Mechanizing and Improving Dependency Pairs *

Jürgen Giesl (giesl@informatik.rwth-aachen.de),
René Thiemann (thiemann@informatik.rwth-aachen.de) and
Peter Schneider-Kamp (psk@informatik.rwth-aachen.de)
LuFG Informatik 2, RWTH Aachen, Ahornstr. 55, 52074 Aachen, Germany

Stephan Falke (spf@cs.unm.edu)
Computer Science Dept., University of New Mexico, Albuquerque, NM 87131, USA

Abstract. The dependency pair technique [1, 11, 12] is a powerful method for automated termination and innermost termination proofs of term rewrite systems (TRSs). For any TRS, it generates inequality constraints that have to be satisfied by well-founded orders. We improve the dependency pair technique 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 mechanize the approach, we show how transformations 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.

Keywords: termination, term rewriting, dependency pairs

1. Introduction

Termination is an essential property of term rewrite systems. Before the development of *dependency pairs* in the mid-90's, most methods to prove termination of TRSs automatically used *simplification orders* [7], where a term is greater than its proper subterms (*subterm property*). Examples for simplification orders include lexicographic or recursive path orders possibly with status (RPOS [7]), the Knuth-Bendix order (KBO [28]), and many polynomial orders [31]. 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 [1, 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 increased significantly.

EXAMPLE 1. *The following TRS from [1] is not simply terminating, since the left-hand side of* div *'s last rule is embedded in the right-hand*

* Supported by the Deutsche Forschungsgemeinschaft DFG, grant GI 274/5-1.

side if y is instantiated with $\mathsf{s}(x)$*. So approaches for termination proofs based on simplification orders fail, while the example is easy to handle with dependency pairs.*

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

In Sect. 2, we recapitulate the dependency pair technique for termination and innermost termination proofs. Then we present new results which *improve* the technique significantly: Sect. 3 shows that for termination, it suffices to require only the same constraints as for innermost termination. Sect. 4 introduces a refinement to reduce the constraints (for both termination and innermost termination) even more by combining the concepts of "usable rules" and "argument filtering". Sect. 5 presents techniques for transforming dependency pairs in order to simplify (innermost) termination proofs further. Compared to previous such transformations, Sect. 5 weakens their applicability conditions, introduces an additional new transformation, and shows how to combine the transformations with the improvements of Sect. 3 and 4.

The remainder of the paper is concerned with *mechanizing* dependency pairs. To this end, we show how to solve the indeterminisms and search problems of the dependency pair technique efficiently. One problem is the question when and how often to apply the dependency pair transformations discussed above. Therefore, in Sect. 5 we also show how to use these transformations in practice in order to guarantee the termination of their application without compromising their power.

For automated (innermost) termination proofs, one tries to solve the constraints generated by the dependency pair technique with standard orders like RPOS, KBO, or polynomial orders. However, if one uses classical simplification orders, then the constraints should first be preprocessed by an *argument filtering* in order to benefit from the full power of dependency pairs. Since the number of possible argument filterings is exponential, the search for a suitable filtering is one of the main problems when automating dependency pairs. We present an algorithm to generate argument filterings efficiently for our improved dependency pair technique in Sect. 6. Instead of using orders like RPOS or KBO in combination with argument filterings, one can also apply polynomial orders, which already simulate the concept of argument filtering themselves. In Sect. 7 we show how to mechanize the dependency pair approach using polynomial orders efficiently.

Our improvements and algorithms are implemented in our termination prover AProVE [17]. In Sect. 8 we give empirical results which

show that they are extremely successful in practice. Thus, the contributions of this paper are also very helpful for other current tools which use dependency pairs (e.g., CiME [6], TORPA [45], TPA [29], TTT [24]). Dependency pairs can also be combined with other termination techniques (e.g., [40] integrates dependency pairs and the *size-change principle* from termination analysis of functional [32] and logic programs [8]). Moreover, the systems TALP [36] and AProVE also use dependency pairs for termination proofs of logic programs. So techniques to mechanize and to improve dependency pairs are useful for termination analysis of other kinds of programming languages as well. Of course, dependency pairs are not the only successful method for automated termination proofs of non-simply terminating TRSs. Other powerful methods include *semantic labelling* [44], *match-bounds* [9], and the *monotonic semantic path order* [5]. For that reason, several tools (including AProVE) also offer other termination techniques, possibly in combination with dependency pairs.

2. Dependency Pairs

We briefly present the *dependency pair* method and refer to [1, 11, 12, 15, 16, 22, 25, 34, 35] for refinements and motivations. Here, we use the new formulation of [15], where the method is presented as a general *framework* for termination proofs which combines several separate sub-techniques. This formulation was inspired by the cycle analysis algorithm of [25] and it is related to the constraint-based approach of [4, Chapter 7]. A main advantage of this formulation is that one can incorporate other termination techniques into the cycle analysis algorithm of [25] which leads to a substantial increase in modularity and power. After presenting the structure of the dependency pair framework in Sect. 2.1, we introduce two of the main components of the framework in Sect. 2.2 and 2.3: the *dependency graph processor* and the *reduction pair processor*.

2.1. The Dependency Pair Framework

We assume familiarity with term rewriting (see, e.g., [3]). For a signature $\mathcal{F}$ and a set of variables $\mathcal{V}$, let $\mathcal{T}(\mathcal{F}, \mathcal{V})$ denote the set of terms over $\mathcal{F}$ and $\mathcal{V}$. For a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined symbols* $\mathcal{D}_\mathcal{R}$ are the root symbols of the left-hand sides of rules. We restrict ourselves to finite signatures and TRSs. For every defined symbol $f \in \mathcal{D}_\mathcal{R}$, we extend the signature $\mathcal{F}$ by a fresh *tuple symbol* $f^\sharp$, where $f^\sharp$ has the same arity as f. To ease readability, in the examples we usually adopt

the original notation of [1] where tuple symbols were written with capital letters, i.e., we often write F for $f^\sharp$, etc. If $t = g(t_1, \ldots, t_m)$ with $g \in \mathcal{D}_\mathcal{R}$, 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 rule $l^\sharp \to t^\sharp$ is a* dependency pair *of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted by $DP(\mathcal{R})$.*

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

$$\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y) \quad (1)$$
$$\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y) \quad (2)$$
$$\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y)) \quad (3)$$

To use dependency pairs for (innermost) termination proofs, we need the notion of (innermost) *chains*. Intuitively, a dependency pair corresponds to a (possibly recursive) function call and a chain of dependency pairs represents a sequence of calls that can occur during a reduction. We always assume that different occurrences of dependency pairs are variable disjoint and consider substitutions whose domains may be infinite. In the following, $\mathcal{P}$ is usually a set of dependency pairs.

DEFINITION 3 (Chain). *Let $\mathcal{P}, \mathcal{R}$ be TRSs. A (possibly infinite) sequence of pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ from $\mathcal{P}$ is a* $(\mathcal{P}, \mathcal{R})$-chain *iff there is a substitution σ with $t_i\sigma \to_\mathcal{R}^* s_{i+1}\sigma$ for all i. The chain is* minimal *if all $t_i\sigma$ are terminating w.r.t. $\mathcal{R}$. The chain is an* innermost $(\mathcal{P}, \mathcal{R})$*-chain iff $t_i\sigma \xrightarrow{\mathsf{i}}_\mathcal{R}^* s_{i+1}\sigma$ and $s_i\sigma$ is in normal form w.r.t. $\mathcal{R}$ for all i. Here, "$\xrightarrow{\mathsf{i}}_\mathcal{R}$" denotes innermost reductions. An innermost $(\mathcal{P}, \mathcal{R})$-chain as above is* minimal *if all $t_i\sigma$ are innermost terminating w.r.t. $\mathcal{R}$.*

In Ex. 1, the following sequence is a minimal (innermost) chain

$$\mathsf{DIV}(\mathsf{s}(x_1), \mathsf{s}(y_1)) \to \mathsf{DIV}(\mathsf{minus}(x_1, y_1), \mathsf{s}(y_1)), \quad (4)$$
$$\mathsf{DIV}(\mathsf{s}(x_2), \mathsf{s}(y_2)) \to \mathsf{DIV}(\mathsf{minus}(x_2, y_2), \mathsf{s}(y_2)) \quad (5)$$

since $\mathsf{DIV}(\mathsf{minus}(x_1, y_1), \mathsf{s}(y_1))\sigma \to_\mathcal{R}^* \mathsf{DIV}(\mathsf{s}(x_2), \mathsf{s}(y_2))\sigma$ holds for a suitable substitution σ. For example, σ could instantiate x_1 with $\mathsf{s}(\mathsf{0})$ and y_1, x_2, y_2 with $\mathsf{0}$. While there are chains of arbitrary finite length in Ex. 1, we have no infinite chains. We obtain the following sufficient and necessary criterion for termination and innermost termination.

THEOREM 4 (Termination Criterion [1]). *A TRS $\mathcal{R}$ is terminating iff there is no infinite minimal $(DP(\mathcal{R}), \mathcal{R})$-chain. $\mathcal{R}$ is innermost terminating iff there is no infinite minimal innermost $(DP(\mathcal{R}), \mathcal{R})$-chain.*

To prove absence of infinite minimal (innermost) chains automatically, we consider so-called *dependency pair problems* ("DP problems")[1]. A DP problem consists of two TRSs $\mathcal{P}$ and $\mathcal{R}$ (where initially, $\mathcal{P} = DP(\mathcal{R})$) and a flag $e \in \{\mathbf{t}, \mathbf{i}\}$ for "**t**ermination" or "**i**nnermost termination". Instead of "$(\mathcal{P}, \mathcal{R})$-chains" we also speak of "$(\mathcal{P}, \mathcal{R}, \mathbf{t})$-chains" and instead of "innermost $(\mathcal{P}, \mathcal{R})$-chains" we speak of "$(\mathcal{P}, \mathcal{R}, \mathbf{i})$-chains". Our goal is to show that there is no infinite minimal $(\mathcal{P}, \mathcal{R}, e)$-chain. In this case, we call the problem *finite*. So Thm. 4 can be reformulated as follows: A TRS $\mathcal{R}$ is terminating iff the DP problem $(DP(\mathcal{R}), \mathcal{R}, \mathbf{t})$ is finite and it is innermost terminating iff $(DP(\mathcal{R}), \mathcal{R}, \mathbf{i})$ is finite.

A DP problem $(\mathcal{P}, \mathcal{R}, e)$ that is not finite is called *infinite*. But in addition, $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is already *infinite* whenever $\mathcal{R}$ is not terminating and $(\mathcal{P}, \mathcal{R}, \mathbf{i})$ is already infinite if $\mathcal{R}$ is not innermost terminating. The reason for this non-symmetric definition of "finite" and "infinite" is that in this way there are more finite resp. infinite DP problems and therefore, it becomes easier to detect (in)finiteness of a problem.[2] While the initial DP problem $(DP(\mathcal{R}), \mathcal{R}, e)$ is either finite or infinite, other DP problems $(\mathcal{P}, \mathcal{R}, e)$ which can occur in termination proofs can be both finite and infinite. For example, the DP problem $(\mathcal{P}, \mathcal{R}, e)$ with $\mathcal{P} = \{\mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(x)\}$ and $\mathcal{R} = \{\mathsf{f}(\mathsf{s}(x)) \to \mathsf{f}(x), \mathsf{a} \to \mathsf{a}\}$ is finite since there is no infinite minimal $(\mathcal{P}, \mathcal{R}, e)$-chain, but also infinite since $\mathcal{R}$ is not (innermost) terminating.

Such DP problems do not cause difficulties. If one detects an infinite problem during a termination proof, one can always abort the proof, since termination has been disproved (if all proof steps were "complete", i.e., if they preserved the termination behavior). If the problem is both finite and infinite, then even if one only considers it as being finite, the proof is still correct, since then there exists another resulting DP problem which is infinite and not finite. The reason is that by Thm. 4, non-termination implies that there is an infinite minimal chain. Indeed, when proving termination of the TRS $\mathcal{R}$ above one also obtains a DP problem with the infinite minimal chain $\mathsf{A} \to \mathsf{A}$, $\mathsf{A} \to \mathsf{A}, \ldots$

Termination techniques should now operate on DP problems instead of TRSs. We refer to such techniques as *dependency pair processors* ("DP processors"). Formally, a DP processor is a function *Proc* which takes a DP problem as input and returns a new set of DP problems which then have to be solved instead. Alternatively, it can also return "no". A DP processor *Proc* is *sound* if for all DP problems d, d is finite

[1] To ease readability we use a simpler definition of *DP problems* than [15], since this simple definition suffices for the presentation of the new results of this paper.

[2] That a DP problem is already "infinite" if $\mathcal{R}$ is not terminating is required for the completeness of the dependency pair transformations in Sect. 5 (cf. Ex. 32 in Sect. 5.1).

whenever $Proc(d)$ is not "no" and all DP problems in $Proc(d)$ are finite. *Proc* is *complete* if for all DP problems d, d is infinite whenever $Proc(d)$ is "no" or when $Proc(d)$ contains an infinite DP problem.

Soundness of *Proc* is required to prove termination (in particular, to conclude that d is finite if $Proc(d) = \varnothing$). Completeness is needed to prove non-termination (in particular, to conclude that d is infinite if $Proc(d) = \mathsf{no}$). Completeness also ensures that one does not transform non-infinite DP problems into infinite ones (i.e., applying the processor does not "harm" – but of course it could still have a negative impact on the success or efficiency of the termination proof attempt).

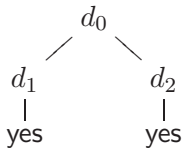
Cor. 5 introduces the DP framework. The idea is to start with the initial DP problem $(DP(\mathcal{R}), \mathcal{R}, e)$, where e depends on whether one wants to prove termination or innermost termination. Then this problem is transformed repeatedly by sound DP processors. If the final processors return empty sets of DP problems, then termination is proved. If one of the processors returns "no" and all processors used before were complete, then one has disproved termination of the TRS $\mathcal{R}$.

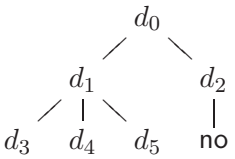
COROLLARY 5 (Dependency Pair Framework). *Let $\mathcal{R}$ be a TRS. We construct a tree whose nodes are labelled with DP problems or "yes" or "no" and whose root is labelled with $(DP(\mathcal{R}), \mathcal{R}, e)$ where $e \in \{\mathbf{t}, \mathbf{i}\}$. For every inner node labelled with d, there is a sound DP processor Proc satisfying one of the following conditions:*

- *$Proc(d) = \mathsf{no}$ and the node has just one child, labelled with "no"*
- *$Proc(d) = \varnothing$ and the node has just one child, labelled with "yes"*
- *$Proc(d) \neq \mathsf{no}$, $Proc(d) \neq \varnothing$, and the children of the node are labelled with the DP problems in $Proc(d)$*

If all leaves of the tree are labelled with "yes", then $\mathcal{R}$ is terminating (if $e = \mathbf{t}$) resp. innermost terminating (if $e = \mathbf{i}$). Otherwise, if there is a leaf labelled with "no" and if all processors used on the path from the root to this leaf are complete, then $\mathcal{R}$ is not terminating (if $e = \mathbf{t}$) resp. not innermost terminating (if $e = \mathbf{i}$).

EXAMPLE 6. *If d_0 is the initial problem $(DP(\mathcal{R}), \mathcal{R}, e)$, if $Proc_0$, $Proc_1$, $Proc_2$ are sound DP processors, and if $Proc_0(d_0) = \{d_1, d_2\}$, $Proc_1(d_1) = \varnothing$, and $Proc_2(d_2) = \varnothing$, then one could obtain the first tree below and conclude (innermost) termination.*





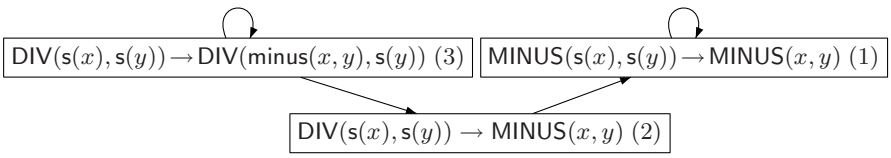
But if $Proc_1(d_1) = \{d_3, d_4, d_5\}$ *and* $Proc_2(d_2) = \mathsf{no}$*, one could obtain the second tree. If both* $Proc_0$ *and* $Proc_2$ *are complete, then now one could conclude non-termination.*

2.2. The Dependency Graph Processor

We first introduce a processor to decompose a DP problem into several sub-problems. To this end, one tries to determine which pairs can follow each other in chains by constructing a so-called *dependency graph*.

DEFINITION 7 (Dependency Graph). *For a problem* $(\mathcal{P}, \mathcal{R}, e)$*, the nodes of the* $(\mathcal{P}, \mathcal{R}, e)$-dependency graph *are the pairs of* $\mathcal{P}$ *and there is an arc from* $s \to t$ *to* $v \to w$ *iff* $s \to t, v \to w$ *is an* $(\mathcal{P}, \mathcal{R}, e)$*-chain.*

In Ex. 1, we have the following dependency graph for both $e \in \{\mathbf{t}, \mathbf{i}\}$.



A set $\mathcal{P}' \neq \varnothing$ of dependency pairs is a *cycle* if for any $s \to t$ and $v \to w$ in $\mathcal{P}'$ there is a non-empty path from $s \to t$ to $v \to w$ in the graph which only traverses pairs of $\mathcal{P}'$. A cycle $\mathcal{P}'$ is a *strongly connected component* ("SCC") if $\mathcal{P}'$ is not a proper subset of another cycle.

Note that in standard graph terminology, a path $n_0 \Rightarrow n_1 \Rightarrow \ldots \Rightarrow n_k$ in a directed graph forms a cycle if $n_0 = n_k$ and $k \geq 1$. In our context we identify cycles with the *set* of elements that occur in it, i.e., we call $\{n_0, n_1, \ldots, n_{k-1}\}$ a cycle, cf. [12]. Since a set never contains multiple occurrences of an element, this results in several cycling paths being identified with the same set. Similarly, an "SCC" is a graph in standard graph terminology, whereas we identify an SCC with the set of elements occurring in it. Then indeed, SCCs are the same as maximal cycles.

In Ex. 1, we have the SCCs $\mathcal{P}_1 = \{(1)\}$ and $\mathcal{P}_2 = \{(3)\}$. Since we only regard finite TRSs, any infinite chain of dependency pairs corresponds to a cycle of the dependency graph. Therefore, one can prove absence of infinite chains separately for every cycle of the dependency graph. As observed in [25], to avoid an exponential blowup, one should not compute all cycles of the dependency graph, but consider the SCCs instead. Therefore, the following DP processor decomposes a DP problem into the sub-problems corresponding to the different SCCs.

THEOREM 8 (Dependency Graph Processor [1, 15, 25]). *Let* $Proc((\mathcal{P}, \mathcal{R}, e)) = \{(\mathcal{P}_1, \mathcal{R}, e), \ldots, (\mathcal{P}_n, \mathcal{R}, e)\}$, *where* $\mathcal{P}_1, \ldots, \mathcal{P}_n$ *are the SCCs of the* $(\mathcal{P}, \mathcal{R}, e)$*-dependency graph. Then Proc is sound and complete.*

The initial problem in Ex. 1 is $(\mathcal{P}, \mathcal{R}, e)$ with $\mathcal{P} = \{(1), (2), (3)\}$. The above processor transforms it into $(\{(1)\}, \mathcal{R}, e)$ and $(\{(3)\}, \mathcal{R}, e)$.

Unfortunately, the dependency graph is not computable. Therefore, for automation one constructs an *estimated* graph containing at least all arcs from the real graph. Obviously, the dependency graph processor of Thm. 8 remains sound and complete if one uses any such estimation.

Let $\text{CAP}_{\mathcal{R}}(t)$ result from replacing all subterms of t with defined root symbol (i.e., with a root symbol from $\mathcal{D}_{\mathcal{R}}$) by different fresh variables. Here, multiple occurrences of the same subterm are also replaced by pairwise different variables. Let $\text{REN}(t)$ result from replacing all occurrences of variables in t by different fresh variables (i.e., $\text{REN}(t)$ is a linear term). So $\text{CAP}_{\mathcal{R}}(\mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y))) = \mathsf{DIV}(z, \mathsf{s}(y))$ and $\text{REN}(\mathsf{DIV}(x, x)) = \mathsf{DIV}(x_1, x_2)$.

DEFINITION 9 (Estimated Dependency Graph). *For a DP problem* $(\mathcal{P}, \mathcal{R}, e)$*, the nodes of the* estimated dependency graph $\text{EDG}(\mathcal{P}, \mathcal{R})$ *are the pairs of* $\mathcal{P}$ *and there is an arc from* $s \to t$ *to* $v \to w$ *iff* $\text{REN}(\text{CAP}_{\mathcal{R}}(t))$ *and* v *are unifiable. In the* estimated innermost dependency graph $\text{EIDG}(\mathcal{P}, \mathcal{R})$ *there is an arc from* $s \to t$ *to* $v \to w$ *iff* $\text{CAP}_{\mathcal{R}}(t)$ *and* v *are unifiable by an mgu* μ *such that* $s\mu$ *and* $v\mu$ *are in normal form.*

The above estimations are sound, i.e., the EDG contains the $(\mathcal{P}, \mathcal{R}, \mathbf{t})$-dependency graph and the EIDG contains the $(\mathcal{P}, \mathcal{R}, \mathbf{i})$-dependency graph. Of course, to check whether there is an arc from $s \to t$ to $v \to w$ in E(I)DG, one has to rename the variables of $s \to t$ and $v \to w$ to make them variable disjoint. In Ex. 1, the E(I)DG is identical to the real dependency graph. Alternative improved techniques to estimate (innermost) dependency graphs can be found in [1, 14, 16, 25, 34, 35]. In particular, the EIDG in Def. 9 is a slightly weaker simplified variant of the "estimated innermost dependency graph" from [1].

2.3. The Reduction Pair Processor

To prove that a DP problem is finite, we now generate constraints which should be satisfied by some *reduction pair* $(\succsim, \succ)$ [30] consisting of a quasi-rewrite order $\succsim$ (i.e., $\succsim$ is reflexive, transitive, monotonic (closed under contexts), and stable (closed under substitutions)) and a stable well-founded order $\succ$ which is compatible with $\succsim$ (i.e., $\succsim \circ \succ \subseteq \succ$ or $\succ \circ \succsim \subseteq \succ$). However, $\succ$ need not be monotonic. For a DP problem $(\mathcal{P}, \mathcal{R}, e)$, the generated constraints ensure that at least one

rule in $\mathcal{P}$ is strictly decreasing (w.r.t. $\succ$) and all remaining rules in $\mathcal{P}$ and $\mathcal{R}$ are weakly decreasing (w.r.t. $\succsim$). Requiring $l \succsim r$ for all $l \to r \in \mathcal{R}$ ensures that in a chain $s_1 \to t_1, s_2 \to t_2, ...$ with $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$, we have $t_i\sigma \succsim s_{i+1}\sigma$ for all i. Hence, if a reduction pair satisfies these constraints, then the strictly decreasing pairs of $\mathcal{P}$ cannot occur infinitely often in chains. Thus, one can delete all these pairs from $\mathcal{P}$.

For innermost termination, a weak decrease is not required for all rules but only for the *usable rules*. These rules are a superset[3] of those rules that may be used to reduce right-hand sides of dependency pairs if their variables are instantiated with normal forms. In Ex. 1, the usable rules of dependency pair (3) are the minus-rules whereas the other dependency pairs have no usable rules.

DEFINITION 10 (Usable Rules). *For $f \in \mathcal{F}$, let $Rls_{\mathcal{R}}(f) = \{l \to r \in \mathcal{R} \mid \mathrm{root}(l) = f\}$ and let $\mathcal{R}' = \mathcal{R} \setminus Rls_{\mathcal{R}}(f)$. For any term, we define*

- $\mathcal{U}_{\mathcal{R}}(x) = \varnothing$ *for* $x \in \mathcal{V}$ *and*
- $\mathcal{U}_{\mathcal{R}}(f(t_1, ..., t_n)) = Rls_{\mathcal{R}}(f) \cup \bigcup_{l \to r \in Rls_{\mathcal{R}}(f)} \mathcal{U}_{\mathcal{R}'}(r) \cup \bigcup_{i=1}^{n} \mathcal{U}_{\mathcal{R}'}(t_i)$.

For any TRS $\mathcal{P}$, we define $\mathcal{U}_{\mathcal{R}}(\mathcal{P}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}_{\mathcal{R}}(t)$.

We want to use standard techniques to synthesize reduction pairs satisfying the constraints generated by the dependency pair technique. Most existing techniques generate monotonic orders $\succ$ like RPOS or KBO. But 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 arguments of function symbols can be eliminated. To perform this elimination, the concept of argument filtering was introduced in [1] (we use the notation of [30]).

DEFINITION 11 (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 $[i_1, \ldots, i_m]$ 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 $\mathcal{T}(\mathcal{F}_\pi, \mathcal{V})$:*

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

An argument filtering with $\pi(f) = i$ for some $f \in \mathcal{F}$ is called collapsing. *For any TRS $\mathcal{R}$, let $\pi(\mathcal{R}) = \{\pi(l) \to \pi(r) \mid l \to r \in \mathcal{R}\}$.*

[3] Improved definitions of the "usable rules" which lead to a better approximation of these rules can be found in [13, 14, 16].

For any relation $\succ$, let $\succ_\pi$ be the relation where $t \succ_\pi u$ holds iff $\pi(t) \succ \pi(u)$. In [1] it was shown that if $(\succsim, \succ)$ is a reduction pair, then $(\succsim_\pi, \succ_\pi)$ is a reduction pair as well. For any TRS $\mathcal{P}$ and any relation $\succ$, let $\mathcal{P}_\succ = \{s \to t \in \mathcal{P} \mid s \succ t\}$, i.e., $\mathcal{P}_\succ$ contains those rules of $\mathcal{P}$ which decrease w.r.t. $\succ$. Now we can define a DP processor which deletes all pairs from $\mathcal{P}$ which are strictly decreasing w.r.t. a reduction pair and an argument filtering (i.e., all pairs of $\mathcal{P}_{\succ_\pi}$).

THEOREM 12 (Reduction Pair Processor [1, 12, 25]). *Let $(\succsim, \succ)$ be a reduction pair and let π be an argument filtering. Then the following DP processor Proc is sound and complete. Here, Proc$(\,(\mathcal{P}, \mathcal{R}, e)\,) =$*

- $\{(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R}, e)\}$, *if the following conditions (a) and (b) hold:*

  (a) $\mathcal{P}_{\succ_\pi} \cup \mathcal{P}_{\succsim_\pi} = \mathcal{P}$ *and* $\mathcal{P}_{\succ_\pi} \neq \varnothing$

  (b) *either* $e = \mathbf{t}$ *and* $\mathcal{R}_{\succsim_\pi} = \mathcal{R}$
  or $e = \mathbf{i}$ *and* $\mathcal{R}_{\succsim_\pi} \supseteq \mathcal{U}_\mathcal{R}(\mathcal{P})$

- $\{(\mathcal{P}, \mathcal{R}, e)\}$, *otherwise*

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

$$\pi_1(\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y))) \succ_1 \pi_1(\mathsf{MINUS}(x, y)) \quad (6)$$

$$\pi_2(\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y))) \succ_2 \pi_2(\mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y))) \quad (7)$$

$$\pi_i(\mathsf{minus}(x, \mathsf{0})) \succsim_i \pi_i(x) \quad (8)$$

$$\pi_i(\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y))) \succsim_i \pi_i(\mathsf{minus}(x, y)) \quad (9)$$

$$\pi_i(\mathsf{div}(\mathsf{0}, \mathsf{s}(y))) \succsim_i \pi_i(\mathsf{0}) \quad (10)$$

$$\pi_i(\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y))) \succsim_i \pi_i(\mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y)))) \quad (11)$$

We use the filtering $\pi_i(\mathsf{minus}) = [1]$ which replaces all terms $\mathsf{minus}(t_1, t_2)$ by $\mathsf{minus}(t_1)$ and does not modify other function symbols. With this filtering, (6) – (11) are satisfied by the lexicographic path order (LPO) with the precedence $\mathsf{div} > \mathsf{s} > \mathsf{minus}$. So one can remove the only dependency pair from the DP problems $(\mathcal{P}_1, \mathcal{R}, \mathbf{t})$ and $(\mathcal{P}_2, \mathcal{R}, \mathbf{t})$, respectively. The remaining DP problems $(\varnothing, \mathcal{R}, \mathbf{t})$ are transformed into the empty set by the dependency graph processor of Def. 8, i.e., termination of the TRS is proved. Similarly, one can also use a collapsing filtering $\pi_i(\mathsf{minus}) = \pi_i(\mathsf{div}) = 1$ which replaces all terms $\mathsf{minus}(t_1, t_2)$ or $\mathsf{div}(t_1, t_2)$ by t_1. Then even the embedding order orients the resulting constraints.

For innermost termination, $(\mathcal{P}_1, \mathcal{R}, \mathbf{i})$ only gives rise to the constraint (6), since $\mathcal{P}_1$ has no usable rules. For $(\mathcal{P}_2, \mathcal{R}, \mathbf{i})$, the constraints (10)

and (11) are not necessary, since the div-rules are not usable. Indeed, the constraints for innermost termination are always a subset of the constraints for termination. So for TRSs where innermost termination already implies termination (e.g., locally confluent overlay systems and in particular, non-overlapping TRSs [20]), one should always use techniques for innermost termination when attempting termination proofs.

Whenever a processor modifies a DP problem, one should apply the dependency graph processor afterwards. This generalizes the strategy of the recursive SCC algorithm of [25] which was suggested for the classical dependency pair approach. Here, SCCs of the dependency graph were re-computed whenever some dependency pairs were strictly oriented and therefore removed. In the DP framework, this would correspond to a repeated alternating application of the processors in Thm. 8 and 12. However, by formulating other termination techniques as DP processors as well, they can now be incorporated into this strategy, too.

3. Improving Termination Proofs by Usable Rules

Now we improve Thm. 12 such that its constraints for termination become as simple as the ones for innermost termination.[4] As observed in [43], the following definition is useful to weaken the constraints.

DEFINITION 13 ($\mathcal{C}_\varepsilon$ [19]). *$\mathcal{C}_\varepsilon$ is the TRS* $\{\mathsf{c}(x,y) \to x, \mathsf{c}(x,y) \to y\}$ *where* c *is a new function symbol. A TRS* $\mathcal{R}$ *is* $\mathcal{C}_\varepsilon$-terminating *iff* $\mathcal{R} \cup \mathcal{C}_\varepsilon$ *is terminating. A relation* $\gtrsim$ *is* $\mathcal{C}_\varepsilon$-compatible[5] *iff* $\mathsf{c}(x,y) \gtrsim x$ *and* $\mathsf{c}(x,y) \gtrsim y$. *A reduction pair* $(\gtrsim, \succ)$ *is* $\mathcal{C}_\varepsilon$-compatible *iff* $\gtrsim$ *is* $\mathcal{C}_\varepsilon$*-compatible.*

Toyama's TRS $\mathcal{R} = \{\mathsf{f}(0,1,x) \to \mathsf{f}(x,x,x)\}$ [41] is terminating, but not $\mathcal{C}_\varepsilon$-terminating, since $\mathcal{R} \cup \mathcal{C}_\varepsilon$ has the infinite reduction $\mathsf{f}(0,1,\mathsf{c}(0,1)) \to \mathsf{f}(\mathsf{c}(0,1),\mathsf{c}(0,1),\mathsf{c}(0,1)) \to^2 \mathsf{f}(0,1,\mathsf{c}(0,1)) \to \ldots$ Thus, requiring $l \gtrsim_\pi r$ only for the usable rules is not sufficient for termination: $\mathcal{R} \cup \mathcal{C}_\varepsilon$'s only SCC $\{\mathsf{F}(0,1,x) \to \mathsf{F}(x,x,x)\}$ has no usable rules and there is a reduction pair $(\gtrsim, \succ)$ such that the dependency pair is strictly decreasing.[6] Hence, $\mathcal{R} \cup \mathcal{C}_\varepsilon$ is innermost terminating, but not terminating, since we cannot satisfy both $\mathsf{F}(0,1,x) \succ_\pi \mathsf{F}(x,x,x)$ and $l \gtrsim_\pi r$ for the $\mathcal{C}_\varepsilon$-rules.

So a reduction of the constraints in Thm. 12 is impossible in general, but it is possible if we restrict ourselves to $\mathcal{C}_\varepsilon$-compatible quasi-orders $\gtrsim$. For automation, this is not a restriction if one uses a *quasi-simplification order* $\gtrsim$ (i.e., a monotonic and stable quasi-order with

[4] Independently, Hirokawa & Middeldorp obtained a corresponding result in [22].
[5] Instead of "$\mathcal{C}_\varepsilon$-compatibility", [43] uses the notion "π expandability".
[6] For example, one can use the reduction pair $(\to^*_{DP(\mathcal{R})\cup\mathcal{R}}, \to^+_{DP(\mathcal{R})\cup\mathcal{R}})$.

the *subterm property* $f(\ldots t \ldots) \gtrsim t$ for any term t and symbol f). Thus, any quasi-simplification order orients $\mathcal{C}_\varepsilon$. A similar observation holds for polynomial orders, although polynomial orders are no quasi-simplification orders if one permits the coefficient 0 in polynomials. However, they can always be extended to orient $\mathcal{C}_\varepsilon$. For example, one could associate c with the polynomial that adds its two arguments, i.e., one could define $\mathcal{P}ol(\mathsf{c}(x, y)) = x + y$.

The first step in this direction was taken by Urbain [43]. He showed that in a hierarchy of $\mathcal{C}_\varepsilon$-terminating TRSs, one can disregard all rules occurring "later" in the hierarchy when proving termination. Hence in Ex. 1, to show the termination of minus, [43] would only require that the MINUS-dependency pair (1) is strictly decreasing and the minus-rules are weakly decreasing. Compared to the reduction pair processor of Thm. 12, the advantage is that no weak decrease of the div-rules is required anymore, since minus does not depend on div. But the constraints are still harder than the ones for innermost termination, since one requires a weak decrease for the minus-rules although they are not *usable* for the MINUS-dependency pair. We will improve this approach further and show in Thm. 17 that even for termination, it suffices only to require a weak decrease for the usable rules. So compared to [43], our result leads to significantly less constraints for termination proofs.

Moreover, due to the restriction to $\mathcal{C}_\varepsilon$-termination, [43] could not use the full power of dependency graphs. For example, recent improved dependency graph estimations [25, 35] can detect that the dependency graph for Toyama's TRS $\mathcal{R}$ has no SCC and thus, it is terminating. But since it is not $\mathcal{C}_\varepsilon$-terminating, it cannot be handled by [43]. In contrast, our result can be combined with arbitrary estimations of dependency graphs. More precisely, before applying the new reduction pair processor of Thm. 17, one can use any other DP processor (e.g., the dependency graph processor with any sound graph estimation). In this way, one can also prove termination of non-$\mathcal{C}_\varepsilon$-terminating TRSs.

To prove that it suffices to regard the usable rules in termination proofs, we show that for every minimal $(\mathcal{P}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \ldots$, there exists a substitution σ such that $t_i\sigma$ reduces to $s_{i+1}\sigma$ using only the rules of $\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon$. In other words, every minimal $(\mathcal{P}, \mathcal{R})$-chain is also a $(\mathcal{P}, \mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon)$-chain. However, the resulting $(\mathcal{P}, \mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon)$-chain is not necessarily minimal.

For example, let $\mathcal{P}$ consist of the DIV-dependency pair (3). Then $\mathcal{U}_\mathcal{R}(\mathcal{P})$ only contains the minus-rules. Two (variable-renamed) occurrences of (3) (like (4) and (5)) form a minimal chain, as $\mathsf{DIV}(\mathsf{minus}(x_1, y_1), \mathsf{s}(y_1))\sigma \to_\mathcal{R}^* \mathsf{DIV}(\mathsf{s}(x_2), \mathsf{s}(y_2))\sigma$ holds for some σ (e.g., $\sigma(x_1) = \mathsf{s}(\mathsf{0})$, $\sigma(y_1) = \mathsf{div}(\mathsf{0}, \mathsf{s}(\mathsf{0}))$, $\sigma(x_2) = \sigma(y_2) = \mathsf{0}$). If one uses this particular substitution σ, then one indeed needs the non-usable rule $\mathsf{div}(\mathsf{0}, \mathsf{s}(y)) \to$

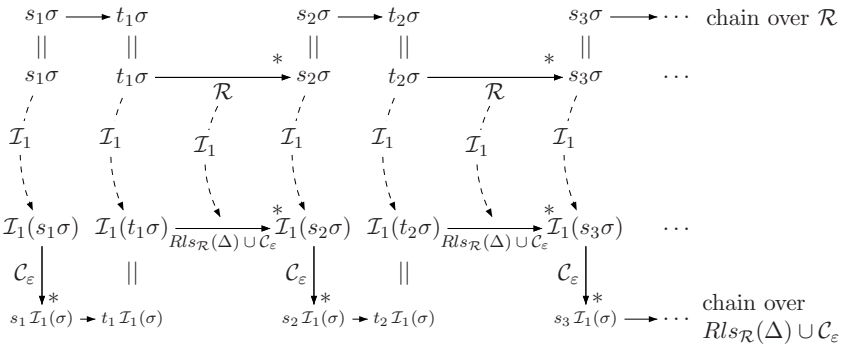


Figure 1. Transformation of chains

$\mathsf{0}$ to reduce $\sigma(y_1)$ to $\mathsf{0}$, i.e., to reduce $\mathsf{DIV}(\mathsf{minus}(x_1, y_1), \mathsf{s}(y_1))\sigma$ to $\mathsf{DIV}(\mathsf{s}(x_2), \mathsf{s}(y_2))\sigma$. However, we will show that for any σ, there is also a substitution $\mathcal{I}_1(\sigma)$ such that $\mathsf{DIV}(\mathsf{minus}(x_1, y_1), \mathsf{s}(y_1))\,\mathcal{I}_1(\sigma)$ reduces to $\mathsf{DIV}(\mathsf{s}(x_2), \mathsf{s}(y_2))\,\mathcal{I}_1(\sigma)$ by applying only usable rules and $\mathcal{C}_\varepsilon$-rules.

We proceed in a similar way as in the proof of [43] and in the original proofs of Gramlich [19]. More precisely, we map any $\mathcal{R}$-reduction to a reduction w.r.t. $\mathcal{U}_{\mathcal{R}}(\mathcal{P}) \cup \mathcal{C}_\varepsilon$. Let Δ contain all function symbols occurring in right-hand sides of $\mathcal{P} \cup \mathcal{U}_{\mathcal{R}}(\mathcal{P})$ (i.e., all *usable symbols* of $\mathcal{P}$). Thus, $\mathcal{U}_{\mathcal{R}}(\mathcal{P}) = Rls_{\mathcal{R}}(\Delta)$ (where $Rls_{\mathcal{R}}(\Delta) = \bigcup_{f \in \Delta} Rls_{\mathcal{R}}(f)$). So for $\mathcal{P} = \{(3)\}$, we have $\Delta = \{\mathsf{DIV}, \mathsf{minus}, \mathsf{s}\}$. Our mapping $\mathcal{I}_1$ modifies the earlier mappings of [19, 43] by treating terms $g(t_1, \ldots, t_n)$ with $g \notin \Delta$ differently. Fig. 1 illustrates that by this mapping, every minimal chain over $\mathcal{R}$ corresponds to a chain over $Rls_{\mathcal{R}}(\Delta) \cup \mathcal{C}_\varepsilon$, but instead of the substitution σ one uses a different substitution $\mathcal{I}_1(\sigma)$.

Intuitively, $\mathcal{I}_1(t)$ "collects" all terms that t can be reduced to in zero or more steps. However, we only regard reductions on or below non-usable symbols, i.e., symbols that are not from Δ. To represent a collection $t_1, \ldots, t_n$ of terms by just one single term, one uses the term $\mathsf{c}(t_1, \mathsf{c}(t_2, \ \ldots\ \mathsf{c}(t_n, \mathsf{d})\ \ldots))$ with a fresh constant d.

DEFINITION 14 ($\mathcal{I}_1$). *Let $\Delta \subseteq \mathcal{F}$ and let $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a terminating term (i.e., t starts no infinite $\mathcal{R}$-reductions). We define $\mathcal{I}_1(t)$:*

$$\begin{array}{rll}
\mathcal{I}_1(x) = x & & \text{for } x \in \mathcal{V} \\
\mathcal{I}_1(f(t_1, \ldots, t_n)) = f(\mathcal{I}_1(t_1), \ldots, \mathcal{I}_1(t_n)) & & \text{for } f \in \Delta \\
\mathcal{I}_1(g(t_1, \ldots, t_n)) = \mathcal{C}omp(\{g(\mathcal{I}_1(t_1), \ldots, \mathcal{I}_1(t_n))\} \cup \mathcal{R}ed_1(g(t_1, \ldots, t_n))) & & \text{for } g \notin \Delta
\end{array}$$

where $\mathcal{R}ed_1(t) = \{\mathcal{I}_1(t') \mid t \to_{\mathcal{R}} t'\}$. Moreover, $\mathcal{C}omp(\{t\} \uplus M) = \mathsf{c}(t, \mathcal{C}omp(M))$ and $\mathcal{C}omp(\varnothing) = \mathsf{d}$, where d is a fresh constant. To make $\mathcal{C}omp$ well defined, in "$\{t\} \uplus M$" we assume that t is smaller than all terms in M w.r.t. some total well-founded order $>_{\mathcal{T}}$ on terms.

For a terminating substitution σ (i.e., $\sigma(x)$ terminates for all $x \in \mathcal{V}$), we define the substitution $\mathcal{I}_1(\sigma)$ as $\mathcal{I}_1(\sigma)\,(x) = \mathcal{I}_1(\sigma(x))$ for all $x \in \mathcal{V}$.

So for $\mathcal{P} = \{(3)\}$ and $\Delta = \{\mathsf{DIV}, \mathsf{minus}, \mathsf{s}\}$, we obtain

$$\begin{aligned}\mathcal{I}_1(\mathsf{div}(\mathsf{0},\mathsf{s}(\mathsf{0}))) &= \mathcal{C}omp(\,\{\,\mathsf{div}(\mathcal{I}_1(\mathsf{0}),\mathcal{I}_1(\mathsf{s}(\mathsf{0}))),\,\mathcal{I}_1(\mathsf{0})\,\}\,)\\ &= \mathsf{c}(\,\mathsf{div}(\mathcal{I}_1(\mathsf{0}),\mathsf{s}(\mathcal{I}_1(\mathsf{0}))),\,\mathsf{c}(\,\mathcal{I}_1(\mathsf{0}),\,\mathsf{d})\,)\\ &= \mathsf{c}(\,\mathsf{div}(\mathsf{c}(\mathsf{0},\mathsf{d}),\mathsf{s}(\mathsf{c}(\mathsf{0},\mathsf{d}))),\,\mathsf{c}(\,\mathsf{c}(\mathsf{0},\mathsf{d}),\,\mathsf{d})\,).\end{aligned}$$

So in contrast to the above substitution σ, the substitution $\mathcal{I}_1(\sigma)$ instantiates y_1 by $\mathcal{I}_1(\mathsf{div}(\mathsf{0},\mathsf{s}(\mathsf{0})))$ instead of $\mathsf{div}(\mathsf{0},\mathsf{s}(\mathsf{0}))$ and it instantiates y_2 by $\mathcal{I}_1(\mathsf{0})$ instead of $\mathsf{0}$. Now one can reduce $\mathcal{I}_1(\sigma)(y_1)$ to $\mathcal{I}_1(\sigma)(y_2)$ by applying $\mathcal{C}_\varepsilon$-rules instead of applying a non-usable div-rule. Thus, the rules of $Rls_{\mathcal{R}}(\Delta) \cup \mathcal{C}_\varepsilon$ suffice to reduce $\mathsf{DIV}(\mathsf{minus}(x_1, y_1), \mathsf{s}(y_1))\,\mathcal{I}_1(\sigma)$ to $\mathsf{DIV}(\mathsf{s}(x_2), \mathsf{s}(y_2))\,\mathcal{I}_1(\sigma)$.

Note that Def. 14 is only possible for terminating terms t, since otherwise, $\mathcal{I}_1(t)$ could be infinite. Before we prove the desired theorem, we need some additional properties of $\mathcal{C}omp$ and $\mathcal{I}_1$. We want to show that for any minimal $(\mathcal{P},\mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \ldots$ with $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$, we also have $t_i\,\mathcal{I}_1(\sigma) \to_{Rls_{\mathcal{R}}(\Delta)\cup\mathcal{C}_\varepsilon}^* s_{i+1}\,\mathcal{I}_1(\sigma)$. In contrast to the corresponding lemmas in [37, 43], Lemma 16 shows that even if the left-hand sides of dependency pairs and rules are not from $\mathcal{T}(\Delta,\mathcal{V})$, the rules of $\mathcal{R} \setminus Rls_{\mathcal{R}}(\Delta)$ are not needed to reduce $t_i\,\mathcal{I}_1(\sigma)$ to $s_{i+1}\,\mathcal{I}_1(\sigma)$. Therefore, Lemma 16 (ii) and (iii) replace equalities from the lemmas of [37, 43] by "$\to_{\mathcal{C}_\varepsilon}^*$". This is possible by including "$g(\mathcal{I}_1(t_1),\ldots,\mathcal{I}_1(t_n))$" in the definition of $\mathcal{I}_1(g(t_1,\ldots,t_n))$ for $g \notin \Delta$.

LEMMA 15 (Properties of $\mathcal{C}omp$). *If $t \in M$ then $\mathcal{C}omp(M) \to_{\mathcal{C}_\varepsilon}^+ t$.*

Proof. For $t_1 <_{\mathcal{T}} \ldots <_{\mathcal{T}} t_n$ and $1 \le i \le n$ we have $\mathcal{C}omp(\{t_1,\ldots,t_n\})$ $= \mathsf{c}(t_1,\ldots\mathsf{c}(t_i,\ldots\mathsf{c}(t_n,\mathsf{d})\ldots)\ldots) \to_{\mathcal{C}_\varepsilon}^* \mathsf{c}(t_i,\ldots\mathsf{c}(t_n,\mathsf{d})\ldots) \to_{\mathcal{C}_\varepsilon} t_i$. □

LEMMA 16 (Properties of $\mathcal{I}_1$). *Let $\Delta \subseteq \mathcal{F}$ such that $f \in \Delta$ implies $g \in \Delta$ whenever g occurs in the right-hand side of a rule from $Rls_{\mathcal{R}}(f)$. Let t, $s, t\sigma \in \mathcal{T}(\mathcal{F},\mathcal{V})$ be terminating and let σ be a terminating substitution.*

(i) If $t \in \mathcal{T}(\Delta,\mathcal{V})$ then $\mathcal{I}_1(t\sigma) = t\,\mathcal{I}_1(\sigma)$.
(ii) $\mathcal{I}_1(t\sigma) \to_{\mathcal{C}_\varepsilon}^* t\,\mathcal{I}_1(\sigma)$.
(iii) If $t \to_{\{l\to r\}} s$ by a root reduction step where $l \to r \in \mathcal{R}$ and $\mathrm{root}(l) \in \Delta$, then $\mathcal{I}_1(t) \to_{\{l\to r\}\cup\mathcal{C}_\varepsilon}^+ \mathcal{I}_1(s)$.
(iv) If $t \to_{\mathcal{R}} s$ with $\mathrm{root}(t) \notin \Delta$, then $\mathcal{I}_1(t) \to_{\mathcal{C}_\varepsilon}^+ \mathcal{I}_1(s)$.
(v) If $t \to_{\{l\to r\}} s$ where $l \to r \in \mathcal{R}$, then $\mathcal{I}_1(t) \to_{\{l\to r\}\cup\mathcal{C}_\varepsilon}^+ \mathcal{I}_1(s)$ if $\mathrm{root}(l) \in \Delta$ and $\mathcal{I}_1(t) \to_{\mathcal{C}_\varepsilon}^+ \mathcal{I}_1(s)$ otherwise.

Proof.

(i) The proof is a straightforward structural induction on t.

(ii) The proof is by structural induction on t. The only interesting case is $t = g(t_1, \ldots, t_n)$ where $g \notin \Delta$. Then we obtain

$$\begin{array}{rcll} \mathcal{I}_1(g(t_1,...,t_n)\sigma) & = & \mathcal{C}omp(\{g(\mathcal{I}_1(t_1\sigma),...,\mathcal{I}_1(t_n\sigma))\} \cup \mathcal{R}ed_1(g(t_1\sigma,...,t_n\sigma))) & \\ & \to^+_{\mathcal{C}_\varepsilon} & g(\mathcal{I}_1(t_1\sigma),...,\mathcal{I}_1(t_n\sigma)) & \text{by Lemma 15} \\ & \to^*_{\mathcal{C}_\varepsilon} & g(t_1\,\mathcal{I}_1(\sigma),\ldots,t_n\,\mathcal{I}_1(\sigma)) & \text{by induction hypothesis} \\ & = & g(t_1,\ldots,t_n)\,\mathcal{I}_1(\sigma) & \end{array}$$

(iii) We have $t = l\sigma \to_{\mathcal{R}} r\sigma = s$ and $r \in \mathcal{T}(\Delta, \mathcal{V})$ by the condition on Δ. By (ii) and (i) we get $\mathcal{I}_1(l\sigma) \to^*_{\mathcal{C}_\varepsilon} l\,\mathcal{I}_1(\sigma) \to_{\{l\to r\}} r\,\mathcal{I}_1(\sigma) = \mathcal{I}_1(r\sigma)$.

(iv) The claim follows by $\mathcal{I}_1(t) = \mathcal{C}omp(\{\ldots\} \cup \mathcal{R}ed_1(t))$, $\mathcal{I}_1(s) \in \mathcal{R}ed_1(t)$, and Lemma 15.

(v) We perform induction on the position p of the redex. If $\mathrm{root}(t) \notin \Delta$, we use (iv). If $\mathrm{root}(t) \in \Delta$ and p is the root position, we apply (iii). Otherwise, p is below the root, $t = f(t_1, \ldots, t_i, \ldots, t_n)$, $s = f(t_1, \ldots, s_i, \ldots, t_n)$, $f \in \Delta$, and $t_i \to_{\{l\to r\}} s_i$. Then the claim follows from the induction hypothesis. □

Now we show the desired theorem which improves upon Thm. 12 since the constraints are reduced significantly. Thus, it becomes easier to find a reduction pair satisfying the resulting constraints.

THEOREM 17 (Reduction Pair Processor Based on Usable Rules). *Let $(\succsim, \succ)$ be a reduction pair and π be an argument filtering. Then the DP processor Proc is sound and complete. Here, Proc$(\,(\mathcal{P}, \mathcal{R}, e)\,) =$*

- $\{(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R}, e)\}$, *if the following conditions (a) and (b) hold:*
  - *(a)* $\mathcal{P}_{\succ_\pi} \cup \mathcal{P}_{\succsim_\pi} = \mathcal{P}$ *and* $\mathcal{P}_{\succ_\pi} \neq \varnothing$
  - *(b)* *either* $e = \mathbf{t}$ *and* $\mathcal{R}_{\succsim_\pi} \supseteq \mathcal{U}_\mathcal{R}(\mathcal{P})$ *and* $\succsim$ *is* $\mathcal{C}_\varepsilon$*-compatible*
    or $e = \mathbf{i}$ *and* $\mathcal{R}_{\succsim_\pi} \supseteq \mathcal{U}_\mathcal{R}(\mathcal{P})$
- $\{(\mathcal{P}, \mathcal{R}, e)\}$, *otherwise*

Proof. $\mathcal{P} \setminus \mathcal{P}_{\succ_\pi} \subseteq \mathcal{P}$ implies completeness. For soundness, we only regard the new case $e = \mathbf{t}$. If $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is not finite, then there is a minimal infinite $(\mathcal{P}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \ldots$ with $t_i\sigma \to^*_\mathcal{R} s_{i+1}\sigma$ for all i.

Since all terms $t_i\sigma$ and $s_{i+1}\sigma$ are terminating, we can apply $\mathcal{I}_1$ to both $t_i\sigma$ and $s_{i+1}\sigma$ (where Δ are the usable symbols of $\mathcal{P}$ again). Using Lemma 16 (v) we obtain $\mathcal{I}_1(t_i\sigma) \to^*_{Rls_\mathcal{R}(\Delta)\cup\mathcal{C}_\varepsilon} \mathcal{I}_1(s_{i+1}\sigma)$.

Moreover, by the definition of $\mathcal{U}_\mathcal{R}$, all t_i are terms over the signature Δ. So by Lemma 16 (i) and (ii) we get $t_i\,\mathcal{I}_1(\sigma) = \mathcal{I}_1(t_i\sigma) \to^*_{Rls_\mathcal{R}(\Delta)\cup\mathcal{C}_\varepsilon} \mathcal{I}_1(s_{i+1}\sigma) \to^*_{\mathcal{C}_\varepsilon} s_{i+1}\,\mathcal{I}_1(\sigma)$ stating that $s_1 \to t_1, s_2 \to t_2, \ldots$ is also a $(\mathcal{P}, Rls_\mathcal{R}(\Delta) \cup \mathcal{C}_\varepsilon)$-chain, i.e., a $(\mathcal{P}, \mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon)$-chain.

Thus, we have

- $s_i\,\mathcal{I}_1(\sigma) \succ_\pi t_i\,\mathcal{I}_1(\sigma)$ for all i where $s_i \to t_i \in \mathcal{P}_{\succ_\pi}$
- $s_i\,\mathcal{I}_1(\sigma) \succsim_\pi t_i\,\mathcal{I}_1(\sigma)$ for all other i
- $t_i\,\mathcal{I}_1(\sigma) \succsim_\pi s_{i+1}\,\mathcal{I}_1(\sigma)$ for all i

Since $\succ_\pi$ is well founded and compatible with $\succsim_\pi$, dependency pairs from $\mathcal{P}_{\succ_\pi}$ cannot occur infinitely often in this chain. Thus, there is an $n \geq 0$ such that all pairs $s_i \to t_i$ with $i \geq n$ are from $\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}$. Therefore, if we omit the first $n-1$ pairs from the original chain, we obtain a minimal infinite $(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R})$-chain $s_n \to t_n, s_{n+1} \to t_{n+1}, \ldots$ Hence, $(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R}, \mathbf{t})$ is not finite either. □

Note that in Thm. 17, one only has to orient the usable rules $\mathcal{U}_\mathcal{R}(\mathcal{P})$, but one keeps *all* rules $\mathcal{R}$ in the resulting DP problem $(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R}, e)$. As an alternative, one might be tempted to replace Thm. 17 by a "usable rule processor" followed by an application of the reduction pair processor of Thm. 12. The *usable rule processor* $Proc_\mathcal{U}$ would remove all non-usable rules and add the $\mathcal{C}_\varepsilon$-rules in the termination case, cf. [15, Thm. 28 and Thm. 37]:

$$\begin{array}{rcl} Proc_\mathcal{U}((\mathcal{P},\mathcal{R},\mathbf{t})) & = & \{(\mathcal{P},\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon,\mathbf{t})\} \\ Proc_\mathcal{U}((\mathcal{P},\mathcal{R},\mathbf{i})) & = & \{(\mathcal{P},\mathcal{U}_\mathcal{R}(\mathcal{P}),\mathbf{i})\} \end{array}$$

However, the following example shows that $Proc_\mathcal{U}$ is not sound in the termination case. In the example, there is an infinite minimal $(\mathcal{P},\mathcal{R})$-chain and thus, there is also an infinite $(\mathcal{P},\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon)$-chain. However, there exists no infinite *minimal* $(\mathcal{P},\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon)$-chain. Therefore, $(\mathcal{P},\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon,\mathbf{t})$ is finite whereas $(\mathcal{P},\mathcal{R},\mathbf{t})$ is not finite.

EXAMPLE 18. *The following DP problem* $(\mathcal{P},\mathcal{R},\mathbf{t})$ *is a variant of Toyama's TRS. Let* $\mathcal{P}$ *consist of*

$$\mathsf{F}(0,1,x_1,0,1,x_2,0,1,x_3,y) \to \mathsf{F}(x_1,x_1,x_1,x_2,x_2,x_2,x_3,x_3,x_3,\mathsf{g}(x_1,x_2,x_3))$$

and let $\mathcal{R}$ consist of the following ten rules:

$$\begin{array}{rcl@{\qquad\qquad}rcl}
\mathsf{a} & \to & \mathsf{0} & \mathsf{g}(x,x,y) & \to & \mathsf{h}(x,x)\\
\mathsf{a} & \to & \mathsf{1} & \mathsf{g}(x,y,x) & \to & \mathsf{h}(x,x)\\
\mathsf{b} & \to & \mathsf{0} & \mathsf{g}(y,x,x) & \to & \mathsf{h}(x,x)\\
\mathsf{b} & \to & \mathsf{1} & \mathsf{h}(\mathsf{0},\mathsf{1}) & \to & \mathsf{h}(\mathsf{0},\mathsf{1})\\
\mathsf{e} & \to & \mathsf{0} & & & \\
\mathsf{e} & \to & \mathsf{1} & & &
\end{array}$$

There is an infinite minimal $(\mathcal{P},\mathcal{R})$-chain, as can be shown using the substitution σ with $\sigma(x_1) = \mathsf{a}$, $\sigma(x_2) = \mathsf{b}$, $\sigma(x_3) = \mathsf{e}$, and $\sigma(y) = \mathsf{g}(\mathsf{a},\mathsf{b},\mathsf{e})$. The reason is that the instantiated right-hand side of $\mathcal{P}$'s only rule now reduces to its instantiated left-hand side:

$$\begin{array}{rl}
 & \mathsf{F}(x_1,x_1,x_1,x_2,x_2,x_2,x_3,x_3,x_3,\mathsf{g}(x_1,x_2,x_3))\sigma\\
= & \mathsf{F}(\mathsf{a},\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b},\mathsf{b},\mathsf{e},\mathsf{e},\mathsf{e},\mathsf{g}(\mathsf{a},\mathsf{b},\mathsf{e}))\\
\to^6_{\mathcal{R}} & \mathsf{F}(\mathsf{0},\mathsf{1},\mathsf{a},\mathsf{0},\mathsf{1},\mathsf{b},\mathsf{0},\mathsf{1},\mathsf{e},\mathsf{g}(\mathsf{a},\mathsf{b},\mathsf{e}))\\
= & \mathsf{F}(\mathsf{0},\mathsf{1},x_1,\mathsf{0},\mathsf{1},x_2,\mathsf{0},\mathsf{1},x_3,y)\sigma
\end{array}$$

The resulting infinite chain is minimal, since the instantiated right-hand side $\mathsf{F}(\mathsf{a},\mathsf{a},\mathsf{a},\mathsf{b},\mathsf{b},\mathsf{b},\mathsf{e},\mathsf{e},\mathsf{e},\mathsf{g}(\mathsf{a},\mathsf{b},\mathsf{e}))$ is terminating. The reason is that $\mathsf{g}(\mathsf{a},\mathsf{b},\mathsf{e})$ can be reduced to $\mathsf{h}(\mathsf{0},\mathsf{0})$ or to $\mathsf{h}(\mathsf{1},\mathsf{1})$ but not to $\mathsf{h}(\mathsf{0},\mathsf{1})$. Thus, the DP problem $(\mathcal{P},\mathcal{R},\mathbf{t})$ is not finite.

However, the DP problem $(\mathcal{P},\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon,\mathbf{t})$ is finite and therefore, any processor which transforms $(\mathcal{P},\mathcal{R},\mathbf{t})$ into $(\mathcal{P},\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon,\mathbf{t})$ is unsound. Here, $\mathcal{U}_\mathcal{R}(\mathcal{P})$ consists of all g-rules and the h-rule, i.e., $\mathcal{U}_\mathcal{R}(\mathcal{P}) = Rls_\mathcal{R}(\{\mathsf{g},\mathsf{h}\})$. To prove that $(\mathcal{P},\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon,\mathbf{t})$ is finite we have to show that there is no infinite minimal $(\mathcal{P},\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon)$-chain.

In any chain of length greater than 1, the right-hand side of $\mathcal{P}$'s rule has to be instantiated by a substitution σ such that all $x_i\sigma$ can be reduced to both $\mathsf{0}$ and $\mathsf{1}$ with $\mathcal{U}_\mathcal{R}(\mathcal{P}) \cup \mathcal{C}_\varepsilon$. It is easy to see that then each $x_i\sigma$ can also be reduced to $\mathsf{c}(\mathsf{0},\mathsf{1})$ or to $\mathsf{c}(\mathsf{1},\mathsf{0})$. So there are at least two $x_i\sigma$ and $x_j\sigma$ with $i \neq j$ which can be reduced to the same term $\mathsf{c}(\mathsf{0},\mathsf{1})$ or $\mathsf{c}(\mathsf{1},\mathsf{0})$. Hence, the subterm $\mathsf{g}(x_1,x_2,x_3)\sigma$ of $\mathcal{P}$'s instantiated right-hand side can be reduced to $\mathsf{h}(\mathsf{c}(\mathsf{0},\mathsf{1}),\mathsf{c}(\mathsf{0},\mathsf{1}))$ or to $\mathsf{h}(\mathsf{c}(\mathsf{1},\mathsf{0}),\mathsf{c}(\mathsf{1},\mathsf{0}))$ and further to the non-terminating term $\mathsf{h}(\mathsf{0},\mathsf{1})$. So the instantiated right-hand side of $\mathcal{P}$'s rule is not terminating and thus, there is no minimal chain of length greater than 1.

The following variant of an example from [1] shows that Thm. 17 not only increases efficiency, but it also leads to a more powerful method than Thm. 12 (if one is restricted to $\mathcal{C}_\varepsilon$-compatible quasi-orders $\succsim$).

EXAMPLE 19. *In the following TRS, $\mathsf{div}(x,y)$ computes $\lfloor \frac{x}{y} \rfloor$ for $x, y \in \mathbb{N}$ if $y \neq 0$ and $\mathsf{quot}(x,y,z)$ computes $1 + \lfloor \frac{x-y}{z} \rfloor$ if $x \geq y$ and $z \neq 0$ and it computes 0 if $x < y$.*

$$\mathsf{div}(0, y) \to 0 \quad (12)$$
$$\mathsf{div}(x, y) \to \mathsf{quot}(x, y, y) \quad (13)$$
$$\mathsf{quot}(0, \mathsf{s}(y), z) \to 0 \quad (14)$$
$$\mathsf{quot}(\mathsf{s}(x), \mathsf{s}(y), z) \to \mathsf{quot}(x, y, z) \quad (15)$$
$$\mathsf{quot}(x, 0, \mathsf{s}(z)) \to \mathsf{s}(\mathsf{div}(x, \mathsf{s}(z))) \quad (16)$$

In contrast to our new processor, Urbain's result [43] is not applicable in this example. The reason is that this TRS is not a hierarchical combination (since div *and* quot *are mutually recursive).*

Note also that this TRS does not belong to known classes of TRSs where innermost termination implies termination, since it is not locally confluent: $\mathsf{div}(0, 0)$ *reduces to the normal forms* 0 *and* $\mathsf{quot}(0, 0, 0)$.

A termination proof is impossible with the previous processors from Thm. 8 and Thm. 12 if one uses standard reduction pairs $(\succsim, \succ)$ *where* $\succsim$ *is a quasi-simplification order. In contrast, innermost termination can easily be proved. We obtain the following dependency pairs which form an SCC of the dependency graph.*

$$\mathsf{DIV}(x, y) \to \mathsf{QUOT}(x, y, y) \quad (17)$$
$$\mathsf{QUOT}(\mathsf{s}(x), \mathsf{s}(y), z) \to \mathsf{QUOT}(x, y, z) \quad (18)$$
$$\mathsf{QUOT}(x, 0, \mathsf{s}(z)) \to \mathsf{DIV}(x, \mathsf{s}(z)) \quad (19)$$

There are no usable rules because the dependency pairs have no defined symbols in their right-hand sides. Thus, the reduction pair processor only requires a decrease for the dependency pairs. Hence, with a filtering $\pi(\mathsf{QUOT}) = \pi(\mathsf{DIV}) = 1$ *and the embedding order, (17) and (19) are weakly decreasing, while (18) is strictly decreasing and can be removed. So the reduction pair processor transforms the initial DP problem* $(\{(17), (18), (19)\}, \mathcal{R}, \mathbf{i})$ *into* $(\{(17), (19)\}, \mathcal{R}, \mathbf{i})$. *With the new improved processor of Thm. 17 this step can now also be done when proving full termination. Afterwards, the remaining DP problem can easily be solved by the existing DP processors: We apply the reduction pair processor once more with a filtering* $\pi(\mathsf{s}) = [\,]$, $\pi(\mathsf{QUOT}) = \pi(\mathsf{DIV}) = 2$ *and the LPO with a precedence* $0 > \mathsf{s}$. *Now (17) is weakly decreasing and (19) is strictly decreasing and can be removed. The resulting DP problem* $(\{(17)\}, \mathcal{R}, e)$ *is solved by the dependency graph processor, since the estimated dependency graph has no cycle anymore.*

Now our technique for termination is nearly as powerful as the one for innermost termination. The remaining difference between termination and innermost termination proofs is that the innermost dependency graph is a subgraph of the dependency graph and may have fewer cycles. Moreover, in Sect. 5 we will see that the conditions for applying dependency pair transformations are less restrictive for innermost termination than for termination. Finally for termination, we

use $\mathcal{C}_\varepsilon$-compatible quasi-orders, which is not necessary for innermost termination. So in general, innermost termination is still easier to prove than termination, but the difference has become much smaller.

4. Improving Termination Proofs by Argument Filtering

Now we introduce a further improvement for both termination and innermost termination proofs in order to reduce the usable rules (and hence, the resulting constraints) further. The idea is to apply the argument filtering first and to determine the usable rules afterwards. The advantage is that after the argument filtering, some symbols g may have been eliminated from the right-hand sides of dependency pairs and thus, the g-rules do not have to be included in the usable rules anymore. Moreover, if f's rules are usable and f calls a function g, then up to now g's rules are also considered usable. However, if all calls of g are only on positions that are eliminated by the argument filtering, now also g's rules are not considered usable anymore.

However, for collapsing argument filterings this refinement is not sound. Consider the non-innermost terminating TRS

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

In the SCC $\{\mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(\mathsf{double}(x))\}$, we can use the 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 SCC has no usable rules. Then we could easily orient the only resulting constraint $\mathsf{F}(\mathsf{s}(x)) \succ \mathsf{F}(x)$ for this SCC and falsely prove (innermost) termination. Note that the elimination of double in $\mathsf{F}(\mathsf{double}(x))$ is not due to the outer symbol F, but due to a collapsing argument filtering for double itself. For that reason, a defined symbol like double may only be ignored when constructing the usable rules, if all its occurrences are in positions which are filtered away by the function symbols *above* them. To capture this formally, we define the *regarded positions* w.r.t. an argument filtering. In this definition, collapsing argument filterings with $\pi(f) = i$ are treated in the same way as filterings of the form $\pi(f) = [i]$.

DEFINITION 20 (Regarded Positions). *Let π be an argument filtering. For an n-ary function symbol f, the set $rp_\pi(f)$ of* regarded positions *is $\{i\}$ if $\pi(f) = i$, and it is $\{i_1, \ldots, i_m\}$ if $\pi(f) = [i_1, \ldots, i_m]$.*

So if $\pi(\mathsf{F}) = [1]$ or $\pi(\mathsf{F}) = 1$, then $rp_\pi(\mathsf{F}) = \{1\}$. Now we can define the usable rules *w.r.t. an argument filtering*. For a term like $\mathsf{F}(\mathsf{double}(x))$, the rules for all symbols on regarded positions are considered usable. So if $rp_\pi(\mathsf{F}) = \{1\}$, then the double-rules are usable.

DEFINITION 21 (Usable Rules w.r.t. Argument Filtering). *For $f \in \mathcal{F}$, let $\mathcal{R}'$ denote $\mathcal{R} \backslash Rls_{\mathcal{R}}(f)$. For any argument filtering π, we define*

- $\mathcal{U}_{\mathcal{R}}(x, \pi) = \varnothing$ *for* $x \in \mathcal{V}$ *and*
- $\mathcal{U}_{\mathcal{R}}(f(t_1, \ldots, t_n), \pi) = Rls_{\mathcal{R}}(f) \cup \bigcup_{l \to r \in Rls_{\mathcal{R}}(f)} \mathcal{U}_{\mathcal{R}'}(r, \pi) \cup \bigcup_{i \in rp_\pi(f)} \mathcal{U}_{\mathcal{R}'}(t_i, \pi)$,

For any TRS $\mathcal{P}$, we define $\mathcal{U}_{\mathcal{R}}(\mathcal{P}, \pi) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}_{\mathcal{R}}(t, \pi)$.

Obviously, this new definition of usable rules improves upon the previous one of Def. 10, i.e., $\mathcal{U}_{\mathcal{R}}(t, \pi) \subseteq \mathcal{U}_{\mathcal{R}}(t)$ for any term t.

EXAMPLE 22. *We illustrate the new definition of usable rules with the following TRS of [27] for list reversal.*

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

For the SCC $\mathcal{P}$ containing the dependency pair $\mathsf{REV2}(x, \mathsf{cons}(y, l)) \to \mathsf{REV}(\mathsf{cons}(x, \mathsf{rev}(\mathsf{rev2}(y, l))))$, up to now all rules were usable, since rev *and* rev2 *occur in the right-hand side and the function* rev *calls* rev1*. In contrast, if one uses an argument filtering with $\pi(\mathsf{cons}) = [2]$, then with our new definition of usable rules from Def. 21, the* rev1*-rules are no longer usable, since* rev1 *is not in right-hand sides of filtered* rev*-rules. This reduction of the set of usable rules is crucial for the success of the (innermost) termination proof with dependency pairs, cf. Ex. 27.*

The following lemma is needed to show that one may replace $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ by $\mathcal{U}_{\mathcal{R}}(\mathcal{P}, \pi)$ in *innermost* termination proofs.

LEMMA 23 (Properties of Usable Rules). *Let $\mathcal{R}$ be a TRS, let π be an argument filtering, and let σ be a normal substitution (i.e., $\sigma(x)$ is in normal form for all $x \in \mathcal{V}$). For all terms t, v we have:*

(i) If $t\sigma \xrightarrow{\mathrm{i}}_{\mathcal{R}} v$, then $\pi(t\sigma) = \pi(v)$ or $\pi(t\sigma) \to_{\pi(\mathcal{U}_{\mathcal{R}}(t,\pi))} \pi(v)$. Moreover, there is a term u and a normal substitution σ' such that $v = u\sigma'$ and $\mathcal{U}_{\mathcal{R}}(u, \pi) \subseteq \mathcal{U}_{\mathcal{R}}(t, \pi)$.
(ii) If $t\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} v$, then $\pi(t\sigma) \to^*_{\pi(\mathcal{U}_{\mathcal{R}}(t,\pi))} \pi(v)$.

Proof.

(i) We perform induction on the position of the reduction. This position must be in t because σ is normal. So t has the form $f(t_1, ..., t_n)$.

If the reduction is on the root position, then we have $t\sigma = l\sigma' \xrightarrow{\mathrm{i}}_{\mathcal{R}} r\sigma' = v$ where $l \to r \in Rls_{\mathcal{R}}(f) \subseteq \mathcal{U}_{\mathcal{R}}(t, \pi)$ and thus $\pi(l) \to \pi(r) \in \pi(\mathcal{U}_{\mathcal{R}}(t, \pi))$. Let σ'_π be the substitution with $\sigma'_\pi(x) = \pi(\sigma'(x))$ for all $x \in \mathcal{V}$. Then, $\pi(t\sigma) = \pi(l)\sigma'_\pi \to_{\pi(\mathcal{U}_{\mathcal{R}}(t,\pi))} \pi(r)\sigma'_\pi = \pi(v)$. Moreover, σ' is a normal substitution due to the innermost strategy and by defining $u = r$, we obtain $v = u\sigma'$. We have $\mathcal{U}_{\mathcal{R}}(u, \pi) = \mathcal{U}_{\mathcal{R}}(r, \pi) \subseteq Rls_{\mathcal{R}}(f) \cup \bigcup_{l' \to r' \in Rls_{\mathcal{R}}(f)} \mathcal{U}_{\mathcal{R}'}(r', \pi) \subseteq \mathcal{U}_{\mathcal{R}}(t, \pi)$.

Otherwise, $t\sigma = f(t_1\sigma \dots t_i\sigma \dots t_n\sigma) \to_{\mathcal{R}} f(t_1\sigma \dots v_i \dots t_n\sigma) = v$ where $t_i\sigma \to_{\mathcal{R}} v_i$. If $\pi(t\sigma) \neq \pi(v)$, then $i \in rp_\pi(f)$ and the induction hypothesis implies $\pi(t_i\sigma) \to_{\pi(\mathcal{U}_{\mathcal{R}}(t_i,\pi))} \pi(v_i)$ and thus, $\pi(t\sigma) \to_{\pi(\mathcal{U}_{\mathcal{R}}(t_i,\pi))} \pi(v)$. As $\mathcal{U}_{\mathcal{R}}(t_i, \pi) \subseteq Rls_{\mathcal{R}}(f) \cup \bigcup_{l' \to r' \in Rls_{\mathcal{R}}(f)} \mathcal{U}_{\mathcal{R}'}(r', \pi) \cup \mathcal{U}_{\mathcal{R}'}(t_i, \pi) \subseteq \mathcal{U}_{\mathcal{R}}(t, \pi)$, we also have $\pi(t\sigma) \to_{\pi(\mathcal{U}_{\mathcal{R}}(t,\pi))} \pi(v)$.

By the induction hypothesis there is some term u_i and some normal substitution σ_i with $v_i = u_i\sigma_i$. Let u'_i result from u_i by replacing its variables x by corresponding fresh variables x'. We define $\sigma'(x') = \sigma_i(x)$ for all these fresh variables and $\sigma'(x) = \sigma(x)$ for all $x \in \mathcal{V}(t)$. Then for $u = f(t_1 \dots u'_i \dots t_n)$ we obtain $v = u\sigma'$. Obviously, we have $\mathcal{U}_{\mathcal{R}}(u'_i, \pi) = \mathcal{U}_{\mathcal{R}}(u_i, \pi)$ and the induction hypothesis implies $\mathcal{U}_{\mathcal{R}}(u_i, \pi) \subseteq \mathcal{U}_{\mathcal{R}}(t_i, \pi)$. As $\mathcal{U}_{\mathcal{R}}(t_i, \pi) \subseteq \mathcal{U}_{\mathcal{R}}(t, \pi)$ (see above), we also have $\mathcal{U}_{\mathcal{R}}(u'_i, \pi) \subseteq \mathcal{U}_{\mathcal{R}}(t, \pi)$. Since $\mathcal{U}_{\mathcal{R}}(u, \pi)$ differs from $\mathcal{U}_{\mathcal{R}}(t, \pi)$ only by containing $\mathcal{U}_{\mathcal{R}'}(u'_i, \pi)$ instead of $\mathcal{U}_{\mathcal{R}'}(t_i, \pi)$ and since $\mathcal{U}_{\mathcal{R}'}(u'_i, \pi) \subseteq \mathcal{U}_{\mathcal{R}}(u'_i, \pi) \subseteq \mathcal{U}_{\mathcal{R}}(t, \pi)$, we also obtain $\mathcal{U}_{\mathcal{R}}(u, \pi) \subseteq \mathcal{U}_{\mathcal{R}}(t, \pi)$.

(ii) The claim immediately follows from (i) by induction on the length of the reduction $t\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} v$. □

The refinement of coupling the usable rules with argument filterings can be used for both innermost and full termination proofs. In the previous section we showed that it suffices for termination proofs if just the usable rules $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ are weakly decreasing. To show that one may replace $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ by $\mathcal{U}_{\mathcal{R}}(\mathcal{P}, \pi)$ here, we define a new mapping $\mathcal{I}_2$ which already incorporates the argument filtering π.

For instance, let $\Delta = \mathcal{F} \setminus \{\mathsf{rev1}\}$ and let $\pi(\mathsf{cons}) = [2]$, $\pi(\mathsf{rev2}) = [2]$. $\mathcal{I}_2(\mathsf{cons}(\mathsf{rev1}(x, l), \mathsf{rev2}(x, l)))$ differs from $\mathcal{I}_1(\mathsf{cons}(\mathsf{rev1}(x, l), \mathsf{rev2}(x, l)))$ by removing the first arguments of all cons- and rev2-terms. So $\mathcal{I}_1$ results in $\mathsf{cons}(\mathsf{c}(\mathsf{rev1}(x, l), \mathsf{d}), \mathsf{rev2}(x, l))$ and $\mathcal{I}_2$ yields $\mathsf{cons}(\mathsf{rev2}(l))$.

DEFINITION 24 ($\mathcal{I}_2$). *Let π be a non-collapsing argument filtering, let $\Delta \subseteq \mathcal{F}$, and let $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ be terminating. We define $\mathcal{I}_2(t)$:*

$$\begin{array}{rll}
\mathcal{I}_2(x) = x & & \text{for } x \in \mathcal{V} \\
\mathcal{I}_2(f(t_1,\ldots,t_n)) = f(\mathcal{I}_2(t_{i_1}),\ldots,\mathcal{I}_2(t_{i_m})) & & \text{for } f \in \Delta, \pi(f) = [i_1,...,i_m] \\
\mathcal{I}_2(g(t_1,\ldots,t_n)) = \mathcal{C}omp(\,\{g(\mathcal{I}_2(t_{i_1}),\ldots,\mathcal{I}_2(t_{i_m}))\} & & \\
\qquad \cup\, \mathcal{R}ed_2(g(t_1,\ldots,t_n))\,) & & \text{for } g \notin \Delta, \pi(g) = [i_1,...,i_m]
\end{array}$$

where $\mathcal{R}ed_2(t) = \{\mathcal{I}_2(t') \mid t \to_{\mathcal{R}} t'\}$. For every terminating substitution σ, we define $\mathcal{I}_2(\sigma)$ as $\mathcal{I}_2(\sigma)\,(x) = \mathcal{I}_2(\sigma(x))$ for all $x \in \mathcal{V}$.

Lemma 25 differs from Lemma 16, since $\mathcal{I}_2$ already applies the filtering π and in (v), we have "$*$" instead of "$+$", as a reduction on a position that is filtered away yields the same transformed terms w.r.t. $\mathcal{I}_2$.

LEMMA 25 (Properties of $\mathcal{I}_2$). *Let π be a non-collapsing filtering and let $\Delta \subseteq \mathcal{F}$ such that $f \in \Delta$ implies $g \in \Delta$ whenever there is a rule $l \to r \in Rls_{\mathcal{R}}(f)$ such that g occurs in $\pi(r)$. Let $t, s, t\sigma \in \mathcal{T}(\mathcal{F},\mathcal{V})$ be terminating and let σ be a terminating substitution.*

(i) *If $\pi(t) \in \mathcal{T}(\Delta_\pi,\mathcal{V})$ then $\mathcal{I}_2(t\sigma) = \pi(t)\,\mathcal{I}_2(\sigma)$.*
(ii) *$\mathcal{I}_2(t\sigma) \to^*_{\mathcal{C}_\varepsilon} \pi(t)\,\mathcal{I}_2(\sigma)$.*
(iii) *If $t \to_{\{l\to r\}} s$ by a root reduction step where $l \to r \in \mathcal{R}$ and $\mathrm{root}(l) \in \Delta$, then $\mathcal{I}_2(t) \to^+_{\{\pi(l)\to\pi(r)\}\cup\mathcal{C}_\varepsilon} \mathcal{I}_2(s)$.*
(iv) *If $t \to_{\mathcal{R}} s$ with $\mathrm{root}(t) \notin \Delta$, then $\mathcal{I}_2(t) \to^+_{\mathcal{C}_\varepsilon} \mathcal{I}_2(s)$.*
(v) *If $t \to_{\{l\to r\}} s$ where $l \to r \in \mathcal{R}$, then $\mathcal{I}_2(t) \to^*_{\{\pi(l)\to\pi(r)\}\cup\mathcal{C}_\varepsilon} \mathcal{I}_2(s)$ if $\mathrm{root}(l) \in \Delta$ and $\mathcal{I}_2(t) \to^*_{\mathcal{C}_\varepsilon} \mathcal{I}_2(s)$ otherwise.*

Proof. The proof is analogous to the proof of Lemma 16. As in Lemma 16, the condition on Δ in the prerequisites of Lemma 25 is needed for (iii). □

Now we can refine the processor of Thm. 17 to the following one where the set of usable rules is reduced significantly.

THEOREM 26 (Reduction Pair Processor Based on Filtering). *Let $(\succsim, \succ)$ be a reduction pair and let π be an argument filtering. Then the following processor Proc is sound and complete. Here, $Proc(\,(\mathcal{P},\mathcal{R},e)\,) =$*

- *$\{(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R}, e)\}$, if the following conditions (a) and (b) hold:*
  - *(a)* *$\mathcal{P}_{\succ_\pi} \cup \mathcal{P}_{\succsim_\pi} = \mathcal{P}$ and $\mathcal{P}_{\succ_\pi} \neq \varnothing$*
  - *(b)* *either* *$e = \mathbf{t}$ and $\mathcal{R}_{\succsim_\pi} \supseteq \mathcal{U}_{\mathcal{R}}(\mathcal{P},\pi)$ and $\succsim$ is $\mathcal{C}_\varepsilon$-compatible*
    or *$e = \mathbf{i}$ and $\mathcal{R}_{\succsim_\pi} \supseteq \mathcal{U}_{\mathcal{R}}(\mathcal{P},\pi)$*

- $\{(\mathcal{P}, \mathcal{R}, e)\}$, *otherwise*

Proof. Again, $\mathcal{P} \setminus \mathcal{P}_{\succ_\pi} \subseteq \mathcal{P}$ implies completeness. For soundness, we first regard the case $e = \mathbf{i}$. If $(\mathcal{P}, \mathcal{R}, \mathbf{i})$ is not finite, then there is an infinite innermost $(\mathcal{P}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \ldots$ So we have $t_i\sigma \xrightarrow{\mathbf{i}}{}^*_{\mathcal{R}} s_{i+1}\sigma$ where all $s_i\sigma$ are in normal form. By Lemma 23 (ii), we obtain $\pi(t_i\sigma) \to^*_{\pi(\mathcal{U}_\mathcal{R}(\mathcal{P},\pi))} \pi(s_{i+1}\sigma)$. All rules of $\mathcal{U}_\mathcal{R}(\mathcal{P}, \pi)$ are decreasing w.r.t. $\succsim_\pi$ and thus, all rules of $\pi(\mathcal{U}_\mathcal{R}(\mathcal{P}, \pi))$ are decreasing w.r.t. $\succsim$. This implies $\pi(t_i\sigma) \succsim \pi(s_{i+1}\sigma)$, i.e., $t_i\sigma \succsim_\pi s_{i+1}\sigma$. Since $\mathcal{P}_{\succ_\pi} \cup \mathcal{P}_{\succsim_\pi} = \mathcal{P}$, pairs of $\mathcal{P}_{\succ_\pi}$ cannot occur infinitely often in the chain.

Now we prove soundness for $e = \mathbf{t}$. If $(\mathcal{P}, \mathcal{R}, \mathbf{t})$ is not finite, then there is a minimal infinite $(\mathcal{P}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \ldots$ with $t_i\sigma \to^*_\mathcal{R} s_{i+1}\sigma$.

Let π' be the *non-collapsing variant* of π where $\pi'(f) = \pi(f)$ if $\pi(f) = [i_1, \ldots, i_k]$ and $\pi'(f) = [i]$ if $\pi(f) = i$. Let Δ be the usable symbols *w.r.t.* π' of $\mathcal{P}$. So Δ is the smallest set of function symbols such that Δ contains all function symbols occurring in right-hand sides of $\pi'(\mathcal{P})$ and $f \in \Delta$ implies $g \in \Delta$ whenever there is a rule $l \to r \in Rls_\mathcal{R}(f)$ such that g occurs in $\pi'(r)$. Thus, $\pi'(\mathcal{U}_\mathcal{R}(\mathcal{P}, \pi')) = \pi'(Rls_\mathcal{R}(\Delta))$. Note that by definition we have $\mathcal{U}_\mathcal{R}(\mathcal{P}, \pi') = \mathcal{U}_\mathcal{R}(\mathcal{P}, \pi)$. Then similar to the proof of Thm. 17, $t_i\sigma \to^*_\mathcal{R} s_{i+1}\sigma$ implies $\pi'(t_i)\mathcal{I}_2(\sigma) = \mathcal{I}_2(t_i\sigma) \to^*_{\pi'(\mathcal{U}_\mathcal{R}(\mathcal{P},\pi))\cup\mathcal{C}_\varepsilon} \mathcal{I}_2(s_{i+1}\sigma) \to^*_{\mathcal{C}_\varepsilon} \pi'(s_{i+1})\mathcal{I}_2(\sigma)$ by Lemma 25 (i), (v), (ii). So $\pi'(s_1) \to \pi'(t_1), \pi'(s_2) \to \pi'(t_2), \ldots$ is an infinite $(\pi'(\mathcal{P}), \pi'(\mathcal{U}_\mathcal{R}(\mathcal{P}, \pi)) \cup \mathcal{C}_\varepsilon)$-chain.

Now we show that in this chain, pairs from $\pi'(\mathcal{P}_{\succ_\pi})$ cannot occur infinitely often. Thus, there is an $n \geq 0$ such that all $s_i \to t_i$ with $i \geq n$ are from $\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}$. Then $s_n \to t_n, s_{n+1} \to t_{n+1}, \ldots$ is a minimal infinite $(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R})$-chain and thus, $(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R}, \mathbf{t})$ is not finite either.

To show that pairs from $\pi'(\mathcal{P}_{\succ_\pi})$ cannot occur infinitely often in the $(\pi'(\mathcal{P}), \pi'(\mathcal{U}_\mathcal{R}(\mathcal{P}, \pi)) \cup \mathcal{C}_\varepsilon)$-chain $\pi'(s_1) \to \pi'(t_1), \pi'(s_2) \to \pi'(t_2), \ldots$, let π'' be the argument filtering for the signature $\mathcal{F}_{\pi'}$ which only performs the collapsing steps of π (i.e., if $\pi(f) = i$ and thus $\pi'(f) = [i]$, we have $\pi''(f) = 1$). All other symbols of $\mathcal{F}_{\pi'}$ are not filtered by π''. Hence, $\pi = \pi'' \circ \pi'$. We extend π'' to the new symbol c by defining $\pi''(\mathsf{c}) = [1, 2]$. Hence, $\mathcal{C}_\varepsilon$-compatibility of $\succsim$ implies $\mathcal{C}_\varepsilon$-compatibility of $\succsim_{\pi''}$.

Now regard the reduction pair $(\succsim_{\pi''}, \succ_{\pi''})$. For all terms s and t, the constraint "$s \succ_\pi t$" implies that the rule $\pi'(s) \to \pi'(t)$ is strictly decreasing (i.e., $\pi'(s) \succ_{\pi''} \pi'(t)$) and the constraint "$s \succsim_\pi t$" implies that the rule $\pi'(s) \to \pi'(t)$ is weakly decreasing (i.e., $\pi'(s) \succsim_{\pi''} \pi'(t)$). Thus by (a), the pairs of $\pi'(\mathcal{P}_{\succ_\pi})$ are strictly decreasing and by (b), all remaining pairs of $\pi'(\mathcal{P})$ and all rules of $\pi'(\mathcal{U}_\mathcal{R}(\mathcal{P}, \pi)) \cup \mathcal{C}_\varepsilon$ are weakly decreasing w.r.t. the reduction pair $(\succ_{\pi''}, \succsim_{\pi''})$. So the chain $\pi'(s_1) \to \pi'(t_1), \pi'(s_2) \to \pi'(t_2), \ldots$ only contains finitely many pairs from $\pi'(\mathcal{P}_{\succ_\pi})$. □

EXAMPLE 27. *The TRS $\mathcal{R}$ for list reversal from Ex. 22 shows the advantages of Thm. 26. The dependency graph processor decomposes the initial DP problem $(DP(\mathcal{R}), \mathcal{R}, e)$ into $(\{(20)\}, \mathcal{R}, e)$ and $(\{(21), (22), (23), (24)\}, \mathcal{R}, e)$ where (20) – (24) are the following dependency pairs:*

$$\mathsf{REV1}(x, \mathsf{cons}(y, l)) \to \mathsf{REV1}(y, l) \quad (20)$$
$$\mathsf{REV}(\mathsf{cons}(x, l)) \to \mathsf{REV2}(x, l) \quad (21)$$
$$\mathsf{REV2}(x, \mathsf{cons}(y, l)) \to \mathsf{REV}(\mathsf{cons}(x, \mathsf{rev}(\mathsf{rev2}(y, l)))) \quad (22)$$
$$\mathsf{REV2}(x, \mathsf{cons}(y, l)) \to \mathsf{REV}(\mathsf{rev2}(y, l)) \quad (23)$$
$$\mathsf{REV2}(x, \mathsf{cons}(y, l)) \to \mathsf{REV2}(y, l) \quad (24)$$

Since $\{(20)\}$ has no usable rules, already the reduction pair processor of Thm. 17 only requires to make the dependency pair (20) strictly decreasing. For example, this is possible using an argument filtering with $\pi(\mathsf{REV1}) = 2$ and the embedding order. The resulting DP problem $(\varnothing, \mathcal{R}, e)$ is then removed by the dependency graph processor.

However, when proving (innermost) termination with Thm. 17, for the problem $(\{(21), (22), (23), (24)\}, \mathcal{R}, e)$ we obtain inequalities from the dependency pairs and $\pi(l) \gtrsim \pi(r)$ for all rules $l \to r$, since all rules are usable. But with standard reduction pairs based on RPOS, KBO, or polynomial orders, these constraints are not satisfiable for any argument filtering [14]. In contrast, with Thm. 26 we can use the filtering $\pi(\mathsf{cons}) = [2]$, $\pi(\mathsf{REV}) = \pi(\mathsf{rev}) = 1$, and $\pi(\mathsf{REV1}) = \pi(\mathsf{REV2}) = \pi(\mathsf{rev2}) = 2$ together with the embedding order. Now we obtain no constraints from the rev1*-rules, cf. Ex. 22. Then the filtered dependency pairs (21), (23), and (24) are strictly decreasing and the filtered pair (22) and all filtered usable rules are at least weakly decreasing. Thus, the reduction pair processor of Thm. 26 results in the DP problem $(\{(22)\}, \mathcal{R}, e)$ which is again removed by the dependency graph processor.*

5. Transforming Dependency Pairs

To increase the power of the dependency pair technique, a dependency pair may be transformed into new pairs. Sect. 5.1 introduces improved versions of these transformations which permit a combination with our new results from the previous sections. Then in Sect. 5.2, we discuss a heuristic in order to mechanize these transformations in practice.

5.1. Improving Dependency Pair Transformations

In [1, 11], techniques were presented to modify dependency pairs by *narrowing*, *rewriting*, and *instantiation*. This is often crucial for the success of a termination proof. The basic idea of *narrowing* and *rewriting* is the following: if there is a chain $s \to t, v \to w$ where $t\sigma \to_{\mathcal{R}}^* v\sigma$ implies that $t\sigma$ must be rewritten at least one step before it reaches $v\sigma$, then these transformations perform this reduction step directly on the pair $s \to t$. The *rewriting* technique can only be used for innermost termination. Here, the right-hand side t must contain a redex. Then one may rewrite this redex, even if t contains other redexes as well. In contrast, for the *narrowing* transformation, one has to build *all* narrowings of t instead of just doing *one* rewrite step. A term t' is an *$\mathcal{R}$-narrowing of t with mgu μ* if a subterm $t|_p \notin \mathcal{V}$ of t unifies with the left-hand side of a (variable-renamed) rule $l \to r \in \mathcal{R}$ with mgu μ, and $t' = t[r]_p\,\mu$. Now $s \to t$ can be replaced by all narrowings $s\mu \to t'$.

The idea of the *instantiation* technique is to examine all pairs $v \to w$ which *precede* $s \to t$ in chains. Here, we analyze what "skeleton" w' of w remains unchanged when we reduce $w\sigma$ to $s\sigma$. Then $s\sigma$ must also be of this form, i.e., w' and s unify. This idea was already used in E(I)DG (Def. 9), where w' is computed by the functions $\text{CAP}_{\mathcal{R}}$ and REN and where one only draws an arc from $v \to w$ to $s \to t$ if w' and s unify with some mgu μ. Then $w\sigma \to_{\mathcal{R}}^* s\sigma$ implies that σ is an instance of μ. Hence, the *instantiation* transformation replaces $s \to t$ by $s\mu \to t\mu$.

The following definition improves the transformations of [1, 11]. Moreover, we introduce a new *forward instantiation* technique. Here, we consider all pairs $v \to w$ which *follow* $s \to t$ in chains. For each such $v \to w$ we have $t\sigma \to_{\mathcal{R}}^* v\sigma$, i.e., $v\sigma \to_{\mathcal{R}^{-1}}^* t\sigma$. Now we compute the mgu between t and a skeleton of v. However, since the $\mathcal{R}$-reduction goes from an instantiation of t to an instantiation of v and not the other way around, the skeleton of v is not constructed with $\text{CAP}_{\mathcal{R}}$, but with the function $\text{CAP}_{\mathcal{R}}^{-1}$. For any set of non-collapsing rules $\mathcal{R}' \subseteq \mathcal{R}$, let $\text{CAP}_{\mathcal{R}'}^{-1}(v)$ result from replacing all subterms of v whose root is from $\{\text{root}(r) \mid l \to r \in \mathcal{R}'\}$ by different fresh variables. If $\mathcal{R}'$ is collapsing, then $\text{CAP}_{\mathcal{R}'}^{-1}(v)$ is a fresh variable. As usual, a rule $l \to r$ is *collapsing* if $r \in \mathcal{V}$. The modifications $\text{CAP}_{\mathcal{R}}^{-1}$ and $\text{CAP}_{\mathcal{U}_{\mathcal{R}}(t)}^{-1}$ were originally introduced in [25, 35, E(I)DG*] in order to improve the estimation of (innermost) dependency graphs. So while the instantiation transformation is influenced by the E(I)DG-estimation, the new forward instantiation is influenced by the E(I)DG*-estimation.

DEFINITION 28 (Transformation Processors). *Let $\mathcal{P}' = \mathcal{P} \uplus \{s \to t\}$.*

(a) For $(\mathcal{P}', \mathcal{R}, e)$, the **n**arrowing processor *$Proc_{\mathbf{n}}$ returns*

- $\{(\mathcal{P} \cup \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\}, \mathcal{R}, e)\}$, *if either*
  - $e = \mathbf{t}$ *and* $t_1, \ldots, t_n$ *are all* $\mathcal{R}$*-narrowings of* t *with 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.*
  - $e = \mathbf{i}$ *and* $t_1, \ldots, t_n$ *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.*
- $\{(\mathcal{P}', \mathcal{R}, e)\}$, *otherwise.*

(b) For $(\mathcal{P}', \mathcal{R}, e)$*, the* **r**ewriting processor $Proc_{\mathbf{r}}$ *returns*

- $\{(\mathcal{P} \cup \{s \to t'\}, \mathcal{R}, e)\}$, *if* $e = \mathbf{i}$*,* $\mathcal{U}_{\mathcal{R}}(t|_p)$ *is non-overlapping, and* $t \to_{\mathcal{R}} t'$*, where* p *is the position of the redex.*
- $\{(\mathcal{P}', \mathcal{R}, e)\}$, *otherwise.*

(c) For $(\mathcal{P}', \mathcal{R}, e)$*, the* **i**nstantiation processor $Proc_{\mathbf{i}}$ *returns*

- $\{(\mathcal{P} \cup \{s\mu \to t\mu \mid \mu = mgu(\text{REN}(\text{CAP}_{\mathcal{R}}(w)), s),\ v \to w \in \mathcal{P}'\}, \mathcal{R}, e)\}$, *if* $e = \mathbf{t}$
- $\{(\mathcal{P} \cup \{s\mu \to t\mu \mid \mu = mgu(\text{CAP}_{\mathcal{R}}(w), s), v \to w \in \mathcal{P}', s\mu, v\mu \text{ normal}\}, \mathcal{R}, e)\}$, *if* $e = \mathbf{i}$

(d) For $(\mathcal{P}', \mathcal{R}, e)$*, the* **f**orward instantiation processor $Proc_{\mathbf{f}}$ *returns*

- $\{(\mathcal{P} \cup \{s\mu \to t\mu \mid \mu = mgu(\text{REN}(\text{CAP}_{\mathcal{R}}^{-1}(v)), t),\ v \to w \in \mathcal{P}'\}, \mathcal{R}, e)\}$, *if* $e = \mathbf{t}$
- $\{(\mathcal{P} \cup \{s\mu \to t\mu \mid \mu = mgu(\text{REN}(\text{CAP}_{\mathcal{U}_{\mathcal{R}}(t)}^{-1}(v)), t),\ v \to w \in \mathcal{P}'\}, \mathcal{R}, e)\}$, *if* $e = \mathbf{i}$

Ex. 29 shows the advantage of the new forward instantiation technique.

EXAMPLE 29. *Without forward instantiation, termination of the TRS* $\{\mathsf{f}(x,y,z) \to \mathsf{g}(x,y,z),\ \mathsf{g}(\mathsf{0},\mathsf{1},x) \to \mathsf{f}(x,x,x)\}$ *cannot be shown by any* $\mathcal{C}_\varepsilon$*-compatible reduction pair. The instantiation processor is useless here, since in chains* $v \to w$*,* $s \to t$*, the mgu of* $\text{REN}(\text{CAP}_{\mathcal{R}}(w))$ *and* s *does not modify* s*. But the forward instantiation processor instantiates* $\mathsf{F}(x,y,z) \to \mathsf{G}(x,y,z)$ *to* $\mathsf{F}(\mathsf{0},\mathsf{1},z) \to \mathsf{G}(\mathsf{0},\mathsf{1},z)$*. The reason is that in the chain* $\mathsf{F}(x,y,z) \to \mathsf{G}(x,y,z)$*,* $\mathsf{G}(\mathsf{0},\mathsf{1},x') \to \mathsf{F}(x',x',x')$*, the mgu of* $\text{REN}(\text{CAP}_{\mathcal{R}}^{-1}(\mathsf{G}(\mathsf{0},\mathsf{1},x'))) = \mathsf{G}(\mathsf{0},\mathsf{1},x'')$ *and* $\mathsf{G}(x,y,z)$ *is* $[x/\mathsf{0},\ y/\mathsf{1},\ x''/z]$*. Now the termination proof succeeds since the dependency graph has no cycle (as detected by the* EDG**-estimation of [25, 35]).*

In addition to forward instantiation, Def. 28 also extends the existing narrowing, rewriting, and instantiation transformations [1, 11] by permitting their application for slightly more TRSs if $e = \mathbf{i}$. In [11], narrowing $s \to t$ was not permitted if t unifies with the left-hand side of a 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, for both instantiation and narrowing, now one only has to consider instantiations which turn left-hand sides of dependency pairs into normal forms.

However, while these liberalized applicability conditions only have a minor impact, the most important improvement of Def. 28 over [1, 11] is that now the transformations are formulated within the DP framework and that they now work for *minimal* instead of ordinary chains of dependency pairs. This is needed to combine the transformations with the improvements of Sect. 3 and 4 which require the consideration of minimal chains. Moreover, due to the formulation as processors, transformations can now be applied at any time during a termination proof.

Before proving soundness and completeness, Ex. 30 illustrates that transformations are often crucial for the success of the proof.

EXAMPLE 30. *The following alternative TRS for division is from [2].*

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

Without transformations, no simplification order satisfies Thm. 26's constraints for innermost termination of the following SCC [14].

$$\mathsf{DIV}(x, \mathsf{s}(y)) \to \mathsf{IF}(\mathsf{le}(\mathsf{s}(y), x), x, \mathsf{s}(y)) \tag{25}$$
$$\mathsf{IF}(\mathsf{true}, x, y) \to \mathsf{DIV}(\mathsf{minus}(x, y), y) \tag{26}$$

But when transforming the dependency pairs, the resulting constraints can easily be satisfied. Intuitively, $x \succ \mathsf{minus}(x, y)$ *only has to be required if* $\mathsf{le}(\mathsf{s}(y), x)$ *reduces to* true. *This argumentation can be simulated using the transformations of Def. 28. By narrowing, we perform a case analysis on how the* le*-term in (25) can be evaluated. In the first narrowing,* x *is instantiated by* $\mathsf{0}$. *This results in a pair* $\mathsf{DIV}(\mathsf{0}, \mathsf{s}(y)) \to \mathsf{IF}(\mathsf{false}, \mathsf{0}, \mathsf{s}(y))$ *which is not in a cycle. The other narrowing is*

$$\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{IF}(\mathsf{le}(y, x), \mathsf{s}(x), \mathsf{s}(y)) \tag{27}$$

which forms an SCC with (26). Now we perform instantiation of (26) and see that x and y must be of the form $\mathsf{s}(\ldots)$. So (26) is replaced by

$$\mathsf{IF}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) \;\to\; \mathsf{DIV}(\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)), \mathsf{s}(y)) \tag{28}$$

that forms an SCC with (27). Finally, by rewriting (28) we obtain

$$\mathsf{IF}(\mathsf{true}, \mathsf{s}(x), \mathsf{s}(y)) \;\to\; \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y)) \tag{29}$$

The constraints of the resulting SCC $\{(27), (29)\}$ (and all other SCCs) are solved by $\pi(\mathsf{minus}) = \pi(\mathsf{DIV}) = 1$, $\pi(\mathsf{IF}) = 2$, and the embedding order.

Thm. 31 states that the transformations in Def. 28 are sound and it gives conditions under which they are also complete. A counterexample which shows that $Proc_{\mathbf{n}}$ is not complete in general is [1, Ex. 43].

THEOREM 31 (Sound- and Completeness). *$Proc_{\mathbf{n}}$, $Proc_{\mathbf{r}}$, $Proc_{\mathbf{i}}$, and $Proc_{\mathbf{f}}$ are sound. Moreover, $Proc_{\mathbf{r}}$, $Proc_{\mathbf{i}}$, and $Proc_{\mathbf{f}}$ are also complete.*[7] *$Proc_{\mathbf{n}}$ is not complete in general, but it is complete for DP problems $(\mathcal{P}, \mathcal{R}, e)$ where $e = \mathbf{t}$ or where $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ is non-overlapping.*

Proof.

Soundness of $Proc_{\mathbf{n}}$

First let $e = \mathbf{t}$. We show that for every minimal $(\mathcal{P}', \mathcal{R})$-chain "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$", there is a narrowing t' of t with the mgu μ such that "$\ldots, v_1 \to w_1, s\mu \to t', v_2 \to w_2, \ldots$" is also a minimal chain. Here, $s \to t$ may also be the first pair (i.e., $v_1 \to w_1$ may be missing). Then all occurrences of $s \to t$ in a chain may be replaced by pairs from $\{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\}$. Hence, every infinite minimal $(\mathcal{P}', \mathcal{R})$-chain results in an infinite minimal $(\mathcal{P} \cup \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\}, \mathcal{R})$-chain. Thus, if $(\mathcal{P} \cup \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\}, \mathcal{R})$ is finite, then so is $(\mathcal{P}', \mathcal{R})$.

If "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$" is a minimal chain, then there must be a substitution such that

(**t1**) every instantiated right-hand side reduces to the instantiated left-hand side of the next pair in the chain and

(**t2**) all instantiated right-hand sides are terminating w.r.t. $\mathcal{R}$.

[7] Related proofs for previous versions of the transformations can be found in [1, 11]. But in contrast to [1, 11], now we regard these techniques within the DP framework, we use them to prove absence of *minimal* chains, and we have more liberal applicability conditions. Finally, the forward instantiation technique is completely new (its soundness proof builds upon proof ideas for the E(I)DG*-approximation of [25, 35]).

Let σ be a substitution satisfying (**t1**) and (**t2**) where the reduction $t\sigma \to_{\mathcal{R}}^* v_2\sigma$ has *minimal* length. Note that $t\sigma \neq v_2\sigma$ as t and v_2 do not unify. Hence, we have $t\sigma \to_{\mathcal{R}} q \to_{\mathcal{R}}^* v_2\sigma$ for some term q.

First, we assume that the reduction $t\sigma \to_{\mathcal{R}} q$ takes place "in σ". Hence, $t|_p = x$ for some position p such that $\sigma(x) \to_{\mathcal{R}} r$ and $q = t[r]_p$. The variable x only occurs *once* in t (as t is linear) and therefore, $q = t\sigma'$ for the substitution σ' with $\sigma'(x) = r$ and $\sigma'(y) = \sigma(y)$ for all variables $y \neq x$. As all (occurrences of) pairs in the chain are variable disjoint, σ' behaves like σ for all pairs except $s \to t$. Here, we have

$$w_1\sigma' = w_1\sigma \to_{\mathcal{R}}^* s\sigma \to_{\mathcal{R}}^* s\sigma' \quad \text{and} \quad t\sigma' = q \to_{\mathcal{R}}^* v_2\sigma = v_2\sigma'.$$

Hence, σ' satisfies (**t1**) and it satisfies (**t2**) as well, since $t\sigma' = q$ is terminating. But as the reduction $t\sigma' \to_{\mathcal{R}}^* v_2\sigma'$ is shorter than the reduction $t\sigma \to_{\mathcal{R}}^* v_2\sigma$, this is a contradiction to the definition of σ.

So the reduction $t\sigma \to_{\mathcal{R}} q$ cannot take place "in σ". Hence, there is a subterm $t|_p \notin \mathcal{V}$ such that a rule $l \to r$ has been applied to $t|_p\sigma$ (i.e. $l\rho = t|_p\sigma$ for some matcher ρ). Hence, the reduction has the form

$$t\sigma = t\sigma[t|_p\sigma]_p = t\sigma[l\rho]_p \to_{\mathcal{R}} t\sigma[r\rho]_p = q.$$

We assume that $\mathcal{V}(l)$ are fresh variables. Then we extend σ to "behave" like ρ on $\mathcal{V}(l)$ (but it remains the same on all other variables). Now σ unifies l and $t|_p$. Hence, there is also an mgu μ with $\sigma = \mu\tau$ for some τ.

Let t' be $t\mu[r\mu]_p$. Then t narrows to t' with the mgu μ. As we may assume $s\mu \to t'$ to be variable disjoint from all other pairs, we can extend σ to behave like τ on the variables of $s\mu$ and t'. Then we have

$$w_1\sigma \to_{\mathcal{R}}^* s\sigma = s\mu\tau = s\mu\sigma \quad \text{and}$$

$$t'\sigma = t'\tau = t\mu\tau[r\mu\tau]_p = t\sigma[r\sigma]_p = t\sigma[r\rho]_p = q \to_{\mathcal{R}}^* v_2\sigma.$$

Hence, "$\ldots, v_1 \to w_1, s\mu \to t', v_2 \to w_2, \ldots$" is also a chain which is minimal, since $t'\sigma = q$ is terminating.

Now we regard the case $e = \mathbf{i}$. Here, we prove that for every minimal *innermost* $(\mathcal{P}', \mathcal{R})$-chain "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$", there exists a narrowing t' of t with the mgu μ such that $s\mu$ is in normal form and such that "$\ldots, v_1 \to w_1, s\mu \to t', v_2 \to w_2, \ldots$" is a minimal *innermost* chain as well. There must be a substitution σ such that

(**i1**) every instantiated right-hand side reduces innermost to the instantiated left-hand side of the next pair in the chain,

(**i2**) all instantiated left-hand sides are normal forms, and

(**i3**) all instantiated right-hand sides terminate innermost w.r.t. $\mathcal{R}$.

Note that $t\sigma \neq v_2\sigma$. Otherwise σ would unify t and v_2, where both $s\sigma$ and $v_2\sigma$ are normal forms. Hence, $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} q \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} v_2\sigma$ for some term q.

The reduction $t\sigma \xrightarrow{\mathsf{i}}_{\mathcal{R}} q$ cannot take place "in σ", because $\mathcal{V}(t) \subseteq \mathcal{V}(s)$. Hence, then $s\sigma$ would not be a normal form, which violates (**i2**). The remainder of the proof is completely analogous to the case $e = \mathbf{t}$.

Completeness of $Proc_{\mathbf{n}}$

We first regard $e = \mathbf{t}$. Let $(\mathcal{P} \cup \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\}, \mathcal{R}, \mathbf{t})$ be infinite. If $\mathcal{R}$ is not terminating, then trivially $(\mathcal{P}', \mathcal{R}, \mathbf{t})$ is infinite as well. Otherwise, we show that any infinite $(\mathcal{P} \cup \{s\mu_1 \to t_1, \ldots, s\mu_n \to t_n\}, \mathcal{R})$-chain can be transformed into an infinite $(\mathcal{P}', \mathcal{R})$-chain. (Now all chains are minimal, since $\mathcal{R}$ is terminating.)

To this end, we prove that if "$\ldots, v_1 \to w_1, s\mu \to t', v_2 \to w_2, \ldots$" is a chain where t narrows to t' with the mgu μ, then "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$" is a chain as well. There is a substitution σ satisfying (**t1**). So in particular we have

$$w_1\sigma \to^*_{\mathcal{R}} s\mu\sigma \quad \text{and} \quad t'\sigma \to^*_{\mathcal{R}} v_2\sigma.$$

As the variables in $s \to t$ are disjoint from all other variables, we may extend σ to behave like $\mu\sigma$ on $\mathcal{V}(s)$. Then $s\sigma = s\mu\sigma$ and hence,

$$w_1\sigma \to^*_{\mathcal{R}} s\sigma. \tag{30}$$

Moreover, by the definition of narrowing, we have $t\mu \to_{\mathcal{R}} t'$. This implies $t\mu\sigma \to_{\mathcal{R}} t'\sigma$ and as $t\sigma = t\mu\sigma$, we obtain

$$t\sigma \to_{\mathcal{R}} t'\sigma \to^*_{\mathcal{R}} v_2\sigma. \tag{31}$$

Hence, "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$" is also a chain.

Now we show completeness for $e = \mathbf{i}$ if $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ is non-overlapping. Again, the non-trivial case is if $\mathcal{R}$ is innermost terminating. We show that if "$\ldots, v_1 \to w_1, s\mu \to t', v_2 \to w_2, \ldots$" is an *innermost* chain where t narrows to t' with mgu μ, then "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$" is an *innermost* chain as well. There is a substitution σ satisfying (**i1**) and (**i2**). Analogous to (30) and (31) in the termination case, one obtains $w_1\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} s\sigma$ and $t\sigma \to_{\mathcal{R}} t'\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} v_2\sigma$ where $v_2\sigma$ is a normal form by (**i2**). Since $\mathcal{R}$ is innermost terminating, repeated application of *innermost* reduction steps to $t\sigma$ also yields some normal form q, i.e., $t\sigma \xrightarrow{\mathsf{i}}{}^*_{\mathcal{R}} q$. Note that all rules used in any reduction of $t\sigma$ are from $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$. So $t\sigma$ is weakly innermost terminating w.r.t. $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$. Since $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ is non-overlapping, $t\sigma$ is terminating and confluent w.r.t. $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ by [21, Thm. 3.2.11] and thus, w.r.t. $\mathcal{R}$ as well. This implies

that $t\sigma$ only has a unique normal form $q = v_2\sigma$, i.e., $t\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} v_2\sigma$. Thus, "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$" is also an *innermost* chain.

Soundness of $Proc_{\mathbf{r}}$

We show that if "$\ldots, s \to t, v \to w, \ldots$" is a minimal innermost $(\mathcal{P}', \mathcal{R})$-chain, then "$\ldots, s \to t', v \to w, \ldots$" is a minimal innermost chain as well. There must be a σ with $t\sigma = t\sigma[t|_p\sigma]_p \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t\sigma[q]_p \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} v\sigma$ where $t|_p\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} q$, the terms q and $v\sigma$ are normal forms, and all instantiated right-hand sides (like $t\sigma$) are innermost terminating.

We proceed as in the completeness proof of $Proc_{\mathbf{n}}$ in the case $e = \mathbf{i}$. All rules applicable in a reduction of $t|_p\sigma$ are contained in $\mathcal{U}_{\mathcal{R}}(t|_p)$. Since $t|_p\sigma$ is weakly innermost terminating and $\mathcal{U}_{\mathcal{R}}(t|_p)$ is non-overlapping, by [21, Thm. 3.2.11] $t|_p\sigma$ is confluent and terminating. Note that $t' = t[r]_p$ where $t|_p \to_{\mathcal{R}} r$ for some term r. Hence, $t|_p\sigma \to_{\mathcal{R}} r\sigma$ and thus, $r\sigma$ is terminating as well. Thus, it also reduces innermost to some normal form q'. Now confluence of $t|_p\sigma$ implies $q = q'$. Hence, $t'\sigma = t\sigma[r\sigma]_p \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t\sigma[q]_p \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} v\sigma$. Therefore, "$\ldots, s \to t', v \to w, \ldots$" is an innermost chain, too. Moreover, the innermost chain is minimal, since $t'\sigma$ is innermost terminating. The reason is that w.l.o.g., every infinite reduction of $t'\sigma$ starts with reducing the subterm at position p, i.e., with $t'\sigma = t\sigma[r\sigma]_p \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t\sigma[q]_p$. However, $t\sigma[q]_p$ is innermost terminating, since $t\sigma$ is innermost terminating and since $t\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t\sigma[q]_p$.

Completeness of $Proc_{\mathbf{r}}$

If $\mathcal{R}$ is not innermost terminating, then completeness is again trivial. Otherwise, we show that if "$\ldots, s \to t', v \to w, \ldots$" is an innermost chain, then so is "$\ldots, s \to t, v \to w, \ldots$". Note that $t' = t[r]_p$ where $t|_p \to_{\mathcal{R}} r$ for some r. There must be a σ with $t'\sigma = t\sigma[r\sigma]_p \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t\sigma[q]_p \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} v\sigma$ where $r\sigma \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} q$ and the terms q and $v\sigma$ are normal forms.

Again, all rules applicable in a reduction of $t|_p\sigma$ are contained in $\mathcal{U}_{\mathcal{R}}(t|_p)$. Since $t|_p\sigma$ is weakly innermost terminating and $\mathcal{U}_{\mathcal{R}}(t|_p)$ is non-overlapping, $t|_p\sigma$ is terminating and confluent [21, Thm. 3.2.11]. Thus, the only normal form of $t|_p\sigma$ is q. Hence, this normal form can also be reached by innermost reductions. This implies $t\sigma = t\sigma[t|_p\sigma]_p \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} t\sigma[q]_p \xrightarrow{\mathrm{i}}{}^*_{\mathcal{R}} v\sigma$. So "$\ldots, s \to t, v \to w, \ldots$" is an innermost chain, too.

Soundness of $Proc_{\mathbf{i}}$

We first regard $e = \mathbf{t}$ and show that if "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$" is a minimal chain, then "$\ldots, v_1 \to w_1, s\mu \to t\mu, v_2 \to w_2, \ldots$" is also a minimal chain, where $\mu = \mathrm{mgu}(\mathrm{REN}(\mathrm{CAP}_{\mathcal{R}}(w_1)), s)$.

Let w_1 have the form $C[p_1, \ldots, p_n]$, where the context C contains no defined symbols or variables and all p_i have a defined root symbol or

they are variables. There is a substitution σ satisfying (**t1**) and (**t2**). Then $s\sigma = C[q_1, \ldots, q_n]$ for some terms q_i with $p_i\sigma \to_{\mathcal{R}}^* q_i$.

We have $\text{REN}(\text{CAP}_{\mathcal{R}}(w_1)) = C[y_1, \ldots, y_n]$ where the y_i are fresh variables. Let σ' be the modification of σ such that $\sigma'(y_i) = q_i$. Then $\text{REN}(\text{CAP}_{\mathcal{R}}(w_1))\sigma' = s\sigma = s\sigma'$, i.e., $\text{REN}(\text{CAP}_{\mathcal{R}}(w_1))$ and s unify. Let $\mu = \text{mgu}(\text{REN}(\text{CAP}_{\mathcal{R}}(w_1)), s)$. Thus, $\sigma' = \mu\tau$ for some substitution τ. As the variables of all (occurrences of all) pairs may be assumed disjoint, we may modify σ to behave like τ on the variables of $s\mu \to t\mu$. Then $w_1\sigma \to_{\mathcal{R}}^* s\sigma = s\sigma' = s\mu\tau = s\mu\sigma$ and $t\mu\sigma = t\mu\tau = t\sigma \to_{\mathcal{R}}^* v_2\sigma$. Thus, "$\ldots, v_1 \to w_1, s\mu \to t\mu, v_2 \to w_2, \ldots$" is a chain, too. Moreover, since $t\sigma$ is terminating and $t\mu\sigma = t\sigma$, the chain is minimal as well.

In this way, one can replace all occurrences of $s \to t$ in chains by instantiated pairs $s\mu \to t\mu$, except for the very first pair in the chain. However, if $s \to t, v_1 \to w_1, v_2 \to w_2, \ldots$ is an infinite minimal chain, then so is $v_1 \to w_1, v_2 \to w_2, \ldots$ Thus, by deleting the possibly remaining first occurrence of $s \to t$ in the end, every infinite minimal $(\mathcal{P}', \mathcal{R})$-chain can indeed be transformed into an infinite minimal chain which only contains instantiations of $s \to t$.

Now we regard $e = \mathbf{i}$. Let "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$" be a minimal *innermost* chain. We show that then "$\ldots, v_1 \to w_1, s\mu \to t\mu, v_2 \to w_2, \ldots$" is a minimal *innermost* chain as well. Here, we have $\mu = \text{mgu}(\text{CAP}_{\mathcal{R}}(w_1), s)$ and $s\mu$ and $v_1\mu$ are in normal form.

Let w_1 have the form $C[p_1, \ldots, p_n]$, where C contains no defined symbols (but C may contain variables) and all $\text{root}(p_i)$ are defined. There is a σ satisfying (**i1**) – (**i3**). Then $s\sigma = C[q_1, \ldots, q_n]$ for some q_i with $p_i\sigma \xrightarrow{\text{i}}_{\mathcal{R}}^* q_i$, since σ instantiates all variables by normal forms.

We have $\text{CAP}_{\mathcal{R}}(w_1) = C[y_1, \ldots, y_n]$ where the y_i are fresh variables. Let σ' be the modification of σ with $\sigma'(y_i) = q_i$. Then $\text{CAP}_{\mathcal{R}}(w_1)\sigma' = s\sigma = s\sigma'$, i.e., $\text{CAP}_{\mathcal{R}}(w_1)$ and s unify. Let $\mu = \text{mgu}(\text{CAP}_{\mathcal{R}}(w_1), s)$. Since $s\sigma$ and $v_1\sigma$ are normal forms by (**i2**) and since μ is more general than σ, $s\mu$ and $v_1\mu$ are normal forms as well. The remainder is as for $e = \mathbf{t}$.

Completeness of $Proc_{\mathbf{i}}$

Again, if $\mathcal{R}$ is not (innermost) terminating, completeness is trivial. Otherwise, let "$\ldots, s\mu \to t\mu, \ldots$" be an (innermost) chain. As different occurrences of pairs may be assumed variable disjoint, we can extend every substitution σ to behave like $\mu\sigma$ on the variables of s. Then one immediately obtains that "$\ldots, s \to t, \ldots$" is also an (innermost) chain.

Soundness of $Proc_{\mathbf{f}}$

We first regard $e = \mathbf{t}$ and show that if "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$" is a minimal chain, then so is "$\ldots, v_1 \to w_1, s\mu \to t\mu, v_2 \to$

$w_2, \ldots$", where $\mu = \mathrm{mgu}(\mathrm{REN}(\mathrm{CAP}_{\mathcal{R}}^{-1}(v_2)), t)$. The non-trivial case is if $\mathcal{R}$ is not collapsing, since otherwise μ does not modify s or t.

Let v_2 have the form $C[p_1, \ldots, p_n]$, where C contains no root symbols of $\mathcal{R}$'s right-hand sides or variables and the p_i are variables or terms where $\mathrm{root}(p_i)$ occurs on root positions of $\mathcal{R}$'s right-hand sides. There is a substitution σ satisfying (**t1**) and (**t2**). Then $t\sigma = C[q_1, \ldots, q_n]$ for some terms q_i with $q_i \to_{\mathcal{R}}^* p_i\sigma$, i.e., $p_i\sigma \to_{\mathcal{R}^{-1}}^* q_i$.

We have $\mathrm{REN}(\mathrm{CAP}_{\mathcal{R}}^{-1}(v_2)) = C[y_1, \ldots, y_n]$ where the y_i are fresh variables. Let σ' be the modification of σ where $\sigma'(y_i) = q_i$. Then $\mathrm{REN}(\mathrm{CAP}_{\mathcal{R}}^{-1}(v_2))\sigma' = t\sigma = t\sigma'$, i.e., $\mathrm{REN}(\mathrm{CAP}_{\mathcal{R}}^{-1}(v_2))$ and t unify. Let $\mu = \mathrm{mgu}(\mathrm{REN}(\mathrm{CAP}_{\mathcal{R}}^{-1}(v_2)), t)$. The rest is as in $Proc_{\mathbf{i}}$'s soundness proof.

Now we regard $e = \mathbf{i}$. We show that if "$\ldots, v_1 \to w_1, s \to t, v_2 \to w_2, \ldots$" is a minimal *innermost* chain, then so is "$\ldots, v_1 \to w_1, s\mu \to t\mu, v_2 \to w_2, \ldots$", where $\mu = \mathrm{mgu}(\mathrm{REN}(\mathrm{CAP}_{\mathcal{U}_{\mathcal{R}}(t)}^{-1}(v_2)), t)$. Again, we only regard the non-trivial case where $\mathcal{U}_{\mathcal{R}}(t)$ is not collapsing.

Let v_2 have the form $C[p_1, \ldots, p_n]$, where C and p_i are as in the termination case. There is a substitution σ satisfying (**i1**) – (**i3**). The only rules applicable in a reduction of $t\sigma$ are from $\mathcal{U}_{\mathcal{R}}(t)$. Thus, $t\sigma = C[q_1, \ldots, q_n]$ for some q_i with $q_i \xrightarrow{\mathrm{i}}_{\mathcal{U}_{\mathcal{R}}(t)}^* p_i\sigma$ and hence, $p_i\sigma \to_{\mathcal{U}_{\mathcal{R}}(t)^{-1}}^* q_i$.

We have $\mathrm{REN}(\mathrm{CAP}_{\mathcal{U}_{\mathcal{R}}(t)}^{-1}(v_2)) = C[y_1, \ldots, y_n]$ where the y_i are fresh variables. Let σ' be the modification of σ such that $\sigma'(y_i) = q_i$. Then $\mathrm{REN}(\mathrm{CAP}_{\mathcal{U}_{\mathcal{R}}(t)}^{-1}(v_2))\sigma' = t\sigma = t\sigma'$, i.e., $\mathrm{REN}(\mathrm{CAP}_{\mathcal{U}_{\mathcal{R}}(t)}^{-1}(v_2))$ and t unify. Let $\mu = \mathrm{mgu}(\mathrm{REN}(\mathrm{CAP}_{\mathcal{U}_{\mathcal{R}}(t)}^{-1}(v_2)), t)$. The remainder of the proof is analogous to the soundness proof of $Proc_{\mathbf{i}}$.

Completeness of $Proc_{\mathbf{f}}$

The proof is analogous to the completeness proof of $Proc_{\mathbf{i}}$. □

The following example shows that the processors $Proc_{\mathbf{n}}$ and $Proc_{\mathbf{r}}$ are not complete if one only considers those DP problems as "infinite" that are not finite. Instead, a DP problem $(\mathcal{P}, \mathcal{R}, e)$ should also be considered "infinite" whenever $\mathcal{R}$ is not (innermost) terminating.

EXAMPLE 32. *For the TRS* $\mathcal{R} = \{\mathsf{f}(\mathsf{b}) \to \mathsf{f}(\mathsf{g}(\mathsf{a})), \mathsf{g}(x) \to \mathsf{b}, \mathsf{a} \to \mathsf{a}\}$, *the dependency graph processor of Thm. 8 results in the DP problems* $(\{\mathsf{A} \to \mathsf{A}\}, \mathcal{R}, e)$ *and* $(\{\mathsf{F}(\mathsf{b}) \to \mathsf{F}(\mathsf{g}(\mathsf{a}))\}, \mathcal{R}, e)$. *The latter DP problem is finite as* $\mathsf{F}(\mathsf{g}(\mathsf{a}))$ *is not terminating and thus, the pair* $\mathsf{F}(\mathsf{b}) \to \mathsf{F}(\mathsf{g}(\mathsf{a}))$ *cannot occur in infinite* minimal *chains. On the other hand, applying* $Proc_{\mathbf{n}}$ *or* $Proc_{\mathbf{r}}$ *on this DP problem leads to a new DP problem* $(\mathcal{P}', \mathcal{R}, e)$ *where* $\mathcal{P}'$ *contains* $\mathsf{F}(\mathsf{b}) \to \mathsf{F}(\mathsf{b})$. *But since* $\mathsf{F}(\mathsf{b}) \to \mathsf{F}(\mathsf{b})$ *leads to an infinite minimal chain, the problem* $(\mathcal{P}', \mathcal{R}, e)$ *is not finite.*

By Thm. 31, the transformations are sound and (under certain conditions) complete. Then they cannot transform a non-infinite DP problem into an infinite one, but they may still be disadvantageous. The reason is that transformations may increase the size of DP problems (and thus, runtimes may increase, too). On the other hand, transformations are often needed to prove (innermost) termination, as shown by Ex. 30.

In practice, the main problem is that these transformations may be applied infinitely many times. For instance, already in our initial Ex. 1 with the dependency pair

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

we can obtain an infinite sequence of narrowing steps. The DP problem $(\{(3)\},\mathcal{R},\mathbf{i})$ can be narrowed to $(\{(32),(33)\},\mathcal{R},\mathbf{i})$ with

$$\mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(\mathsf{0})) \to \mathsf{DIV}(x,\mathsf{s}(\mathsf{0})) \tag{32}$$
$$\mathsf{DIV}(\mathsf{s}(\mathsf{s}(x)),\mathsf{s}(\mathsf{s}(y))) \to \mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(\mathsf{s}(y))) \tag{33}$$

Another narrowing step yields $(\{(32),(34),(35)\},\mathcal{R},\mathbf{i})$ with

$$\mathsf{DIV}(\mathsf{s}(\mathsf{s}(x)),\mathsf{s}(\mathsf{s}(\mathsf{0}))) \to \mathsf{DIV}(x,\mathsf{s}(\mathsf{s}(\mathsf{0}))) \tag{34}$$
$$\mathsf{DIV}(\mathsf{s}(\mathsf{s}(\mathsf{s}(x))),\mathsf{s}(\mathsf{s}(\mathsf{s}(y)))) \to \mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(\mathsf{s}(\mathsf{s}(y)))) \tag{35}$$

Obviously, narrowing can be repeated infinitely many times here, although the DP problems are finite and the TRS is terminating.

Therefore, we have developed restricted *safe* transformations which are guaranteed to terminate. Our experiments in Sect. 8 show that applying transformations only in these *safe* cases is indeed successful in practice. The experiments also demonstrate that this is clearly advantageous to the alternative straightforward heuristic which simply applies transformations a fixed number of times.

A narrowing, instantiation, or forward instantiation step is *safe* if it reduces the number of pairs in SCCs of the estimated (innermost) dependency graph. Let $\mathrm{SCC}(\mathcal{P})$ be the set of SCCs built from the pairs in $\mathcal{P}$. Then the transformation is safe if $|\bigcup_{\mathcal{S}\in\mathrm{SCC}(\mathcal{P})}\mathcal{S}|$ decreases. So the forward instantiation in Ex. 29 was safe, since the estimated dependency graph had an SCC before, but not afterwards. Moreover, a transformation step is also considered safe if by this step, all descendants of an original dependency pair disappear from SCCs. For every pair $s\to t$, $o(s\to t)$ denotes an 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. Finally, for each pair that was not narrowed

yet, one single narrowing step which does not satisfy the above requirements is also considered safe. The benefits of this are demonstrated by our experiments in Sect. 8. We use a similar condition for instantiation and forward instantiation, where for forward instantiation, we require that the transformation may only yield *one* new pair. So the narrowing and instantiation steps in Ex. 30 were safe as well.

The rewriting transformation may be applied without any restriction provided that the rules used for (innermost) rewriting are (innermost) terminating. Therefore, one should consider the recursion hierarchy in termination proofs. A symbol f *depends on* the symbol h ($f \geq_d h$) if $f = h$ or if some symbol g occurs in the right-hand side of an f-rule where g depends on h. Note that $\geq_d$ also corresponds to the definition of usable rules in Def. 10, since $\mathcal{U}_\mathcal{R}(f(x_1, \ldots, x_n))$ consists of $Rls_\mathcal{R}(g)$ for all $f \geq_d g$. We define $>_d = \geq_d \setminus \leq_d$. For example, the div- and if-rules depend on the minus-and le-rules in Ex. 30. If one has to solve two DP problems $(\mathcal{P}_1, \mathcal{R}, e)$ and $(\mathcal{P}_2, \mathcal{R}, e)$ where there exist $l_1^\sharp \to r_1^\sharp \in \mathcal{P}_1$ and $l_2^\sharp \to r_2^\sharp \in \mathcal{P}_2$ with $\mathrm{root}(l_1) >_d \mathrm{root}(l_2)$, then it is advantageous to treat $(\mathcal{P}_2, \mathcal{R}, e)$ before $(\mathcal{P}_1, \mathcal{R}, e)$. In other words, in Ex. 30 one should solve the DP problems for minus and le before handling the DP problem of div and if. Then innermost termination of minus is already verified when proving innermost termination of div and therefore, innermost rewriting the DIV-dependency pair with the minus-rules is guaranteed to terminate. Thus, the rewrite step from (28) to (29) was safe.

DEFINITION 33 (Safe Transformations). *Let $\mathcal{Q}$ result from the set $\mathcal{P}$ by transforming $s \to t \in \mathcal{P}$ as in Def. 28. The transformation is* safe *if*

(1) $s \to t$ was transformed by $Proc_\mathbf{n}$, $Proc_\mathbf{i}$, or $Proc_\mathbf{f}$ and
- $|\bigcup_{\mathcal{S} \in \mathrm{SCC}(\mathcal{P})} \mathcal{S}| > |\bigcup_{\mathcal{S} \in \mathrm{SCC}(\mathcal{Q})} \mathcal{S}|$, *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 and $\mathcal{U}_\mathcal{R}(\mathcal{P})$ is innermost terminating

(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

(5) $s \to t$ was transformed by forward instantiation, there is only one pair $v \to w$ where $\mathrm{REN}(\mathrm{CAP}_\mathcal{R}^{-1}(v))$ and t (resp. $\mathrm{REN}(\mathrm{CAP}_{\mathcal{U}_\mathcal{R}(t)}^{-1}(v))$ and t) are unifiable, and all previous steps which transformed $o(s \to t)$ to $s \to t$ were not forward instantiation steps

Thm. 34 proves that the application of safe transformations terminates.

THEOREM 34 (Termination). *Let $(\mathcal{P}, \mathcal{R}, e)$ be a DP problem. Then any repeated application of safe transformations on $\mathcal{P}$ terminates.*

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

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

These 4-tuples are compared lexicographically by the usual order on naturals for components (a) – (c). For (d), we use the multiset extension of the innermost rewrite relation of $\mathcal{U}_{\mathcal{R}}(\mathcal{P})$ if it is innermost terminating. 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 $\mathcal{P}_2$'s measure. Due to (a), (b), and (d), any safe transformation of $\mathcal{P}$ with (1) or (2) decreases $\mathcal{P}$'s measure.

Let $w(\mathcal{P}) = \langle \mathcal{P}_{\neg n,\neg i,\neg f}, \mathcal{P}_{n,\neg i,\neg f}, \mathcal{P}_{\neg n,i,\neg f}, \mathcal{P}_{\neg n,\neg i,f}, \mathcal{P}_{\neg n,i,f}, \mathcal{P}_{n,\neg i,f}, \mathcal{P}_{n,i,\neg f}, \mathcal{P}_{n,i,f} \rangle$. $\mathcal{P}_{\neg n,\neg i,\neg f}$ consists of those $s \to t \in \mathcal{P}$ where no **n**arrowing, **i**nstantiation, or **f**orward instantiation was used to transform $o(s \to t)$ to $s \to t$. $\mathcal{P}_{n,\neg i,\neg f}$ are the pairs where narrowing, but no instantiation or forward instantiation was used, etc. Every safe transformation step decreases $w(\mathcal{P})$ lexicographically w.r.t. $\succ$: The leftmost component of $w(\mathcal{P})$ that is changed decreases w.r.t. $\succ$, whereas components on its right-hand side may increase. In particular, transformations with (3) – (5) decrease one component of $w(\mathcal{P})$ w.r.t. $\succ$ by (c). □

A good strategy is to apply the processors of this paper according to the following precedence. This strategy is also used in our experiments in Sect. 8. Here, one always uses the first processor in the list which modifies the current DP problem. More elaborate strategies which also take other processors into account can be found in [15].

1. Dependency Graph Processor (Thm. 8)
2. Transformation Processor (Def. 28) restricted to Def. 33 (1), (2)
3. Reduction Pair Processor (Thm. 12, Thm. 17, or Thm. 26)
4. Transformation Processor (Def. 28) restricted to Def. 33 (3) – (5)

So after each transformation, one should re-compute the dependency graph. Here, one only has to regard the former neighbors of the transformed pair in the old graph. The reason is that only former neighbors may have arcs to or from the new pairs resulting from the transformation. Regarding neighbors in the graph also suffices for the unifications

required for narrowing, instantiation, and forward instantiation. In this way, the transformations can be performed efficiently.

6. Computing Argument Filterings

One of the most important processors of the DP framework is the reduction pair processor (Thm. 12) which we improved considerably in Thm. 17 and 26. Here, we may apply an argument filtering π to the constraints before orienting them with a reduction pair. When using reduction pairs based on monotonic orders $\succ$ like RPOS or KBO, this is necessary to benefit from the fact that in a reduction pair $(\succsim, \succ)$, $\succ$ need not be monotonic. However, the number of argument filterings is exponential in the number and the arities of the function symbols. We now show how to search for suitable filterings efficiently. More precisely, for each DP problem $(\mathcal{P}, \mathcal{R}, e)$, we show how to compute a small set $\Pi(\mathcal{P}, \mathcal{R})$. This set includes all argument filterings that may satisfy the constraints of the reduction pair processor. A corresponding algorithm was presented in [25] for termination proofs with Thm. 12. However, in Sect. 6.1 and Sect. 6.2 we now develop algorithms which can also be used for the improved versions of the reduction pair processor from Thm. 17 and 26. In particular for Thm. 26, the algorithm is considerably more involved since the set of constraints depends on the argument filtering used.

6.1. Argument Filterings for the Processor of Thm. 17

We use the approach of [25] to consider partial argument 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 domain of π may even be empty, i.e., $DOM(\pi) = \varnothing$. An argument filtering π is sufficiently defined for $f(t_1, \ldots, t_n)$ iff $f \in DOM(\pi)$ and π is sufficiently defined for all t_i with $i \in rp_\pi(f)$. 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, [25] introduced the following relation "$\subseteq$": $\pi \subseteq \pi'$ iff $DOM(\pi) \subseteq DOM(\pi')$ and $\pi(f) = \pi'(f)$ for all $f \in DOM(\pi)$.

In [25], one regards all $\subseteq$-minimal filterings which permit a term to be evaluated. We now use the same concept to define the set $\Pi(\mathcal{P})$

of those argument filterings where at least one pair in $\mathcal{P}$ is strictly decreasing and the remaining ones are weakly decreasing. Here, $\Pi(\mathcal{P})$ should only contain $\subseteq$-minimal elements, i.e., if $\pi' \in \Pi(\mathcal{P})$, then $\Pi(\mathcal{P})$ does not contain any $\pi \subset \pi'$. Of course, all filterings in $\Pi(\mathcal{P})$ must be sufficiently defined for the terms in the pairs of $\mathcal{P}$. Let $\mathcal{RP}$ be a class of reduction pairs describing the particular base order used (e.g., $\mathcal{RP}$ may contain all reduction pairs based on LPO). In the termination case, we restrict ourselves to $\mathcal{C}_\varepsilon$-compatible reduction pairs.

DEFINITION 35 ($\Pi(\mathcal{P})$). *For a set $\mathcal{P}$ of pairs of terms, let $\Pi(\mathcal{P})$ consist of all $\subseteq$-minimal elements of $\{\pi \mid$ there is a $(\succsim, \succ) \in \mathcal{RP}$ such that $\pi(s) \succ \pi(t)$ for some $s \to t \in \mathcal{P}$ and $\pi(s) \succsim \pi(t)$ for all other $s \to t \in \mathcal{P}\}$.*

In Ex. 1, if $\mathcal{P} = \{\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y))\}$ and $\mathcal{RP}$ are all reduction pairs based on LPO, then $\Pi(\mathcal{P})$ consists of the 12 filterings π where $DOM(\pi) = \{\mathsf{DIV}, \mathsf{s}, \mathsf{minus}\}$, $\pi(\mathsf{DIV}) \in \{1, [1], [1, 2]\}$, and either $\pi(\mathsf{minus}) \in \{[\,], 1, [1]\}$ and $\pi(\mathsf{s}) = [1]$ or both $\pi(\mathsf{minus}) = \pi(\mathsf{s}) = [\,]$.

For any DP problem $(\mathcal{P}, \mathcal{R}, e)$, we now define a superset $\Pi(\mathcal{P}, \mathcal{R})$ of all argument filterings where the constraints of the reduction pair processor from Thm. 17 are satisfied by some reduction pair of $\mathcal{RP}$. So only these filterings have to be regarded when automating Thm. 17, cf. Thm. 43. As in [25], one therefore has to *extend* partial filterings in order to obtain all filterings which can possibly satisfy certain inequalities.

DEFINITION 36 (Ex_f, $\Pi(\mathcal{P}, \mathcal{R})$). *For a partial filtering π and $f \in \mathcal{D}_\mathcal{R}$, $Ex_f(\pi)$ consists of all $\subseteq$-minimal filterings π' with $\pi \subseteq \pi'$ such that there is a $(\succsim, \succ) \in \mathcal{RP}$ with $\pi'(l) \succsim \pi'(r)$ for all $l \to r \in Rls_\mathcal{R}(f)$. For a set Π of filterings, let $Ex_f(\Pi) = \bigcup_{\pi \in \Pi} Ex_f(\pi)$. We define $\Pi(\mathcal{P}, \mathcal{R}) = Ex_{f_k}(\ldots Ex_{f_1}(\Pi(\mathcal{P}))\ldots)$, where $f_1, \ldots, f_k$ are $\mathcal{U}_\mathcal{R}(\mathcal{P})$'s defined symbols. Note that the order of $f_1, \ldots, f_k$ is irrelevant for the definition of $\Pi(\mathcal{P}, \mathcal{R})$.*

So for the DIV-SCC $\mathcal{P}$ in Ex. 1, we have $\Pi(\mathcal{P}, \mathcal{R}) = Ex_{\mathsf{minus}}(\Pi(\mathcal{P}))$. Note that all $\pi \in \Pi(\mathcal{P})$ remove the second argument of minus. Therefore in $Ex_{\mathsf{minus}}(\Pi(\mathcal{P}))$, their domain does not have to be extended to the symbol 0, since 0 only occurs in minus' second argument. However, in $Ex_{\mathsf{minus}}(\Pi(\mathcal{P}))$, all filterings π with $\pi(\mathsf{minus}) = [\,]$ are eliminated, since they contradict the weak decrease of the first minus-rule. Thus, while there exist $6 \cdot 3 \cdot 6 \cdot 1 = 144$ possible argument filterings for the symbols DIV, s, minus, and 0, our algorithm reduces this set to only $|\Pi(\mathcal{P}, \mathcal{R})| = 6$ candidates. For TRSs with more function symbols, the reduction of the search space is of course even more dramatic.

Moreover, in successful proofs we only compute a small subset of $\Pi(\mathcal{P}, \mathcal{R})$, since its elements are determined step by step in a *depth-first*

search until a proof is found. To this end, we start with a $\pi \in \Pi(\mathcal{P})$ and extend it to a minimal π' such that f_1's rules are weakly decreasing. Then π' is extended such that f_2's rules are weakly decreasing, etc. Here, f_1 is considered before f_2 if $f_1 >_d f_2$, where $>_d$ is again the "*dependence*" relation from Sect. 5.2. When we have $\Pi(\mathcal{P},\mathcal{R})$'s first element π_1, we check whether the constraints of Thm. 17 are satisfiable with π_1. In case of success, we do not compute further elements of $\Pi(\mathcal{P},\mathcal{R})$. Only if the constraints are not satisfiable with π_1, we determine $\Pi(\mathcal{P},\mathcal{R})$'s next element, etc. The advantage of this approach is that $\Pi(\mathcal{P})$ is usually small, since it only contains filterings that satisfy a *strict* inequality. Thus, by taking $\Pi(\mathcal{P})$'s restrictions into account, only a fraction of the search space is examined. This depth-first strategy differs from the corresponding algorithm in [25] where the constraints are treated separately in order to share and re-use results.

EXAMPLE 37. *The following TRS illustrates the depth-first algorithm.*

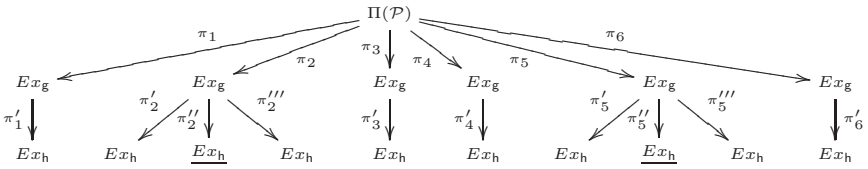
$$\begin{array}{rcl@{\qquad}rcl} \mathsf{f}(\mathsf{s}(\mathsf{s}(x))) & \to & \mathsf{f}(\mathsf{g}(x)) & \mathsf{g}(x) & \to & \mathsf{h}(x) \\ \mathsf{f}(\mathsf{s}(x)) & \to & \mathsf{f}(x) & \mathsf{h}(x) & \to & \mathsf{s}(x) \end{array}$$

The only SCC of the dependency graph is $\mathcal{P} = \{\mathsf{F}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{F}(\mathsf{g}(x)), \mathsf{F}(\mathsf{s}(x)) \to \mathsf{F}(x)\}$. *Let* $\mathcal{RP}$ *contains all reduction pairs based on LPO. Then* $\Pi(\mathcal{P})$ *consists of the six partial argument filterings where* $\pi(\mathsf{F})$ *is 1 or* $[1]$, $\pi(\mathsf{s}) = [1]$, *and* $\pi(\mathsf{g}) \in \{1, [1], [\,]\}$.

To demonstrate the depth-first algorithm, we inspect filterings in the order where collapsing filterings are checked first and otherwise, filterings which do not filter anything are preferred (so the order of preference is 1, $[1]$, $[\,]$*). Then we start with* $\pi_1 \in \Pi(\mathcal{P})$ *where* $\pi_1(\mathsf{F}) = 1$, $\pi_1(\mathsf{s}) = [1]$, *and* $\pi_1(\mathsf{g}) = 1$. *When computing* $Ex_{\mathsf{g}}(\pi_1)$, *the argument filtering has to be extended in order to filter* h *as well. Moreover, we must ensure that the* g*-rule can be made weakly decreasing by some LPO. Thus, we have to use the extension* π_1' *of* π_1 *where* $\pi_1'(\mathsf{h}) = 1$. *But then, when trying to compute* $Ex_{\mathsf{h}}(\pi_1')$, *it turns out that this set is empty since the* h*-rule cannot be weakly decreasing with this filtering.*

So one backtracks and regards $\pi_2 \in \Pi(\mathcal{P})$ *with* $\pi_2(\mathsf{F}) = 1$, $\pi_2(\mathsf{s}) = [1]$, *and* $\pi_2(\mathsf{g}) = [1]$. *Now we compute the first element* π_2' *of* $Ex_{\mathsf{g}}(\pi_2)$. *We have* $\pi_2'(\mathsf{h}) = 1$ *and again,* $Ex_{\mathsf{h}}(\pi_2')$ *is empty. Hence, we backtrack and compute the next element* π_2'' *of* $Ex_{\mathsf{g}}(\pi_2)$. *Now* $\pi_2''(\mathsf{h}) = [1]$ *and* $Ex_{\mathsf{h}}(\pi_2'')$ *consists of* π_2''. *Thus, we have found the first element of* $\Pi(\mathcal{P},\mathcal{R}) = Ex_{\mathsf{h}}(Ex_{\mathsf{g}}(\Pi(\mathcal{P})))$. *Hence, we stop the computation of* $\Pi(\mathcal{P},\mathcal{R})$ *and check whether the reduction pair processor is successful with* π_2''. *Indeed, the constraints can be solved using an LPO where* s, g, *and* h *have equal precedence. Then both dependency pairs are decreasing and can be removed. Thus, termination can immediately be proved.*

The example demonstrates that the depth-first search only generates a small part of the search space when looking for argument filterings. This is also illustrated by the following tree which depicts the whole search space for determining $\Pi(\mathcal{P},\mathcal{R})$*. Since we use depth-first search and stop as soon as the first solution is found, we do not compute this full tree in a successful termination proof. Instead we stop as soon as we reach the third leaf, which corresponds to the first element of* $\Pi(\mathcal{P},\mathcal{R})$*. Here, those leaves where* Ex_{h} *is underlined denote success (i.e., computing* Ex_{h} *does not result in the empty set).*



6.2. Argument Filterings for the Processor of Thm. 26

When automating the improved reduction pair processor of Thm. 26 instead of Thm. 17, 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 g's rules as well, since all occurrences of g in f-rules may have been deleted by the argument filtering. To formalize this, we define a relation "$\Vdash_{\mathcal{P},\mathcal{R}}$" on sets of argument filterings. We extend rp_π to *partial* filterings by defining $rp_\pi(f) = \varnothing$ for $f \notin DOM(\pi)$. Now $\mathcal{U}_\mathcal{R}(\mathcal{P},\pi)$ is also defined for partial filterings by disregarding all subterms of function symbols where π is not defined.

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

DEFINITION 38 ($\Vdash_{\mathcal{P},\mathcal{R}}$). *Each argument filtering* π *is labelled with a set* $\mathcal{G} \subseteq \mathcal{D}_\mathcal{R}$ *and we denote a labelled argument filtering by* $\pi_\mathcal{G}$*. For labelled argument filterings, we define* $\pi_\mathcal{G} \Vdash_{\mathcal{P},\mathcal{R}} \pi'_{\mathcal{G}\cup\{f\}}$ *if* $f \in \mathcal{D}_\mathcal{R} \setminus \mathcal{G}$*,* $Rls_\mathcal{R}(f) \subseteq \mathcal{U}_\mathcal{R}(\mathcal{P},\pi)$*, and* $\pi' \in Ex_f(\pi)$*. We extend "*$\Vdash_{\mathcal{P},\mathcal{R}}$*" to sets of labelled filterings as follows:* $\Pi \uplus \{\pi_\mathcal{G}\} \Vdash_{\mathcal{P},\mathcal{R}} \Pi \cup \{\pi'_{\mathcal{G}'} \mid \pi_\mathcal{G} \Vdash_{\mathcal{P},\mathcal{R}} \pi'_{\mathcal{G}'}\}$.

To automate the reduction pair processor of Thm. 26, we only regard filterings that result from $\Pi(\mathcal{P})$ by applying $\Vdash_{\mathcal{P},\mathcal{R}}$-reductions as long as possible. So each $\pi \in \Pi(\mathcal{P})$ is extended individually by $\Vdash_{\mathcal{P},\mathcal{R}}$ instead of building $Ex_f(\Pi(\mathcal{P}))$ as in Thm. 17's automation. The advantage of $\Vdash_{\mathcal{P},\mathcal{R}}$ is that only those filterings π are extended to include f in their

domain where this is required by the usable rules $\mathcal{U}_\mathcal{R}(\mathcal{P},\pi)$. We denote the set of filterings that should be regarded when automating Thm. 26. by $\Pi'(\mathcal{P},\mathcal{R})$ and compute it by depth-first search, similar to $\Pi(\mathcal{P},\mathcal{R})$.

EXAMPLE 39. *We regard the SCC* $\mathcal{P} = \{(21),(22),(23),(24)\}$ *of the* REV- *and* REV2*-dependency pairs in Ex. 22 and 27. Here, the filterings in* $\Pi(\mathcal{P})$ *are not defined on* rev1*, since* rev1 *does not occur in* $\mathcal{P}$*. When performing* $\Vdash_{\mathcal{P},\mathcal{R}}$*-reductions, those* $\pi \in \Pi(\mathcal{P})$ *which eliminate the first argument of* cons *will never be extended to* rev1*, whereas this is necessary for other filterings in* $\Pi(\mathcal{P})$*.*

For example (possibly after some backtracking), the depth-first search could consider the partial filtering $\pi \in \Pi(\mathcal{P})$*:* $\pi(\mathsf{cons}) = [2]$*,* $\pi(\mathsf{REV}) = \pi(\mathsf{rev}) = 1$*, and* $\pi(\mathsf{REV1}) = \pi(\mathsf{REV2}) = \pi(\mathsf{rev2}) = 2$*, where* π *is undefined on* rev1*. Initially, this filtering is labelled with* $\varnothing$*, since no rule is guaranteed to be weakly decreasing yet. We have* $Rls_\mathcal{R}(\mathsf{rev}) \subseteq \mathcal{U}_\mathcal{R}(\mathcal{P},\pi)$*. To make* rev*'s rules weakly decreasing,* π *has to be extended to a filtering* π' *which is also defined on* nil*, i.e.,* $\pi_\varnothing \Vdash_{\mathcal{P},\mathcal{R}} \pi'_{\{\mathsf{rev}\}}$*. Since* $\mathcal{U}_\mathcal{R}(\mathcal{P},\pi') = Rls_\mathcal{R}(\mathsf{rev}) \cup Rls_\mathcal{R}(\mathsf{rev2})$ *and since* π' *also allows us to make all* rev2*-rules weakly decreasing, we obtain* $\pi'_{\{\mathsf{rev}\}} \Vdash_{\mathcal{P},\mathcal{R}} \pi'_{\{\mathsf{rev},\mathsf{rev2}\}}$*. Thus,* π' *is a normal form w.r.t.* $\Vdash_{\mathcal{P},\mathcal{R}}$ *and therefore, it is an element of* $\Pi'(\mathcal{P},\mathcal{R})$*. As soon as one finds a filtering where the constraints of Thm. 26 can be solved, the depth-first search stops and we do not determine further elements of* $\Pi'(\mathcal{P},\mathcal{R})$*.*

So to compute the set of regarded argument filterings $\Pi'(\mathcal{P},\mathcal{R})$ for Thm. 26, our aim is to construct the normal form of $\Pi(\mathcal{P})$ w.r.t. $\Vdash_{\mathcal{P},\mathcal{R}}$. To ensure that this normal form always exists and that it is unique, we will prove that $\Vdash_{\mathcal{P},\mathcal{R}}$ is confluent and terminating.

For this proof, 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. [25].[8]

LEMMA 40 (Incompatibility). *Let* T *be a finite set of terms.*

(a) Let π, π_1, π_2 *be (partial) argument filterings. Let* $\pi_1, \pi_2 \in \{\pi' \mid \pi \subseteq \pi'$ *and* π' *is sufficiently defined for* $T\}$*, where* π_1 *is a* $\subseteq$*-minimal element of this set. If* π_1 *and* π_2 *are compatible, then* $\pi_1 \subseteq \pi_2$*.*

[8] The notion of *compatibility* was introduced in [25] for a different purpose: there it was used to merge sets of argument filterings, whereas a statement like Lemma 40 was not presented in [25]. In the setting of [25], Lemma 40 would mean that "$\mathsf{AF}(T)$" consists of pairwise incompatible argument filterings. Here, "$\mathsf{AF}(T)$" are all minimal argument filterings that are sufficiently defined for all terms in the set T.

(b) If $\pi_1, \pi_2 \in Ex_f(\pi)$ are compatible, then $\pi_1 = \pi_2$. In other words, $Ex_f(\pi)$ 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, too.

Proof.

(a) We perform induction on T using the (multi)set 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, $\pi_1 = \pi \subseteq \pi_2$.

Next let $T = T' \uplus \{x\}$ for $x \in \mathcal{V}$. Now the claim follows from the induction hypothesis, since π_1 and π_2 are also sufficiently defined for T' and π_1 is a minimal extension of π with this property.

If $T = T' \uplus \{f(t_1, \ldots, t_n)\}$, then $f \in DOM(\pi_1)$. Let $T'' = T' \cup \{t_i \mid i \in rp_{\pi_1}(f)\}$. 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, $f \notin DOM(\pi)$ and we obtain the following minimal extension π_1' of π that is sufficiently defined for T'': $DOM(\pi_1') = DOM(\pi_1) \setminus \{f\}$ and $\pi_1'(g) = \pi_1(g)$ for all $g \in DOM(\pi_1')$. Then the induction hypothesis implies $\pi_1' \subseteq \pi_2$. Since π_1 only differs from π_1' on f and since $\pi_1(f) = \pi_2(f)$, we obtain $\pi_1 \subseteq \pi_2$.

(b) Let $\pi_1, \pi_2 \in Ex_f(\pi)$ 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 $Rls_{\mathcal{R}}(f)$, we use (a) to conclude both $\pi_1 \subseteq \pi_2$ and $\pi_2 \subseteq \pi_1$, which implies $\pi_1 = \pi_2$.

(c) Let $\pi_1', \pi_2' \in \Pi$, $\pi_1 \in Ex_f(\pi_1')$, $\pi_2 \in Ex_f(\pi_2')$, and $\pi_1 \neq \pi_2$. If $\pi_1' = \pi_2'$, then π_1 and π_2 are incompatible by (b). Otherwise $\pi_1' \neq \pi_2'$, and π_1' and π_2' are incompatible by the assumption about Π. As $\pi_1' \subseteq \pi_1$ and $\pi_2' \subseteq \pi_2$, then π_1 and π_2 are incompatible as well. □

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

THEOREM 41. *$\Vdash_{\mathcal{P},\mathcal{R}}$ is terminating and confluent on sets of filterings.*

Proof. Termination of $\Vdash_{\mathcal{P},\mathcal{R}}$ is obvious as the labellings increase in every $\Vdash_{\mathcal{P},\mathcal{R}}$-step. So for confluence, it suffices to show local confluence. The only crucial indeterminism in the definition of $\Vdash_{\mathcal{P},\mathcal{R}}$ is the choice of

f. Let $f_0, f_1 \in \mathcal{D}_\mathcal{R} \backslash \mathcal{G}$ with $f_0 \neq f_1$ and $Rls_\mathcal{R}(f_0) \cup Rls_\mathcal{R}(f_1) \subseteq \mathcal{U}_\mathcal{R}(\mathcal{P}, \pi)$ for some labelled filtering $\pi_\mathcal{G}$. This leads to two possible steps:

$$\Pi \uplus \{\pi_\mathcal{G}\} \ \Vdash_{\mathcal{P},\mathcal{R}} \ \Pi \cup \Pi_0, \quad \text{where } \Pi_0 = \{\pi^0_{\mathcal{G}\cup\{f_0\}} \mid \pi^0 \in Ex_{f_0}(\pi)\}$$
$$\Pi \uplus \{\pi_\mathcal{G}\} \ \Vdash_{\mathcal{P},\mathcal{R}} \ \Pi \cup \Pi_1, \quad \text{where } \Pi_1 = \{\pi^1_{\mathcal{G}\cup\{f_1\}} \mid \pi^1 \in Ex_{f_1}(\pi)\}$$

Note that $\mathcal{U}_\mathcal{R}(\mathcal{P}, \pi) \subseteq \mathcal{U}_\mathcal{R}(\mathcal{P}, \pi^i)$ holds for all $\pi^i \in Ex_{f_i}(\pi)$. Thus, for all $\pi^i_{\mathcal{G}\cup\{f_i\}} \in \Pi_i$, we have $f_{1-i} \in \mathcal{D}_\mathcal{R} \setminus (\mathcal{G} \cup \{f_i\})$ and $Rls_\mathcal{R}(f_{1-i}) \subseteq \mathcal{U}_\mathcal{R}(\mathcal{P}, \pi^i)$. Hence, we obtain the following reductions (where we also apply Ex_f to labelled filterings by simply ignoring their labels).

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

where $Ex_{f_0}(\Pi \cap \Pi_1) \subseteq Ex_{f_0}(Ex_{f_1}(\pi))$, $Ex_{f_1}(\Pi \cap \Pi_0) \subseteq Ex_{f_1}(Ex_{f_0}(\pi))$. To finish the proof we show $Ex_{f_0}(Ex_{f_1}(\pi)) = Ex_{f_1}(Ex_{f_0}(\pi))$. By symmetry, it suffices to prove $Ex_{f_0}(Ex_{f_1}(\pi)) \subseteq Ex_{f_1}(Ex_{f_0}(\pi))$. Here, we only have to show that for every $\pi_{01} \in Ex_{f_0}(Ex_{f_1}(\pi))$ there is a $\pi_{10} \in Ex_{f_1}(Ex_{f_0}(\pi))$ with $\pi_{10} \subseteq \pi_{01}$. 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 $\pi'_{01} \subseteq \pi_{10}$. Hence, $\pi'_{01} \subseteq \pi_{10} \subseteq \pi_{01}$. By Lemma 40 (b), $Ex_{f_1}(\pi)$ consists of pairwise incompatible argument filterings and hence by Lemma 40 (c), this also holds for $Ex_{f_0}(Ex_{f_1}(\pi))$. However, $\pi'_{01} \subseteq \pi_{01}$ implies that π'_{01} and π_{01} are compatible. So $\pi'_{01} = \pi_{01}$ and thus, $\pi'_{01} = \pi_{10} = \pi_{01}$.

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

Now we can define the set of argument filterings $\Pi'(\mathcal{P},\mathcal{R})$ that are regarded when automating the reduction pair processor of Thm. 26. Due to Thm. 41, we can define $\Pi'(\mathcal{P},\mathcal{R})$ in an unambiguous way (as the unique normal form of $\Pi(\mathcal{P})$).

DEFINITION 42 ($\Pi'(\mathcal{P},\mathcal{R})$). *Let $Nf_{\Vdash_{\mathcal{P},\mathcal{R}}}(\Pi)$ be the normal form of Π w.r.t. $\Vdash_{\mathcal{P},\mathcal{R}}$. Then we define $\Pi'(\mathcal{P},\mathcal{R}) = Nf_{\Vdash_{\mathcal{P},\mathcal{R}}}(\{\pi_\varnothing \mid \pi \in \Pi(\mathcal{P})\})$.*

Thm. 43 states that $\Pi(\mathcal{P},\mathcal{R})$ resp. $\Pi'(\mathcal{P},\mathcal{R})$ indeed contain all argument filterings which could possibly solve the constraints of Thm. 17 resp. Thm. 26. In this way the set of argument filterings is reduced dramatically and thus, efficiency is increased.

THEOREM 43. *Let $(\mathcal{P},\mathcal{R},e)$ be a DP problem. If the constraints of Thm. 17 (26) are satisfied by some reduction pair from $\mathcal{RP}$ and argument filtering π, then $\pi' \subseteq \pi$ for some $\pi' \in \Pi(\mathcal{P},\mathcal{R})$ ($\pi' \in \Pi'(\mathcal{P},\mathcal{R})$).*

Proof. Let the constraints (a) and (b) from Thm. 17 or 26 be solved by some filtering π and some $(\succsim,\succ) \in \mathcal{RP}$. We first consider Thm. 17. Let $f_1,\ldots,f_k$ be the defined symbols of $\mathcal{U}_\mathcal{R}(\mathcal{P})$. So we have $\Pi(\mathcal{P},\mathcal{R}) = Ex_{f_k}(\ldots Ex_{f_1}(\Pi(\mathcal{P}))\ldots)$. We show that for all $0 \leq j \leq k$ there is a $\pi_j \in Ex_{f_j}(\ldots Ex_{f_1}(\Pi(\mathcal{P}))\ldots)$ with $\pi_j \subseteq \pi$ by induction on j. For $j = 0$, since π solves the constraints in (a), there is a minimal filtering $\pi_0 \in \Pi(\mathcal{P})$ with $\pi_0 \subseteq \pi$. For $j > 0$, we assume that there is a $\pi_{j-1} \in Ex_{f_{j-1}}(\ldots Ex_{f_1}(\Pi(\mathcal{P}))\ldots)$ with $\pi_{j-1} \subseteq \pi$. As $\pi(l) \succsim \pi(r)$ for all f_j-rules $l \to r$, there is a filtering $\pi_j \in Ex_{f_j}(\pi_{j-1})$ with $\pi_j \subseteq \pi$.

For Thm. 26, let $\Pi(\mathcal{P}) = \Pi_0 \Vdash_{\mathcal{P},\mathcal{R}} \Pi_1 \Vdash_{\mathcal{P},\mathcal{R}} \ldots \Vdash_{\mathcal{P},\mathcal{R}} \Pi_k = \Pi'(\mathcal{P},\mathcal{R})$ be a $\Vdash_{\mathcal{P},\mathcal{R}}$-reduction to normal form. We show that for all $0 \leq j \leq k$ there is a $\pi_j \in \Pi_j$ with $\pi_j \subseteq \pi$ by induction on j. For $j = 0$, since π solves the constraints in (a), there is again a minimal filtering $\pi_0 \in \Pi(\mathcal{P})$ with $\pi_0 \subseteq \pi$. For $j > 0$, we assume that there is a $\pi_{j-1} \in \Pi_{j-1}$ with $\pi_{j-1} \subseteq \pi$. So 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 $Rls_\mathcal{R}(f) \subseteq \mathcal{U}_\mathcal{R}(\mathcal{P},\pi_{j-1})$. Since $\pi_{j-1} \subseteq \pi$, we have $\mathcal{U}_\mathcal{R}(\mathcal{P},\pi_{j-1}) \subseteq \mathcal{U}_\mathcal{R}(\mathcal{P},\pi)$ and hence, π also makes f's rules weakly decreasing by the constraints in (b). Thus, there is a $\pi_j \in Ex_f(\pi_{j-1}) \subseteq \Pi_j$ with $\pi_j \subseteq \pi$. □

The converse directions of this theorem do not hold, since in the computation of $\Pi(\mathcal{P},\mathcal{R})$ and $\Pi'(\mathcal{P},\mathcal{R})$, 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 from $\mathcal{RP}$.

7. Using Polynomial Orders for Dependency Pairs

In Sect. 6 we showed how to mechanize the reduction pair processor with argument filterings and monotonic orders like RPOS or KBO. Now we regard reduction pairs based on polynomial orders instead, which are not necessarily monotonic if one also permits the coefficient 0 in polynomials.[9] In contrast to RPOS and KBO, it is undecidable whether a set of constraints is satisfiable by polynomial orders, and thus one can only use sufficient criteria to automate them. However, in combination with dependency pairs, even linear polynomial interpretations with coefficients from $\{0, 1\}$ are already very powerful, cf. Sect. 8.

An advantage of polynomial orders is that one does not need any extra argument filtering anymore, since argument filtering can be simulated directly by the corresponding polynomials. If

$$\mathcal{P}ol(f(x_1, ..., x_n)) = a_1\, x_1^{b_{1,1}} \dots\, x_n^{b_{n,1}} + \dots + a_m\, x_1^{b_{1,m}} \dots\, x_n^{b_{n,m}} \quad (36)$$

for coefficients $a_j \geq 0$, then this corresponds to the argument filtering $\pi_{\mathcal{P}ol}$ with $\pi_{\mathcal{P}ol}(f) = [\, i \mid a_j > 0 \wedge b_{i,j} > 0 \text{ for some } 1 \leq j \leq m\,]$. However, disregarding argument filterings is a problem in our improved reduction pair processor (Thm. 26), since the filtering π is needed to compute the constraints resulting from the usable rules $\mathcal{U}_{\mathcal{R}}(\mathcal{P}, \pi)$. Hence, in Sect. 7.1 we adapt the reduction pair processor to polynomial orders such that it does not use argument filterings anymore. In Sect. 7.2 we improve this special reduction pair processor by eliminating the indeterminism in the constraints of type (a). Here, for any DP problem $(\mathcal{P}, \mathcal{R}, e)$ one had to select at least one pair from $\mathcal{P}$ which should be strictly decreasing.

7.1. Reduction Pair Processor With Polynomial Orders

When automating Thm. 26 with polynomial orders, one fixes the degree of the polynomials and then suitable coefficients have to be found automatically. So one starts with an *abstract* polynomial interpretation $\mathcal{P}ol$. For every function symbol f, $\mathcal{P}ol(f(x_1, \dots, x_n))$ is as in (36), but m and $b_{i,j}$ are fixed numbers, whereas a_j are *variable* coefficients. Then the constraints (a) of Thm. 26 have the form

$$\begin{array}{l} \bullet\ \mathcal{P}ol(s) - \mathcal{P}ol(t) > 0 \ \text{ for all } s \to t \in \mathcal{P}' \text{ for a non-empty } \mathcal{P}' \subseteq \mathcal{P} \\ \bullet\ \mathcal{P}ol(s) - \mathcal{P}ol(t) \geq 0 \ \text{ for all } s \to t \in \mathcal{P} \setminus \mathcal{P}' \end{array} \quad (37)$$

An abstract polynomial interpretation $\mathcal{P}ol$ can be turned into a concrete one by assigning a natural number to each variable coefficient a_j.

[9] We only consider polynomial orders with natural coefficients. Approaches for polynomials with negative or real coefficients can be found in [23, 33].

We denote such *assignments* by α and let $\alpha(\mathcal{P}ol)$ be the corresponding concrete polynomial interpretation. A set of constraints of the form $p \,{}_{(}\!\geq_{)}\, 0$ is *satisfiable* iff there exists an assignment α such that all instantiated constraints $\alpha(p) \,{}_{(}\!\geq_{)}\, 0$ hold. Here $\alpha(p)$ still contains the variables $x, y, \ldots$ occurring in $\mathcal{P}$ and we say that $\alpha(p) \,{}_{(}\!\geq_{)}\, 0$ *holds* iff it is true for all instantiations of $x, y, \ldots$ by natural numbers. For example, $a_1x + a_2 - a_3y > 0$ is satisfied by the assignment α where $\alpha(a_1) = 1$, $\alpha(a_2) = 1$, and $\alpha(a_3) = 0$. The reason is that α turns the above constraint into $x + 1 > 0$ which holds for all instantiations of x with natural numbers.

The constraints of type (b) in Thm. 26 require $l \gtrsim r$ for all $l \to r \in \mathcal{U}_{\mathcal{R}}(\mathcal{P}, \pi_{\alpha(\mathcal{P}ol)})$. The problem is how to determine these constraints in the case of polynomial orders where the argument filtering $\pi_{\alpha(\mathcal{P}ol)}$ is not given explicitly but depends on the assignment α of natural numbers to variable coefficients a_j. Thus, $\pi_{\alpha(\mathcal{P}ol)}$ is not available yet when building the constraints although we need it to compute $\mathcal{U}_{\mathcal{R}}(\mathcal{P}, \pi_{\alpha(\mathcal{P}ol)})$.

The solution is to translate the constraints of type (b) to polynomial constraints of the following form:

$$q \cdot (\mathcal{P}ol(l) - \mathcal{P}ol(r)) \geq 0 \;\text{ for all rules } l \to r \text{ of } \mathcal{U}_{\mathcal{R}}(\mathcal{P}) \qquad (38)$$

Here, q will be a polynomial containing only variable coefficients a_j but no variables $x, y, \ldots$ from the rules of $\mathcal{R}$. So for any assignment α, $\alpha(q)$ is a number. We generate the constraints such that for *any* assignment α, we have $\alpha(q) = 0$ iff $l \to r \notin \mathcal{U}_{\mathcal{R}}(\mathcal{P}, \pi_{\alpha(\mathcal{P}ol)})$. So for any α, the constraints (38) are equivalent to requiring $\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0$ for all $l \to r$ in $\mathcal{U}_{\mathcal{R}}(\mathcal{P}, \pi_{\alpha(\mathcal{P}ol)})$, but the advantage is that the constraints (38) can be constructed before determining the assignment α.

Let $\mathcal{P}ol$ be an abstract polynomial interpretation as in (36). To generate the constraints (38), we first define a polynomial which sums up the coefficients of those monomials of $\mathcal{P}ol(f(x_1, ..., x_n))$ that contain x_i.

$$rp_{\mathcal{P}ol}(f, i) = \sum\nolimits_{1 \leq j \leq m,\, b_{i,j} > 0} a_j$$

So for any assignment α, $\alpha(rp_{\mathcal{P}ol}(f, i))$ is a number which is greater than 0 iff $i \in rp_{\pi_{\alpha(\mathcal{P}ol)}}(f)$. Now the constraints corresponding to (b) in Thm. 26 can be built similar to the definition of usable rules (Def. 21).

DEFINITION 44 (Usable Rules for Polynomial Orders). *Let $\mathcal{P}ol$ be an abstract polynomial interpretation. Again, let $\mathcal{R}' = \mathcal{R} \setminus Rls_{\mathcal{R}}(f)$. For any term t, we define the* usable rule constraints *$Con_{\mathcal{R}}(t, \mathcal{P}ol)$ as*

- *$Con_{\mathcal{R}}(x, \mathcal{P}ol) = \varnothing$ for $x \in \mathcal{V}$ and*
- *$Con_{\mathcal{R}}(f(t_1, \ldots, t_n), \mathcal{P}ol) = \{\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0 \mid l \to r \in Rls_{\mathcal{R}}(f)\}$*
  $\cup \;\bigcup_{l \to r \in Rls_{\mathcal{R}}(f)} Con_{\mathcal{R}'}(r, \mathcal{P}ol)$
  $\cup \;\bigcup_{1 \leq i \leq n} \{rp_{\mathcal{P}ol}(f, i) \cdot p \;\geq\; 0 \;\mid\; \text{“}p \geq 0\text{”} \in Con_{\mathcal{R}'}(t_i, \mathcal{P}ol)\}$.

For any TRS $\mathcal{P}$, let $Con_{\mathcal{R}}(\mathcal{P}, \mathcal{P}ol) = \bigcup_{s \to t \in \mathcal{P}} Con_{\mathcal{R}}(t, \mathcal{P}ol)$.

EXAMPLE 45. *Consider the TRS from Ex. 22 again. We use the linear abstract polynomial interpretation $\mathcal{P}ol(\mathsf{nil}) = a_{\mathsf{nil}}$, $\mathcal{P}ol(f(x)) = a_{f,0} + a_{f,1}\ x$ for $f \in \{\mathsf{rev}, \mathsf{REV}\}$, and $\mathcal{P}ol(f(x,y)) = a_{f,0} + a_{f,1}\ x + a_{f,2}\ y$ for all other f. Thus, $rp_{\mathcal{P}ol}(f,i) = a_{f,i}$ for all symbols f and positions i.*

We compute $Con_{\mathcal{R}}$ for the right-hand side of $\mathsf{REV2}(x, \mathsf{cons}(y,z)) \to \mathsf{REV}(\mathsf{rev2}(y,z))$. Let $\mathcal{R}' = \mathcal{R} \setminus Rls_{\mathcal{R}}(\mathsf{rev2})$ and $\mathcal{R}'' = \mathcal{R}' \setminus Rls_{\mathcal{R}'}(\mathsf{rev})$.

$$
\begin{aligned}
Con_{\mathcal{R}}(\mathsf{REV}(\mathsf{rev2}(y,z)), \mathcal{P}ol) &= \{\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0 \mid l \to r \in Rls_{\mathcal{R}}(\mathsf{REV}) = \varnothing\} \cup \\
&\quad \textstyle\bigcup_{l \to r \in Rls_{\mathcal{R}}(\mathsf{REV}) = \varnothing} Con_{\mathcal{R}}(r, \mathcal{P}ol) \cup \\
&\quad \{a_{\mathsf{REV},1} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\mathcal{R}}(\mathsf{rev2}(y,z), \mathcal{P}ol)\} \\
&= \{a_{\mathsf{REV},1} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\mathcal{R}}(\mathsf{rev2}(y,z), \mathcal{P}ol)\}
\end{aligned}
$$

$$
\begin{aligned}
Con_{\mathcal{R}}(\mathsf{rev2}(y,z), \mathcal{P}ol) &= \{\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0 \mid l \to r \in Rls_{\mathcal{R}}(\mathsf{rev2})\} \cup \\
&\quad Con_{\mathcal{R}'}(\mathsf{nil}, \mathcal{P}ol) \cup Con_{\mathcal{R}'}(\mathsf{rev}(\mathsf{cons}(x,\ldots)), \mathcal{P}ol) \cup \\
&\quad \{a_{\mathsf{rev2},1} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\mathcal{R}'}(y, \mathcal{P}ol)\} \cup \\
&\quad \{a_{\mathsf{rev2},2} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\mathcal{R}'}(z, \mathcal{P}ol)\} \\
&= \{\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0 \mid l \to r \in Rls_{\mathcal{R}}(\mathsf{rev2})\} \cup \\
&\quad Con_{\mathcal{R}'}(\mathsf{rev}(\mathsf{cons}(x,\ldots)), \mathcal{P}ol)
\end{aligned}
$$

$$
\begin{aligned}
Con_{\mathcal{R}'}(\mathsf{rev}(\mathsf{cons}(x,\ldots)), \mathcal{P}ol) &= \{\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0 \mid l \to r \in Rls_{\mathcal{R}'}(\mathsf{rev})\} \cup \\
&\quad Con_{\mathcal{R}''}(\mathsf{nil}, \mathcal{P}ol) \cup Con_{\mathcal{R}''}(\mathsf{cons}(\mathsf{rev1}(\ldots),\ldots), \mathcal{P}ol) \cup \\
&\quad \{a_{\mathsf{rev},1} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\mathcal{R}''}(\mathsf{cons}(x,\ldots), \mathcal{P}ol)\} \\
&= \{\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0 \mid l \to r \in Rls_{\mathcal{R}'}(\mathsf{rev})\} \cup \\
&\quad Con_{\mathcal{R}''}(\mathsf{cons}(\mathsf{rev1}(\ldots),\ldots), \mathcal{P}ol)
\end{aligned}
$$

$$
\begin{aligned}
Con_{\mathcal{R}''}(\mathsf{cons}(\mathsf{rev1}(..),..), \mathcal{P}ol) &= \{\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0 \mid l \to r \in Rls_{\mathcal{R}''}(\mathsf{cons}) = \varnothing\} \cup \\
&\quad \textstyle\bigcup_{l \to r \in Rls_{\mathcal{R}''}(\mathsf{cons}) = \varnothing} Con_{\mathcal{R}''}(r, \mathcal{P}ol) \cup \\
&\quad \{a_{\mathsf{cons},1} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\mathcal{R}''}(\mathsf{rev1}(x,l), \mathcal{P}ol)\} \cup \\
&\quad \{a_{\mathsf{cons},2} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\mathcal{R}''}(\mathsf{rev2}(x,l), \mathcal{P}ol)\} \\
&= \{a_{\mathsf{cons},1} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\mathcal{R}''}(\mathsf{rev1}(x,l), \mathcal{P}ol)\}
\end{aligned}
$$

$$
\begin{aligned}
Con_{\mathcal{R}''}(\mathsf{rev1}(x,l), \mathcal{P}ol) &= \{\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0 \mid l \to r \in Rls_{\mathcal{R}''}(\mathsf{rev1})\} \cup \\
&\quad \textstyle\bigcup_{l \to r \in Rls_{\mathcal{R}''}(\mathsf{rev1})} Con_{\varnothing}(r, \mathcal{P}ol) \cup \\
&\quad \{a_{\mathsf{rev1},1} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\varnothing}(x, \mathcal{P}ol)\} \cup \\
&\quad \{a_{\mathsf{rev1},2} \cdot p \geq 0 \mid \text{“}p \geq 0\text{”} \in Con_{\varnothing}(l, \mathcal{P}ol)\} \\
&= \{\mathcal{P}ol(l) - \mathcal{P}ol(r) \geq 0 \mid l \to r \in Rls_{\mathcal{R}''}(\mathsf{rev1})\}
\end{aligned}
$$

So $Con_{\mathcal{R}}(\mathsf{REV}(\mathsf{rev2}(y,z)), \mathcal{P}ol)$ contains $a_{\mathsf{REV},1} \cdot (\mathcal{P}ol(l) - \mathcal{P}ol(r)) \geq 0$ for rev2*- and* rev*-rules and $a_{\mathsf{REV},1} \cdot a_{\mathsf{cons},1} \cdot (\mathcal{P}ol(l) - \mathcal{P}ol(r)) \geq 0$ for* rev1*-rules.*

This indicates that if cons *is mapped to a polynomial which disregards its first argument (i.e., if $a_{\mathsf{cons},1} = 0$), then one does not have to require that the* rev1*-rules are weakly decreasing. As shown in Ex. 27, this observation is crucial for the success of the innermost termination proof. For example, all constraints (for all SCCs) are satisfied by the assignment α which maps $a_{\mathsf{cons},0}$, $a_{\mathsf{cons},2}$, $a_{\mathsf{REV},1}$, $a_{\mathsf{rev},1}$, $a_{\mathsf{REV1},2}$, $a_{\mathsf{REV2},2}$, and $a_{\mathsf{rev2},2}$ to 1 and all other variable coefficients to 0. So α turns $\mathcal{P}ol$ into a*

concrete polynomial interpretation where nil *and* $\mathsf{rev1}(x,y)$ *are mapped to 0,* $\mathsf{cons}(x,y)$ *is mapped to* $1+y$*,* $\mathsf{REV}(x)$ *and* $\mathsf{rev}(x)$ *are mapped to* x*, and* $\mathsf{REV1}(x,y)$*,* $\mathsf{REV2}(x,y)$*, and* $\mathsf{rev2}(x,y)$ *are mapped to* y*.*

The following lemma shows that $Con_{\mathcal{R}}$ indeed corresponds to the constraints resulting from the usable rules.

LEMMA 46 (*Con* and $\mathcal{U}$). *Let* $\mathcal{P}ol$ *be an abstract polynomial interpretation and* t *be a term. An assignment* α *for* $\mathcal{P}ol$*'s coefficients satisfies* $Con_{\mathcal{R}}(t,\mathcal{P}ol)$ *iff* α *satisfies* $\mathcal{P}ol(l)-\mathcal{P}ol(r)\geq 0$ *for all* $l\to r\in\mathcal{U}_{\mathcal{R}}(t,\pi_{\alpha(\mathcal{P}ol)})$*.*

Proof. We use induction over the sizes of $\mathcal{R}$ and t. If $t\in\mathcal{V}$, then the claim is trivial. Otherwise, let $t=f(t_1,\ldots,t_n)$. The assignment α satisfies $Con_{\mathcal{R}}(f(t_1,\ldots,t_n),\mathcal{P}ol)$ iff it satisfies $\mathcal{P}ol(l)-\mathcal{P}ol(r)\geq 0$ and $Con_{\mathcal{R}'}(r,\mathcal{P}ol)$ for all $l\to r\in Rls_{\mathcal{R}}(f)$, and if it also satisfies $rp_{\mathcal{P}ol}(f,i)\cdot p \geq 0$ for all constraints $p\geq 0$ from $Con_{\mathcal{R}'}(t_i,\mathcal{P}ol)$ where $i\in\{1,\ldots,n\}$.

We have $\alpha(rp_{\mathcal{P}ol}(f,i))>0$ if $i\in rp_{\pi_{\alpha(\mathcal{P}ol)}}(f)$ and $\alpha(rp_{\mathcal{P}ol}(f,i))=0$, otherwise. So α satisfies the last condition iff it satisfies $Con_{\mathcal{R}'}(t_i,\mathcal{P}ol)$ for $i\in rp_{\pi_{\alpha(\mathcal{P}ol)}}(f)$. Now the claim follows by the induction hypothesis. □

Now the reduction pair processor from Thm. 26 can be reformulated to permit the use of reduction pairs based on polynomial orders.

THEOREM 47 (Reduction Pair Processor with Polynomials). *Let* $\mathcal{P}ol$ *be an abstract polynomial interpretation and let* α *be an assignment. Then the following DP processor Proc is sound and complete. We define* $Proc(\,(\mathcal{P},\mathcal{R},e)\,) =$

- $\{(\mathcal{P}\backslash\mathcal{P}',\mathcal{R},e)\}$*, if* α *satisfies the constraints (37) and* $Con_{\mathcal{R}}(\mathcal{P},\mathcal{P}ol)$
- $\{(\mathcal{P},\mathcal{R},e)\}$*, otherwise*

Proof. The theorem follows from Thm. 26: for the reduction pair "$(\succsim_\pi,\succ_\pi)$" in Thm. 26 we choose the polynomial order $\alpha(\mathcal{P}ol)$ and for computing the usable rules "$\mathcal{U}_{\mathcal{R}}(\mathcal{P},\pi)$" in Thm. 26 we use the filtering $\pi_{\alpha(\mathcal{P}ol)}$. Then (37) clearly corresponds to the constraints (a) in Thm. 26 and $Con_{\mathcal{R}}(\mathcal{P},\mathcal{P}ol)$ corresponds to the constraints (b) by Lemma 46. □

7.2. Finding Strict Constraints Automatically

For termination proofs with dependency pairs and polynomial orders, we have to solve constraints like (37) which have the form

$$p_j\geq 0 \text{ for all } 1\leq j\leq k \qquad \text{and} \qquad p_j>0 \text{ for one } 1\leq j\leq k \quad (39)$$

for polynomials p_j. The reason is that for a DP problem $(\mathcal{P}, \mathcal{R}, e)$, all pairs in $\mathcal{P}$ must be weakly decreasing but at least one has to be strictly decreasing. The basic approach is to iterate over all k possible choices for the strict constraint. So in the worst case, the satisfiability checker for polynomial inequalities is called k times in order to find an assignment α satisfying (39). We present an equivalent, but more efficient method where the satisfiability checker is only called once. The solution is to transform (39) into the following constraint.

$$p_j \geq 0 \text{ for all } 1 \leq j \leq k \qquad \text{and} \qquad \textstyle\sum_{1 \leq j \leq k} p_j > 0 \tag{40}$$

THEOREM 48 ((39) iff (40)). *Let the p_j be variable disjoint, except for variable coefficients. An assignment α satisfies (39) iff it satisfies (40).*

Proof. If α satisfies (39), then w.l.o.g. we have $\alpha(p_1) > 0$. Using $\alpha(p_j) \geq 0$ for all $j \in \{2, \ldots, k\}$ we obtain $\alpha(\sum_{1 \leq j \leq k} p_j) > 0$.

For the other direction, let α satisfy (40) and assume that $\alpha(p_j) \not> 0$ for all $j \in \{1, \ldots, k\}$. Hence, for all j there exists a variable assignment β_j of the variables $x, y, \ldots$ in $\alpha(p_j)$ such that $\beta_j(\alpha(p_j)) = 0$. Since the polynomials $\alpha(p_j)$ are pairwise variable disjoint, the assignments β_j can be combined to one assignment β which coincides with each β_j on β_j's domain. Thus, $\beta(\alpha(p_j)) = 0$ for all j and therefore $\beta(\alpha(\sum_{1 \leq j \leq k} p_j)) = 0$. But then α cannot satisfy $\sum_{1 \leq j \leq k} p_j > 0$ which gives a contradiction. □

There exist several methods to find variable assignments α satisfying polynomial constraints $p \geq 0$ or $p > 0$. Most of them search for an assignment α where p is *absolutely positive* [26]. A polynomial p is absolutely positive if, in addition to $p \geq 0$ or $p > 0$, all successive partial derivatives of p are non-negative (i.e., $\frac{\partial p(x_1,\ldots,x_n)}{\partial x_i} \geq 0$ for all i, $\frac{\partial^2 p(x_1,\ldots,x_n)}{\partial x_i \partial x_j} \geq 0$ for all i and j, etc.). Examples for such approaches to determine polynomial interpretations are the *method of partial derivation* [10, 31] and the *shifting method* [26].

If in addition to $p \geq 0$ or $p > 0$, the satisfiability checker for polynomial constraints ensures that at least the first partial derivatives are non-negative (i.e., $\frac{\partial p(x_1,\ldots,x_n)}{\partial x_i} \geq 0$ for all i), then (40) can be simplified further to

$$p_j \geq 0 \text{ for all } 1 \leq j \leq k \qquad \text{and} \qquad \textstyle\sum_{1 \leq j \leq k} p_j(0, \ldots, 0) > 0 \tag{41}$$

The reason is that then the constraints $p_j \geq 0$ ensure that all first partial derivatives of p_j must be at least 0. But then, the first partial derivatives of $\sum_{1 \leq j \leq k} p_j$ are also at least 0. Thus, it suffices to require $\sum_{1 \leq j \leq k} p_j > 0$ only for the instantiation of all variables $x, y, \ldots$ by 0.

EXAMPLE 49. *For the TRS of Ex. 19 we obtain constraints like (39):*

$$\mathcal{P}ol(\mathsf{DIV}(x_1, y_1)) - \mathcal{P}ol(\mathsf{QUOT}(x_1, y_1, y_1)) \ {}_{(}\geq_{)}\ 0 \quad (42)$$

$$\mathcal{P}ol(\mathsf{QUOT}(\mathsf{s}(x_2), \mathsf{s}(y_2), z_2)) - \mathcal{P}ol(\mathsf{QUOT}(x_2, y_2, z_2)) \ {}_{(}\geq_{)}\ 0 \quad (43)$$

$$\mathcal{P}ol(\mathsf{QUOT}(x_3, \mathsf{0}, \mathsf{s}(z_3))) - \mathcal{P}ol(\mathsf{DIV}(x_3, \mathsf{s}(z_3))) \ {}_{(}\geq_{)}\ 0 \quad (44)$$

Instead of choosing one of the above constraints to be strict, with our refinement in (40) one obtains the constraints (42) – (44) with weak inequalities (i.e., with "≥") and the additional constraint

$$\begin{array}{l} \mathcal{P}ol(\mathsf{DIV}(x_1, y_1)) - \mathcal{P}ol(\mathsf{QUOT}(x_1, y_1, y_1)) \\ + \mathcal{P}ol(\mathsf{QUOT}(\mathsf{s}(x_2), \mathsf{s}(y_2), z_2)) - \mathcal{P}ol(\mathsf{QUOT}(x_2, y_2, z_2)) \\ + \mathcal{P}ol(\mathsf{QUOT}(x_3, \mathsf{0}, \mathsf{s}(z_3))) - \mathcal{P}ol(\mathsf{DIV}(x_3, \mathsf{s}(z_3))) \end{array} \quad > 0 \quad (45)$$

If the satisfiability checker guarantees absolute positiveness, then the above constraint may be simplified by instantiating all x_i, y_i, and z_i with 0. If we use a linear abstract polynomial interpretation $\mathcal{P}ol$ with $\mathcal{P}ol(\mathsf{DIV}(x, y)) = a_{\mathsf{DIV},0} + a_{\mathsf{DIV},1}x + a_{\mathsf{DIV},2}y$, $\mathcal{P}ol(\mathsf{QUOT}(x, y, z)) = a_{\mathsf{QUOT},0} + a_{\mathsf{QUOT},1}x + a_{\mathsf{QUOT},2}y + a_{\mathsf{QUOT},3}z$, $\mathcal{P}ol(\mathsf{s}(x)) = a_{\mathsf{s},0} + a_{\mathsf{s},1}\ x$, and $\mathcal{P}ol(\mathsf{0}) = a_{\mathsf{0}}$, then instead of (45) we obtain

$$(a_{\mathsf{QUOT},1} + a_{\mathsf{QUOT},2} + a_{\mathsf{QUOT},3} - a_{\mathsf{DIV},2})\, a_{\mathsf{s},0} + a_{\mathsf{QUOT},2}\, a_{\mathsf{0}} > 0. \quad (46)$$

The resulting constraints are satisfied by the assignment which maps $a_{\mathsf{DIV},1}$, $a_{\mathsf{QUOT},1}$, $a_{\mathsf{s},0}$, and $a_{\mathsf{s},1}$ to 1 and all other variable coefficients to 0. In this way, the QUOT*-dependency pair corresponding to Constraint (43) is strictly decreasing and can be removed. The remaining proof is similar to the one in Ex. 19.*[10]

8. Conclusion and Empirical Results

We improved the dependency pair technique by significantly reducing the sets of constraints to be solved for (innermost) termination proofs. To combine these improvements with dependency pair transformations, we extended the transformations and developed a criterion to ensure that their application is terminating without compromising their power in practice. Afterwards, we introduced new techniques to mechanize the approach both with polynomial orders and with monotonic orders combined with argument filterings. These implementation techniques are tailored to the improvements of dependency pairs presented before.

[10] In order to remove the dependency pair $\mathsf{QUOT}(x, \mathsf{0}, \mathsf{s}(z)) \to \mathsf{DIV}(x, \mathsf{s}(z))$, one uses $\mathcal{P}ol(\mathsf{QUOT}(x, y, z)) = \mathcal{P}ol(\mathsf{DIV}(x, y)) = y$, $\mathcal{P}ol(\mathsf{s}) = 0$, and $\mathcal{P}ol(\mathsf{0}) = 1$. Afterwards, the estimated dependency graph has no cycle anymore.

Preliminary versions of parts of this paper appeared in [13, 39]. The present article extends [13, 39] substantially, e.g., by detailed proofs, by extending all results to the new DP framework of [15], by the new forward instantiation transformation, by a new section on automating dependency pairs with polynomial orders, by a detailed description of our experiments, and by several additional explanations and examples.

We implemented the results of the paper in the system AProVE [17], available at `http://aprove.informatik.rwth-aachen.de/`. Due to the results of this paper, AProVE was the most powerful tool for (innermost) termination proofs of TRSs at the *International Annual Competition of Termination Tools* in 2004, 2005, and 2006 [42].

In our experiments, we tested AProVE on the examples of the *Termination Problem Data Base*. This is the collection of problems used in the annual termination competition [42]. It contains 773 TRSs from different sources as a benchmark for termination analysis of term rewriting. Our experiments are presented in two tables. In the first table, we tried to prove (full) termination[11] and in the second, we tried to prove innermost termination of all examples.

In our experiments, we used AProVE with the following techniques:

- *Thm. 12* applies the dependency graph processor with the estimation of Def. 9 and the reduction pair processor of Thm. 12. This is the basic dependency pair technique without new improvements.
- *Thm. 17* uses the dependency graph and the reduction pair processor of Thm. 17. Thus, usable rules are also applied for termination proofs. For innermost termination, this is the same as *Thm. 12*.
- *Thm. 26* uses the dependency graph and the reduction pair processor of Thm. 26. Thus, now one applies the usable rules w.r.t. argument filterings for both termination and innermost termination.

Moreover, we use five different kinds of reduction pairs for the reduction pair processor:

- *EMB* is the embedding order.
- *LPO* is the lexicographic path order where we allow different symbols to be equal in the precedence.

[11] As mentioned, there are classes of TRSs where innermost termination implies full termination. Thus, here one should only prove innermost termination. However, this observation was not used in the first table in order to obtain a clearer evaluation of our contributions for full termination. Similarly, we also did not use additional recent refinements of the dependency pair technique in our experiments in order to assess only the impact of contributions which come from the present paper.

– *POLO-filter* searches for linear polynomial interpretations with coefficients from $\{0, 1\}$. Here, we do not yet use the results of Sect. 7.1 which combine the search for a polynomial order with the search for an argument filtering. Instead, we first determine an argument filtering and search for a monotonic polynomial order afterwards. Moreover, we also do not use the improvement of Sect. 7.2. Instead, in the reduction pair processor, we try all possibilities in order to make one dependency pair in the DP problem strictly decreasing.

– *POLO-7.1* differs from *POLO-filter* by using the results of Sect. 7.1. Now we do not search for an argument filtering anymore but we look for a (not necessarily monotonic) polynomial order. So in particular, when applying usable rules w.r.t. an argument filtering in Thm. 26, we proceed as in Thm. 47.

– *POLO-7.1+7.2* is like *POLO-7.1*, but it also uses the refinements of Sect. 7.2 which avoid the search for a strictly decreasing dependency pair when applying the reduction pair processor.

For the first three reduction pairs, the reduction pair processor has to search for argument filterings. Here, we use the method of Sect. 6.

Finally, in each of the above settings, we experiment with different variants for the application of the transformation processors of Def. 28:

– *"no"* means that we do not use any transformations at all.

– *"older"* is the heuristic of Def. 33 for safe transformations. The combination with the reduction pair processor is done as described at the end of Sect. 5. However, we do not use the new forward instantiation transformation and instead of Def. 28, we use the previous applicability conditions for the transformations from [1, 11].

– *"old"* is like *"older"*, but with the new more liberal applicability conditions from Def. 28 instead.

– *"safe"* is the heuristic of Def. 33 with the transformations from Def. 28. So in contrast to *"old"*, now we also use forward instantiation.

– *"(1)+(2)"* differs from *"safe"* by only regarding those transformation steps as "safe" which satisfy condition (1) or (2) of Def. 33.

– *"lim"* uses the transformations of Def. 28 with a different heuristic than Def. 33. Now at most five transformations are allowed for each dependency pair. To combine the transformations with the reduction pair processor, the strategy at the end of Sect. 5 is

modified as follows: At most two transformation steps are allowed before applying the reduction pair processor, while the remaining transformation steps (up to five in total) are performed afterwards.

For each example we used a time limit of 60 seconds. This corresponds to the way that tools were evaluated in the annual competitions for termination tools. The computer used was an AMD Athlon 64 at 2.2 GHz running Sun's J2SE 1.5.0 under GNU/Linux 2.6.10 which is similar in speed to the computer used in the 2005 competition. In the tables, we give the number of examples where termination could be proved ("Y"), where AProVE failed within the time limit of 60 seconds ("F"), and where it failed due to a time-out ("TO"). In square brackets, we give the average runtime (in seconds) needed for TRSs where AProVE could prove termination and where it failed within the time limit. The detailed results of our experiments (including experimental data for all possible combinations of our settings) can be found at `http://aprove.informatik.rwth-aachen.de/eval/JAR06/`. At this URL one can also download a special version of AProVE with all settings described above. So this version of AProVE permits to re-run all our experiments.

Termination Proofs

Line	Algorithm	Order	Tr.	Y	F	TO
1	*Thm. 12*	*EMB*	*no*	89 [0.8]	680 [1.6]	4
2	*Thm. 12*	*LPO*	*no*	245 [3.0]	399 [3.6]	129
3	*Thm. 12*	*POLO-filter*	*no*	234 [1.7]	426 [5.3]	113
4	*Thm. 12*	*POLO-7.1*	*no*	251 [0.9]	512 [1.6]	10
5	*Thm. 12*	*POLO-7.1+7.2*	*no*	251 [0.9]	520 [1.5]	2
6	*Thm. 17*	*EMB*	*no*	174 [0.7]	588 [1.4]	11
7	*Thm. 17*	*LPO*	*no*	277 [1.5]	387 [2.4]	109
8	*Thm. 17*	*POLO-filter*	*no*	331 [1.8]	361 [2.0]	81
9	*Thm. 17*	*POLO-7.1*	*no*	341 [1.0]	425 [1.4]	7
10	*Thm. 17*	*POLO-7.1+7.2*	*no*	341 [1.0]	432 [1.4]	0
11	*Thm. 26*	*EMB*	*no*	233 [0.8]	529 [1.6]	11
12	*Thm. 26*	*LPO*	*no*	292 [1.6]	374 [2.1]	107
13	*Thm. 26*	*POLO-filter*	*no*	361 [1.8]	334 [1.8]	78
14	*Thm. 26*	*POLO-7.1*	*no*	368 [1.1]	388 [1.8]	17
15	*Thm. 26*	*POLO-7.1+7.2*	*no*	369 [1.1]	399 [1.7]	5
16/17	*Thm. 26*	*POLO-7.1+7.2*	*older/old*	390 [1.2]	349 [2.4]	34
18	*Thm. 26*	*POLO-7.1+7.2*	*(1)+(2)*	374 [1.2]	394 [1.8]	5
19	*Thm. 26*	*POLO-7.1+7.2*	*lim*	396 [2.1]	268 [1.9]	109
20	*Thm. 26*	*POLO-7.1+7.2*	*safe*	406 [1.1]	330 [2.5]	37

Innermost Termination Proofs

Line	Algorithm	Order	Tr.	Y	F	TO
1/6	*Thm. 12/17*	*EMB*	*no*	246 [0.8]	516 [1.8]	11
2/7	*Thm. 12/17*	*LPO*	*no*	330 [2.1]	350 [3.2]	93
3/8	*Thm. 12/17*	*POLO-filter*	*no*	378 [2.1]	324 [2.7]	71
4/9	*Thm. 12/17*	*POLO-7.1*	*no*	388 [1.0]	379 [1.6]	6
5/10	*Thm. 12/17*	*POLO-7.1+7.2*	*no*	388 [1.2]	384 [1.7]	1
11	*Thm. 26*	*EMB*	*no*	289 [0.9]	473 [1.6]	11
12	*Thm. 26*	*LPO*	*no*	341 [1.6]	341 [2.4]	91
13	*Thm. 26*	*POLO-filter*	*no*	402 [1.7]	305 [1.9]	66
14	*Thm. 26*	*POLO-7.1*	*no*	408 [1.2]	350 [1.9]	15
15	*Thm. 26*	*POLO-7.1+7.2*	*no*	408 [1.1]	360 [2.0]	5
16	*Thm. 26*	*POLO-7.1+7.2*	*older*	465 [1.3]	253 [5.6]	55
17	*Thm. 26*	*POLO-7.1+7.2*	*old*	467 [1.3]	250 [5.5]	56
18	*Thm. 26*	*POLO-7.1+7.2*	*(1)+(2)*	441 [1.6]	314 [3.5]	18
19	*Thm. 26*	*POLO-7.1+7.2*	*lim*	386 [3.7]	144 [2.4]	243
20	*Thm. 26*	*POLO-7.1+7.2*	*safe*	480 [1.7]	217 [6.9]	76

Comparing the results for *Thm. 12* (lines 1 – 5), *Thm. 17* (lines 6 – 10), and *Thm. 26* (lines 11 – 15) shows the benefits of our contributions from Sect. 3 and 4. Irrespective of the underlying reduction pair, *Thm. 17* is always more powerful than *Thm. 12* and increases the number of examples where termination can be proved by up to 95.5%. *Thm. 26* improves upon *Thm. 17* further and increases power by up to 33.9% (up to 17.5% for innermost termination).

To measure the impact of our contributions in Sect. 7, we compare the naive use of monotonic polynomial orders (*POLO-filter*, line 13) with the more sophisticated approaches of Sect. 7.1 and 7.2 (lines 14 – 15). As all three methods are equally powerful in principle, the difference is in efficiency. Indeed, the step from *POLO-filter* to *POLO-7.1* and further to *POLO-7.1+7.2* reduces the overall runtime dramatically. The reason is that now a failure can often be detected quickly for examples which led to a time-out before. Thus, while the number of examples where termination can be proved within the time limit only increases slightly, the number of time-outs reduces substantially (by at least 77.2 % when going from *POLO-filter* to *POLO-7.1* and by at least 66.7 % when going from *POLO-7.1* to *POLO-7.1+7.2*). This fast-failure behavior of *POLO-7.1+7.2* permits the use of further techniques in order to attempt a termination proof within the time limit.

Examples for such additional techniques are the dependency pair transformations from Sect. 5. If one uses the existing "*older*" transformations with our new heuristic for their application (line 16), then power is increased by 5.7% (14.0% for innermost termination). While the use of dependency pair transformations increases power, it can of course also increase runtimes. This holds especially for innermost termination proofs, since here the transformations are much more often applicable.

Def. 28 extends the dependency pair transformations in two ways: we presented more liberal applicability conditions for the transformations in the innermost case and we introduced a new forward instantiation transformation. While the new applicability conditions only lead to a minor improvement of 0.4% in power (cf. the "*old*" heuristic, line 17), the forward instantiation technique increases power again by 4.1% (2.8% for innermost termination), cf. the "*safe*" heuristic in line 20.

Finally, we also evaluate our heuristic from Def. 33 which describes when to apply dependency pair transformations. We consider two possible alternatives. The first alternative is a restriction to those transformations which make the DP problem "smaller", i.e., which correspond to Def. 33 (1) or (2), cf. line 18. Our experiments show that allowing one additional narrowing, instantiation, and forward instantiation step (as in the "*safe*" heuristic, line 20) has considerable advantages since it increases power by 8.6% (8.8% for innermost termination). As a second alternative, we tried a "*lim*"-heuristic, which simply applies transformations up to a certain limit. The experiments demonstrate that our "*safe*"-heuristic is significantly better: it increases power (in particular for innermost termination, where the success rate is improved by 24.4%) and it reduces runtimes dramatically since the "*lim*"-heuristic leads to extremely many time-outs.

In the end, we also ran the experiment with the automatic mode of the newest AProVE-version (AProVE 1.2) which features several additional techniques (e.g., *semantic labelling* [44] and *match-bounds* [9]) in addition to the results of this paper. Now AProVE could prove termination of 576 examples while it disproved termination for 94 TRSs. Innermost termination could be shown for 617 examples while it could be disproved for 61 TRSs. This corresponds exactly to the results of AProVE in the termination competition 2005 [42]. The average runtime for a successful proof was 2.3 s (2.6 s for innermost termination) and the average time for a successful disproof was 2.7 s (2.9 s for innermost termination). Finally, AProVE failed on 11 examples within the time limit (14 for innermost termination) and the average runtime for these failures was 26.7 s for termination and 8.7 s for innermost termination.

To summarize, our experiments indicate that the contributions of the paper are indeed very useful in practice. This also holds if our results are combined with other termination techniques.

While our experiments show that termination analysis of TRSs has reached a stage where already many realistic examples can be treated automatically, future work should be devoted to the question on how these methods can be used in order to analyze termination for programs from "real" programming languages. These languages pose several additional challenges such as evaluation strategies, built-in data types,

partial functions, non-termination analysis, higher-order functions, etc. We report our first results in this direction in [16, 18, 38].

Acknowledgements. We thank the referees for many helpful remarks.

References

1. Arts, T. and J. Giesl: 2000, 'Termination of Term Rewriting Using Dependency Pairs'. *Theoretical Computer Science* **236**, 133–178.
2. Arts, T. and J. Giesl: 2001, 'A Collection of Examples for Termination of Term Rewriting Using Dependency Pairs'. Technical Report AIB-2001-09, RWTH Aachen, Germany. Available from `http://aib.informatik.rwth-aachen.de`.
3. Baader, F. and T. Nipkow: 1998, *Term Rewriting and All That*. Cambridge University Press.
4. Borralleras, C.: 2003, 'Ordering-based methods for proving termination automatically'. Ph.D. thesis, Universitat Politècnica de Catalunya, Spain.
5. Borralleras, C., M. Ferreira, and A. Rubio: 2000, 'Complete Monotonic Semantic Path Orderings'. In: *Proc. 17th CADE*. pp. 346–364. LNAI 1831.
6. Contejean, E., C. Marché, B. Monate, and X. Urbain: 2000, 'CiME version 2'. Available from `http://cime.lri.fr`.
7. Dershowitz, N.: 1987, 'Termination of Rewriting'. *Journal of Symbolic Computation* **3**, 69–116.
8. Dershowitz, N., N. Lindenstrauss, Y. Sagiv, and A. Serebrenik: 2001, 'A General Framework for Automatic Termination Analysis of Logic Programs'. *Appl. Algebra in Engineering, Communication and Computing* **12**(1,2), 117–156.
9. Geser, A., D. Hofbauer, and J. Waldmann: 2004, 'Match-Bounded String Rewriting Systems'. *Appl. Algebra in Eng., Comm. & Comp.* **15**(3,4), 149–171.
10. Giesl, J.: 1995, 'Generating Polynomial Orderings for Termination Proofs'. In: *Proc. 6th RTA*. pp. 426–431. LNCS 914.
11. Giesl, J. and T. Arts: 2001, 'Verification of Erlang Processes by Dependency Pairs'. *Appl. Algebra in Engineering, Communication & Comp.* **12**(1,2), 39–72.
12. Giesl, J., T. Arts, and E. Ohlebusch: 2002, 'Modular Termination Proofs for Rewriting Using Dependency Pairs'. *J. Symbolic Computation* **34**(1), 21–58.
13. Giesl, J., R. Thiemann, P. Schneider-Kamp, and S. Falke: 2003a, 'Improving Dependency Pairs'. In: *Proc. 10th LPAR*. pp. 165–179. LNAI 2850.
14. Giesl, J., R. Thiemann, P. Schneider-Kamp, and S. Falke: 2003b, 'Mechanizing Dependency Pairs'. Technical Report AIB-2003-08, RWTH Aachen, Germany. Available from `http://aib.informatik.rwth-aachen.de`.
15. Giesl, J., R. Thiemann, and P. Schneider-Kamp: 2005a, 'The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs'. In: *Proc. 11th LPAR*. pp. 301–331. LNAI 3452.
16. Giesl, J., R. Thiemann, and P. Schneider-Kamp: 2005b, 'Proving and Disproving Termination of Higher-Order Functions'. In: *Proc. 5th FroCoS*. pp. 216–231. LNAI 3717.
17. Giesl, J., P. Schneider-Kamp, and R. Thiemann: 2006a, 'AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework'. In: *Proc. 3rd IJCAR*. pp. 281–286. LNAI 4130.

18. Giesl, J., S. Swiderski, P. Schneider-Kamp, and R. Thiemann: 2006b, 'Automated Termination Analysis for Haskell: From Term Rewriting to Programming Languages'. In: *Proc. 17th RTA*. pp. 297–312. LNCS 4098.
19. Gramlich, B.: 1994, 'Generalized Sufficient Conditions for Modular Termination of Rewriting'. *Appl. Algebra in Eng., Comm. & Comp.* **5**, 131–158.
20. Gramlich, B.: 1995, 'Abstract Relations between Restricted Termination and Confluence Properties of Rewrite Systems'. *Fundamenta Informaticae* **24**, 3–23.
21. Gramlich, B.: 1996, 'Termination and Confluence Properties of Structured Rewrite Systems'. Ph.D. thesis, Universität Kaiserslautern, Germany.
22. Hirokawa, N. and A. Middeldorp: 2004a, 'Dependency Pairs Revisited'. In: *Proc. 15th RTA*. pp. 249–268. LNCS 3091.
23. Hirokawa, N. and A. Middeldorp: 2004b, 'Polynomial Interpretations with Negative Coefficients'. In: *Proc. AISC '04*. pp. 185–198. LNAI 3249.
24. Hirokawa, N. and A. Middeldorp: 2005a, 'Tyrolean Termination Tool'. In: *Proc. 16th RTA*. pp. 175–184. LNCS 3467.
25. Hirokawa, N. and A. Middeldorp: 2005b, 'Automating the Dependency Pair Method'. *Information and Computation* **199**(1,2), 172–199.
26. Hong, H. and D. Jakuš: 1998, 'Testing Positiveness of Polynomials'. *Journal of Automated Reasoning* **21**(1), 23–38.
27. Huet, G. and J.-M. Hullot: 1982, 'Proofs by Induction in Equational Theories with Constructors'. *Journal of Computer and System Sciences* **25**, 239–299.
28. Knuth, D. and P. Bendix: 1970, 'Simple Word Problems in Universal Algebras'. In: J. Leech (ed.): *Computational Problems in Abstract Algebra*. pp. 263–297.
29. Koprowski, A.: 2005, 'TPA'. Available from `http://www.win.tue.nl/tpa/`.
30. Kusakari, K., M. Nakamura, and Y. Toyama: 1999, 'Argument Filtering Transformation'. In: *Proc. 1st PPDP*. pp. 48–62. LNCS 1702.
31. Lankford, D.: 1979, 'On Proving Term Rewriting Systems are Noetherian'. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA.
32. Lee, C. S., N. D. Jones, and A. M. Ben-Amram: 2001, 'The Size-Change Principle for Program Termination'. In: *Proc. 28th POPL*. pp. 81–92.
33. Lucas, S.: 2005, 'Polynomials Over the Reals in Proofs of Termination: From Theory to Practice'. *RAIRO Theor. Informatics and Appl.* **39**(3), 547–586.
34. Middeldorp, A.: 2001, 'Approximating Dependency Graphs Using Tree Automata Techniques'. In: *Proc. 1st IJCAR*. pp. 593–610. LNAI 2083.
35. Middeldorp, A.: 2002, 'Approximations for Strategies and Termination'. In: *Proc. 2nd WRS*. ENTCS 70(6).
36. Ohlebusch, E., C. Claves, and C. Marché: 2000, 'TALP: A Tool for Termination Analysis of Logic Programs'. In: *Proc. 11th RTA*. pp. 270–273. LNCS 1833.
37. Ohlebusch, E.: 2002, *Advanced Topics in Term Rewriting*. Springer.
38. Schneider-Kamp, P., J. Giesl, A. Serebrenik, and R. Thiemann: 2006, 'Automated Termination Analysis for Logic Programs by Term Rewriting'. In: *Proc. 16th LOPSTR*. LNCS. To appear.
39. Thiemann, R., J. Giesl, and P. Schneider-Kamp: 2004, 'Improved Modular Termination Proofs Using Dependency Pairs'. In: *Proc. 2nd IJCAR*. pp. 75–90. LNAI 3097.
40. Thiemann, R. and J. Giesl: 2005, 'The Size-Change Principle and Dependency Pairs for Termination of Term Rewriting'. *Applicable Algebra in Engineering, Communication and Computing* **16**(4), 229–270.
41. Toyama, Y.: 1987, 'Counterexamples to the Termination for the Direct Sum of Term Rewriting Systems'. *Information Processing Letters* **25**, 141–143.
42. TPDB web page. `http://www.lri.fr/~marche/termination-competition/`.

43. Urbain, X.: 2004, 'Modular & Incremental Automated Termination Proofs'. *Journal of Automated Reasoning* **32**(4), 315–355.
44. Zantema, H.: 1995, 'Termination of Term Rewriting by Semantic Labelling'. *Fundamenta Informaticae* **24**, 89–105.
45. Zantema, H.: 2005, 'Termination of String Rewriting Proved Automatically'. *Journal of Automated Reasoning* **34**(2), 105–139.