	OK	0.9	OK	0.8	OK	0.8	OK	0.6	OK
3.56	0.0	OK	0.0	OK	0.0	OK	0.1	-	0.0	-	0.1	-	0.1	OK
3.57	2.1	OK	2.1	OK	1.2	OK	0.9	-	0.6	OK	0.4	OK	0.5	OK
4.1	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.2	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.3	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.4	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.4a	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.5	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.6	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.7	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.8	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.9	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.10	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.11	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.12	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.12a	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.13	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.14	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.15	0.0	OK	0.0	OK	0.0	OK	0.0	-	0.0	-	0.0	-	0.0	OK
4.16	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.17	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.18	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.19	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.0	OK	0.0	OK	0.0	OK
4.20	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.20a	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.21	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.22	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.23	0.2	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.0	OK
4.25	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
4.26	1.8	OK	1.2	OK	0.9	OK	0.6	OK	0.2	OK	0.7	OK	0.7	OK
4.27	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.0	OK

Algorithm Orders	Normal LPO	Type LPO	Emb LPO	Bottom-Up LPO	Bottom-Up Polo	Bottom-Up Polo, LPO	Combi
4.28	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
4.29	7.1 OK	4.9 OK	1.1 OK	0.9 OK	0.5 OK	0.9 OK	1.0 OK
4.30	4.7 OK	3.4 OK	1.4 OK	0.7 OK	0.3 OK	0.6 OK	0.9 OK
4.30a	0.2 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.2 OK	0.1 OK
4.30b	6.3 OK	5.3 OK	1.5 OK	1.0 OK	0.5 OK	0.9 OK	1.0 OK
4.30c	29.4 -	19.9 -	3.4 -	3.0 -	0.8 OK	1.2 OK	1.3 OK
4.31	2.2 OK	2.2 OK	0.2 OK	0.2 OK	0.1 OK	0.2 OK	0.2 OK
4.32	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
4.33	0.2 OK	0.2 OK	0.2 OK	0.2 OK	0.1 OK	0.1 OK	0.1 OK
4.34	0.3 OK	0.3 OK	1.0 -	0.5 -	0.5 -	0.5 -	1.2 OK
4.35	∞ [-]	∞ [-]	5.7 OK	20.6 -	20.9 -	20.4 -	5.3 OK
4.36	9.6 OK	4.3 OK	6.6 OK	0.9 OK	0.8 OK	0.8 OK	0.7 OK
4.37	0.0 OK	0.0 OK	0.0 OK	0.1 -	0.0 -	0.1 -	0.1 OK
4.37a	0.0 OK	0.0 OK	0.0 OK	0.1 -	0.0 -	0.1 -	0.1 OK
S.1	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.2	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.3	0.1 OK	0.1 OK	0.1 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.4	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.5	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.6	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.7	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.10	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.11	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.12	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.14	0.2 OK	0.2 OK	0.2 OK	0.2 -	0.1 -	0.2 -	0.2 OK
S.15	0.1 -	0.1 -	0.1 -	0.2 -	0.2 -	0.2 -	0.3 -
S.17	0.8 -	0.7 -	0.6 -	0.4 -	0.4 -	0.5 -	0.8 -
S.18	0.0 -	0.0 -	0.0 -	0.0 -	0.1 -	0.1 -	0.1 OK
S.22	0.2 OK	0.2 OK	0.2 OK	0.1 OK	0.1 OK	0.2 OK	0.1 OK
S.24	3.1 OK	2.4 OK	0.2 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.25	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.26	1.2 OK	1.2 OK	0.6 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.27	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
S.28	0.2 -	0.2 -	0.1 -	0.1 -	0.0 -	0.1 -	0.5 -
S.29	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.30	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
S.31	3.8 OK	2.8 OK	2.5 OK	0.9 -	0.8 -	0.9 -	1.6 OK
D.1	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.2	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.3	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.6	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK	0.0 OK
D.7	0.0 OK	0.0 OK	0.0 OK	0.0 -	0.0 -	0.0 -	0.0 OK
D.8	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 OK
D.9	0.0 OK	0.0 OK	0.0 OK	0.0 -	0.0 -	0.0 -	0.0 OK
D.11	0.6 OK	0.6 OK	0.6 OK	1.4 OK	0.9 OK	0.9 OK	0.5 OK
D.12	0.1 -	0.1 -	0.1 -	0.1 -	0.1 -	0.2 -	0.1 OK
D.13	0.9 -	0.8 -	0.7 -	0.1 -	0.1 -	0.1 -	0.9 -
D.17	0.1 OK	0.1 OK	0.1 OK	0.1 OK	0.1 -	0.1 OK	0.1 OK

Algorithm	Normal		Type		Emb		Bottom-Up		Bottom-Up		Bottom-Up		Combi	
Orders	LPO		LPO		LPO		LPO		Polo		Polo, LPO			
D.18	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK	0.0	OK
D.20	2.0	OK	1.9	OK	1.9	OK	1.8	OK	1.8	OK	1.9	OK	1.7	OK
D.21	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.0	OK
D.28	0.0	OK	0.0	OK	0.0	OK	0.0	-	0.0	-	0.0	-	0.0	OK
D.29	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK	0.0	OK
D.30	0.5	OK	0.5	OK	0.3	OK	7.3	-	7.3	-	7.4	-	0.3	OK
D.32	1.1	OK	0.7	OK	0.7	OK	0.1	OK	0.1	OK	0.1	OK	0.1	OK
D.33	∞	[-]	∞	[-]	7.5	-	0.2	-	0.2	-	0.2	-	∞	[-]
Sum:	353	131	316	131	137	128	115	113	60	125	63	126	86	146
Avg/%:	2.3	86.7	2.0	86.7	0.9	84.7	0.7	74.8	0.4	82.7	0.4	83.4	0.5	96.6

Aachener Informatik-Berichte

This is a list of recent technical reports. To obtain copies of technical reports please consult http://aib.informatik.rwth-aachen.de/ or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

95-11 * M. Staudt / K. von Thadden: Subsumption Checking in Knowledge Bases

95-12 * G.V. Zemanek / H.W. Nissen / H. Hubert / M. Jarke: Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modeling Technology

95-13 * M. Staudt / M. Jarke: Incremental Maintenance of Externally Materialized Views

95-14 * P. Peters / P. Szczurko / M. Jeusfeld: Business Process Oriented Information Management: Conceptual Models at Work

95-15 * S. Rams / M. Jarke: Proceedings of the Fifth Annual Workshop on Information Technologies & Systems

95-16 * W. Hans / St. Winkler / F. Sáenz: Distributed Execution in Functional Logic Programming

96-1 * Jahresbericht 1995

96-2 M. Hanus / Chr. Prehofer: Higher-Order Narrowing with Definitional Trees

96-3 * W. Scheufele / G. Moerkotte: Optimal Ordering of Selections and Joins in Acyclic Queries with Expensive Predicates

96-4 K. Pohl: PRO-ART: Enabling Requirements Pre-Traceability

96-5 K. Pohl: Requirements Engineering: An Overview

96-6 * M. Jarke / W. Marquardt: Design and Evaluation of Computer–Aided Process Modelling Tools

96-7 O. Chitil: The ς-Semantics: A Comprehensive Semantics for Functional Programs

96-8 * S. Sripada: On Entropy and the Limitations of the Second Law of Thermodynamics

96-9 M. Hanus (Ed.): Proceedings of the Poster Session of ALP'96 — Fifth International Conference on Algebraic and Logic Programming

96-10 R. Conradi / B. Westfechtel: Version Models for Software Configuration Management

96-11 * C. Weise / D. Lenzkes: A Fast Decision Algorithm for Timed Refinement

96-12 * R. Dömges / K. Pohl / M. Jarke / B. Lohmann / W. Marquardt: PRO-ART/CE* — An Environment for Managing the Evolution of Chemical Process Simulation Models

96-13 * K. Pohl / R. Klamma / K. Weidenhaupt / R. Dömges / P. Haumer / M. Jarke: A Framework for Process-Integrated Tools

96-14 * R. Gallersdörfer / K. Klabunde / A. Stolz / M. Eßmajor: INDIA — Intelligent Networks as a Data Intensive Application, Final Project Report, June 1996

96-15 * H. Schimpe / M. Staudt: VAREX: An Environment for Validating and Refining Rule Bases

96-16 * M. Jarke / M. Gebhardt, S. Jacobs, H. Nissen: Conflict Analysis Across Heterogeneous Viewpoints: Formalization and Visualization

96-17 M. Jeusfeld / T. X. Bui: Decision Support Components on the Internet

96-18 M. Jeusfeld / M. Papazoglou: Information Brokering: Design, Search and Transformation

96-19 * P. Peters / M. Jarke: Simulating the impact of information flows in networked organizations

96-20 M. Jarke / P. Peters / M. Jeusfeld: Model-driven planning and design of cooperative information systems

96-21 * G. de Michelis / E. Dubois / M. Jarke / F. Matthes / J. Mylopoulos / K. Pohl / J. Schmidt / C. Woo / E. Yu: Cooperative information systems: a manifesto

96-22 * S. Jacobs / M. Gebhardt, S. Kethers, W. Rzasa: Filling HTML forms simultaneously: CoWeb architecture and functionality

96-23 * M. Gebhardt / S. Jacobs: Conflict Management in Design

97-01 Jahresbericht 1996

97-02 J. Faassen: Using full parallel Boltzmann Machines for Optimization

97-03 A. Winter / A. Schürr: Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems

97-04 M. Mohnen / S. Tobies: Implementing Context Patterns in the Glasgow Haskell Compiler

97-05 * S. Gruner: Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge

97-06 M. Nicola / M. Jarke: Design and Evaluation of Wireless Health Care Information Systems in Developing Countries

97-07 P. Hofstedt: Taskparallele Skelette für irregulär strukturierte Probleme in deklarativen Sprachen

97-08 D. Blostein / A. Schürr: Computing with Graphs and Graph Rewriting

97-09 C.-A. Krapp / B. Westfechtel: Feedback Handling in Dynamic Task Nets

97-10 M. Nicola / M. Jarke: Integrating Replication and Communication in Performance Models of Distributed Databases

97-13 M. Mohnen: Optimising the Memory Management of Higher-Order Functional Programs

97-14 R. Baumann: Client/Server Distribution in a Structure-Oriented Database Management System

97-15 G. H. Botorog: High-Level Parallel Programming and the Efficient Implementation of Numerical Algorithms

98-01 * Jahresbericht 1997

98-02 S. Gruner/ M. Nagel / A. Schürr: Fine-grained and Structure-oriented Integration Tools are Needed for Product Development Processes

98-03 S. Gruner: Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr

98-04 * O. Kubitz: Mobile Robots in Dynamic Environments

98-05 M. Leucker / St. Tobies: Truth — A Verification Platform for Distributed Systems

98-07 M. Arnold / M. Erdmann / M. Glinz / P. Haumer / R. Knoll / B. Paech / K. Pohl / J. Ryser / R. Studer / K. Weidenhaupt: Survey on the Scenario Use in Twelve Selected Industrial Projects

98-08 * H. Aust: Sprachverstehen und Dialogmodellierung in natürlichsprachlichen Informationssystemen

98-09 * Th. Lehmann: Geometrische Ausrichtung medizinischer Bilder am Beispiel intraoraler Radiographien

98-10 * M. Nicola / M. Jarke: Performance Modeling of Distributed and Replicated Databases

98-11 * A. Schleicher / B. Westfechtel / D. Jäger: Modeling Dynamic Software Processes in UML

98-12 * W. Appelt / M. Jarke: Interoperable Tools for Cooperation Support using the World Wide Web

98-13 K. Indermark: Semantik rekursiver Funktionsdefinitionen mit Striktheitsinformation

99-01 * Jahresbericht 1998

99-02 * F. Huch: Verifcation of Erlang Programs using Abstract Interpretation and Model Checking — Extended Version

99-03 * R. Gallersdörfer / M. Jarke / M. Nicola: The ADR Replication Manager

99-04 M. Alpuente / M. Hanus / S. Lucas / G. Vidal: Specialization of Functional Logic Programs Based on Needed Narrowing

99-07 Th. Wilke: CTL+ is exponentially more succinct than CTL

99-08 O. Matz: Dot-Depth and Monadic Quantifier Alternation over Pictures

2000-01 * Jahresbericht 1999

2000-02 Jens Vöge / Marcin Jurdzinski: A Discrete Strategy Improvement Algorithm for Solving Parity Games

2000-04 Andreas Becks / Stefan Sklorz / Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach

2000-05 Mareike Schoop: Cooperative Document Management

2000-06 Mareike Schoop / Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling

2000-07 * Markus Mohnen / Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages

2000-08 Thomas Arts / Thomas Noll: Verifying Generic Erlang Client-Server Implementations

2001-01 * Jahresbericht 2000

2001-02 Benedikt Bollig / Martin Leucker: Deciding LTL over Mazurkiewicz Traces

2001-03 Thierry Cachat: The power of one-letter rational languages

2001-04 Benedikt Bollig / Martin Leucker / Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus

2001-05 Benedikt Bollig / Martin Leucker / Thomas Noll: Regular MSC Languages

2001-06 Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic

2001-07 Martin Grohe / Stefan Wöhrle: An Existential Locality Theorem

2001-08 Mareike Schoop / James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling

2001-09 Thomas Arts / Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs

2001-10 Achim Blumensath: Axiomatising Tree-interpretable Structures

2001-11 Klaus Indermark / Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung

2002-01 * Jahresbericht 2001

2002-02 Jürgen Giesl / Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems

2002-03 Benedikt Bollig / Martin Leucker / Thomas Noll: Generalised Regular MSC Languages

2002-04 Jürgen Giesl / Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting

2002-05 Horst Lichter / Thomas von der Maßen / Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines

2002-06 Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata

2002-07 Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities

2002-08 Markus Mohnen: An Open Framework for Data-Flow Analysis in Java

2002-09 Markus Mohnen: Interfaces with Default Implementations in Java

2002-10 Martin Leucker: Logics for Mazurkiewicz traces

2002-11 Jürgen Giesl / Hans Zantema: Liveness in Rewriting

2003-01 * Jahresbericht 2002

2003-02 René Thiemann / Jürgen Giesl: Size-Change Termination for Term Rewriting

2003-03 Jürgen Giesl / Deepak Kapur: Deciding Inductive Validity of Equations

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.