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 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

Most traditional methods to prove termination of TRSs (automatically) use *simplification orders* [7, 24], where a term is greater than its proper subterms. 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, 11, 12] 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.

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

In Sect. 2, we recapitulate the dependency pair approach for termination and innermost termination proofs. Then we 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 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

* *Proceedings 10th Int. Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR '03)*, Almaty, Kazakhstan, LNAI, Springer-Verlag, 2003.

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. Thus, our contributions are also very helpful for other tools based on dependency pairs ([1], CiME [6], TTT [16]) and we conjecture that they can also be used in other recent approaches for termination of TRSs [5, 10] which have several aspects in common with dependency pairs. Finally, dependency pairs can be combined with other termination techniques (e.g., in [25] we integrated dependency pairs and the *size-change principle* from termination analysis of functional [19] and logic programs [9]). Moreover, the system TALP [22] 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. All proofs and details on our experiments can be found in [13].

2 Dependency Pairs

We briefly present the *dependency pair* approach of Arts and Giesl and refer to [2, 11, 12] 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

$$\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y) \quad (1) \qquad \mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y) \quad (2)$$

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

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, $\xrightarrow{\mathsf{i}}_{\mathcal{R}}$ denotes innermost reductions.

Definition 3 ($\mathcal{R}$-Chain). *A sequence of dependency pairs $s_1 \to t_1$, $s_2 \to t_2, \ldots$ 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 $t_j\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}}^* s_{j+1}\sigma$ and if $s_j\sigma$ is a normal form for all j.*

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) ⇒ (1), (2) ⇒ (1), (3) ⇒ (2), and (3) ⇒ (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 $\textsc{ren}(\mathsf{QUOT}(x, x)) = \mathsf{QUOT}(x_1, x_2)$. In the *estimated dependency graph*, 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. Here, $\textsc{cap}_v$ is defined like $\textsc{cap}$ except that subterms with defined root that already occur in v are not replaced by new variables. In Ex. 1, the estimated dependency and the estimated innermost dependency graph are identical to the real dependency graph. For alternative approximations of dependency graphs see [15, 20].

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, one proves absence of infinite (innermost) chains separately for every cycle. To this end, one generates sets of constraints which should be satisfied by a *reduction pair* $(\succsim, \succ)$ [18] consisting of a quasi-rewrite order $\succsim$ (i.e., $\succsim$ is reflexive, transitive, monotonic and stable (closed under contexts and 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 5 (Usable Rules). *For $f \in \mathcal{F}$, let $Rls(f) = \{l \to r \in \mathcal{R} | \mathrm{root}(l) = f\}$. 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, 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 (we use the notation of [18]).

Definition 6 (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 7 (Automating Dependency Pairs [2, 12]). *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 argument filtering* π *with*

(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 proving termination.

As shown in [15], to implement Thm. 7, one does 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. 7 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. 7 for 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 [21], then the constraints (a) and (b) of Thm. 7 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 [27] how to use dependency pairs for modular termination proofs of hierarchical combinations of C_ε-terminating TRSs. However in the results of [27], he did not integrate the consideration of cycles in (estimated) dependency graphs and required all dependency pairs to be strictly decreasing. Thm. 8 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. 7 is reduced significantly.

The crucial idea of [27] 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 $Rls(f) \cup Rls(g) \subseteq \mathcal{R}_i$, 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 $Rls(f) \subseteq \mathcal{R}_i$ and $Rls(g) \subseteq \mathcal{R}_j$. For any i, let $\mathcal{R}'_i$ denote the rules that $\mathcal{R}_i$ depends on, i.e., $\mathcal{R}'_i = \bigcup_{\mathcal{R}_i \geq_d \mathcal{R}_j} \mathcal{R}_j$.

Clearly, a cycle only consists of dependency pairs from one $\mathcal{R}_i$. Thus, in Thm. 7 we only have to regard cycles $\mathcal{P}$ with 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 consider $\mathcal{R}$-chains, not just $\mathcal{R}_i$- or $\mathcal{R}'_i$-chains.[1]

Thm. 8 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

[1] To see this, consider Toyama's TRS [26] 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.

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 8 (Improved Termination Proofs with DPs). *Let $\mathcal{R}_1, \ldots, \mathcal{R}_n$ be a* separation *of $\mathcal{R}$. $\mathcal{R}$ is terminating if for all $1 \leq i \leq n$ and any cycle $\mathcal{P}$ of $\mathcal{R}$'s (estimated) dependency graph 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*

(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 $l \to r \in \mathcal{R}'_i$

Example 9. This TRS of [23] shows that Thm. 8 not only increases efficiency, but also leads to a more powerful method. Here, $\mathsf{int}(\mathsf{s}^n(0), \mathsf{s}^m(0))$ computes $[\mathsf{s}^n(0), \mathsf{s}^{n+1}(0), \ldots, \mathsf{s}^m(0)]$, nil is the empty list, and cons represents list insertion.

$$\mathsf{intlist}(\mathsf{nil}) \to \mathsf{nil} \quad (10) \qquad \mathsf{intlist}(\mathsf{cons}(x, y)) \to \mathsf{cons}(\mathsf{s}(x), \mathsf{intlist}(y)) \quad (13)$$

$$\mathsf{int}(\mathsf{s}(x), 0) \to \mathsf{nil} \quad (11) \qquad \mathsf{int}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{intlist}(\mathsf{int}(x, y)) \quad (14)$$

$$\mathsf{int}(0, 0) \to \mathsf{cons}(0, \mathsf{nil}) \quad (12) \qquad \mathsf{int}(0, \mathsf{s}(y)) \to \mathsf{cons}(0, \mathsf{int}(\mathsf{s}(0), \mathsf{s}(y))) \quad (15)$$

The TRS is separated into the intlist-rules $\mathcal{R}_1$ and the int-rules $\mathcal{R}_2 >_d \mathcal{R}_1$. The constraints of Thm. 7 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. In contrast, by using Thm. 8, only $\mathcal{R}'_1 = \mathcal{R}_1$ must be weakly decreasing when examining $\mathcal{P}$. These constraints are satisfied by the embedding order using the argument filtering $\pi(\mathsf{cons}) = [2]$, $\pi(\mathsf{intlist}) = \pi(\mathsf{INTLIST}) = 1$, $\pi(s) = [1]$.

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 $\pi(\mathsf{cons}) = 1$, $\pi(\mathsf{INT}) = 2$). 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 where both dependency pairs of $\mathcal{R}_2$ are strictly decreasing. So if one only considers cycles or only uses Urbain's modularity result [27], then Ex. 9 fails with simplification orders. Instead, both refinements should be combined as in Thm. 8.

4 Improved Innermost Termination Proofs

Innermost termination is easier to prove than termination: the innermost dependency graph has less arcs than the dependency graph and we only require $l \succsim r$ for *usable* instead of *all* rules. In Sect. 3 we showed that for termination, it suffices to require $l \succsim r$ only for rules of $\mathcal{R}'_i$ if the current cycle consists of $\mathcal{R}_i$-dependency pairs. Still, $\mathcal{R}'_i$ is a superset of the usable rules. Now we present an improvement of Thm. 7 for innermost termination to reduce the 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

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

[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.

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, as in 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.

Definition 10 (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 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 $\mathcal{V}(t) \subseteq \mathcal{V}(s)$, let $\mathcal{U}_s(t, \pi) = \varnothing$ if t is a subterm of s. Otherwise, $\mathcal{U}_s(f(t_1, \ldots, t_n), \pi) = Rls(f) \;\cup\; \bigcup_{l \to r \in Rls(f)} \mathcal{U}(r, \pi) \;\cup\; \bigcup_{j \in \mathcal{R}egPos_\pi(f)} \mathcal{U}_s(t_j, \pi)$. Moreover, for any set $\mathcal{P}$ of dependency pairs, let $\mathcal{U}(\mathcal{P}, \pi) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}_s(t, \pi)$.*

Now we can refine the innermost termination technique of Thm. 7 (c) and (d) to the following one where the set of usable rules is reduced significantly.

Theorem 11 (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*

(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 $l \to r \in \mathcal{U}(\mathcal{P}, \pi)$

Example 12. This TRS of [17] for list reversal shows the advantages of Thm. 11.

$$
\begin{array}{rlrl}
\mathsf{rev}(\mathsf{nil}) \to \mathsf{nil} & & \mathsf{rev}(\mathsf{cons}(x, l)) \to \mathsf{cons}(\mathsf{rev1}(x, l), \mathsf{rev2}(x, l)) \\
\mathsf{rev1}(x, \mathsf{nil}) \to x & & \mathsf{rev1}(x, \mathsf{cons}(y, l)) \to \mathsf{rev1}(y, l) \\
\mathsf{rev2}(x, \mathsf{nil}) \to \mathsf{nil} & & \mathsf{rev2}(x, \mathsf{cons}(y, l)) \to \mathsf{rev}(\mathsf{cons}(x, \mathsf{rev}(\mathsf{rev2}(y, l))))
\end{array}
$$

For innermost termination with Thm. 7, from the cycle of the REV and REV2-dependency pairs, we get inequalities for the dependency pairs and $\pi(l) \succsim \pi(r)$ for all rules $l \to r$, since all rules are usable. But standard reduction pairs based on recursive path orders possibly with status (RPOS), Knuth-Bendix orders (KBO), or polynomial orders do not satisfy these constraints for any argument filtering. In contrast, with Thm. 11 and a filtering with $\pi(\mathsf{cons}) = [2], \pi(\mathsf{REV}) = \pi(\mathsf{rev}) = 1, \pi(\mathsf{REV2}) = \pi(\mathsf{rev2}) = 2$, 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. 8 and 11 indeed improve upon Thm. 7 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).

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, 11]. 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 13 (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 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*
  $\mathcal{P} \uplus \{s\mu \to t\mu \mid \mu = mgu(\textsc{cap}_v(w), s), v \to w \in \mathcal{P}, s\mu, v\mu \text{ are normal forms}\}$.

Theorem 14 (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, 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 $DP(\mathcal{R})'$. Here, we estimate (innermost) dependency graphs as in Sect. 2 when computing the constraints.*

By Thm. 14, these transformations never complicate 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). On the other hand, the transformations are often crucial for the success of the proof.

Example 15. In this TRS [3], the minus-rules of Ex. 1 are extended with

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

When trying to prove innermost termination, no simplification order satisfies the constraints of Thm. 11 for the following cycle.

$$\mathsf{QUOT}(x, \mathsf{s}(y)) \to \mathsf{IF}(\mathsf{le}(\mathsf{s}(y), x), x, \mathsf{s}(y)) \quad (16) \qquad \mathsf{IF}(\mathsf{true}, x, y) \to \mathsf{QUOT}(\mathsf{minus}(x, y), y) \quad (17)$$

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 above transformations.

By i-narrowing, we perform a case analysis on how the le-term in (16) 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

$$\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{IF}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \tag{18}$$

which forms a new cycle with (17). Now we perform i-instantiation of (17) and see that x and y must be of the form $\mathsf{s}(\ldots)$. So (17) is replaced by the new pair

$$\mathsf{IF}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{QUOT}(\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y)) \tag{19}$$

that forms a cycle with (18). Finally, we do a rewriting step on (19) and obtain

$$\mathsf{IF}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{QUOT}(\mathsf{minus}(x, y), \mathsf{s}(y)) \tag{20}$$

The constraints from the resulting cycle $\{(18), (20)\}$ (and from all other cycles) can be solved by $\pi(\mathsf{minus}) = \pi(\mathsf{QUOT}) = 1$, $\pi(\mathsf{IF}) = 2$, and the embedding order.

For innermost termination, Def. 13 and Thm. 14 extend the results of [2, 11] by permitting these transformations for a larger set of TRSs. In [11], narrowing a pair $s \to t$ was not permitted if t unifies with the left-hand side of some dependency pair. Rewriting dependency pairs was only allowed if all usable rules for the current cycle were non-overlapping. Finally, when instantiating dependency pairs, in contrast to [11] 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 crucial problem 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, 8, 23] 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, let $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{g}(\mathsf{b}),\ \mathsf{g}(x) \to \mathsf{f}(x)\}$. 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 and 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. 15 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. 15, when proving innermost termination of the QUOT-cycle, we may assume innermost termination of minus and thus, the rewrite step from (19) to (20) was safe.

Definition 16 (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. 13. 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

Theorem 17 (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.*

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 always regards SCCs first and then, one builds new SCCs from the remaining pairs which were not strictly decreasing (Sect. 2) [15]. 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 before orienting constraints with reduction pairs. Since there are exponentially many argument filterings, we now show how to search for suitable filterings efficiently. For every cycle $\mathcal{P}$, we compute small sets $\Pi^t(\mathcal{P})$ and $\Pi^i(\mathcal{P})$ containing all filterings which could possibly satisfy the constraints for termination or innermost termination, respectively. A corresponding algorithm was presented in [15] for termination proofs w.r.t. Thm. 7. We now develop such an algorithm for the improved dependency pair approach from Thm. 8 and 11. In particular for Thm. 11, 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 [15], we process them according to an efficient depth-first strategy.

Let $\mathcal{RP}$ be a class of reduction pairs (e.g., $\mathcal{RP}$ may contain all LPOs with arbitrary precedences). 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 [15] 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 if π is defined on all function symbols needed. For two (partial) argument filterings, we define $\pi \sqsubseteq \pi'$ iff $DOM(\pi) \subseteq DOM(\pi')$ and $\pi(f) = \pi'(f)$ for all $f \in DOM(\pi)$. 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 $\pi \sqsubset \pi'$.

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. 8. To this end, we have to *extend* partial argument filterings.

Definition 18 (Ex_f, $\Pi^t(\mathcal{P})$). *For $f \in \mathcal{D}$, $Ex_f(\pi)$ 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 a $\pi \in \Pi(\mathcal{P})$ 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. 8 are satisfiable with π_1. In case of success, we do not compute further elements of $\Pi^t(\mathcal{P})$. Otherwise, we determine $\Pi^t(\mathcal{P})$'s next element, etc. The advantage of this approach is that $\Pi(\mathcal{P})$ is usually 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. 11.

We 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 $\mathcal{P}$, we use a relation "$\vdash_\mathcal{P}$" to 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 19 ($\vdash_{\mathcal{P}}$, $\Pi^i(\mathcal{P})$)**.** *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 a 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 $Rls(f) \subseteq \mathcal{U}(\mathcal{P}, \pi)$. Note that $\vdash_{\mathcal{P}}$ is confluent and well founded, since the labellings increase in every $\vdash_{\mathcal{P}}$-step. 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 a $\pi \in \Pi(\mathcal{P})$. If $Rls(f) \subseteq \mathcal{U}(\mathcal{P}, \pi)$, then π is extended to make f's rules weakly decreasing. If by this extension, the rules for g become usable, then we have to extend with Ex_g afterwards, etc.

Thm. 20 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. 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 20. *Let $\mathcal{P}$ be a cycle. If the constraints (a) and (b) of Thm. 8 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. 11 for innermost termination are satisfied for some reduction pair from $\mathcal{RP}$ and argument filtering π', then $\pi \sqsubseteq \pi'$ for some $\pi \in \Pi^i(\mathcal{P})$.*

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, 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 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 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 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. Our experiments show that all examples in the collections of [3, 8, 23] 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 $\pi(l) \succsim \pi(r)$ for rules $l \to r$, one may apply more complex quasi-orders. The advantage is that now $\Pi(\mathcal{P})$ is much smaller. Our experiments show that due to the improvements in Sect. 3 and 4, this heuristic succeeds for more than 96 % of those examples of [3, 8, 23] where a full LPO was successful, while reducing runtimes by 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. If we are not able to orient the rules of f, then we mark f as not orientable. Finally, 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).

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. The bottom-up algorithm reduces the search space enormously. The number of TRSs from [3, 8, 23] 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 presented improvements of the dependency pair approach which significantly reduce the sets of constraints $\pi(l) \succsim \pi(r)$ for 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 gave an algorithm for computing argument filterings which is tailored to the improvements presented before. Finally, we developed heuristics to increase efficiency which proved successful in large case studies.

We implemented these results in the system AProVE (Automated Program Verification Environment), 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. 16
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. 16.
   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, 8, 23] (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[3], RWTH Aachen, 2001.
4. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambr. Univ. Pr., 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. Termination of rewriting. *J. Symbolic Comp.*, 3:69–116, 1987.
8. N. Dershowitz. 33 examples of termination. In *Proc. French Spring School of Theoretical Computer Science*, LNCS 909, pages 16–26, 1995.
9. 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.
10. 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.
11. J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Appl. Algebra in Engineering, Communication and Computing*, 12(1,2):39–72, 2001.
12. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
13. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. Technical Report AIB-2003-04[3], RWTH Aachen, Germany, 2003.
14. B. Gramlich. On proving termination by innermost termination. In *Proc. 7th RTA*, LNCS 1103, pages 97–107, 1996.
15. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. In *Proc. 19th CADE*, LNAI 2741, 2003.
16. N. Hirokawa and A. Middeldorp. Tsukuba termination tool. In *Proc. 14th RTA*, LNCS 2706, pages 311–320, 2003.
17. G. Huet and J.-M. Hullot. Proofs by induction in equational theories with constructors. *Journal of Computer and System Sciences*, 25:239–299, 1982.
18. K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proc. 1st PPDP*, LNCS 1702, pages 48–62, 1999.
19. 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.
20. A. Middeldorp. Approximating dependency graphs using tree automata techniques. In *Proc. IJCAR 2001*, LNAI 2083, pages 593–610, 2001.
21. E. Ohlebusch. Hierarchical termination revisited. *IPL*, 84(4):207–214, 2002.
22. 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.
23. J. Steinbach. Automatic termination proofs with transformation orderings. In *Proc. 6th RTA*, LNCS 914, pages 11–25, 1995. Full version appeared as Technical Report SR-92-23, Universität Kaiserslautern, Germany.
24. J. Steinbach. Simplification orderings: History of results. *Fund. I.*, 24:47–87, 1995.
25. R. Thiemann and J. Giesl. Size-change termination for term rewriting. In *Proc. 14th RTA*, LNCS 2706, pages 264–278, 2003.
26. Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.
27. X. Urbain. Automated incremental termination proofs for hierarchically defined term rewriting systems. In *Proc. IJCAR 2001*, LNAI 2083, pages 485–498, 2001.

[3] Available from `http://aib.informatik.rwth-aachen.de`