The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs[⋆]

Jürgen Giesl, René Thiemann, Peter Schneider-Kamp

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

Abstract. The dependency pair approach is one of the most powerful techniques for automated termination proofs of term rewrite systems. Up to now, it was regarded as one of several possible methods to prove termination. In this paper, we show that dependency pairs can instead be used as a general concept to integrate arbitrary techniques for termination analysis. In this way, the benefits of different techniques can be combined and their modularity and power are increased significantly. We refer to this new concept as the "dependency pair *framework*" to distinguish it from the old "dependency pair *approach*". Moreover, this framework facilitates the development of new methods for termination analysis. To demonstrate this, we present several new techniques within the dependency pair framework which simplify termination problems considerably. We implemented the dependency pair framework in our termination prover AProVE and evaluated it on large collections of examples.

1 Introduction

Termination of term rewrite systems (TRSs) has been studied for decades and several methods were developed to prove termination of TRSs automatically (one of the most powerful techniques is the *dependency pair approach* [1, 6, 7]). Up to now, all these methods were seen as separate approaches on their own.

In contrast, this paper shows that dependency pairs are suitable as a general framework to integrate arbitrary techniques for termination proofs. In this way, the benefits of all available methods can be combined and the classical dependency pair *approach* is just a special case of this new dependency pair *framework*. By combining termination techniques within the dependency pair framework (instead of trying to apply them on a TRS directly, one after another), the flexibility, modularity, and power of these techniques are increased significantly.

We introduce the dependency pair framework in Sect. 2. Here, each technique for termination proofs is seen as a *dependency pair processor* that transforms a *dependency pair problem* into a set of new (and hopefully, simpler) ones. Sect. 3 shows how to formulate the components of the classical dependency pair approach as processors. This increases their applicability substantially and it demonstrates that the dependency pair approach is indeed a special case of the dependency pair framework. In Sect. 4 we introduce new processors which simplify dependency pair problems significantly and which are only possible due to the new formulation of the dependency pair framework. This demonstrates that

[⋆] *Proceedings of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2004)*, LNCS 3452, pp. 301-331, 2005.

the dependency pair framework is particularly well suitable as a basis for future developments in automated termination proving. In Sect. 5 we discuss how to integrate other (existing) methods for automated termination proofs into the dependency pair framework by formulating them as processors as well.

Finally, to construct an automatic termination prover using the dependency pair framework, a main problem is to develop strategies to decide which processors should be applied to a given DP problem. Suitable heuristics and our implementation in the termination prover AProVE are discussed in Sect. 6.

2 The Dependency Pair Framework

We extend the *dependency pair approach* to a framework for the combination of arbitrary termination techniques. The reader is referred to [2] for the basics of term rewriting and to [1, 7] for further details on the dependency pair approach.

We restrict ourselves to finite signatures $\mathcal{F}$ and TRSs. $\mathcal{T}(\mathcal{F},\mathcal{V})$ denotes the set of terms over $\mathcal{F}$ and the infinite set of variables $\mathcal{V}$. A *TRS over* $\mathcal{F}$ is a set of rules $l \to r$ where l and r are from $\mathcal{T}(\mathcal{F},\mathcal{V})$, $\mathcal{V}(r) \subseteq \mathcal{V}(l)$, and $l \notin \mathcal{V}$. To handle different evaluation strategies (like innermost or full rewriting) in a uniform way, we introduce the new notion of *$\mathcal{Q}$-restricted rewriting.* In $\mathcal{Q}$-restricted rewriting, one may only perform a rewrite step if the proper subterms of the redex are not reducible w.r.t. $\mathcal{Q}$ (i.e., if they are *$\mathcal{Q}$-normal forms*). This notion is particularly useful when defining techniques for innermost termination proofs later on.

Definition 1 ($\mathcal{Q}$-restricted Rewriting). *Let $\mathcal{R}$ and $\mathcal{Q}$ be TRSs. We define the* $\mathcal{Q}$-restricted rewrite relation *as $s \overset{\mathcal{Q}}{\to}_{\mathcal{R},p} t$ iff p is a position of s, $s|_p = l\sigma$ for some rule $l \to r \in \mathcal{R}$ and some substitution σ, $t = s[r\sigma]_p$, and all proper subterms of the redex $s|_p$ are in $\mathcal{Q}$-normal form. Moreover, $s \overset{\mathcal{Q}}{\to}_{\mathcal{R}} t$ means that $s \overset{\mathcal{Q}}{\to}_{\mathcal{R},p} t$ for some position p. A TRS $\mathcal{R}$ is* $\mathcal{Q}$-terminating *iff $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ is well founded.*[1]

Example 2. Consider $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}), \mathsf{a} \to \mathsf{b}\}$. If $\mathcal{Q}$ contains the rule $\mathsf{a} \to \mathsf{b}$, then the step from $\mathsf{f}(\mathsf{a})$ to $\mathsf{f}(\mathsf{a})$ is no longer possible with $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ since the proper subterm a of the redex $\mathsf{f}(\mathsf{a})$ is not a $\mathcal{Q}$-normal form. Thus, $\mathcal{R}$ is $\mathcal{Q}$-terminating.

$\mathcal{Q}$-restricted rewriting subsumes both innermost and ordinary rewriting. Ordinary rewriting is $\mathcal{Q}$-restricted rewriting for $\mathcal{Q} = \varnothing$ and innermost rewriting is $\mathcal{Q}$-restricted rewriting with $\mathcal{Q} = \mathcal{R}$ ($\to_{\mathcal{R}} = \overset{\varnothing}{\to}_{\mathcal{R}}$ and $\overset{\mathrm{i}}{\to}_{\mathcal{R}} = \overset{\mathcal{R}}{\to}_{\mathcal{R}}$). The following lemma shows that $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ is "increasing" if $\mathcal{R}$ is "increasing" and $\mathcal{Q}$ is "decreasing".

Lemma 3 (Monotonicity of $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$). *$\mathcal{R}' \subseteq \mathcal{R}$ and $\mathcal{Q}' \supseteq \mathcal{Q}$ implies $\overset{\mathcal{Q}'}{\to}_{\mathcal{R}'} \subseteq \overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.*

Proof. Obvious. □

This lemma already indicates why $\mathcal{Q}$-restricted rewriting is better suitable for termination analysis than innermost rewriting. There exist several techniques which can simplify termination proofs by removing rules from the TRS $\mathcal{R}$. For full rewriting and also for $\mathcal{Q}$-restricted rewriting, removal of rules is always advantageous, since it can never introduce non-termination (termination of $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$

[1] Since the right-hand sides of $\mathcal{Q}$'s rules are not needed to define $\mathcal{Q}$-restricted rewriting, Def. 1 could also be formulated if $\mathcal{Q}$ is a set of *terms* instead of *rules*.

implies termination of $\xrightarrow{\mathcal{Q}}_{\mathcal{R}'}$ if $\mathcal{R}' \subseteq \mathcal{R}$). But for innermost rewriting, this is not true. For instance, by removing the rule $\mathsf{a} \to \mathsf{b}$ from the innermost terminating TRS $\mathcal{R}$ of Ex. 2, we result in a TRS $\mathcal{R}'$ that is not innermost terminating (hence, $\xrightarrow{\mathrm{i}}_{\mathcal{R}'} \not\subseteq \xrightarrow{\mathrm{i}}_{\mathcal{R}}$). Here, $\mathcal{Q}$-restricted rewriting has the advantage that the rules $\mathcal{Q}$ which restrict the set of possible redexes are separated from the rules $\mathcal{R}$ used for rewriting and thus, $\mathcal{R}$ and $\mathcal{Q}$ can be changed independently.

Now we present a termination criterion for $\mathcal{Q}$-restricted rewriting based on dependency pairs. For a TRS $\mathcal{R}$ over $\mathcal{F}$, the *defined symbols* are $\mathcal{D} = \{\mathrm{root}(l) \mid l \to r \in \mathcal{R}\}$ and the *constructors* are $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. For every $f \in \mathcal{D}$ we extend the signature $\mathcal{F}$ by a fresh *tuple symbol* $f^\sharp$, where $f^\sharp$ has the same arity as f and we often write F for $f^\sharp$. If $t = g(t_1, \ldots, t_m)$ with $g \in \mathcal{D}$, we let $t^\sharp$ denote $g^\sharp(t_1, \ldots, t_m)$.

Definition 4 (Dependency Pair). *The set of* dependency pairs *for a TRS $\mathcal{R}$ is $DP(\mathcal{R}) = \{l^\sharp \to t^\sharp \mid l \to r \in \mathcal{R}$, $\mathrm{root}(t) \in \mathcal{D}$, t is a subterm of $r\}$.*[2]

Example 5. The following TRS computes subtraction and division, cf. [1, Ex. 2].

$$\mathsf{minus}(x, \mathsf{0}) \to x \quad (1) \qquad \mathsf{div}(\mathsf{0}, \mathsf{s}(y)) \to \mathsf{0} \quad (4)$$
$$\mathsf{minus}(\mathsf{0}, \mathsf{s}(y)) \to \mathsf{0} \quad (2) \qquad \mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y))) \quad (5)$$
$$\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{minus}(x, y) \quad (3)$$

Here, the defined symbols are minus and div and the symbols $\mathsf{0}$ and s are constructors. We obtain the following dependency pairs:

$$\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y) \quad (6) \qquad \mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y) \quad (7)$$
$$\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y)) \quad (8)$$

To verify $\mathcal{Q}$-termination, we use the notion of *chains*. Intuitively, a dependency pair corresponds to a function call and a chain represents a possible sequence of calls that can occur in a reduction. For termination, we try to prove that there are no infinite chains. We always assume that different occurrences of dependency pairs are variable disjoint and consider substitutions whose domains may be infinite. In the following definition, $\mathcal{P}$ is usually a set of dependency pairs.

Definition 6 (Chain). *Let $\mathcal{P}, \mathcal{Q}, \mathcal{R}$ be TRSs. A (possibly infinite) sequence of pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ from $\mathcal{P}$ is a* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain *iff there is a substitution σ such that $t_i\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} s_{i+1}\sigma$ for all i and all $s_i\sigma$ are in $\mathcal{Q}$-normal form. A chain is* minimal *iff there is a σ as above where all $t_i\sigma$ are terminating w.r.t. $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$.*

Example 7. If $\mathcal{Q} \subseteq \mathcal{R}$, then the TRS of Ex. 5 has the following chain which consists of two occurrences of the dependency pair (8).

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

The reason is that $\mathsf{DIV}(\mathsf{minus}(x_1, y_1), \mathsf{s}(y_1))\sigma \xrightarrow{\mathcal{Q}}{}^*_{\mathcal{R}} \mathsf{DIV}(\mathsf{s}(x_2), \mathsf{s}(y_2))\sigma$ holds for some substitution σ (e.g., $\sigma(x_1) = \mathsf{s}(\mathsf{0})$ and $\sigma(x_2) = \sigma(y_i) = \mathsf{0}$ for $i \in \{1, 2\}$) such that all instantiated left-hand sides $\mathsf{DIV}(\mathsf{s}(x_i), \mathsf{s}(y_i))\sigma$ are in $\mathcal{Q}$-normal form. Moreover, the chain is minimal, since all instantiated right-hand sides of the

[2] It even suffices only to regard dependency pairs where t is no proper subterm of l [4].

dependency pairs are terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.

As mentioned above, termination corresponds to absence of infinite chains. Here, it suffices to consider minimal chains, since minimal non-terminating terms (whose proper subterms are terminating) correspond to infinite minimal chains. The following termination criterion is immediately obtained from [1, Thm. 6].

Theorem 8 (Termination Criterion). *These three properties are equivalent:*

- *$\mathcal{R}$ is $\mathcal{Q}$-terminating*
- *there is no infinite $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R})$-chain*
- *there is no infinite* minimal *$(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R})$-chain*

With this criterion, we can now state the dependency pair framework. The basic idea of this framework is to examine a set of dependency pairs $\mathcal{P}$ together with the TRSs $\mathcal{Q}$ and $\mathcal{R}$ and to prove absence of (minimal) infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chains instead of examining the relation $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ directly. The advantages of this approach will become clear in the sequel. For example, it will be possible to decompose this so-called *dependency pair problem* into several independent sub-problems. These problems can then be solved separately using different techniques, which leads to a very modular approach to termination proving.

More precisely, a dependency pair problem ("DP problem", for short) consists of three TRSs $\mathcal{P}$, $\mathcal{Q}$, and $\mathcal{R}$ (where initially, $\mathcal{P} = DP(\mathcal{R})$) and a flag $f \in \{\mathbf{m}, \mathbf{a}\}$ which stands for "$\underline{\mathbf{m}}$inimal" or "$\underline{\mathbf{a}}$rbitrary". Initially, we have $f = \mathbf{m}$. Our goal is to show that there is no infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain if $f = \mathbf{m}$ and that there is no infinite (possibly non-minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain if $f = \mathbf{a}$. If this is possible, then we call the problem *finite*.

A DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ that is not finite is called *infinite*. But in addition, $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is already *infinite* whenever $\mathcal{R}$ is not $\mathcal{Q}$-terminating. So in particular, the existence of any (possibly non-minimal) infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain suffices to conclude that $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is infinite, even if $f = \mathbf{m}$. While the initial DP problem $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$ is either finite or infinite, other DP problems $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ which can occur in termination proofs can be both finite and infinite. For example, the DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ with $\mathcal{P} = \{\mathsf{A} \to \mathsf{B}\}$, $\mathcal{Q} = \varnothing$ and $\mathcal{R} = \{\mathsf{a} \to \mathsf{a}, \mathsf{a} \to \mathsf{b}, \mathsf{b} \to \mathsf{c}\}$ is finite since there is no infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, but also infinite since $\mathcal{R}$ is not $(\mathcal{Q}$-$)$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 (provided that all proof steps were "complete", i.e., that 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. 8, 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. They should transform a DP problem into a new set of problems which then have to be solved instead. Alternatively, they can also return the answer "no". We refer to such techniques as *dependency pair processors* ("DP processors").

Definition 9 (DP Problems and Processors). *A* DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *consists of three TRSs* $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$ *and a flag* $f \in \{\mathbf{m}, \mathbf{a}\}$. *A DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ *is* finite *iff there is no infinite minimal* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$*-chain and* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{a})$ *is* finite *iff there is no infinite* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$*-chain. A DP problem* $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ *is* infinite *iff it is not finite or if* $\mathcal{R}$ *is not* $\mathcal{Q}$*-terminating.*

A DP processor *is a function Proc which takes a DP problem as input and returns either a set of DP problems or the result "*no*". A DP processor Proc is* sound *if for all DP problems d, d is finite whenever Proc(d) is not "*no*" and all DP problems in Proc(d) are finite. A DP processor 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.*

Thus, soundness is required in order to use a DP processor *Proc* to prove termination (in particular, to conclude that d is finite if $Proc(d) = \varnothing$). Completeness is needed in order to use *Proc* to prove non-termination (in particular, to conclude that d is infinite if $Proc(d) = \mathsf{no}$). Even if one is only interested in proving termination, completeness is still advantageous, since it ensures that one does not transform non-infinite DP problems into infinite ones (i.e., applying the processor does not "harm"). The reason for the above 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.[3]

The following corollary introduces the dependency pair framework ("DP framework", for short). The idea is to start with the initial DP problem where $\mathcal{P} = DP(\mathcal{R})$ and $f = \mathbf{m}$. 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 proved that the original TRS is not terminating. The proof of Cor. 10 is immediate from Def. 9 and Thm. 8.

Corollary 10 (Dependency Pair Framework). *Let* $\mathcal{R}$ *and* $\mathcal{Q}$ *be TRSs. We construct a tree whose nodes are labelled with DP problems or "*yes*" or "*no*" and whose root is labelled with* $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$. *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* $\mathcal{Q}$*-terminating. 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* $\mathcal{Q}$*-terminating.*

[3] That a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ is already "finite" if there are no infinite *minimal* chains will be required for the soundness of many processors (cf. Thm. 19, Thm. 28, Thm. 32, and Thm. 37) and that a DP problem is already "infinite" if $\mathcal{R}$ is not $\mathcal{Q}$-terminating will be required for the completeness of most processors for dependency pair transformations (cf. Ex. 24 in Sect. 3.3).

Example 11. If d_0 is the initial problem $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$, $Proc_0$, $Proc_1$, $Proc_2$ are sound DP processors, and $Proc_0(d_0) = \{d_1, d_2\}$, $Proc_1(d_1) = \varnothing$, $Proc_2(d_2) = \varnothing$, then one could obtain the first tree below and conclude termination.

$$\begin{tikzcd}
& d_0 \arrow[dl, dash] \arrow[dr, dash] & \\
d_1 \arrow[d, dash] & & d_2 \arrow[d, dash] \\
\mathsf{yes} & & \mathsf{yes}
\end{tikzcd}
\qquad
\begin{tikzcd}
& & d_0 \arrow[dl, dash] \arrow[dr, dash] & \\
& d_1 \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & & d_2 \arrow[d, dash] \\
d_3 & d_4 & d_5 & \mathsf{no}
\end{tikzcd}$$

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

In the remainder of the paper, we present several sound DP processors which can be used for termination proofs within the DP framework. Of course, it is desirable to find DP processors which transform a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ into a set of "simpler" problems and whose application can never "harm". Therefore, we are particularly interested in processors which decrease $\mathcal{P}$ and $\mathcal{R}$ and which increase $\mathcal{Q}$. As stated by Lemma 3, decreasing the set of rules $\mathcal{R}$ and increasing $\mathcal{Q}$ leads to a more restricted rewrite relation and thus, it can never transform a non-infinite DP problem into an infinite one. In other words, any DP processor which removes rules from $\mathcal{P}$ and $\mathcal{R}$ and which adds rules to $\mathcal{Q}$ is complete.

Lemma 12 (Completeness of DP Processors). *Let Proc be a DP processor where for all DP problems $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ and all $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f') \in Proc((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f))$ we have $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{R}' \subseteq \mathcal{R}$, and $\mathcal{Q}' \supseteq \mathcal{Q}$. Then Proc is complete.*

Proof. Let some $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f') \in Proc((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f))$ be infinite and suppose that $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is not infinite. Thus, $\mathcal{R}$ is $\mathcal{Q}$-terminating and $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ is finite. Due to $\mathcal{Q}$-termination of $\mathcal{R}$, every $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain is minimal and thus, there is no infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, even if $f = \mathbf{m}$.

Note that $\mathcal{Q}'$-termination of $\mathcal{R}'$ follows from $\mathcal{Q}$-termination of $\mathcal{R}$ by Lemma 3. So if $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f')$ is infinite, there must be an infinite $(\mathcal{P}', \mathcal{Q}', \mathcal{R}')$-chain. But then this is also an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain which contradicts the observation above. The reason is that $t_i\sigma \overset{\mathcal{Q}'}{\to}{}^*_{\mathcal{R}'} s_{i+1}\sigma$ implies $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s_{i+1}\sigma$ by Lemma 3 and if $s_i\sigma$ is a $\mathcal{Q}'$-normal form then it is also a $\mathcal{Q}$-normal form. $\square$

3 DP Processors From the Dependency Pair Approach

In the classical dependency pair approach, finiteness of a DP problem was shown by first modularizing the proof using the *dependency graph* (cf. Sect. 3.1.) Afterwards, a set of inequalities was generated and one tried to find certain orders satisfying these inequalities (cf. Sect. 3.2). Moreover, before constructing the dependency graph, it was possible to transform dependency pairs (cf. Sect. 3.3). In this section, we develop DP processors which perform these three tasks. Thus, the whole dependency pair approach is now presented as a set of processors working on DP problems. Each step of the dependency pair approach is formulated as a DP processor on its own. In this way, flexibility is increased substantially: now these DP processors may be applied repeatedly and in any order, whereas their order of application was fixed in the original dependency pair approach.

All processors in this section only modify the set of pairs $\mathcal{P}$ in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$. Processors which also modify the sets $\mathcal{Q}$ and $\mathcal{R}$ will be discussed in Sect. 4 and processors which also modify f will be shown in Sect. 5.

3.1 A DP Processor Based on the Dependency Graph

We now present a processor to decompose a DP problem into several separate 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 13 (Dependency Graph). *Let $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ be a DP problem. The nodes of the* $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graph *are the pairs of $\mathcal{P}$ and there is an arc from $s \to t$ to $u \to v$ iff $s \to t, u \to v$ is a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain.*

Example 14. We regard the TRS for subtraction and division from Ex. 5 again. Here we obtain the following $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graph for all $\mathcal{Q} \subseteq \mathcal{R}$.

$$\begin{tikzcd}
\boxed{\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y))\ (8)} \arrow[loop above] \arrow[dr] & & \boxed{\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)\ (6)} \arrow[loop above] \\
& \boxed{\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)\ (7)} \arrow[ur] &
\end{tikzcd}$$

Obviously, every infinite chain corresponds to a cycle in the dependency graph. Since this graph is in general not computable, for automation one constructs an *estimated* graph. To this end, one has to approximate whether two pairs $s \to t, u \to v$ form a $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. In this case, one draws an arc from $s \to t$ to $u \to v$. As long as the approximation is sound (i.e., as long as it is an over-approximation), the resulting graph contains the real dependency graph and thus, every infinite chain also corresponds to a cycle in the estimated graph.

In the classical dependency pair approach, several such approximations were developed (e.g., [1, 11]) and for example, all of them would be able to compute the graph given in Ex. 14. However, instead of $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chains, here one only considered chains where $\mathcal{Q} = \varnothing$ (for full termination) or where $\mathcal{Q} = \mathcal{R}$ (for innermost termination). The latter were called "*innermost* chains". By Lemma 3, every $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain is also a $(\mathcal{P}, \varnothing, \mathcal{R})$-chain (i.e., an ordinary chain in the classical dependency pair approach) and if $\mathcal{Q} \supseteq \mathcal{R}$, it is also an innermost chain. Thus, all existing methods to (over-)approximate chains in the dependency pair approach can also be used to approximate $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chains for any $\mathcal{Q}$. Moreover, if $\mathcal{Q} \supseteq \mathcal{R}$, then all approximations for innermost chains can be applied as well. Hence, one can still use the existing estimation techniques for (innermost) dependency graphs in order to estimate $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graphs.

Now it is sufficient to prove absence of infinite (minimal) chains for maximal cycles (so-called *strongly connected components*, SCCs) of the dependency graph. Here, a subset $\mathcal{P}'$ of dependency pairs is called a *cycle* iff for all pairs $s \to t$ and $u \to v$ in $\mathcal{P}'$, there is a path from $s \to t$ to $u \to v$ traversing only pairs from $\mathcal{P}'$. A cycle $\mathcal{P}'$ is called an *SCC* if $\mathcal{P}'$ is not a proper subset of any other cycle.

Theorem 15 (DP Processor Based on the Dependency Graph). *The following DP Processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ Proc returns $\{(\mathcal{P}_1, \mathcal{Q}, \mathcal{R}, f), \ldots, (\mathcal{P}_n, \mathcal{Q}, \mathcal{R}, f)\}$, where $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are the SCCs of the (estimated) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-dependency graph.*

Proof. Completeness follows from Lemma 12, since $\mathcal{P}_i \subseteq \mathcal{P}$ for all i. *Proc* is sound since after a finite number of pairs in the beginning, any infinite (minimal) $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain only contains pairs from some SCC. Hence, there would also be an infinite (minimal) $(\mathcal{P}_i, \mathcal{Q}, \mathcal{R})$-chain for some $\mathcal{P}_i$. $\square$

Example 16. To prove $\mathcal{Q}$-termination of the TRS $\mathcal{R}$ for subtraction and division from Ex. 5, we start with the initial DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$, where $\mathcal{P}$ is the set of dependency pairs $\{(6), (7), (8)\}$. As shown in Ex. 14, the SCCs of the dependency graph consist of (6) and (8), respectively. Hence, the above DP processor transforms the initial DP problem into the two new problems $(\{(6)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ and $(\{(8)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$. These two problems can now be solved independently. In other words, we can now prove termination of minus and div separately.

In contrast to the classical dependency pair approach, now the dependency graph can be (re-)computed at any time during the termination proof. This leads to very modular proofs, since one may always decompose DP problems into subproblems which can be solved independently, e.g., by different DP processors.

3.2 A DP Processor Based on Orders

Classical techniques for automated termination proofs try to find a *reduction order* $\succ$, i.e., an order which is well-founded, monotonic, and stable (closed under contexts and substitutions), such that $l \succ r$ holds for all rules $l \to r$ of the TRS. In practice, one mainly uses *simplification orders*, where a term is always greater than its proper subterms [3, 19]. Examples for such orders are the *lexicographic* or *recursive path order* [3, 14], the *Knuth-Bendix order* [15], and (most) *polynomial orders* [17]. However, the power of this approach is limited, since termination of many important TRSs cannot be proved with simplification orders. For instance, simplification orders fail on the TRS of Ex. 5, since the left-hand side of Rule (5) is embedded in its right-hand side if y is instantiated with $\mathsf{s}(x)$.

The dependency pair approach was introduced to overcome the limitations of classical simplification orders. For any TRS, it generates a set of inequality constraints and if there exists a well-founded order satisfying the constraints, then termination is proved. Hence, one can use existing techniques to search for suitable orders and it turns out that in this way, classical simplification orders can prove termination of numerous TRSs where they would have failed otherwise.

We now formalize this idea in the context of the DP framework. To remove pairs from $\mathcal{P}$ in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, one can generate constraints which should be satisfied by a *reduction pair* [16] $(\succsim, \succ)$ where $\succsim$ is reflexive, transitive, monotonic, and stable and $\succ$ is a stable well-founded order compatible with $\succsim$ (i.e., $\succsim \circ \succ \subseteq \succ$ and $\succ \circ \succsim \subseteq \succ$). But $\succ$ does not have to be monotonic. The constraints require that at least one pair in $\mathcal{P}$ is strictly decreasing (w.r.t. $\succ$) and all remaining pairs in $\mathcal{P}$ and all rules in $\mathcal{R}$ are weakly decreasing (w.r.t. $\succsim$).

Requiring $l \succsim r$ for all rules $l \to r \in \mathcal{R}$ ensures that in chains $s_1 \to t_1, s_2 \to t_2, \ldots$ with $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s_{i+1}\sigma$, we have $t_i\sigma \succsim s_{i+1}\sigma$. Hence, the existence of such a reduction pair implies that there is no chain which contains the strictly decreasing pairs of $\mathcal{P}$ infinitely often. Thus, all of these pairs can be deleted from $\mathcal{P}$.

If the $\mathcal{Q}$-restricted rewrite relation is contained in the innermost rewrite relation (i.e., if $\mathcal{Q} \supseteq \mathcal{R}$), this approach can be improved. Now a weak decrease $l \succsim r$ is not required for all rules but only for the *usable rules*. For any term t, all function symbols occurring in t are "usable". Moreover, if some symbol f is usable and there is a rule $f(\ldots) \to r$ in $\mathcal{R}$ whose right-hand side r contains a symbol g, then g is usable as well. Let $\mathcal{US}(t, \mathcal{R})$ denote the set of *usable symbols* and we define the *usable rules* $\mathcal{U}(t, \mathcal{R})$ as the set of those rules $f(\ldots) \to \ldots$ from $\mathcal{R}$ where $f \in \mathcal{US}(t, \mathcal{R})$. Analogously, for a TRS $\mathcal{P}$, the usable symbols and rules are defined as $\mathcal{US}(\mathcal{P}, \mathcal{R}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{US}(t, \mathcal{R})$ and $\mathcal{U}(\mathcal{P}, \mathcal{R}) = \bigcup_{s \to t \in \mathcal{P}} \mathcal{U}(t, \mathcal{R})$. Further refinements to reduce the set of usable rules can be found in [8, 12, 21].

Example 17. Let $\mathcal{R}$ be the TRS of Ex. 5. For the problem $(\{\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ one would now have to find a reduction pair $(\succsim, \succ)$ such that $\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \succ \mathsf{MINUS}(x, y)$ and $l \succsim r$ for all rules. But if one only wants to prove innermost termination (or $\mathcal{Q}$-termination for $\mathcal{Q} \supseteq \mathcal{R}$), it suffices to require $l \succsim r$ just for the usable rules. Since the only usable symbol of the dependency pair's right-hand side $\mathsf{MINUS}(x, y)$ is MINUS, there are no usable rules. So for this DP problem, one only has to find a stable well-founded order $\succ$ satisfying $\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \succ \mathsf{MINUS}(x, y)$. This can easily be done with any of the existing classical reduction orders. Thus, the dependency pair can be deleted and the resulting DP problem $(\varnothing, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ is transformed into the empty set by the dependency graph processor of Thm. 15.

In [21], we recently extended previous results of [8, 23] and showed that if one only has to prove absence of *minimal* chains (i.e., if $f = \mathbf{m}$), then usable rules can also be used for proving full instead of just innermost termination, provided that $\succsim$ is *$\mathcal{C}_\varepsilon$-compatible* (i.e., $\mathsf{c}(x, y) \succsim x$ and $\mathsf{c}(x, y) \succsim y$ for a fresh function symbol c). This holds for virtually all relations $\succsim$ generated automatically by standard techniques. Then, to prove $\mathcal{Q}$-termination (for arbitrary $\mathcal{Q}$) it is enough to require $l \succsim r$ just for the usable rules. In this way, the DP problem for minus can be solved as in Ex. 17 for arbitrary $\mathcal{Q}$.

To generate reduction pairs $(\succsim, \succ)$ automatically, one often uses classical (monotonic) simplification orders. However, $\succ$ does not have to be monotonic. To benefit from this possibility and to build non-monotonic orders from simplification orders, one may pre-process the constraints first and delete certain function symbols and arguments by an *argument filtering* π [1]. For example, if π_1 eliminates the second argument of the function symbol minus, then for any term t, $\pi_1(t)$ results from replacing all subterms $\mathsf{minus}(t_1, t_2)$ by $\mathsf{minus}(t_1)$. Moreover, one can also use argument filterings which collapse function symbols to one of their arguments (i.e., one could also define an argument filtering π_2 with $\pi_2(\mathsf{minus}(t_1, t_2)) = t_1$). Now instead of a reduction pair $(\succsim, \succ)$, one may use the reduction pair $(\succsim_\pi, \succ_\pi)$ with $s \succsim_\pi t$ iff $\pi(s) \succsim \pi(t)$ and $s \succ_\pi t$ iff $\pi(s) \succ \pi(t)$. Techniques to search for argument filterings efficiently were developed in [8, 11].

Example 18. Regard the other DP problem $(\{\mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x,y), \mathsf{s}(y))\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ resulting from the TRS $\mathcal{R}$ of Ex. 5. The usable rules of the term $\mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(y))$ are just the minus-rules. Thus, if we use $\mathcal{C}_\varepsilon$-compatible relations $\succsim$, it suffices to find a reduction pair $(\succsim, \succ)$ and an argument filtering π such that $\mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) \succ_\pi \mathsf{DIV}(\mathsf{minus}(x,y),\mathsf{s}(y))$ and $l \succsim_\pi r$ for all minus-rules.

If we apply the argument filtering π_1 above, the constraint from the dependency pair becomes $\mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) \succ \mathsf{DIV}(\mathsf{minus}(x),\mathsf{s}(y))$ and if we use the argument filtering π_2, it becomes $\mathsf{DIV}(\mathsf{s}(x),\mathsf{s}(y)) \succ \mathsf{DIV}(x,\mathsf{s}(y))$. In both cases, all resulting constraints can easily be satisfied (e.g., by the lexicographic path order). Thus, the dependency pair can be deleted from this DP problem as well and in this way, termination of the TRS in Ex. 5 can easily be shown automatically.

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 $\mathcal{P}_{\succ_\pi}$). The reason is that they cannot occur infinitely many times in a chain.

Theorem 19 (DP Processor Based on Reduction Pairs). *Let $(\succsim, \succ)$ be a reduction pair and π be an argument filtering. Then the following DP processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{Q}, \mathcal{R}, f)\}$, *if*
  - $\mathcal{P}_{\succ_\pi} \cup \mathcal{P}_{\succsim_\pi} = \mathcal{P}$ *and*
  - $\mathcal{R}_{\succsim_\pi} \supseteq \mathcal{U}(\mathcal{P}, \mathcal{R})$ *and*
  - $\mathcal{Q} \supseteq \mathcal{R}$ *or* $\mathcal{R}_{\succsim_\pi} = \mathcal{R}$ *or* ($\succsim$ *is* $\mathcal{C}_\varepsilon$*-compatible and* $f = \mathbf{m}$)
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

Proof. Completeness follows from Lemma 12. If $\succsim$ is $\mathcal{C}_\varepsilon$-compatible, soundness is proved as in [21, Thm. 22]. (The extension from ordinary to $\mathcal{Q}$-restricted rewriting is completely straightforward.) The case $\mathcal{R}_{\succsim_\pi} = \mathcal{R}$ is the classical dependency pair approach for termination and its soundness is proved in [7, Thm. 3.5] and [1, Thm. 11]. Finally, for the case $\mathcal{Q} \supseteq \mathcal{R}$ recall that the $\mathcal{Q}$-restricted rewrite relation is contained in the innermost rewrite relation and thus, every $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain is an innermost chain. Hence, then the soundness follows from the corresponding result of [7, Thm. 5.6] for innermost termination. $\square$

Whenever a processor modifies a DP problem, it is usually advantageous to apply the dependency graph processor of Thm. 15 afterwards. The reason is that in this way one can split the DP problem into new subproblems and probably also remove further rules of $\mathcal{P}$. This is a generalization of a strategy which was originally suggested within the classical dependency pair approach in [11]. 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 now correspond to a repeated alternating application of the processors in Thm. 15 and in Thm. 19. However, by formalizing termination techniques as DP processors, many additional strategies can now easily be formulated as well, cf. Sect. 6.

3.3 DP Processors Based on Dependency Pair Transformations

As shown in [1, 6, 8], to increase the power of the dependency pair approach, a dependency pair may be transformed by *rewriting*, *narrowing*, or *instantiation*. We now adapt these transformations to the DP framework. Given a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, they replace one of the pairs $s \to t$ in $\mathcal{P}$ by several new ones (i.e., the result is of the form $(\mathcal{P}', \mathcal{Q}, \mathcal{R}, f)$). In contrast to the previous processors, we usually do not have $\mathcal{P}' \subseteq \mathcal{P}$, but $\mathcal{P}'$ is obtained by replacing $s \to t$ with new pairs resulting from rewriting, narrowing, or instantiating $s \to t$.

Compared to the original versions of these transformations in the dependency pair approach, they are now improved and modularized considerably. The reason is that now these transformations can be applied at any time during the proof and the conditions for their applicability only have to take the pairs and rules in the current DP problem into account. In this way, these conditions are much more often satisfied than in the original dependency pair approach, where such transformations were only permitted in the very beginning.

In the dependency pair approach, there were two variants of the narrowing and the instantiation transformation (for termination and for innermost termination, respectively), cf. [8]. The DP processors for narrowing and instantiation are immediately obtained from the original transformations, by applying the variants for termination if $\mathcal{Q} = \varnothing$ and by applying the variants for innermost termination if $\mathcal{Q} \supseteq \mathcal{R}$.[4] The soundness and completeness results for these transformations directly carry over from the classical dependency pair approach to their versions in the DP framework: instantiation is sound and complete and narrowing is sound. Completeness of narrowing holds as well in the termination case (i.e., if $\mathcal{Q} = \varnothing$), but not if $\mathcal{Q} \supseteq \mathcal{R}$, cf. [1, Ex. 43]. The proofs for these results are completely analogous to the ones in the dependency pair approach (i.e., to [1, Thm. 27 and 42] for narrowing and to [6, Thm. 20] for instantiation).

While adapting narrowing and instantiation to the DP framework is straightforward, the adaption of the *rewriting* transformation is more problematic. In the classical dependency pair approach, rewriting is only applicable for innermost termination proofs (i.e., if $\mathcal{Q} = \mathcal{R}$). The problem is that the original proofs for its soundness and completeness [6, Thm. 14, 15, 18, 19] do not extend to the case where $\mathcal{Q} \supseteq \mathcal{R}$.[5] However, such an extension is urgently required in the DP framework, since in Sect. 4 we will introduce new powerful DP processors which reduce the set of rules $\mathcal{R}$ in a DP problem. Thus, even if one starts with an innermost termination problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ where $\mathcal{Q} = \mathcal{R}$, after application of some

[4] Of course, when applying the transformations for $\mathcal{Q} \supseteq \mathcal{R}$ instead of "normal forms" in [8] one always has to regard "$\mathcal{Q}$-normal forms".

[5] In contrast to the narrowing and instantiation transformations which perform *all* possible narrowings or instantiations, here one may replace $s \to t$ by *any* pair resulting from rewriting t. The soundness proof relies on the result that weak innermost termination and non-overlappingness imply confluence and termination [10] and the completeness proof relies on the result that innermost termination implies normalization. Obviously, these results do not extend to $\mathcal{Q} \supseteq \mathcal{R}$, i.e., in general $\mathcal{Q}$-termination and non-overlappingness do not imply confluence, termination, or normalization.

DP processors one might result in a problem $(\mathcal{P}', \mathcal{Q}', \mathcal{R}', f')$ where $\mathcal{Q}' \supset \mathcal{R}'$. Now it would be desirable if one could still apply the rewriting transformation (this will be demonstrated in Ex. 29). Therefore, we now present a new proof to show that the rewriting transformation is sound for any $\mathcal{Q} \supseteq \mathcal{R}$ in the DP framework. Moreover, if for all subterms q below the position where the rewriting took place we have $\mathcal{U}(q, \mathcal{Q}) \subseteq \mathcal{R}$, then it is also complete. So q's usable rules w.r.t. $\mathcal{Q}$ also have to be contained in $\mathcal{R}$. In the special case of innermost termination where $\mathcal{Q} = \mathcal{R}$, this completeness condition is always fulfilled (i.e., our results encompass the completeness result for innermost termination from [6, Thm. 19]).[6]

For the proof, we need the following sufficient criterion for confluence of the $\mathcal{Q}$-restricted rewrite relation. It states that if there are no critical pairs on the root level, then we even have the *diamond property* (i.e., if $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} t_1$ and $t \overset{\mathcal{Q}}{\to}_{\mathcal{R}} t_2$, then $t_1 = t_2$ or there exists a t' such that $t_1 \overset{\mathcal{Q}}{\to}_{\mathcal{R}} t'$ and $t_2 \overset{\mathcal{Q}}{\to}_{\mathcal{R}} t'$).

Lemma 20 (Confluence of $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$). *Let $\mathcal{Q} \supseteq \mathcal{R}$ and let $\mathcal{R}$ be non-overlaying (i.e., left-hand sides of different $\mathcal{R}$-rules do not unify after variable renaming). Then $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ has the diamond property and hence, it is confluent.*

Proof. Let $t \overset{\mathcal{Q}}{\to}_{\mathcal{R},p_1} t_1$ and $t \overset{\mathcal{Q}}{\to}_{\mathcal{R},p_2} t_2$. Since $t|_{p_1}$ and $t|_{p_2}$ are no $\mathcal{R}$-normal forms and thus no $\mathcal{Q}$-normal forms (by $\mathcal{Q} \supseteq \mathcal{R}$), p_2 cannot be strictly above p_1 and p_1 cannot be strictly above p_2. If $p_1 = p_2$, then $t_1 = t_2$, as we used the same rule in both reductions since $\mathcal{R}$ is non-overlaying. Otherwise, p_1 and p_2 are not above each other, and thus, t_1 and t_2 can obviously be joined in one step. $\square$

Now we introduce the rewriting processor. It states that in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ with $\mathcal{Q} \supseteq \mathcal{R}$, any $s \to t \in \mathcal{P}$ can be replaced by $s \to t'$ if t rewrites to t' at some position p. The only applicability condition is that the usable rules for the redex $t|_p$ must be non-overlapping (i.e., they must not have critical pairs).

Theorem 21 (Rewriting Processor). *Let Proc be a processor which transforms a DP problem $(\mathcal{P} \cup \{s \to t\}, \mathcal{Q}, \mathcal{R}, f)$ either into $\{(\mathcal{P} \cup \{s \to t'\}, \mathcal{Q}, \mathcal{R}, f)\}$ or into $\{(\mathcal{P} \cup \{s \to t\}, \mathcal{Q}, \mathcal{R}, f)\}$ again. In the first case, the following two conditions must be satisfied:*

- $t \to_{\mathcal{R},p} t'$ *and* $\mathcal{U}(t|_p, \mathcal{R})$ *is non-overlapping*
- $\mathcal{Q} \supseteq \mathcal{R}$

Proc is sound, and it is complete if $\mathcal{U}(q, \mathcal{Q}) \subseteq \mathcal{R}$ for all proper subterms q of $t|_p$.

Proof. Let $t = t[l\mu]_p \to_{\mathcal{R}} t[r\mu]_p = t'$ for some $l \to r \in \mathcal{R}$ and a substitution μ.

We first prove the soundness and only consider the case where $f = \mathbf{m}$. The case $f = \mathsf{a}$ is analogous. Let $s \to t, u \to v$ be a minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. Thus, $t\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} u\sigma$, both $s\sigma$ and $u\sigma$ are in $\mathcal{Q}$-normal form, and $t\sigma$ and $v\sigma$ are terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$. We want to show that $t'\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} u\sigma$ and that $t'\sigma$ is also terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$. Then we can exchange all occurrences of $s \to t$ in chains by $s \to t'$.

[6] A similar completeness result also holds for narrowing: if $\mathcal{Q} \supseteq \mathcal{R}$, then the narrowing processor is still complete if for all narrowed subterms, the usable rules w.r.t. $\mathcal{R}$ are non-overlapping and for all subterms q below those positions that were narrowed we have $\mathcal{U}(q, \mathcal{Q}) \subseteq \mathcal{R}$. Again, the latter condition is always satisfied if $\mathcal{Q} = \mathcal{R}$. Thus, this encompasses the completeness result of [6, Thm. 17] for innermost termination.

We consider the reduction $t\sigma = t\sigma[l\mu\sigma]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} u\sigma$. As $u\sigma$ is in $\mathcal{Q}$-normal form and as $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$-reductions cannot take place above $\mathcal{Q}$-redexes, w.l.o.g. we first reduce $l\mu\sigma$ to some $\mathcal{Q}$-normal form w. Thus, $t\sigma = t\sigma[l\mu\sigma]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} t\sigma[w]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} u\sigma$ where $l\mu\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} w$. Since σ instantiates all variables by normal forms w.r.t. $\mathcal{Q} \supseteq \mathcal{R}$, the only rules applicable to $t|_p\sigma = l\mu\sigma$ or its reducts are from $\mathcal{U}(t|_p, \mathcal{R})$. For readability, we abbreviate $\mathcal{U}(t|_p, \mathcal{R})$ by $\mathcal{U}$. Hence, $l\mu\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{U}} w$. As w is a $\mathcal{Q}$-normal form, w.l.o.g. one first reduces all terms $x\mu\sigma$ with $x \in \mathcal{V}(l)$ to $\mathcal{Q}$-normal forms. As $\mathcal{U}$ is non-overlapping and $\mathcal{Q} \supseteq \mathcal{R} \supseteq \mathcal{U}$, by Lemma 20 these normal forms are unique. Thus, $l\mu\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{U}} l\delta \overset{\mathcal{Q}}{\to}^*_{\mathcal{U}} w$ for some $\mathcal{Q}$-normal substitution δ (i.e., $\delta(x)$ is in $\mathcal{Q}$-normal form for all $x \in \mathcal{V}$) and for all $x \in \mathcal{V}(l)$ we have $x\mu\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{U}} x\delta$. Note that $l\delta$ is not yet a $\mathcal{Q}$-normal form as $l \to r \in \mathcal{R} \subseteq \mathcal{Q}$. Thus, we need at least one more step to get from $l\delta$ to w. As δ is a $\mathcal{Q}$-normal substitution, the reduction is above δ and as $\mathcal{U}$ is non-overlapping, the only possible reduction is $l\delta \overset{\mathcal{Q}}{\to}_{\mathcal{U}} r\delta \overset{\mathcal{Q}}{\to}^*_{\mathcal{U}} w$. This finally proves $t'\sigma = t\sigma[r\mu\sigma]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} t\sigma[r\delta]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} t\sigma[w]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} u\sigma$.

Now minimality (i.e., termination of $t'\sigma$ w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$), can be proved in an analogous way. As before, w.l.o.g. any infinite $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$-reduction of $t'\sigma = t\sigma[r\mu\sigma]_p$ first reduces all redexes in $x\mu\sigma$ for $x \in \mathcal{V}(r)$. These reductions either lead to non-termination or they end in some $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$-normal forms. Since $x \in \mathcal{V}(r) \subseteq \mathcal{V}(l)$, all $x\mu$ are contained in $t|_p$. As σ instantiates variables by normal forms w.r.t. $\mathcal{Q} \supseteq \mathcal{R}$, again the only rules applicable to subterms of $t|_p\sigma$ (like $x\mu\sigma$) are from $\mathcal{U}$. As $\overset{\mathcal{Q}}{\to}_{\mathcal{U}}$ is confluent by Lemma 20, the reduction must begin with $r\mu\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} r\delta$. Hence, whenever $t'\sigma$ is non-terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ then so is $t\sigma[r\delta]_p$. But this would contradict the termination of t w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ as we know that $t \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} t\sigma[r\delta]_p$.

The completeness of the rewriting processor is obvious if $\mathcal{R}$ is not $\mathcal{Q}$-terminating. Otherwise, we show that if there is a reduction $t'\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} u\sigma$ such that $s\sigma$ and $u\sigma$ are in $\mathcal{Q}$-normal form, then $t\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} u\sigma$ also holds. We use the same way of reasoning as for the soundness proof. So if $t'\sigma = t\sigma[r\mu\sigma]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} u\sigma$, we may assume that we first reduce $r\mu\sigma$ to some $\mathcal{Q}$-normal form w which is again done with $\overset{\mathcal{Q}}{\to}_{\mathcal{U}}$ reductions. By the confluence of $\overset{\mathcal{Q}}{\to}_{\mathcal{U}}$ (Lemma 20), we have $r\mu\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{U}} r\delta \overset{\mathcal{Q}}{\to}^*_{\mathcal{U}} w$ for some $\mathcal{Q}$-normal substitution δ. In the same way as before we obtain $t'\sigma = t\sigma[r\mu\sigma]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} t\sigma[r\delta]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} t\sigma[w]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} u\sigma$. It remains to show that $l\mu\sigma \overset{\mathcal{Q}}{\to}_{\mathcal{R}}{}^* r\delta$, as this implies $t\sigma = t\sigma[l\mu\sigma]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} t\sigma[r\delta]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} t\sigma[w]_p \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} u\sigma$.

We know that $x\mu\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} x\delta$ for all $x \in \mathcal{V}(r)$, where $x\delta$ is in $\mathcal{Q}$-normal form. By $\mathcal{Q} \supseteq \mathcal{R}$, $x\delta$ is in normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$, too. As $\mathcal{R}$ is $\mathcal{Q}$-terminating, we can extend δ such that $x\delta$ is a normal form of $x\mu\sigma$ w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ for every variable $x \in \mathcal{V}(l) \setminus \mathcal{V}(r)$. Then we have $l\mu\sigma \overset{\mathcal{Q}}{\to}_{\mathcal{R}}{}^* l\delta$. To prove the desired result $l\delta \overset{\mathcal{Q}}{\to}_{\mathcal{R}} r\delta$, we show that $l\delta$ does not contain $\mathcal{Q}$-redexes as proper subterms.

First suppose that $l\delta$ contains a $\mathcal{Q}$-redex at a position o of l where $l|_o \notin \mathcal{V}$. If a rule $l' \to r' \in \mathcal{Q}$ can be applied to $l|_o\delta$ at root position, then we have $l' \to r' \in \mathcal{U}(l|_o, \mathcal{Q}) \subseteq \mathcal{U}(l|_o\mu, \mathcal{Q}) \subseteq \mathcal{R}$, by the prerequisite of the theorem. Since $\mathrm{root}(l')$ occurs in $l|_o$ and hence in l, we also have $l' \to r' \in \mathcal{U}(t|_p, \mathcal{R})$. But then $l' \to r'$ and $l \to r$ are two rules from $\mathcal{U}(t|_p, \mathcal{R})$ which form a critical pair. This contradicts the requirement that $\mathcal{U}(t|_p, \mathcal{R})$ is non-overlapping.

Now we show that $x\delta$ cannot contain a $\mathcal{Q}$-redex for $x \in \mathcal{V}(l)$. Note that if $x\delta$ is not a $\mathcal{Q}$-normal form, it is also not a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{Q}}$. Thus, there is a term w' such that $x\mu\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{Q}} x\delta \overset{\mathcal{Q}}{\to}_{\mathcal{Q}} w'$. We now show that $\mathcal{Q}$-reductions starting from $x\mu\sigma$ can only use rules from $\mathcal{R}$. Then we also have $x\mu\sigma \overset{\mathcal{Q}}{\to}^*_{\mathcal{R}} x\delta \overset{\mathcal{Q}}{\to}_{\mathcal{R}} w'$. But this contradicts the fact that $x\delta$ is in normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.

Any $\mathcal{Q}$-reduction starting from $x\mu\sigma$ only uses rules from $\mathcal{U}(x\mu, \mathcal{Q})$, since σ is a $\mathcal{Q}$-normal substitution. As $x\mu$ is a subterm of $t|_p$, by the prerequisite of the theorem we have $\mathcal{U}(x\mu, \mathcal{Q}) \subseteq \mathcal{R}$. Thus, any $\mathcal{Q}$-reduction starting from $x\mu\sigma$ can indeed only use rules from $\mathcal{R}$. $\square$

Example 22. We replace the minus-rule (3) of Ex. 5 by the following rules:

$$\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{minus}(\mathsf{p}(\mathsf{s}(x)), \mathsf{p}(\mathsf{s}(y))) \quad (9) \qquad \mathsf{p}(\mathsf{s}(x)) \to x \quad (11)$$
$$\mathsf{minus}(x, \mathsf{plus}(y, z)) \to \mathsf{minus}(\mathsf{minus}(x, y), z) \quad (10) \qquad \mathsf{plus}(\mathsf{0}, y) \to y \quad (12)$$
$$\mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{plus}(x, y)) \quad (13)$$

Now (innermost) termination cannot be shown by the previous processors (if one uses reduction pairs based on (quasi-)simplification orders in Thm. 19).[7] The reason is that the dependency pair $\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(\mathsf{p}(\mathsf{s}(x)), \mathsf{p}(\mathsf{s}(y)))$ from Rule (9) is not strictly decreasing w.r.t. any classical reduction order.

However, by the rewrite processor, the right-hand side of this dependency pair can be rewritten twice to obtain $\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)$. The processor is applicable, since the only usable rule of the redexes $\mathsf{p}(\mathsf{s}(x))$ and $\mathsf{p}(\mathsf{s}(y))$ is (11) which is non-overlapping (although the whole TRS is overlapping). Afterwards, innermost termination can easily be shown. However, since rewriting is only possible for $\mathcal{Q} \supseteq \mathcal{R}$, this step is not permitted if one wants to prove termination (where $\mathcal{Q} = \varnothing$). Note that this TRSs does not belong to those classes of TRSs where it is known that innermost termination implies termination. (A well-known such class are locally confluent overlay systems [10], but due to Rule (10) this system is neither locally confluent nor an overlay system.) In Sect. 4, we will introduce new DP processors to simplify DP problems and it will turn out that then the termination of this example is very easy to prove, cf. Ex. 29 and 31.[8]

The following examples show why straightforward extensions of the rewriting processor would destroy soundness or completeness, respectively.

Example 23. Since non-overlayingness already implies confluence of the $\mathcal{Q}$-restricted rewrite relation by Lemma 20, a natural question is whether the rewrite

[7] Here, $\succsim$ is a *quasi-simplification order* if $s \succsim t$ for all subterms t of s. However, a proof with Thm. 19 using the very recently developed negative polynomial orders of [13] would be possible. An example where negative polynomials fail as well is Ex. 33.

[8] Alternatively, termination can also be proved using the *narrowing* transformation. Then the right-hand side $\mathsf{MINUS}(\mathsf{p}(\mathsf{s}(x)), \mathsf{p}(\mathsf{s}(y)))$ would be replaced by its narrowings $\mathsf{MINUS}(x, \mathsf{p}(\mathsf{s}(y)))$ and $\mathsf{MINUS}(\mathsf{p}(\mathsf{s}(x)), y)$. However, narrowing is only permitted for right-hand sides of dependency pairs which do not unify with left-hand sides. Thus, narrowing would no longer be possible if one adds the rule $\mathsf{minus}(\mathsf{p}(x), y) \to \mathsf{p}(\mathsf{minus}(x, y))$, since $\mathsf{MINUS}(\mathsf{p}(\mathsf{s}(x)), \mathsf{p}(\mathsf{s}(y)))$ unifies with the left-hand side of the new dependency pair $\mathsf{MINUS}(\mathsf{p}(x), y) \to \mathsf{MINUS}(x, y)$. In contrast, the termination proof with the new DP processors in Ex. 29 and 31 would still work.

processor in Thm. 21 would already be sound if the usable rules $\mathcal{U}(t|_p, \mathcal{R})$ are just non-overlaying instead of non-overlapping. This is refuted by the following counterexample. Let $\mathcal{R} = \mathcal{Q} = \{\mathsf{f}(\mathsf{c}) \to \mathsf{d}, \mathsf{f}(\mathsf{h}(x)) \to \mathsf{a}, \mathsf{h}(\mathsf{b}) \to \mathsf{c}, \mathsf{g}(\mathsf{d}, x) \to \mathsf{g}(\mathsf{f}(\mathsf{h}(x)), x)\}$. Then $\mathcal{R}$ is not innermost terminating (i.e., not $\mathcal{Q}$-terminating):

$$\mathsf{g}(\mathsf{d}, \mathsf{b}) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{g}(\mathsf{f}(\mathsf{h}(\mathsf{b})), \mathsf{b}) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{g}(\mathsf{f}(\mathsf{c}), \mathsf{b}) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \mathsf{g}(\mathsf{d}, \mathsf{b}) \xrightarrow{\mathrm{i}}_{\mathcal{R}} \ldots$$

The dependency graph has only one SCC $\{\mathsf{G}(\mathsf{d}, x) \to \mathsf{G}(\mathsf{f}(\mathsf{h}(x)), x)\}$. Since $\mathcal{R}$ is non-overlaying and $\mathsf{G}(\mathsf{f}(\mathsf{h}(x)), x) \to_{\mathcal{R}} \mathsf{G}(\mathsf{a}, x)$, rewriting would transform the dependency pair of the SCC into the new pair $\mathsf{G}(\mathsf{d}, x) \to \mathsf{G}(\mathsf{a}, x)$. Now the dependency graph processor would delete this pair, since it is obviously not on any cycle of the (new) dependency graph. Thus, we could falsely "prove" termination.

The problem is that although the dependency pair was rewritten by a $\mathcal{Q}$-restricted step, it is no longer $\mathcal{Q}$-restricted if one instantiates x with b. So to guarantee that any reduction from $\mathsf{G}(\mathsf{f}(\mathsf{h}(x)), x)\sigma$ to an instantiated left-hand side of a dependency pair is also possible from $\mathsf{G}(\mathsf{a}, x)\sigma$, one needs non-overlappingness and not just confluence of $\mathcal{U}(t|_p, \mathcal{R})$'s $\mathcal{Q}$-restricted rewrite relation.

Example 24. This processor shows why we defined a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ to be "infinite" if it is not finite *or if $\mathcal{R}$ is not $\mathcal{Q}$-terminating*, cf. Def. 9. The reason is that if "infinite" would be defined as "not finite", then the rewriting processor would be incomplete, i.e., it could transform DP problems that are not infinite into problems with infinite chains, even if $\mathcal{Q} = \mathcal{R}$. Let $\mathcal{P} = \{\mathsf{F}(x, x) \to \mathsf{F}(\mathsf{b}, \mathsf{g}(\mathsf{a}(x), x))\}$ and $\mathcal{Q} = \mathcal{R} = \{\mathsf{g}(x, y) \to y, \mathsf{a}(\mathsf{b}) \to \mathsf{a}(\mathsf{b})\}$. Obviously, $\mathcal{R}$ is not $\mathcal{Q}$-terminating. But there is no infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain as $\mathsf{F}(\mathsf{b}, \mathsf{g}(\mathsf{a}(x_1), x_1))\sigma \xrightarrow{\mathcal{Q}}_{\mathcal{R}}^{*} \mathsf{F}(x_2, x_2)\sigma$ implies $\sigma(x_2) = \mathsf{b}$. Thus $\mathsf{F}(\mathsf{b}, \mathsf{g}(\mathsf{a}(x_2), x_2))\sigma = \mathsf{F}(\mathsf{b}, \mathsf{g}(\mathsf{a}(\mathsf{b}), \mathsf{b}))$ can only be reduced by $\xrightarrow{\mathcal{Q}}_{\mathcal{R}}$ to itself, but it does not unify with $\mathsf{F}(x_3, x_3)$.

However, the rewriting processor could rewrite the right-hand side $\mathsf{F}(\mathsf{b}, \mathsf{g}(\mathsf{a}(x), x))$ of the dependency pair to $\mathsf{F}(\mathsf{b}, x)$. This results in $\mathcal{P}' = \{\mathsf{F}(x, x) \to \mathsf{F}(\mathsf{b}, x)\}$. Now there is clearly an infinite (minimal) $(\mathcal{P}', \mathcal{Q}, \mathcal{R})$-chain.

Example 25. Finally, we also show that the condition $\mathcal{U}(q, \mathcal{Q}) \subseteq \mathcal{R}$ for all proper subterms q of $t|_p$ is required for completeness. We again let $\mathcal{P} = \{\mathsf{F}(x, x) \to \mathsf{F}(\mathsf{b}, \mathsf{g}(\mathsf{a}(x), x))\}$. Moreover, $\mathcal{R} = \{\mathsf{g}(x, y) \to y\}$ and $\mathcal{Q} = \mathcal{R} \cup \{\mathsf{a}(\mathsf{b}) \to \mathsf{a}(\mathsf{b})\}$. Now $\mathcal{R}$ is $\mathcal{Q}$-terminating. As in Ex. 24, there is no infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. But the rewriting processor would replace $\mathcal{P}$ by $\mathcal{P}' = \{\mathsf{F}(x, x) \to \mathsf{F}(\mathsf{b}, x)\}$ and there is again a minimal infinite $(\mathcal{P}', \mathcal{Q}, \mathcal{R})$-chain. Note however, that the redex $\mathsf{g}(\mathsf{a}(x), x)$ of the reduction has a proper subterm $\mathsf{a}(x)$ whose usable rules w.r.t. $\mathcal{Q}$ are not contained in $\mathcal{R}$. So the condition for completeness in Thm. 21 is violated.

4 New Dependency Pair Processors

Now we introduce new processors which improve the power of termination analysis considerably. The techniques of the classical dependency pair approach from Sect. 3 only modify the pairs $\mathcal{P}$ in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$. In contrast, we now present techniques to decrease the underlying set of rules $\mathcal{R}$ (Sect. 4.1 and 4.2) or to increase the set $\mathcal{Q}$ that restricts possible redexes (Sect. 4.3).

4.1 DP Processors Based on Usable Rules

The first processor shows that in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ one can remove all non-usable rules from $\mathcal{R}$ if $\mathcal{Q} \supseteq \mathcal{R}$ (i.e., if $\overset{\mathcal{Q}}{\to}_{\mathcal{R}} \subseteq \overset{\mathsf{i}}{\to}_{\mathcal{R}}$).

Example 26. After applying the dependency graph processor, the TRS $\mathcal{R}$ for division from Ex. 5 results in the problems $(\{\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ and $(\{\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y))\}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$, cf. Ex. 14.

When proving innermost termination (or if $\mathcal{Q} \supseteq \mathcal{R}$), one can replace $\mathcal{R}$ by the usable rules of $\mathsf{MINUS}(x, y)$ and $\mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y))$, respectively. So the problems become $(\{\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)\}, \mathcal{Q}, \varnothing, \mathbf{m})$ and $(\{\mathsf{DIV}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{DIV}(\mathsf{minus}(x, y), \mathsf{s}(y))\}, \mathcal{Q}, \mathcal{R}', \mathbf{m})$, where $\mathcal{R}'$ are the minus-rules.

A similar restriction to the usable rules was already possible with the DP processor based on reduction pairs in Thm. 19. However, with that processor one immediately had to find a reduction pair which orients the usable rules $\mathcal{U}(\mathcal{P}, \mathcal{R})$ and the pairs in $\mathcal{P}$, and afterwards one remained with a DP problem $(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{Q}, \mathcal{R}, f)$ which still contains the full set of rules $\mathcal{R}$. In contrast, this new processor requires no orientation with reduction pairs and it has the advantage that subsequent DP processors can benefit from the removal of non-usable rules. Therefore whenever $\mathcal{Q} \supseteq \mathcal{R}$, this processor should be applied first.

Theorem 27 (DP Processor Based on Usable Rules). *The following DP processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns $\{\, (\mathcal{P},\ \mathcal{Q},\ \mathcal{U}(\mathcal{P}, \mathcal{R}),\ f) \,\}$ if $\mathcal{Q} \supseteq \mathcal{R}$ and $\{\, (\mathcal{P}, \mathcal{Q}, \mathcal{R}, f) \,\}$ otherwise.*

Proof. Completeness follows from Lemma 12. For soundness, let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. Thus there is a σ such that $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s_{i+1}\sigma$, $s_i\sigma$ is in $\mathcal{Q}$-normal form, and if the chain is minimal then $t_i\sigma$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ for all i. By Lemma 3 then $t_i\sigma$ also terminates w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{U}(\mathcal{P},\mathcal{R})}$.

Since σ instantiates all variables by normal forms w.r.t. $\mathcal{Q} \supseteq \mathcal{R}$, the only rules applicable to $t_i\sigma$ or its reducts are from $\mathcal{U}(\mathcal{P}, \mathcal{R})$. Hence, $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{U}(\mathcal{P},\mathcal{R})} s_{i+1}\sigma$. So $s_1 \to t_1, s_2 \to t_2, \ldots$ is also an infinite (minimal) $(\mathcal{P},\ \mathcal{Q}, \mathcal{U}(\mathcal{P}, \mathcal{R}))$-chain. $\square$

Note that completeness of this processor is due to our new notions of "$\mathcal{Q}$-restricted rewriting" and of "$(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chains", which use two different TRSs $\mathcal{Q}$ and $\mathcal{R}$ to restrict potential redexes and to determine possible rewrite steps, respectively. In other words, this processor would be incomplete in the original dependency pair approach, where one regarded innermost termination and "innermost chains". As an example let $\mathcal{P} = \{\mathsf{F}(\mathsf{a}, x) \to \mathsf{F}(x, x)\}$ and $\mathcal{R} = \{\mathsf{f}(\mathsf{a}, x) \to \mathsf{f}(x, x), \mathsf{a} \to \mathsf{b}\}$. Now there is no infinite innermost chain (i.e., no infinite $(\mathcal{P}, \mathcal{R}, \mathcal{R})$-chain), since the left-hand side of the dependency pair in $\mathcal{P}$ is not in $\mathcal{R}$-normal form. As there are no usable rules, this processor would replace $\mathcal{R}$ by the empty set. In the DP framework, one would obtain the DP problem $(\mathcal{P}, \mathcal{R}, \varnothing, f)$ which still has no infinite chain, but in the classical dependency pair approach, the second component of this DP problem would be disregarded. Since there is an infinite (minimal) innermost chain of $\mathcal{P}$'s dependency pair if the underlying TRS is empty, then this processor would be incomplete.

A similar processor can also be defined in the general case (for arbitrary $\mathcal{Q}$).

Thus, this processor is also suitable for full (instead of just innermost) termination proofs. In contrast to Thm. 27, in order to apply this processor one has to satisfy a set of constraints and one can only use it if one tries to prove absence of *minimal* chains. On the other hand, this processor does not only delete non-usable rules, but it also removes all rules from $\mathcal{P}$ and $\mathcal{R}$ that contain non-usable symbols on their left-hand sides. To this end, for any TRS $\mathcal{P}$ and any subset $\mathcal{F}'$ of the signature we define $\mathcal{P}_{\neg\mathcal{F}'}$ as the set of those rules of $\mathcal{P}$ which contain symbols on left-hand sides that are not in $\mathcal{F}'$, i.e., $\mathcal{P}_{\neg\mathcal{F}'} = \{s \to t \in \mathcal{P} \mid s \notin \mathcal{T}(\mathcal{F}', \mathcal{V})\}$.

As in Thm. 19, to apply this processor, all pairs in $\mathcal{P}$ and all their usable rules have to be (weakly) decreasing. But in contrast to Thm. 19, none of the pairs in $\mathcal{P}$ has to be strictly decreasing. On the other hand, now we require monotonicity and $\mathcal{C}_\varepsilon$-compatibility of the order $\succ$ and one may not use argument filterings.

Theorem 28 (DP Processor Based on Usable Rules and Reduction Pairs). *Let $(\succsim, \succ)$ be a reduction pair where $\succ$ is monotonic and $\mathcal{C}_\varepsilon$-compatible. The following DP processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{\,(\mathcal{P} \setminus \mathcal{P}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})},\ \mathcal{Q},\ \mathcal{U}(\mathcal{P}, \mathcal{R}) \setminus \mathcal{R}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})}, f)\,\}$*, if*
  - $\mathcal{P}_{\succsim} = \mathcal{P}$ *and*
  - $\mathcal{R}_{\succsim} \supseteq \mathcal{U}(\mathcal{P}, \mathcal{R})$ *and*
  - $f = \mathbf{m}$
- $\{\,(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\,\}$*, otherwise*

Proof. Completeness follows by Lemma 12. For soundness, let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. So there is a substitution σ and rules $\{l_{i,1} \to r_{i,1}, \ldots, l_{i,m_i} \to r_{i,m_i}\} \subseteq \mathcal{R}$ with $m_i \geq 0$ where $t_i\sigma = v_{i,0} \overset{\mathcal{Q}}{\to}_{\{l_{i,1}\to r_{i,1}\}} v_{i,1} \overset{\mathcal{Q}}{\to}_{\{l_{i,2}\to r_{i,2}\}} \cdots \overset{\mathcal{Q}}{\to}_{\{l_{i,m_i}\to r_{i,m_i}\}} v_{i,m_i} = s_{i+1}\sigma$, $s_i\sigma$ is in $\mathcal{Q}$-normal form, and the term $t_i\sigma$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ for all i. We now show that this chain ends in an infinite minimal $(\mathcal{P} \setminus \mathcal{P}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})}, \mathcal{Q}, \mathcal{U}(\mathcal{P}, \mathcal{R}) \setminus \mathcal{R}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})})$-chain.

By [21, Lemma 16], there exists a mapping $\mathcal{I} : \mathcal{T}(\mathcal{F}, \mathcal{V}) \to \mathcal{T}(\mathcal{F} \cup \{\mathsf{c}\}, \mathcal{V})$ such that[9] we have $t_i\,\mathcal{I}(\sigma) = \mathcal{I}(t_i\sigma) = \mathcal{I}(v_{i,0}) \to^{+}_{\mathcal{C}_\varepsilon \cup (\mathcal{U}(\mathcal{P},\mathcal{R}) \cap \{l_{i,1}\to r_{i,1}\})} \mathcal{I}(v_{i,1}) \to^{+}_{\mathcal{C}_\varepsilon \cup (\mathcal{U}(\mathcal{P},\mathcal{R}) \cap \{l_{i,2}\to r_{i,2}\})} \cdots \to^{+}_{\mathcal{C}_\varepsilon \cup (\mathcal{U}(\mathcal{P},\mathcal{R}) \cap \{l_{i,m_i}\to r_{i,m_i}\})} \mathcal{I}(v_{i,m_i}) = \mathcal{I}(s_{i+1}\sigma) \to^{*}_{\mathcal{C}_\varepsilon} s_{i+1}\,\mathcal{I}(\sigma)$. Here, "$\to_{\mathcal{C}_\varepsilon \cup (\mathcal{U}(\mathcal{P},\mathcal{R}) \cap \{l_{i,j}\to r_{i,j}\})}$" denotes a reduction with $\mathcal{C}_\varepsilon \cup \{l_{i,j} \to r_{i,j}\}$, where the rule $l_{i,j} \to r_{i,j}$ may only be used if it is contained in $\mathcal{U}(\mathcal{P}, \mathcal{R})$. Moreover, $\mathcal{I}(\sigma)$ is the substitution with $\mathcal{I}(\sigma)\,(x) = \mathcal{I}(\sigma(x))$ for all $x \in \mathcal{V}$.

As $\mathcal{P} \cup \mathcal{U}(\mathcal{P}, \mathcal{R}) \subseteq \succsim$ and $\succ$ is $\mathcal{C}_\varepsilon$-compatible, the reduction $s_1\mathcal{I}(\sigma) \to_{\mathcal{P}} t_1\mathcal{I}(\sigma) \to^{*}_{\mathcal{C}_\varepsilon \cup \mathcal{U}(\mathcal{P},\mathcal{R})} s_2\mathcal{I}(\sigma) \to_{\mathcal{P}} t_2\mathcal{I}(\sigma) \to^{*}_{\mathcal{C}_\varepsilon \cup \mathcal{U}(\mathcal{P},\mathcal{R})} \ldots$ only has finitely many $\to_{\mathcal{C}_\varepsilon}$-steps. So there is an n where for all $i \geq n$ we have $t_i\,\mathcal{I}(\sigma) = \mathcal{I}(t_i\sigma) = \mathcal{I}(v_{i,0}) \to^{+}_{\{l_{i,1}\to r_{i,1}\}} \mathcal{I}(v_{i,1}) \to^{+}_{\{l_{i,2}\to r_{i,2}\}} \cdots \to^{+}_{\{l_{i,m_i}\to r_{i,m_i}\}} \mathcal{I}(v_{i,m_i}) = \mathcal{I}(s_{i+1}\sigma) = s_{i+1}\,\mathcal{I}(\sigma)$.

Due to the definition of $\mathcal{I}$ [21, Def. 14], for any term $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ one proves by structural induction that symbols from $\mathcal{F} \setminus \mathcal{US}(\mathcal{P}, \mathcal{R})$ only occur below c-symbols in $\mathcal{I}(s)$. So since $s_i\,\mathcal{I}(\sigma) = \mathcal{I}(s_i\sigma)$ for $i > n$, symbols from $\mathcal{F} \setminus \mathcal{US}(\mathcal{P}, \mathcal{R})$ only

[9] As in the proof of Thm. 19, the lemma and the mapping $\mathcal{I}$ have to be adapted to $\mathcal{Q}$-restricted instead of full rewriting, which however is completely straightforward.

occur below c in $s_i\,\mathcal{I}(\sigma)$. As s_i does not contain c, we have $s_i \in \mathcal{T}(\mathcal{US}(\mathcal{P},\mathcal{R}),\mathcal{V})$. Thus, $s_i \to t_i \in \mathcal{P} \setminus \mathcal{P}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})}$ for all $i \geq n$.

Inspection of the proof of [21, Lemma 16] reveals that the reductions from $\mathcal{I}(t_i\sigma)$ to $\mathcal{I}(s_{i+1}\sigma)$ never take place below any c-symbol. Hence by the observation above, in the redexes of this reduction, symbols from $\mathcal{F} \setminus \mathcal{US}(\mathcal{P},\mathcal{R})$ only occur below c-symbols. Thus the rules $l_{i,1} \to r_{i,1},\ldots,l_{i,m_i} \to r_{i,m_i}$ used for the reductions are from $\mathcal{U}(\mathcal{P},\mathcal{R}) \setminus \mathcal{R}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})}$.

Hence, even if one uses the *original* substitution σ instead of $\mathcal{I}(\sigma)$, all rules $l_{i,j} \to r_{i,j}$ used in the reduction from $t_i\sigma$ to $s_{i+1}\sigma$ are from $\mathcal{U}(\mathcal{P},\mathcal{R})\setminus\mathcal{R}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})}$. Since these reductions were performed with $\mathcal{Q}$-restricted rewriting, the tail $s_{n+1} \to t_{n+1}, s_{n+2} \to t_{n+2},\ldots$ of the chain is an infinite $(\mathcal{P} \setminus \mathcal{P}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})}, \mathcal{Q}, \mathcal{U}(\mathcal{P},\mathcal{R}) \setminus \mathcal{R}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})})$-chain. The chain is also minimal: since all $t_i\sigma$ are terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$, they are also terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{U}(\mathcal{P},\mathcal{R})\setminus\mathcal{R}_{\neg\mathcal{US}(\mathcal{P},\mathcal{R})}}$ by Lemma 3. $\square$

Example 29. We regard the termination proof of the TRS $\mathcal{R}$ from Ex. 22, i.e.

$\mathsf{minus}(x,\mathsf{0}) \to x$	(1)	$\mathsf{p}(\mathsf{s}(x)) \to x$	(11)
$\mathsf{minus}(\mathsf{0},\mathsf{s}(y)) \to \mathsf{0}$	(2)	$\mathsf{plus}(\mathsf{0},y) \to y$	(12)
$\mathsf{minus}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{minus}(\mathsf{p}(\mathsf{s}(x)),\mathsf{p}(\mathsf{s}(y)))$	(9)	$\mathsf{plus}(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{plus}(x,y))$	(13)
$\mathsf{minus}(x,\mathsf{plus}(y,z)) \to \mathsf{minus}(\mathsf{minus}(x,y),z)$	(10)		

together with the rules (4) and (5) for division. By the dependency graph processor (Thm. 15) we get two DP problems corresponding to the termination of div and plus (which can easily be solved) and the problem $(\{(14),(15),(16)\},\varnothing,\mathcal{R},\mathbf{m})$ with the following dependency pairs:

$$\mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{MINUS}(\mathsf{p}(\mathsf{s}(x)),\mathsf{p}(\mathsf{s}(y))) \quad (14)$$

$$\mathsf{MINUS}(x,\mathsf{plus}(y,z)) \to \mathsf{MINUS}(\mathsf{minus}(x,y),z) \quad (15)$$

$$\mathsf{MINUS}(x,\mathsf{plus}(y,z)) \to \mathsf{MINUS}(x,y) \quad (16)$$

As discussed in Ex. 22, this problem cannot be solved by the previous processors if one uses reduction pairs based on (quasi-)simplification orders.

We now show how the processor of Thm. 28 simplifies this DP problem. An efficient approach to mechanize this processor is to use reduction pairs $(\succsim,\succ)$ based on linear polynomial interpretations $\mathcal{P}ol$ with coefficients from $\{0,1\}$. Due to the monotonicity of $\succ$, for an n-ary function symbol $f \in \mathcal{F}$, we either have $\mathcal{P}ol(f(t_1,\ldots,t_n)) = \mathcal{P}ol(t_1)+\ldots+\mathcal{P}ol(t_n)$ or $\mathcal{P}ol(f(t_1,\ldots,t_n)) = \mathcal{P}ol(t_1)+\ldots+\mathcal{P}ol(t_n)+1$. Since there are just two possible interpretations for each function symbol, the search space is small and our experiments show that with these orders, the processor of Thm. 28 is already very successful.

Here, we use the polynomial interpretation $\mathcal{P}ol$ with $\mathcal{P}ol(f(t_1,\ldots,t_n)) = \mathcal{P}ol(t_1)+\ldots+\mathcal{P}ol(t_n)$ for every function symbol $f \in \mathcal{F}$ (i.e., $\mathcal{P}ol(f) = 0$ for constants f). With this reduction pair, the conditions of Thm. 28 are satisfied whenever $\mathcal{P}$ and $\mathcal{U}(\mathcal{P},\mathcal{R})$ are non-duplicating. Therefore, we can now remove the dependency pairs (15) and (16) and the rule (10) which contain the non-usable symbol plus on their left-hand sides. Moreover, we can delete all non-usable rules (i.e., all rules except the ones for p and minus). So the DP problem $(\{(14),(15),(16)\},\varnothing,\mathcal{R},\mathbf{m})$ is transformed to $(\{(14)\},\varnothing,\{(1),(2),(9),(11)\},\mathbf{m})$.

The only defined usable symbol in (14) is p. Hence, if we apply the processor again with the same reduction pair as above, we can remove the non-usable minus-rules. Thus, we obtain the DP problem $(\{(14)\}, \varnothing, \{(11)\}, \mathbf{m})$. To solve such DP problems, we will introduce another processor in the next section.

Obviously, the processor of Thm. 28 can also be applied for innermost termination proofs in the same way. Then we would obtain the resulting DP problem $(\{(14)\}, \mathcal{R}, \{(11)\}, \mathbf{m})$ instead, i.e., then the second component would be the original TRS. Note that since the processor of Thm. 28 removes rules from the third component of the DP problem, the resulting problem is not a real "innermost termination problem" anymore. In other words, now the second component $\mathcal{R}$ is a proper superset of the third component $\{(11)\}$. Due to our extension of the rewriting transformation to this case in Thm. 21, we can still apply the rewriting processor and replace the dependency pair (14) by $\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(x, y)$ as in Ex. 22. Afterwards, the proof can be completed immediately. But if we would only have the classical rewrite transformation from the dependency pair approach, this step would not be possible.

4.2 A DP Processor Based on Removal of Rules

Now we introduce a processor to remove further rules from $\mathcal{R}$. As in Thm. 19, for a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, all rules in $\mathcal{P}$ and $\mathcal{R}$ are oriented with a reduction pair $(\succsim, \succ)$. The processor in Thm. 19 was used to remove pairs from $\mathcal{P}$ which could be oriented with $\succ$. In contrast, the present processor removes rules from both $\mathcal{P}$ and $\mathcal{R}$ if they can be oriented with $\succ$. On the other hand, here we are again restricted to monotonic orders $\succ$ and we may not use argument filterings.

Theorem 30 (DP Processor Based on Rule Removal). *Let $(\succsim, \succ)$ be a reduction pair where $\succ$ is monotonic. The following DP processor Proc is sound and complete. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{\, (\mathcal{P} \setminus \mathcal{P}_\succ,\ \mathcal{Q},\ \mathcal{R} \setminus \mathcal{R}_\succ,\ f) \,\}$, *if*
  - $\mathcal{P}_\succ \cup \mathcal{P}_\succsim = \mathcal{P}$ *and*
  - $\mathcal{R}_\succ \cup \mathcal{R}_\succsim = \mathcal{R}$
- $\{\, (\mathcal{P}, \mathcal{Q}, \mathcal{R}, f) \,\}$, *otherwise*

Proof. Completeness follows by Lemma 12. For soundness, let $s_1 \to t_1, s_2 \to t_2, \ldots$ be an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain. So there is a σ with $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s_{i+1}\sigma$ and if the chain is minimal, then $t_i\sigma$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ for all i.

As in the proof of [21, Thm. 23], one can show that in the reductions $t_i\sigma \overset{\mathcal{Q}}{\to}_{\mathcal{R}}{}^* s_{i+1}\sigma$, $\mathcal{R}_\succ$-rules are only applied for finitely many i and that $s_i \to t_i \in \mathcal{P}_\succ$ only holds for finitely many i as well. So $s_i \to t_i \in \mathcal{P} \setminus \mathcal{P}_\succ$ and $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}\setminus\mathcal{R}_\succ} s_{i+1}\sigma$ for all $i \geq n$ for some $n \in \mathbb{N}$. Moreover, if all $t_i\sigma$ are terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$, then by Lemma 3 they are terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}\setminus\mathcal{R}_\succ}$ as well. Thus, $s_n \to t_n, s_{n+1} \to t_{n+1}, \ldots$ is an infinite (minimal) $(\mathcal{P} \setminus \mathcal{P}_\succ, \mathcal{Q}, \mathcal{R} \setminus \mathcal{R}_\succ)$-chain. $\square$

Example 31. We continue the termination proof of Ex. 29. To finish the proof, we only have to solve the problem $(\{(14)\}, \varnothing, \{(11)\},\ \mathbf{m})$, i.e., $(\{\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(\mathsf{p}(\mathsf{s}(x)), \mathsf{p}(\mathsf{s}(y)))\}, \varnothing, \{\mathsf{p}(\mathsf{s}(x)) \to x\}, \mathbf{m})$. With the DP processor of

Thm. 30 this can easily be done, whereas it is difficult with the previous processors. As with Thm. 28, for efficiency it is often enough to restrict oneself to linear polynomial interpretations with coefficients from $\{0, 1\}$. We use the reduction pair based on the interpretation $\mathcal{P}ol$ with $\mathcal{P}ol(\mathsf{s}(t)) = \mathcal{P}ol(t) + 1$ and $\mathcal{P}ol(f(t_1, \ldots, t_n)) = \mathcal{P}ol(t_1) + \ldots + \mathcal{P}ol(t_n)$ for every other symbol $f \in \mathcal{F}$. In general, this reduction pair satisfies $l \succsim r$ whenever a rule $l \to r$ is non-duplicating and the number of s-symbols in l is greater than or equal to the number of s-symbols in r. By Thm. 30, those rules where l contains more s-symbols than r can be removed. So in our example, the rule (11) (i.e., $\mathsf{p}(\mathsf{s}(x)) \to x$) can be deleted. The resulting DP problem is $(\{\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(\mathsf{p}(\mathsf{s}(x)), \mathsf{p}(\mathsf{s}(y)))\}, \varnothing, \varnothing, \mathbf{m})$.

Note that now p is not a defined symbol anymore. Therefore, the dependency pair $\mathsf{MINUS}(\mathsf{s}(x), \mathsf{s}(y)) \to \mathsf{MINUS}(\mathsf{p}(\mathsf{s}(x)), \mathsf{p}(\mathsf{s}(y)))$ is no longer on a cycle of the dependency graph (since now terms starting with p cannot reduce to terms starting with s anymore). This can be detected by all existing estimations of dependency graphs (e.g., [1, 11]). Hence by the the dependency graph processor of Thm. 15, we obtain the empty set of DP problems, i.e., the termination proof is completed. This demonstrates an advantage of the DP framework, since now it is possible to apply techniques like the dependency graph repeatedly at any point during the termination proof. In contrast, in the classical dependency pair approach this technique was only applied at the beginning.

Note that similar to Thm. 27, in the classical dependency pair approach the processor of Thm. 30 would not be complete for innermost termination. This is shown by the example $\mathcal{P} = \{\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{a})\}$ and $\mathcal{Q} = \mathcal{R} = \{\mathsf{a} \to \mathsf{b}\}$. There is no infinite innermost chain, but if one uses an order where $\mathsf{a} \succ \mathsf{b}$, one can replace $\mathcal{R}$ by the empty set. Obviously, now one obtains an infinite innermost chain (i.e., an infinite $(\mathcal{P}, \varnothing, \varnothing, f)$-chain), but there is no infinite $(\mathcal{P}, \mathcal{R}, \varnothing, f)$-chain.

In [21, Thm. 23] we presented a first method within the dependency pair approach to remove rules of the TRS $\mathcal{R}$ that are not relevant for termination. The processors in Thm. 28 and 30 are significant improvements of this earlier technique: there one could only eliminate strictly decreasing (usable) rules as in Thm. 30, but it was impossible to remove non-usable rules and rules with non-usable symbols in their left-hand sides as in Thm. 28. This removal of non-usable rules is often crucial, since these rules often block the application of other important processors, as will be shown in the next section (cf. Ex. 33).

4.3 A DP Processor to Switch to Innermost Termination

The following processor replaces a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ with $\mathcal{Q} \subset \mathcal{R}$ by $(\mathcal{P}, \mathcal{R}, \mathcal{R}, \mathbf{m})$, i.e., under certain conditions it suffices to prove innermost instead of full termination. Proving innermost termination is significantly simpler: the dependency graph is smaller (Sect. 3.1), there are less restrictions when applying reduction pairs (Sect. 3.2), more dependency pair transformations are applicable (Sect. 3.3), one may directly remove all non-usable rules (Sect. 4.1), etc.

So while the previous processors modified the first and third components $\mathcal{P}$ and $\mathcal{R}$ in a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, this processor modifies the second component $\mathcal{Q}$. As shown in Lemma 12, while for $\mathcal{P}$ and $\mathcal{R}$ it is advantageous to remove rules, for $\mathcal{Q}$ it is advantageous to add rules.

In the classical dependency pair approach, a switch from termination to innermost termination was only possible if the *whole* TRS belongs to a class where innermost termination implies termination. An example for such a class are locally confluent overlay systems [10] or in particular, non-overlapping TRSs.

Instead, the following processor only requires local confluence for the rules $\mathcal{R}$ of the current DP problem. After applying the processors of Sect. 4.1 and 4.2, $\mathcal{R}$ is usually just a small subset of the whole TRS. Moreover, $\mathcal{R}$ does not have to be an overlay system. One only requires that $\mathcal{R}$ may not overlap with the pairs in $\mathcal{P}$. But the rules $\mathcal{R}$ themselves may have arbitrary critical pairs. This clearly extends the known classes where innermost termination implies termination.

Theorem 32 (DP Processor for Modular Non-Overlap Check). *The following processor is sound and complete. For a problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, Proc returns*

- $\{(\mathcal{P}, \mathcal{R}, \mathcal{R}, f)\}$, *if*
  - *for all $s \to t \in \mathcal{P}$, non-variable subterms of s do not unify with left-hand sides of rules from $\mathcal{R}$ (after variable renaming) and*
  - $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ *is locally confluent and*
  - $\mathcal{Q} \subseteq \mathcal{R}$ *and*
  - $f = \mathbf{m}$
- $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

Proof. Completeness follows from Lemma 12. For soundness, we prove that under the conditions of the first case in Thm. 32, every minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain $s_1 \to t_1, s_2 \to t_2, \ldots$ also results in a minimal $(\mathcal{P}, \mathcal{R}, \mathcal{R})$-chain. There is a substitution σ such that we have the following conditions for all i:

(a) $t_i\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} s_{i+1}\sigma$
(b) $s_i\sigma$ is in $\mathcal{Q}$-normal form
(c) $t_i\sigma$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$

By (a) and (c), $\sigma(x)$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ for all $x \in \mathcal{V}(s_2) \cup \mathcal{V}(s_3) \cup \ldots$ Since $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ is locally confluent, every $\sigma(x)$ has a unique normal form $\sigma(x)\!\downarrow$ w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by Newman's lemma. Let σ' be a substitution with $\sigma'(x) = \sigma(x)\!\downarrow$ for all $x \in \mathcal{V}(s_2) \cup \mathcal{V}(s_3) \cup \ldots$ and $\sigma'(x) = \sigma(x)$ otherwise. For all $i > 1$ we obtain:

(i) For all terms t we have $t\sigma \overset{\mathcal{Q}}{\to}{}^*_{\mathcal{R}} t\sigma'$.
(ii) If non-variable subterms of s_i do not unify with left-hand sides of rules from $\mathcal{R}$, then $s_i\sigma'$ is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.
(iii) A term is an $\mathcal{R}$-normal form iff it is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.

The observations (i) and (ii) are obvious. For (iii), the "only if" direction follows from $\overset{\mathcal{Q}}{\to}_{\mathcal{R}} \subseteq \to_{\mathcal{R}}$ (by Lemma 3). For the "if" direction, let t be a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ and assume that t contains $\mathcal{R}$-redexes. Let t' be an "innermost" $\mathcal{R}$-redex, i.e., all proper subterms of t' are in $\mathcal{R}$-normal form. Since $\mathcal{Q} \subseteq \mathcal{R}$, they are also in $\mathcal{Q}$-normal form. But then t' is also a redex w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$. This contradicts the assumption that t is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$.

Now we show that $s_2 \to t_2, s_3 \to t_3, \ldots$ is also a minimal $(\mathcal{P}, \mathcal{R}, \mathcal{R})$-chain. To this end, we use the substitution σ' instead of σ. For all $i > 1$ we have to prove:

(a$'$) $t_i\sigma' \overset{\mathcal{R}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma'$
(b$'$) $s_i\sigma'$ is in normal form w.r.t. $\mathcal{R}$
(c$'$) $t_i\sigma'$ is terminating w.r.t. $\overset{\mathcal{R}}{\to}_{\mathcal{R}}$

For (a$'$), note that $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma'$ by (a) and (i), where $s_{i+1}\sigma'$ is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (ii). Moreover, since $t_i\sigma$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ and since $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ is locally confluent, $s_{i+1}\sigma'$ is the *unique* normal form of $t_i\sigma$ w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by Newman's lemma. Since $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} t_i\sigma'$ by (i), $t_i\sigma'$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (c) and since $\overset{\mathcal{R}}{\to}_{\mathcal{R}} \subseteq \overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by Lemma 3, $t_i\sigma'$ is also terminating w.r.t. $\overset{\mathcal{R}}{\to}_{\mathcal{R}}$. Let w be a normal form of $t_i\sigma'$ w.r.t. $\overset{\mathcal{R}}{\to}_{\mathcal{R}}$. As $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} w$ and as w is also a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (iii), w must be the unique normal form $s_{i+1}\sigma'$. Hence, $t_i\sigma' \overset{\mathcal{R}}{\to}{}^{*}_{\mathcal{R}} s_{i+1}\sigma'$.

For (b$'$), $s_i\sigma'$ is a normal form w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (ii). Thus, (iii) implies that it is also a normal form w.r.t. $\mathcal{R}$.

For (c$'$), we have $t_i\sigma \overset{\mathcal{Q}}{\to}{}^{*}_{\mathcal{R}} t_i\sigma'$ by (i). Thus, $t_i\sigma'$ is terminating w.r.t. $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by (c). Hence, $t_i\sigma'$ is also terminating w.r.t. $\overset{\mathcal{R}}{\to}_{\mathcal{R}}$ since $\overset{\mathcal{R}}{\to}_{\mathcal{R}} \subseteq \overset{\mathcal{Q}}{\to}_{\mathcal{R}}$ by Lemma 3. $\square$

To apply this processor to a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$, one only has to check that $\mathcal{R}$ does not overlap with $\mathcal{P}$ and one has to prove local confluence of $\overset{\mathcal{Q}}{\to}_{\mathcal{R}}$. In practice, Thm. 32 is usually applied for $\mathcal{Q} = \varnothing$ (i.e., to switch from full to innermost termination). Then local confluence is equivalent to joinability of critical pairs and, for example, it suffices if $\mathcal{R}$ is non-overlapping. With such syntactic sufficient conditions for its applicability, Thm. 32 can easily be automated.

Example 33. $\mathcal{R}$ results from replacing the plus-rules (12) and (13) in Ex. 29 by

$$\mathsf{plus}(\mathsf{0}, y) \to y \quad (17) \qquad \mathsf{plus}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{plus}(y, \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(\mathsf{0})))) \quad (18)$$

and by adding the rule $\mathsf{div}(\mathsf{plus}(x, y), z) \to \mathsf{plus}(\mathsf{div}(x, z), \mathsf{div}(y, z))$.

To prove termination, now the dependency graph processor of Thm. 15 results in three DP problems (corresponding to the termination of div, minus, and plus). While the DP problems for div and minus can be solved as before, the DP problem $(\{(19)\}, \varnothing, \mathcal{R}, \mathbf{m})$ for plus cannot be handled with the existing processors if one uses base orders based on (quasi-)simplification orders or on (possibly negative) polynomial interpretations. In contrast, innermost termination of the TRS is easy to show. Here, (19) is the following dependency pair from Rule (18).

$$\mathsf{PLUS}(\mathsf{s}(x), y) \;\to\; \mathsf{PLUS}(y, \mathsf{minus}(\mathsf{s}(x), \mathsf{s}(\mathsf{0}))) \qquad (19)$$

Since $\mathcal{R}$ is non-duplicating, we can apply the usable rule processor of Thm. 28 with the same polynomial interpretation as in Ex. 29 (i.e., $\mathcal{P}ol(f(t_1, \ldots, t_n)) = \mathcal{P}ol(t_1) + \ldots + \mathcal{P}ol(t_n)$ for all $f \in \mathcal{F}$) and replace $\mathcal{R}$ by the usable rules, i.e., by the p- and minus-rules. Moreover, Rule (10) can also be deleted, since it contains the non-usable symbol plus on its left-hand side. Thus, the DP problem is transformed into $(\{(19)\}, \varnothing, \{(1), (2), (9), (11)\}, \mathbf{m})$. The TRS $\{(1), (2), (9), (11)\}$ is non-overlapping and thus, locally confluent. Moreover, no non-variable subterm of the left-hand side of (19) unifies with a left-hand side of these rules after variable renaming. Hence, we can apply the new DP processor of Thm. 32 to switch to an innermost termination problem.[10] To this end, we enlarge the second component of the DP problem from the empty set to $\{(1),(2),(9),(11)\}$. So now we have to solve the problem $(\{(19)\},\ \{(1),(2),(9),(11)\},\ \{(1),(2),(9),(11)\},\ \mathbf{m})$.

Note that the whole TRS $\mathcal{R}$ is overlapping and not locally confluent. Thus, it does not belong to a known class where innermost termination implies termination. Hence, this switch would not have been possible with existing results.

Since we now have an innermost termination problem, we may use the rewriting processor of Thm. 21 repeatedly to transform the pair (19) to $\mathsf{PLUS}(\mathsf{s}(x),y) \to \mathsf{PLUS}(y,x)$. Then the usable rule processor of Thm. 27 allows us to delete all rules, since they are not usable anymore. Hence, we obtain the problem $(\{(19)\}, \varnothing, \varnothing, \mathbf{m})$. By Thm. 19, now it suffices to solve the constraint $\mathsf{PLUS}(\mathsf{s}(x),y) \succ \mathsf{PLUS}(y,x)$ which is trivial by polynomial orders or recursive path orders.

By using Thm. 32 instead of removing rules as in Thm. 30, one also obtains an alternative proof for the DP problem of minus. As seen in Ex. 29, after removing all non-usable rules, one obtains the DP problem $(\{\mathsf{MINUS}(\mathsf{s}(x),\mathsf{s}(y)) \to \mathsf{MINUS}(\mathsf{p}(\mathsf{s}(s)),\mathsf{p}(\mathsf{s}(y)))\}, \varnothing, \{\mathsf{p}(\mathsf{s}(x)) \to x\}, \mathbf{m})$. Since the rule is locally confluent and does not overlap with the remaining dependency pair, now one can use Thm. 32 to switch to the innermost case and easily solve the DP problem.

Ex. 31 already showed the advantages of re-computing the dependency graph later during the termination proof. Now we have demonstrated that it can also be beneficial to use dependency pair transformations after some other processors have been applied (like the usable rule processor of Thm. 28 and the processor for the modular non-overlap check of Thm. 32). In contrast, in the classical dependency pair approach, transformations could only be used in the very beginning of a proof, but not after deleting rules, pairs, etc. This demonstrates an advantage of the new modular DP framework.

Note that by Thm. 32, the observation that innermost termination implies termination for locally confluent overlay systems is obtained as a corollary. While the original proof for this important result of Gramlich [10] is not at all trivial, the proof of Thm. 32 is quite simple. While there already exists another easy proof [18], in this way we get an alternative simple proof for this crucial result.

Corollary 34 (Thm. 3.23. in [10]). *Let $\mathcal{R}$ be a locally confluent overlay system. If $\mathcal{R}$ is innermost terminating, then it is terminating.*

[10] Deleting all non-usable rules with Thm. 28 is often needed to enable an application of the modular non-overlap check from Thm. 32 afterwards. In this example, the non-usable rules are not locally confluent due to the new additional div-rule and there is no other processor which can remove these rules (if one uses reduction pairs based on (quasi-)simplification orders). Therefore, if one would replace the new processor of Thm. 28 with our previous related technique in [21, Thm. 23], then one cannot switch to an innermost termination problem with Thm. 32 anymore and thus, the termination proof would fail.

Proof. $\mathcal{R}$ terminates if the DP problem $(DP(\mathcal{R}), \varnothing, \mathcal{R}, \mathbf{m})$ is finite by Thm. 8. For overlay systems, no non-variable subterms of left-hand sides from $DP(\mathcal{R})$ unify with variable-renamed left-hand sides from $\mathcal{R}$. Thus by Thm. 32, it is sufficient if the DP problem $(DP(\mathcal{R}), \mathcal{R}, \mathcal{R}, \mathbf{m})$ is finite. This follows from innermost termination (i.e., $\mathcal{R}$-termination) of $\mathcal{R}$ by Thm. 8. □

However, Thm. 32 improves Gramlich's result significantly. Even if $\mathcal{R}$ is not a locally confluent overlay system, by representing the termination task as a DP problem, one may first apply other processors to obtain sub-problems $(\mathcal{P}', \varnothing, \mathcal{R}', \mathbf{m})$ where $\mathcal{R}'$ is indeed locally confluent and does not overlap with $\mathcal{P}'$. For these sub-problems, one can now switch to the innermost case, whereas Gramlich's result would not be applicable. As demonstrated in Ex. 33, this switch can be crucial for the termination proof.

5 DP Processors From Other Techniques

Now we show how to integrate existing termination techniques in the DP framework. In this way, these techniques can benefit from other DP processors which were applied before. This increases their applicability and power considerably.

Definition 35 (Termination Technique). *A* termination technique TT *maps TRSs to TRSs such that termination of* $TT(\mathcal{R})$ *implies termination of* $\mathcal{R}$.

Note that the above definition captures both transformational techniques (which transform a TRS $\mathcal{R}$ into a new TRS $\mathcal{R}'$ whose termination is sufficient for termination of $\mathcal{R}$) and traditional techniques which simply give a "yes" or "no" answer when trying to prove termination. For those techniques, we would define $TT(\mathcal{R}) = \varnothing$ if termination of $\mathcal{R}$ can be proved and $TT(\mathcal{R}) = \mathcal{R}$, otherwise.

Now at any point during the termination proof, instead of solving a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ one can use a termination technique and verify termination of $TT(\mathcal{P} \cup \mathcal{R})$. To this end, one should of course use the DP framework again. Hence, we now define a processor to integrate arbitrary termination techniques.

Theorem 36 (DP Processor for Termination Techniques). *Let* TT *be a termination technique and let Proc be a DP processor with* $Proc((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)) = \{(DP(\mathcal{R}'), \varnothing, \mathcal{R}', \mathbf{m})\}$ *where* $TT(\mathcal{P} \cup \mathcal{R}) = \mathcal{R}'$. *Then Proc is sound.*

Proof. Obvious from Thm. 8. □

It is easy to show that if $\mathcal{P}$'s rules have tuple symbols on their root positions,[11] if $\mathcal{Q} = \varnothing$, and if TT is "complete" (i.e., $TT(\mathcal{P} \cup \mathcal{R})$ terminates iff $\mathcal{P} \cup \mathcal{R}$ terminates), then the above processor is also complete.

Of course, if a termination technique TT is capable of handling $\mathcal{Q}$-restricted rewriting, then one could easily take this into account: Now TT would be applied to *pairs* $(\mathcal{Q}, \mathcal{R})$ and return a pair $(\mathcal{Q}', \mathcal{R}')$ such that $\mathcal{Q}'$-termination of $\mathcal{R}'$ is sufficient for $\mathcal{Q}$-termination of $\mathcal{R}$. Hence, a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ may now be transformed into $(DP(\mathcal{R}'), \mathcal{Q}', \mathcal{R}', \mathbf{m})$ where $TT(\mathcal{Q}, \mathcal{P} \cup \mathcal{R}) = (\mathcal{Q}', \mathcal{R}')$.

[11] More precisely, one requires $\text{root}(s), \text{root}(t) \in \mathcal{F}'$ for all $s \to t \in \mathcal{P}$ and some $\mathcal{F}' \subseteq \mathcal{F}$, while $\mathcal{F}'$-symbols do not occur anywhere else in $\mathcal{P}$ and they also do not occur in $\mathcal{R}$.

To improve the applicability of termination techniques, one may pre-process a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$ before. The reason is that there exist powerful termination techniques which can only be applied to subclasses of TRSs. For example, the *RFC matchbounds* technique of [5] or the method of *string reversal* only operate on *string rewrite systems* (SRSs), i.e., on TRSs where all occurring function symbols have arity 1. To make such techniques applicable, one may perform a pre-processing step which transforms a DP problem with non-unary symbols into a problem on SRSs.

Note that the processors of the previous sections never change the flag f when transforming a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$. Thus, when starting with the initial DP problem $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$, all resulting problems have the flag $f = \mathbf{m}$. However, for the pre-processing mentioned above, we will also introduce processors which modify the flag by changing it to $\mathbf{a}$. In other words, while for the original DP problem it was sufficient to prove absence of *minimal* chains, for the problems resulting from these processors one has to prove absence of *all arbitrary* chains.

Applying such processors usually has the disadvantage that afterwards, many other processors are no longer applicable, since they only work on DP problems with $f = \mathbf{m}$, i.e., on problems where one only examines minimal chains. However, if one re-builds the dependency pairs afterwards as in Thm. 36, then any DP problem is changed back again into a problem with the flag $f = \mathbf{m}$. For this reason, if one has obtained a DP problem with the flag $\mathbf{a}$, it can even be useful to apply Thm. 36 with the "empty" termination technique where $TT(\mathcal{P} \cup \mathcal{R}) = \mathcal{P} \cup \mathcal{R}$, since afterwards it is again sufficient to regard only *minimal* chains.

We now introduce two processors which are very useful as pre-processing steps before applying termination techniques. The first processor $Proc_{\mathcal{U}}$ removes all non-usable rules from a reduction pair (without checking any further conditions as in Thm. 28). This corresponds to the usable rule processor of Thm. 27, but in contrast to Thm. 27 the new processor is applicable for arbitrary $\mathcal{Q}$, not just for $\mathcal{Q} \supseteq \mathcal{R}$. However, one now has to add $\mathcal{C}_\varepsilon = \{\mathsf{c}(x, y) \to x, \mathsf{c}(x, y) \to y\}$ to the usable rules. Moreover, we introduce a processor which allows us to apply argument filterings to the rules and pairs in a DP problem. Here, we define $\pi(\mathcal{R}) = \{\pi(l) \to \pi(r) \mid l \to r \in \mathcal{R}\}$ for any TRS $\mathcal{R}$.

Theorem 37 (Pre-Processing DP Processors). *The following DP processors $Proc_{\mathcal{U}}$ and $Proc_\pi$ are sound.*

- *For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, $Proc_{\mathcal{U}}$ returns*
  - $\{\, (\mathcal{P},\ \varnothing,\ \mathcal{U}(\mathcal{P}, \mathcal{R}) \cup \mathcal{C}_\varepsilon,\ \mathbf{a})\,\}$, *if* $f = \mathbf{m}$
  - $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*
- *Let π be an argument filtering. For a DP problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$, $Proc_\pi$ returns*
  - $\{(\pi(\mathcal{P}), \varnothing, \pi(\mathcal{R}), \mathbf{a})\}$, *if* $\pi(\mathcal{P})$ *and* $\pi(\mathcal{R})$ *are TRSs*
  - $\{(\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)\}$, *otherwise*

Proof. $Proc_{\mathcal{U}}$ is sound since every minimal $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain is a (not necessarily minimal) $(\mathcal{P},\ \varnothing,\ \mathcal{U}(\mathcal{P}, \mathcal{R}) \cup \mathcal{C}_\varepsilon)$-chain, cf. [21, Thm. 17].[12]

[12] The extension from ordinary to $\mathcal{Q}$-restricted rewriting is again straightforward.

We now show soundness of Proc_π: If $s_1 \to t_1, s_2 \to t_2, \ldots$ is an infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain, then there is a substitution σ with $t_i\sigma \to_{\mathcal{R}}^* s_{i+1}\sigma$ for all i. Hence, $\pi(t_i)\sigma_\pi \to_{\pi(\mathcal{R})}^* \pi(s_{i+1})\sigma_\pi$ for the substitution σ_π with $\sigma_\pi(x) = \pi(\sigma(x))$ for all $x \in \mathcal{V}$. So $\pi(s_1) \to \pi(t_1), \pi(s_2) \to \pi(t_2), \ldots$ is an infinite $(\pi(\mathcal{P}), \varnothing, \pi(\mathcal{R}))$-chain. $\square$

The following example shows that the processors are incomplete, even if $\mathcal{Q} = \varnothing$.

Example 38. Let $\mathcal{R} = \{\mathsf{f}(\mathsf{a}, \mathsf{b}, x) \to \mathsf{f}(x, x, x)\}$ [22]. $\mathcal{R}$ is terminating and thus, the resulting DP problem $(\{\mathsf{F}(\mathsf{a}, \mathsf{b}, x) \to \mathsf{F}(x, x, x)\}, \varnothing, \mathcal{R}, \mathbf{m})$ is not infinite. However, by the processor $\mathit{Proc}_{\mathcal{U}}$ we obtain the infinite DP problem $(\{\mathsf{F}(\mathsf{a}, \mathsf{b}, x) \to \mathsf{F}(x, x, x)\}, \varnothing, \mathcal{C}_\varepsilon, \mathbf{a})$, since now the instantiated right-hand side of the dependency pair reduces to the instantiated left-hand side if x is substituted by $\mathsf{c}(\mathsf{a}, \mathsf{b})$.

Incompleteness of Proc_π is shown by $(\{\mathsf{F}(\mathsf{a}) \to \mathsf{F}(\mathsf{b})\}, \varnothing, \varnothing, f)$ which is not infinite, but the filtering $\pi(\mathsf{F}(x)) = \mathsf{F}$ produces the infinite problem $(\{\mathsf{F} \to \mathsf{F}\}, \varnothing, \varnothing, \mathbf{a})$.

Now we demonstrate why the argument filtering processor Proc_π has to set the flag f to $\mathbf{a}$, i.e., we show why one has to prove absence of *arbitrary* (possibly non-minimal) chains after the filtering. For the processor $\mathit{Proc}_{\mathcal{U}}$ it is currently open whether changing f to $\mathbf{a}$ is really needed for soundness. In other words, it is not known whether a processor which transforms $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ into $\{\,(\,\mathcal{P},\ \varnothing,\ \mathcal{U}(\mathcal{P}, \mathcal{R}) \cup \mathcal{C}_\varepsilon,\ \mathbf{m})\,\}$ would be sound.

Example 39. An argument filtering processor which replaces $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ by $\{(\pi(\mathcal{P}), \varnothing, \pi(\mathcal{R}), \mathbf{m})\}$ is not sound, even if $\mathcal{Q} = \varnothing$ and if π does not modify the function symbols of $\mathcal{R}$: The DP problem $(\mathcal{P}, \varnothing, \mathcal{R}, \mathbf{m})$ with $\mathcal{P} = \{\mathsf{F}(\mathsf{g}(\mathsf{s}(\mathsf{a}))) \to \mathsf{F}(\mathsf{g}(\mathsf{s}(\mathsf{a})))\}$ and $\mathcal{R} = \{\mathsf{g}(\mathsf{a}) \to \mathsf{g}(\mathsf{a})\}$ has an infinite minimal chain, but if one filters $\mathcal{P}$ with $\pi(\mathsf{s}(x)) = x$, then there is no infinite *minimal* chain anymore. The reason is that then the filtered right-hand side $\mathsf{F}(\mathsf{g}(\mathsf{a}))$ of the pair in $\pi(\mathcal{P})$ is no longer terminating w.r.t. $\mathcal{R}$.

However, it is easy to show that if $\mathcal{P}$'s rules have tuple symbols on their root positions (as in Footnote 11) and if the filtering π only modifies these tuple symbols, then one could define $\mathit{Proc}_\pi((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)) = \{(\pi(\mathcal{P}), \mathcal{Q}, \mathcal{R}, f)\}$ if $\pi(\mathcal{P})$ is a TRS. In other words, then both the TRS $\mathcal{Q}$ and the flag f could remain unchanged and the resulting processor Proc_π would still be sound. (This observation can also be extended to more general forms of $\mathcal{P}$.)

The next example shows that replacing $\mathcal{Q}$ by $\varnothing$ after the filtering is needed for the soundness of Proc_π, even if π does not modify any symbols of $\mathcal{Q}$.

Example 40. Consider the problem $(\mathcal{P}, \mathcal{Q}, \mathcal{R}, \mathbf{m})$ with $\mathcal{P} = \{\mathsf{F}(x) \to \mathsf{F}(\mathsf{g}(\mathsf{s}(x)))\}$, $\mathcal{Q} = \{\mathsf{g}(\mathsf{g}(x)) \to x\}$, and $\mathcal{R} = \varnothing$. Clearly, there is a (minimal) infinite $(\mathcal{P}, \mathcal{Q}, \mathcal{R})$-chain as $\mathsf{F}(x) \to \mathsf{F}(\mathsf{g}(\mathsf{s}(x)))$, $\mathsf{F}(\mathsf{g}(\mathsf{s}(x))) \to \mathsf{F}(\mathsf{g}(\mathsf{s}(\mathsf{g}(\mathsf{s}(x)))))$, $\ldots$, are instantiations of the pair in $\mathcal{P}$ which are in $\mathcal{Q}$-normal form. However, if one uses the argument filtering with $\pi(\mathsf{s}(x)) = x$, we would replace $\mathcal{P}$ by $\pi(\mathcal{P}) = \{\mathsf{F}(x) \to \mathsf{F}(\mathsf{g}(x))\}$ whereas $\mathcal{Q}$ would remain unchanged. Now there is no infinite $(\pi(\mathcal{P}), \mathcal{Q}, \mathcal{R})$-chain anymore, since terms of the form $\mathsf{F}(\mathsf{g}(\mathsf{g}(\ldots)))$ are not in $\mathcal{Q}$-normal form.

As a larger last example, we now demonstrate the benefits of Thm. 37 for the integration of *string reversal* into the DP framework. As mentioned before, string reversal is a transformational termination technique which is only applicable to SRSs. The *reversal* t^{-1} of a term t with only unary symbols is obtained by reversing the order of its function symbols (e.g., the reversal of $f(g(h(x)))$ is $h(g(f(x)))$). For a TRS $\mathcal{R}$, its reversal is $\mathcal{R}^{-1} = \{l^{-1} \to r^{-1} \mid l \to r \in \mathcal{R}\}$. It is well known that an SRS $\mathcal{R}$ is terminating iff $\mathcal{R}^{-1}$ is terminating. Thus, we can use the termination technique TT_{REV} with $TT_{REV}(\mathcal{R}) = \mathcal{R}^{-1}$ if $\mathcal{R}$ is an SRS and $TT_{REV}(\mathcal{R}) = \mathcal{R}$, otherwise.

Example 41. The TRS $\mathcal{R}$ contains the following rules together with the plus-rules (12) and (13) from Ex. 22. Here, $\mathsf{mult}(x, y, z)$ computes $x * y + z$.

$$\mathsf{times}(x, y) \to \mathsf{mult}(x, y, \mathsf{0}) \quad (20) \qquad \mathsf{p}(\mathsf{s}(\mathsf{0})) \to \mathsf{0} \quad (24)$$
$$\mathsf{mult}(\mathsf{0}, y, z) \to z \quad (21) \qquad \mathsf{p}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{p}(\mathsf{s}(x))) \quad (25)$$
$$\mathsf{mult}(\mathsf{s}(x), y, z) \to \mathsf{mult}(\mathsf{p}(\mathsf{s}(x)), y, \mathsf{plus}(y, z)) \quad (22)$$
$$\mathsf{times}(\mathsf{plus}(x, y), z) \to \mathsf{plus}(\mathsf{times}(x, z), \mathsf{times}(y, z)) \quad (23)$$

By the processor based on the dependency graph of Thm. 15, we obtain four DP problems corresponding to the termination of p, plus, times, and mult. The first three are easy to handle, but the problem $(\{\mathsf{MULT}(\mathsf{s}(x), y, z) \to \mathsf{MULT}(\mathsf{p}(\mathsf{s}(x)), y, \mathsf{plus}(y, z))\}, \varnothing, \mathcal{R}, \mathbf{m})$ cannot be solved by the processors of the previous sections if one uses reduction pairs based on (quasi-)simplification orders.

However, by applying the new processors of this section, we can transform this DP problem into an SRS and apply the termination technique "string reversal". Afterwards, it can easily be solved. We first apply the processor $Proc_{\mathcal{U}}$ of Thm. 37 to remove the non-usable rules for times and mult which results in $(\{\mathsf{MULT}(\mathsf{s}(x), y, z) \to \mathsf{MULT}(\mathsf{p}(\mathsf{s}(x)), y, \mathsf{plus}(y, z))\}, \varnothing, \{(12), (13), (24), (25)\} \cup \mathcal{C}_\varepsilon, \mathbf{a})$. Next we eliminate the second and third argument of MULT by the argument filtering processor $Proc_\pi$ of Thm. 37 and replace the dependency pair by $\mathsf{MULT}(\mathsf{s}(x)) \to \mathsf{MULT}(\mathsf{p}(\mathsf{s}(x)))$. Now the processor for removal of rules from Thm. 30 is used with a polynomial interpretation where $\mathcal{P}ol(\mathsf{c}(x, y)) = x + y + 1$, $\mathcal{P}ol(\mathsf{plus}(x, y)) = 2\,x + y + 1$, $\mathcal{P}ol(\mathsf{p}(x)) = x$, and $\mathcal{P}ol(\mathsf{s}(x)) = x + 1$. Then the rules (12), (13), and (24) are strictly decreasing and can be removed, i.e., we result in $(\{\mathsf{MULT}(\mathsf{s}(x)) \to \mathsf{MULT}(\mathsf{p}(\mathsf{s}(x)))\}, \varnothing, \{\mathsf{p}(\mathsf{s}(\mathsf{s}(x))) \to \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))\}, \mathbf{a})$.

Note that we have obtained a DP problem containing only symbols of arity 1. Therefore, we can apply the termination technique TT_{REV} and try to prove termination of the reversed TRS $\mathcal{R}' = \{\mathsf{s}(\mathsf{MULT}(x)) \to \mathsf{s}(\mathsf{p}(\mathsf{MULT}(x))), \mathsf{s}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{s}(\mathsf{p}(\mathsf{s}(x)))\}$. The resulting DP problem $(DP(\mathcal{R}'), \varnothing, \mathcal{R}', \mathbf{m})$ is easy to solve: the dependency graph processor yields $(\{\mathsf{S}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{S}(x)\}, \varnothing, \mathcal{R}', \mathbf{m})$ and by the usable rule processor we can delete all rules of $\mathcal{R}'$ and also the remaining pair $\mathsf{S}(\mathsf{s}(\mathsf{p}(x))) \to \mathsf{S}(x)$ which contains the non-usable symbol p on the left-hand side.

To summarize, the advantage of integrating termination techniques like string reversal into the DP framework is that they can solve certain parts of the termination proof, whereas different techniques are used for other parts (e.g., because these parts do not correspond to an SRS). Moreover, as shown in the above example, since one can modify DP problems by argument filterings, one can also apply SRS-termination techniques for DP problems which originally contained non-unary function symbols. So in general, the applicability, modularity, and power of existing termination techniques is increased significantly by the integration into the DP framework. While Thm. 36 shows how to integrate arbitrary techniques, certain termination techniques may also be adapted in order to operate on DP problems directly instead of TRSs. Then instead of integrating them with Thm. 36, they could be directly used as processors in the DP framework.

6 Strategies for the Dependency Pair Framework

The DP framework allows us to combine DP processors in a completely modular and flexible way. A system for termination proofs with the DP framework tries to prove $\mathcal{Q}$-termination of $\mathcal{R}$ for two TRSs $\mathcal{Q}$ and $\mathcal{R}$. It starts with the initial DP problem $(DP(\mathcal{R}), \mathcal{Q}, \mathcal{R}, \mathbf{m})$ and then constructs a tree as in Cor. 10. As long as there is a DP problem d left, it chooses a DP processor *Proc* and computes $Proc(d)$. If $Proc(d) = \mathsf{no}$, the proof is stopped. Then the system returns "no" if *Proc* and all processors used on the path from the initial DP problem to d are complete and otherwise it returns "maybe". If $Proc(d) \neq \mathsf{no}$, then d is replaced by $Proc(d)$ and the procedure continues. As soon as there are no DP problems left anymore, the system returns "yes". To avoid non-termination of the system, it can also abort the proof after some time limit and return "maybe". This algorithm and a large number of DP processors (including those presented in this paper) have been implemented in our automated termination tool AProVE [9] which can be obtained from `http://www-i2.informatik.rwth-aachen.de/AProVE`.

To obtain a powerful system for termination proofs, a main challenge is to develop strategies to decide which DP processor should be applied to a DP problem d. A general heuristic is to apply fast processors first and to use more powerful slower processors later on in order to handle those problems which cannot already be solved by fast processors. The strategy of AProVE is to select the first DP processor *Proc* from the following list which satisfies $Proc(d) \neq \{d\}$.

1. DP processor based on the dependency graph (Thm. 15)
2. DP processor based on usable rules (Thm. 27)
3. DP processor for modular non-overlap check (Thm. 32)
4. Narrowing, rewriting, and instantiation processors in "safe" cases [8] where they "simplify" the DP problem (Thm. 21)
5. DP processor based on usable rules and reduction pairs (Thm. 28)
6. DP processor based on rule removal (Thm. 30)
7. DP processor based on red. pairs: linear polynomials over $\{0, 1\}$ (Thm. 19)
8. DP processor for non-termination analysis[13]
9. Narrowing, rewriting, and instantiation (up to a certain limit) (Thm. 21)

[13] A simple sound and complete DP processor *Proc* for non-termination analysis is the following: $Proc((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)) = \mathsf{no}$ if $\mathcal{P}$ contains a rule of the form $s \to s$ where s is in $\mathcal{Q}$-normal form. Otherwise, $Proc((\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)) = (\mathcal{P}, \mathcal{Q}, \mathcal{R}, f)$. Obviously, this processor can be improved to detect more cases of non-termination.

10. DP processor based on red. pairs: linear polynomials over $\{0, 1, 2\}$ (Thm. 19)
11. DP processor based on reduction pairs: lexicographic path orders (Thm. 19)
12. DP processor based on reduction pairs: non-linear polynomials (Thm. 19)
13. DP processor based on string reversal (Thm. 36)

Of course, one can also use different strategies for different forms of TRSs. For example, if the underlying TRS is an SRS, AProVE uses a slightly different strategy, which also includes DP processors based on other techniques like RFC matchbounds [5] and semantic labelling [24], cf. Sect. 5.

Due to the DP framework (with the above strategies), AProVE was the most powerful system at the competition of termination tools at the *7th International Workshop on Termination* (WST '04). Here, the tools were tested on 936 examples from the *termination problem data base* (TPDB) [20], a collection of termination problems from several sources and different areas of computer science. This demonstrates that the DP framework is indeed very well suited for automation and for application in practice.

7 Conclusion and Future Work

We introduced the dependency pair framework for termination proofs (Sect. 2) which generalizes the classical dependency pair approach into a general basis for automated termination proofs. We first showed how to formulate the existing components of the dependency pair approach as DP processors within this framework (Sect. 3). Now these components can be applied at any time during the termination proof and their applicability conditions only concern the current DP problem, not the whole TRSs. Afterwards, we developed several new DP processors to simplify termination problems (Sect. 4) and we showed how to integrate arbitrary existing termination techniques into the DP framework (Sect. 5). For all processors, we also investigated their completeness which allows us to use them also when proving non-termination. As demonstrated in Sect. 6, this framework is indeed suitable for automation in practice. For future work, we see two main directions of research:

While there already exist several powerful DP processors, these processors are not yet sufficient to handle all termination problems occurring in practice. Therefore, one important topic for further work is the improvement of the existing DP processors and the development of new DP processors which are particularly fast or particularly powerful for certain classes of DP problems.

The other important line of research is the development of new strategies to decide which DP processor should be applied next on a particular DP problem. We have presented such a strategy in Sect. 6, but depending on the area of application, other strategies can be advantageous.

To summarize, in this paper we have shown that the combination of techniques for termination proofs within the dependency pair framework leads to a very modular, flexible, and powerful approach. Therefore, we think that this framework is particularly suitable as a basis for future research on automated termination proving.

References

1. T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236:133–178, 2000.
2. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
3. N. Dershowitz. Termination of rewriting. *J. Symb. Computation*, 3:69–116, 1987.
4. N. Dershowitz. Termination by abstraction. In *Proc. ICLP '04*, LNCS 3132, pages 1–18, 2004.
5. A. Geser, D. Hofbauer, and J. Waldmann. Match-bounded string rewriting systems. In *Proc. MFCS '03*, LNCS 2747, pages 449–459, 2003.
6. J. Giesl and T. Arts. Verification of Erlang processes by dependency pairs. *Appl. Algebra in Engineering, Communication and Computing*, 12(1,2):39–72, 2001.
7. J. Giesl, T. Arts, and E. Ohlebusch. Modular termination proofs for rewriting using dependency pairs. *Journal of Symbolic Computation*, 34(1):21–58, 2002.
8. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Improving dependency pairs. In *Proc. LPAR '03*, LNAI 2850, pages 165–179, 2003.
9. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVE. In *Proc. RTA '04*, LNCS 3091, pages 210–220, 2004.
10. B. Gramlich. Abstract relations between restricted termination and confluence properties of rewrite systems. *Fundamenta Informaticae*, 24:3–23, 1995.
11. N. Hirokawa and A. Middeldorp. Automating the dependency pair method. In *Proc. CADE '03*, LNAI 2741, pages 32–46, 2003. Full version to appear in *Information and Computation*.
12. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *Proc. RTA '04*, LNCS 3091, pages 249–268, 2004.
13. N. Hirokawa and A. Middeldorp. Polynomial interpretations with negative coefficients. In *Proc. AISC '04*, LNAI 3249, pages 185–198, 2004.
14. S. Kamin and J. J. Lévy. Two generalizations of the recursive path ordering. Unpublished Manuscript, University of Illinois, IL, USA, 1980.
15. D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297, 1970.
16. K. Kusakari, M. Nakamura, and Y. Toyama. Argument filtering transformation. In *Proc. PPDP '99*, LNCS 1702, pages 48–62, 1999.
17. D. Lankford. On proving term rewriting systems are Noetherian. Technical Report MTP-3, Louisiana Technical University, Ruston, LA, USA, 1979.
18. Aart Middeldorp. A simple proof to a result of Bernhard Gramlich. Presented at the 5th Japanese Term Rewriting Meeting, Tsukuba, 1994. Available from `http://informatik.uibk.ac.at/users/ami/research/papers/bg.pdf`.
19. J. Steinbach. Simplification orderings: History of results. *Fundamenta Informaticae*, 24:47–87, 1995.
20. Termination Problem Data Base (TPDB). Available from `http://www.lri.fr/~marche/wst2004-competition/tpdb.html`.
21. R. Thiemann, J. Giesl, and P. Schneider-Kamp. Improved modular termination proofs using dependency pairs. In *Proc. IJCAR '04*, LNAI 3097, pages 75–90, 2004.
22. Y. Toyama. Counterexamples to the termination for the direct sum of term rewriting systems. *Information Processing Letters*, 25:141–143, 1987.
23. X. Urbain. Modular and incremental automated termination proofs. *Journal of Automated Reasoning*, 32(4): 315–355, 2004.
24. H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24:89–105, 1995.