Eliminating Dummy Elimination

Jürgen Giesl[1] and Aart Middeldorp[2]

[1] Computer Science Department
University of New Mexico, Albuquerque, NM 87131, USA
giesl@cs.unm.edu

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

Abstract. This paper is concerned with methods that automatically prove termination of term rewrite systems. The aim of dummy elimination, a method to prove termination introduced by Ferreira and Zantema, is to transform a given rewrite system into a rewrite system whose termination is easier to prove. We show that dummy elimination is subsumed by the more recent dependency pair method of Arts and Giesl. More precisely, if dummy elimination succeeds in transforming a rewrite system into a so-called simply terminating rewrite system then termination of the given rewrite system can be directly proved by the dependency pair technique. Even stronger, using dummy elimination as a preprocessing step to the dependency pair technique does not have any advantages either. We show that to a large extent these results also hold for the argument filtering transformation of Kusakari *et al.*

1 Introduction

Traditional methods to prove termination of term rewrite systems are based on simplification orders, like polynomial interpretations [6, 12, 17], the recursive path order [7, 14], and the Knuth-Bendix order [9, 15]. However, the restriction to simplification orders represents a significant limitation on the class of rewrite systems that can be proved terminating. Indeed, there are numerous important and interesting rewrite systems which are not *simply terminating*, i.e., their termination cannot be proved by simplification orders. Transformation methods (e.g. [5, 10, 11, 16, 18, 20–22]) aim to prove termination by transforming a given term rewrite system into a term rewrite system whose termination is easier to prove. The success of such methods has been measured by how well they transform non-simply terminating rewrite systems into simply terminating rewrite systems, since simply terminating systems were the only ones where termination could be established automatically.

In recent years, the dependency pair technique of Arts and Giesl [1, 2] emerged as the most powerful automatic method for proving termination of rewrite systems. For any given rewrite system, this technique generates a set of constraints which may then be solved by standard simplification orders. In this way, the

power of traditional termination proving methods has been increased significantly, i.e., the class of systems where termination is provable mechanically by the dependency pair technique is much larger than the class of simply terminating systems. In light of this development, it is no longer sufficient to base the claim that a particular transformation method is successful on the fact that it may transform non-simply terminating rewrite systems into simply terminating ones. In this paper we compare two transformation methods, dummy elimination [11] and the argument filtering transformation [16], with the dependency pair technique. With respect to dummy elimination we obtain the following results:

1. If dummy elimination transforms a given rewrite system $\mathcal{R}$ into a simply terminating rewrite system $\mathcal{R}'$, then the termination of $\mathcal{R}$ can also be proved by the most basic version of the dependency pair technique.
2. If dummy elimination transforms a given rewrite system $\mathcal{R}$ into a *DP simply terminating* rewrite system $\mathcal{R}'$, i.e., the termination of $\mathcal{R}'$ can be proved by a simplification order in combination with the dependency pair technique, then $\mathcal{R}$ is also DP simply terminating.

These results are constructive in the sense that the constructions in the proofs are solely based on the termination proof of $\mathcal{R}'$. This shows that proving termination of $\mathcal{R}$ directly by dependency pairs is never more difficult than proving termination of $\mathcal{R}'$. The second result states that dummy elimination is useless as a preprocessing step to the dependency pair technique. Not surprisingly, the reverse statements do not hold. In other words, as far as automatic termination proofs are concerned, dummy elimination is no longer needed.

The recent argument filtering transformation of Kusakari, Nakamura, and Toyama [16] can be viewed as an improvement of dummy elimination by incorporating ideas of the dependency pair technique. We show that the first result above also holds for the argument filtering transformation. The second result does not extend in its full generality, but we show that under a suitable restriction on the argument filtering applied in the transformation of $\mathcal{R}$ to $\mathcal{R}'$, DP simple termination of $\mathcal{R}'$ also implies DP simple termination of $\mathcal{R}$.

The remainder of the paper is organized as follows. In the next section we briefly recall some definitions and results pertaining to termination of rewrite systems and in particular, the dependency pair technique. In Section 3 we relate the dependency pair technique to dummy elimination. Section 4 is devoted to the comparison of the dependency pair technique and the argument filtering transformation. We conclude in Section 5.

2 Preliminaries

An introduction to term rewrite systems (TRSs) can be found in [4], for example. We first introduce the dependency pair technique. Our presentation combines features of [2, 13, 16]. Apart from the presentation, all results stated below are due to Arts and Giesl. We refer to [2, 3] for motivations and proofs. Let $\mathcal{R}$ be a (finite) TRS over a signature $\mathcal{F}$. As usual, all root symbols of left-hand

sides of rewrite rules are called *defined*, whereas all other function symbols are *constructors*. Let $\mathcal{F}^\sharp$ denote the union of $\mathcal{F}$ and $\{f^\sharp \mid f \text{ is a defined symbol of } \mathcal{R}\}$ where $f^\sharp$ has the same arity as f. Given a term $t = f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with f defined, we write $t^\sharp$ for the term $f^\sharp(t_1, \ldots, t_n)$. If $l \to r \in \mathcal{R}$ and t is a subterm of r with defined root symbol then the rewrite rule $l^\sharp \to t^\sharp$ is called a *dependency pair* of $\mathcal{R}$. The set of all dependency pairs of $\mathcal{R}$ is denoted by $\mathrm{DP}(\mathcal{R})$. In examples we often write F for $\mathsf{f}^\sharp$.

For instance, consider the following well-known one-rule TRS $\mathcal{R}$ from [8]:

$$\mathsf{f}(\mathsf{f}(x)) \to \mathsf{f}(\mathsf{e}(\mathsf{f}(x))) \tag{1}$$

Here f is defined, e is a constructor, and $\mathrm{DP}(\mathcal{R})$ consists of the two dependency pairs

$$\mathsf{F}(\mathsf{f}(x)) \to \mathsf{F}(\mathsf{e}(\mathsf{f}(x))) \qquad \mathsf{F}(\mathsf{f}(x)) \to \mathsf{F}(x)$$

An *argument filtering* [2] for a signature $\mathcal{F}$ is a mapping π that associates with every n-ary function symbol an argument position $i \in \{1, \ldots, n\}$ or a (possibly empty) list $[i_1, \ldots, i_m]$ of argument positions with $1 \leqslant i_1 < \cdots < i_m \leqslant n$. The signature $\mathcal{F}_\pi$ consists of all function symbols f such that $\pi(f)$ is some list $[i_1, \ldots, i_m]$, where in $\mathcal{F}_\pi$ the arity of f is m. Every argument filtering π induces a mapping from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}_\pi, \mathcal{V})$, also denoted by π:

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

Thus, an argument filtering is used to replace function symbols by one of their arguments or to eliminate certain arguments of function symbols. For example, if $\pi(\mathsf{f}) = \pi(\mathsf{F}) = [1]$ and $\pi(\mathsf{e}) = 1$, then we have $\pi(\mathsf{F}(\mathsf{e}(\mathsf{f}(x)))) = \mathsf{F}(\mathsf{f}(x))$. However, if we change $\pi(\mathsf{e})$ to $[\,]$, then we obtain $\pi(\mathsf{F}(\mathsf{e}(\mathsf{f}(x)))) = \mathsf{F}(\mathsf{e})$.

A *preorder* (or *quasi-order*) is a transitive and reflexive relation. A *rewrite preorder* is a preorder $\succsim$ on terms that is closed under contexts and substitutions. A *reduction pair* [16] consists of a rewrite preorder $\succsim$ and a compatible well-founded order $>$ which is closed under substitutions. Here compatibility means that the inclusion $\succsim \cdot > \;\subseteq\; >$ or the inclusion $> \cdot \succsim \;\subseteq\; >$ holds. In practice, $>$ is often chosen to be the strict part $\succ$ of $\succsim$ (or the order where $s > t$ iff $s\sigma \succ t\sigma$ for all ground substitutions σ). The following theorem presents the (basic) dependency pair approach of Arts and Giesl.

Theorem 1. *A TRS $\mathcal{R}$ over a signature $\mathcal{F}$ is terminating if and only if there exists an argument filtering π for $\mathcal{F}^\sharp$ and a reduction pair $(\succsim, >)$ such that $\pi(\mathcal{R}) \subseteq \succsim$ and $\pi(\mathrm{DP}(\mathcal{R})) \subseteq >$.*

Because rewrite rules are just pairs of terms, $\pi(\mathcal{R}) \subseteq \succsim$ is a shorthand for $\pi(l) \succsim \pi(r)$ for every rewrite rule $l \to r \in \mathcal{R}$. In our example, when using $\pi(\mathsf{e}) = [\,]$, the inequalities $\mathsf{f}(\mathsf{f}(x)) \succsim \mathsf{f}(\mathsf{e})$, $\mathsf{F}(\mathsf{f}(x)) > \mathsf{F}(\mathsf{e})$, and $\mathsf{F}(\mathsf{f}(x)) > \mathsf{F}(x)$ resulting from the dependency pair technique are satisfied by the recursive path order, for instance. Hence, termination of this TRS is proved.

Rather than considering all dependency pairs at the same time, like in the above theorem, it is advantageous to treat groups of dependency pairs separately. These groups correspond to *clusters* in the *dependency graph* of $\mathcal{R}$. The nodes of the dependency graph are the dependency pairs of $\mathcal{R}$ and there is an arrow from node $l_1^\sharp \to t_1^\sharp$ to $l_2^\sharp \to t_2^\sharp$ if there exist substitutions σ_1 and σ_2 such that $t_1^\sharp\sigma_1 \to_\mathcal{R}^* l_2^\sharp\sigma_2$. (By renaming variables in different occurrences of dependency pairs we may assume that $\sigma_1 = \sigma_2$.) The dependency graph of $\mathcal{R}$ is denoted by $\mathrm{DG}(\mathcal{R})$. We call a non-empty subset $\mathcal{C}$ of dependency pairs of $\mathrm{DP}(\mathcal{R})$ a *cluster* if for every two (not necessarily distinct) pairs $l_1^\sharp \to t_1^\sharp$ and $l_2^\sharp \to t_2^\sharp$ in $\mathcal{C}$ there exists a non-empty path in $\mathcal{C}$ from $l_1^\sharp \to t_1^\sharp$ to $l_2^\sharp \to t_2^\sharp$.

Theorem 2. *A TRS $\mathcal{R}$ is terminating if and only if for every cluster $\mathcal{C}$ in* $\mathrm{DG}(\mathcal{R})$ *there exists an argument filtering π and a reduction pair $(\succsim, >)$ such that $\pi(\mathcal{R}) \subseteq \succsim$, $\pi(\mathcal{C}) \subseteq \succsim \cup >$, and $\pi(\mathcal{C}) \cap > \neq \varnothing$.*

Note that $\pi(\mathcal{C}) \cap > \neq \varnothing$ denotes the situation that $\pi(l^\sharp) > \pi(t^\sharp)$ for at least one dependency pair $l^\sharp \to t^\sharp \in \mathcal{C}$.

In the above example, the dependency graph only contains an arrow from $\mathsf{F}(\mathsf{f}(x)) \to \mathsf{F}(x)$ to itself and thus $\{\mathsf{F}(\mathsf{f}(x)) \to \mathsf{F}(x)\}$ is the only cluster. Hence, with the refinement of Theorem 2 the inequality $\mathsf{F}(\mathsf{f}(x)) > \mathsf{F}(\mathsf{e})$ is no longer necessary. See [3] for further examples which illustrate the advantages of regarding clusters separately.

Note that while in general the dependency graph cannot be computed automatically (since it is undecidable whether $t_1^\sharp\sigma \to_\mathcal{R}^* l_2^\sharp\sigma$ holds for some σ), one can nevertheless approximate this graph automatically, cf. [1–3, "estimated dependency graph"]. In this way, the criterion of Theorem 2 can be mechanized.

Most classical methods for automated termination proofs are restricted to simplification (pre)orders, i.e., to (pre)orders satisfying the subterm property $f(\ldots t \ldots) \succ t$ or $f(\ldots t \ldots) \succsim t$, respectively. Hence, these methods cannot prove termination of TRSs like (1), as the left-hand side of its rule is embedded in the right-hand side (so the TRS is not simply terminating). However, with the development of the dependency pair technique now the TRSs where an automated termination proof is potentially possible are those systems where the inequalities generated by the dependency pair technique are satisfied by simplification (pre)orders.

A straightforward way to generate a simplification preorder $\succeq$ from a simplification order $\succ$ is to define $s \succeq t$ if $s \succ t$ or $s = t$, where $=$ denotes syntactic equality. Such relations $\succeq$ are particularly relevant, since many existing techniques generate simplification *orders* rather than *preorders*. By restricting ourselves to this class of simplification preorders, we obtain the notion of DP simple termination.

Definition 1. *A TRS $\mathcal{R}$ is called* DP simply terminating *if for every cluster $\mathcal{C}$ in* $\mathrm{DG}(\mathcal{R})$ *there exists an argument filtering π and a simplification order $\succ$ such that $\pi(\mathcal{R} \cup \mathcal{C}) \subseteq \succeq$ and $\pi(\mathcal{C}) \cap \succ \neq \varnothing$.*

Simple termination implies DP simple termination, but not vice versa. For example, the TRS (1) is DP simply terminating, but not simply terminating. The above definition coincides with the one in [13] except that we use the real dependency graph instead of the *estimated* dependency graph of [1–3]. The reason for this is that we do not want to restrict ourselves to a particular computable approximation of the dependency graph, for the same reason that we do not insist on a particular simplification order to make the conditions effective.

3 Dummy Elimination

In [11], Ferreira and Zantema defined an automatic transformation technique which transforms a TRS $\mathcal{R}$ into a new TRS dummy($\mathcal{R}$) such that termination of dummy($\mathcal{R}$) implies termination of $\mathcal{R}$. The advantage of this transformation is that non-simply terminating systems like (1) may be transformed into simply terminating ones. Thus, after the transformation, standard techniques may be used to prove termination.

Below we define Ferreira and Zantema's dummy elimination transformation. While our formulation of dummy($\mathcal{R}$) is different from the one in [11], it is easily seen to be equivalent.

Definition 2. *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$. Let e be a distinguished function symbol in $\mathcal{F}$ of arity $m \geqslant 1$ and let $\diamond$ be a fresh constant. We write $\mathcal{F}_\diamond$ for $(\mathcal{F} \setminus \{e\}) \cup \{\diamond\}$. The mapping* $\mathrm{cap}\colon \mathcal{T}(\mathcal{F},\mathcal{V}) \to \mathcal{T}(\mathcal{F}_\diamond,\mathcal{V})$ *is inductively defined as follows:*

$$\mathrm{cap}(t) = \begin{cases} t & \text{if } t \in \mathcal{V}, \\ \diamond & \text{if } t = e(t_1,\ldots,t_m), \\ f(\mathrm{cap}(t_1),\ldots,\mathrm{cap}(t_n)) & \text{if } t = f(t_1,\ldots,t_n) \text{ with } f \neq e. \end{cases}$$

The mapping dummy *assigns to every term in $\mathcal{T}(\mathcal{F},\mathcal{V})$ a subset of $\mathcal{T}(\mathcal{F}_\diamond,\mathcal{V})$, as follows:*

$$\mathrm{dummy}(t) = \{\mathrm{cap}(t)\} \cup \{\mathrm{cap}(s) \mid s \text{ is an argument of an } e \text{ symbol in } t\}.$$

Finally, we define

$$\mathrm{dummy}(\mathcal{R}) = \{\mathrm{cap}(l) \to r' \mid l \to r \in \mathcal{R} \text{ and } r' \in \mathrm{dummy}(r)\}.$$

The mappings cap and dummy are illustrated in Figure 1, where we assume that the numbered contexts do not contain any occurrences of e. Ferreira and Zantema [11] showed that dummy elimination is sound.

Theorem 3. *Let $\mathcal{R}$ be a TRS. If* dummy($\mathcal{R}$) *is terminating then $\mathcal{R}$ is terminating.*

For the one-rule TRS (1), dummy elimination yields the TRS consisting of the two rewrite rules

$$\mathsf{f}(\mathsf{f}(x)) \to \mathsf{f}(\diamond) \qquad \mathsf{f}(\mathsf{f}(x)) \to \mathsf{f}(x)$$

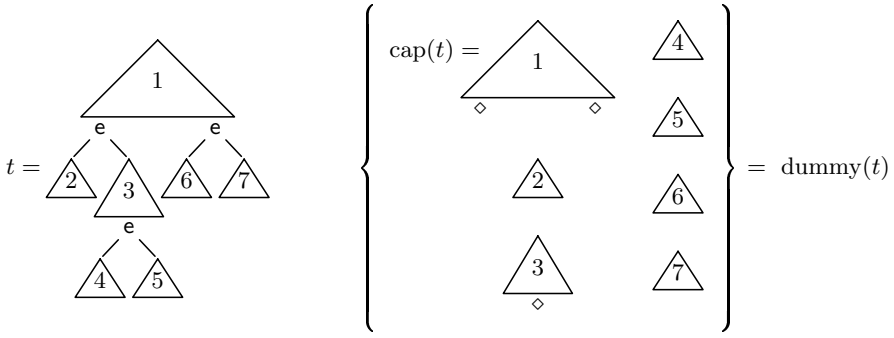

Fig. 1. The mappings cap and dummy.

In contrast to the original system, the new TRS is simply terminating and its termination is easily shown automatically by standard techniques like the recursive path order. Hence, dummy elimination can transform non-simply terminating TRSs into simply terminating ones. However, as indicated in the introduction, nowadays the right question to ask is whether it can transform non-DP simply terminating TRSs into DP simply terminating ones. Before answering this question we show that if dummy elimination succeeds in transforming a TRS into a simply terminating TRS then the original TRS is DP simply terminating. Even stronger, whenever termination of dummy($\mathcal{R}$) can be proved by a simplification order, then the *same* simplification order satisfies the constraints of the dependency pair approach. Thus, the termination proof using dependency pairs is not more difficult or more complex than the one with dummy elimination.

Theorem 4. *Let $\mathcal{R}$ be a TRS. If* dummy($\mathcal{R}$) *is simply terminating then $\mathcal{R}$ is DP simply terminating.*

Proof. Let $\mathcal{F}$ be the signature of $\mathcal{R}$. We show that $\mathcal{R}$ is DP simply terminating even without considering the dependency graph refinement. So we define an argument filtering π for $\mathcal{F}^\sharp$ and a simplification order $\succ$ on $\mathcal{T}(\mathcal{F}^\sharp_\pi, \mathcal{V})$ such that $\pi(\mathcal{R}) \subseteq \succeq$ and $\pi(\mathrm{DP}(\mathcal{R})) \subseteq \succ$. The argument filtering π is defined as follows: $\pi(e) = [\,]$ and $\pi(f) = [1, \ldots, n]$ for every n-ary symbol $f \in (\mathcal{F} \setminus \{e\})^\sharp$. Moreover, if e is a defined symbol, we define $\pi(e^\sharp) = [\,]$. Let $\sqsupset$ be any simplification order that shows the simple termination of dummy($\mathcal{R}$). We define the simplification order $\succ$ on $\mathcal{T}(\mathcal{F}^\sharp_\pi, \mathcal{V})$ as follows: $s \succ t$ if and only if $s' \sqsupset t'$ where $(\cdot)'$ denotes the mapping from $\mathcal{T}(\mathcal{F}^\sharp_\pi, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}_\diamond, \mathcal{V})$ that first replaces every marked symbol F by f and afterwards replaces every occurrence of the constant e by $\diamond$. Note that $\succ$ and $\sqsupset$ are essentially the same. It is very easy to show that $\pi(t)' = \pi(t^\sharp)' = \mathrm{cap}(t)$ for every term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$. Let $l \to r \in \mathcal{R}$. Because $\mathrm{cap}(l) \to \mathrm{cap}(r)$ is a rewrite rule in dummy($\mathcal{R}$), we get $\pi(l)' = \mathrm{cap}(l) \sqsupset \mathrm{cap}(r) = \pi(r)'$ and thus $\pi(l) \succ \pi(r)$. Hence $\pi(\mathcal{R}) \subseteq \succ$ and thus certainly $\pi(\mathcal{R}) \subseteq \succeq$. Now let $l^\sharp \to t^\sharp$ be a dependency pair of $\mathcal{R}$, originating from the rewrite rule $l \to r \in \mathcal{R}$. From $t \trianglelefteq r$ ($\trianglelefteq$ denotes the subterm relation) we easily infer the existence of a term $u \in \mathrm{dummy}(r)$ such that $\mathrm{cap}(t) \trianglelefteq u$. Since $\mathrm{cap}(l) \to u$ is a rewrite rule in dummy($\mathcal{R}$), we have

$\pi(l^\sharp)' = \mathrm{cap}(l) \sqsupset u$. The subterm property of $\sqsupset$ yields $u \sqsupseteq \mathrm{cap}(t) = \pi(t^\sharp)'$. Hence $\pi(l^\sharp)' \sqsupset \pi(t^\sharp)'$ and thus $\pi(l^\sharp) \succ \pi(t^\sharp)$. We conclude that $\pi(\mathrm{DP}(\mathcal{R})) \subseteq \succ$. □

The previous result states that dummy elimination offers no advantage compared to the dependency pair technique. On the other hand, dependency pairs succeed for many systems where dummy elimination fails [1, 2] (an example is given in the next section). One could imagine that dummy elimination may nevertheless be helpful in combination with dependency pairs. Then to show termination of a TRS one would first apply dummy elimination and afterwards prove termination of the transformed TRS with the dependency pair technique. In the remainder of this section we show that such a scenario cannot handle TRSs which cannot already be handled by the dependency pair technique directly. In short, dummy elimination is useless for automated termination proofs. We proceed in a stepwise manner. First we relate the dependency pairs of $\mathcal{R}$ to those of dummy($\mathcal{R}$).

Lemma 1. *If* $l^\sharp \to t^\sharp \in \mathrm{DP}(\mathcal{R})$ *then* $\mathrm{cap}(l)^\sharp \to \mathrm{cap}(t)^\sharp \in \mathrm{DP}(\mathrm{dummy}(\mathcal{R}))$.

Proof. In the proof of Theorem 4 we observed that there exists a rewrite rule $\mathrm{cap}(l) \to u$ in dummy($\mathcal{R}$) with $\mathrm{cap}(t) \trianglelefteq u$. Since $\mathrm{root}(\mathrm{cap}(t))$ is a defined symbol in dummy($\mathcal{R}$), $\mathrm{cap}(l)^\sharp \to \mathrm{cap}(t)^\sharp$ is a dependency pair of dummy($\mathcal{R}$). □

Now we prove that reducibility in $\mathcal{R}$ implies reducibility in dummy($\mathcal{R}$).

Definition 3. *Given a substitution* σ*, the substitution* σ_{cap} *is defined as* $\mathrm{cap} \circ \sigma$ *(i.e., the composition of* cap *and* σ *where* σ *is applied first).*

Lemma 2. *For all terms* t *and substitutions* σ*, we have* $\mathrm{cap}(t\sigma) = \mathrm{cap}(t)\sigma_{\mathrm{cap}}$.

Proof. Easy induction on the structure of t. □

Lemma 3. *If* $s \to_{\mathcal{R}}^* t$ *then* $\mathrm{cap}(s) \to_{\mathrm{dummy}(\mathcal{R})}^* \mathrm{cap}(t)$.

Proof. It is sufficient to show that $s \to_{\mathcal{R}} t$ implies $\mathrm{cap}(s) \to_{\mathrm{dummy}(\mathcal{R})}^* \mathrm{cap}(t)$. There must be a rule $l \to r \in \mathcal{R}$ and a position p such that $s|_\pi = l\sigma$ and $t = s[r\sigma]_p$. If p is below the position of an occurrence of e, then we have $\mathrm{cap}(s) = \mathrm{cap}(t)$. Otherwise, $\mathrm{cap}(s)|_p = \mathrm{cap}(l\sigma) = \mathrm{cap}(l)\sigma_{\mathrm{cap}}$ by Lemma 2. Thus, $\mathrm{cap}(s) \to_{\mathrm{dummy}(\mathcal{R})} \mathrm{cap}(s)[\mathrm{cap}(r)\sigma_{\mathrm{cap}}]_p = \mathrm{cap}(s)[\mathrm{cap}(r\sigma)]_p = \mathrm{cap}(t)$. □

Next we show that if there is an arrow between two dependency pairs in the dependency graph of $\mathcal{R}$ then there is an arrow between the corresponding dependency pairs in the dependency graph of dummy($\mathcal{R}$).

Lemma 4. *Let* s*,* t *be terms with defined root symbols. If* $s^\sharp\sigma \to_{\mathcal{R}}^* t^\sharp\sigma$ *for some substitution* σ*, then* $\mathrm{cap}(s)^\sharp\sigma_{\mathrm{cap}} \to_{\mathrm{dummy}(\mathcal{R})}^* \mathrm{cap}(t)^\sharp\sigma_{\mathrm{cap}}$.

Proof. Let $s = f(s_1, \ldots, s_n)$. We have $s^\sharp\sigma = f^\sharp(s_1\sigma, \ldots, s_n\sigma)$. Since $f^\sharp$ is a constructor, no step in the sequence $s^\sharp\sigma \to_{\mathcal{R}}^* t^\sharp\sigma$ takes place at the root position and thus $t^\sharp = f^\sharp(t_1, \ldots, t_n)$ with $s_i\sigma \to_{\mathcal{R}}^* t_i\sigma$ for all $1 \leqslant i \leqslant n$. We obtain $\mathrm{cap}(s_i)\sigma_{\mathrm{cap}} = \mathrm{cap}(s_i\sigma) \to_{\mathrm{dummy}(\mathcal{R})}^* \mathrm{cap}(t_i\sigma) = \mathrm{cap}(t_i)\sigma_{\mathrm{cap}}$ for all $1 \leqslant i \leqslant n$ by Lemmata 2 and 3. Hence $\mathrm{cap}(s)^\sharp\sigma_{\mathrm{cap}} \to_{\mathrm{dummy}(\mathcal{R})}^* \mathrm{cap}(t)^\sharp\sigma_{\mathrm{cap}}$. □

Finally we are ready for the main theorem of this section.

Theorem 5. *Let $\mathcal{R}$ be a TRS. If* dummy($\mathcal{R}$) *is DP simply terminating then $\mathcal{R}$ is DP simply terminating.*

Proof. Let $\mathcal{C}$ be a cluster in the dependency graph of $\mathcal{R}$. From Lemmata 1 and 4 we infer the existence of a corresponding cluster, denoted by dummy($\mathcal{C}$), in the dependency graph of dummy($\mathcal{R}$). By assumption, there exists an argument filtering π' and a simplification order $\sqsupset$ such that $\pi'(\text{dummy}(\mathcal{R}) \cup \text{dummy}(\mathcal{C})) \subseteq \sqsupseteq$ and $\pi'(\text{dummy}(\mathcal{C})) \cap \sqsupset \neq \varnothing$. Let $\mathcal{F}$ be the signature of $\mathcal{R}$. We define an argument filtering π for $\mathcal{F}^\sharp$ as follows: $\pi(f) = \pi'(f)$ for every $f \in (\mathcal{F} \setminus \{e\})^\sharp$, $\pi(e) = [\,]$ and, if e is a defined symbol of $\mathcal{R}$, $\pi(e^\sharp) = [\,]$. Slightly different from the proof of Theorem 4, let $(\cdot)'$ denote the mapping that just replaces every occurrence of the constant e by $\diamond$ and every occurrence of $e^\sharp$ by $\diamond^\sharp$. It is easy to show that $\pi(t)' = \pi'(\text{cap}(t))$ for every term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\pi(t^\sharp)' = \pi'(\text{cap}(t)^\sharp)$ for every term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with a defined root symbol. Similar to Theorem 4, we define the simplification order $\succ$ on $\mathcal{F}_\pi$ as $s \succ t$ if and only if $s' \sqsupset t'$. We claim that π and $\succ$ satisfy the constraints for $\mathcal{C}$, i.e., $\pi(\mathcal{R} \cup \mathcal{C}) \subseteq \succeq$ and $\pi(\text{dummy}(\mathcal{C})) \cap \succ \neq \varnothing$. If $l \to r \in \mathcal{R}$, then $\text{cap}(l) \to \text{cap}(r) \in \text{dummy}(\mathcal{R})$ and thus $\pi(l)' = \pi'(\text{cap}(l)) \sqsupseteq \pi'(\text{cap}(r)) = \pi(r)'$. Hence $\pi(l) \succeq \pi(r)$. If $l^\sharp \to t^\sharp \in \mathcal{C}$, then $\text{cap}(l)^\sharp \to \text{cap}(t)^\sharp \in \text{dummy}(\mathcal{C})$ by Lemma 1 and thus $\pi(l^\sharp)' = \pi'(\text{cap}(l)^\sharp) \sqsupseteq \pi'(\text{cap}(t)^\sharp) = \pi(t^\sharp)'$. Hence $\pi(l^\sharp) \succeq \pi(t^\sharp)$ and if $\pi'(\text{cap}(l)^\sharp) \sqsupset \pi'(\text{cap}(t)^\sharp)$, then $\pi(l^\sharp) \succ \pi(t^\sharp)$. □

We stress that the proof is constructive in the sense that a DP simple termination proof of dummy($\mathcal{R}$) can be automatically transformed into a DP simple termination proof of $\mathcal{R}$ (i.e., the orders and argument filterings required for the DP simple termination proofs of dummy($\mathcal{R}$) and $\mathcal{R}$ are essentially the same). Thus, the termination proof of dummy($\mathcal{R}$) is not simpler than a direct proof for $\mathcal{R}$.

Theorem 5 also holds if one uses the *estimated* dependency graph of [1–3] instead of the real dependency graph. As mentioned in Section 2, such a computable approximation of the dependency graph must be used in implementations, since constructing the real dependency graph is undecidable in general. The proof is similar to the one of Theorem 5, since again for every cluster in the estimated dependency graph of $\mathcal{R}$ there is a corresponding one in the estimated dependency graph of dummy($\mathcal{R}$).

4 Argument Filtering Transformation

By incorporating argument filterings, a key ingredient of the dependency pair technique, into dummy elimination, Kusakari, Nakamura, and Toyama [16] recently developed the argument filtering transformation. In their paper they proved the soundness of their transformation and they showed that it improves upon dummy elimination. In this section we compare their transformation to the dependency pair technique. We proceed as in the previous section. First we recall the definition of the argument filtering transformation.

Definition 4. *Let π be an argument filtering, f a function symbol, and $1 \leqslant i \leqslant \mathrm{arity}(f)$. We write $f \perp_\pi i$ if neither $i \in \pi(f)$ nor $i = \pi(f)$. Given two terms s and t, we say that s is a* preserved *subterm of t with respect to π and we write $s \trianglelefteq_\pi t$, if $s \trianglelefteq t$ and either $s = t$ or $t = f(t_1, \ldots, t_n)$, s is a preserved subterm of t_i, and $f \not\perp_\pi i$.*

Definition 5. *Given an argument filtering π, the argument filtering $\bar{\pi}$ is defined as follows:*

$$\bar{\pi}(f) = \begin{cases} \pi(f) & \text{if } \pi(f) = [i_1, \ldots, i_m], \\ [\pi(f)] & \text{if } \pi(f) = i. \end{cases}$$

The mapping AFT_π assigns to every term in $\mathcal{T}(\mathcal{F}, \mathcal{V})$ a subset of $\mathcal{T}(\mathcal{F}_\pi, \mathcal{V})$, as follows:

$$\mathrm{AFT}_\pi(t) = \{\pi(t) \mid \bar{\pi}(t) \text{ contains a defined symbol}\} \cup \bigcup_{s \in S} \mathrm{AFT}_\pi(s)$$

with S denoting the set of outermost non-preserved subterms of t. Finally, we define

$$\mathrm{AFT}_\pi(\mathcal{R}) = \{\pi(l) \to r' \mid l \to r \in \mathcal{R} \text{ and } r' \in \mathrm{AFT}_\pi(r) \cup \{\pi(r)\}\}.$$

Consider the term t of Figure 1. Figure 2 shows $\mathrm{AFT}_\pi(t)$ for the two argument filterings with $\pi(\mathsf{e}) = [1]$ and $\pi(\mathsf{e}) = 2$, respectively, and $\pi(f) = [1, \ldots, n]$ for every other n-ary function symbol f. Here we assume that all numbered contexts contain defined symbols, but no occurrence of e.

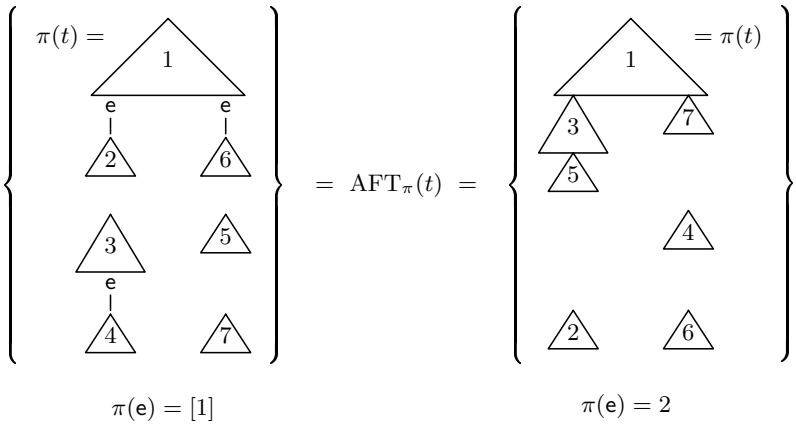

Fig. 2. The mappings π and AFT_π.

So essentially, $\mathrm{AFT}_\pi(t)$ contains $\pi(s)$ for $s = t$ and for all (maximal) subterms s of t which are eliminated if the argument filtering π is applied to t. However, one only needs terms $\pi(s)$ in $\mathrm{AFT}_\pi(t)$ where s contained a defined

symbol outside eliminated arguments (otherwise the original subterm s cannot have been responsible for a potential non-termination). Kusakari *et al.* [11] proved the soundness of the argument filtering transformation.

Theorem 6. *If* $\mathrm{AFT}_\pi(\mathcal{R})$ *is terminating then* $\mathcal{R}$ *is terminating.*

We show that if $\mathrm{AFT}_\pi(\mathcal{R})$ is simply terminating then $\mathcal{R}$ is DP simply terminating and again, a termination proof by dependency pairs works with the *same* argument filtering π and the simplification order used to orient $\mathrm{AFT}_\pi(\mathcal{R})$. Thus, the argument filtering transformation has no advantage compared to dependency pairs. We start with two easy lemmata.[1]

Lemma 5. *Let* s *and* t *be terms. If* $s \trianglelefteq_\pi t$ *then* $\pi(s) \trianglelefteq \pi(t)$.

Proof. By induction on the definition of $\trianglelefteq_\pi$. If $s = t$ then the result is trivial. Suppose $t = f(t_1, \ldots, t_n)$, $s \trianglelefteq_\pi t_i$, and $f \not\perp_\pi i$. The induction hypothesis yields $\pi(s) \trianglelefteq \pi(t_i)$. Because $f \not\perp_\pi i$, $\pi(t_i)$ is a subterm of $\pi(t)$ and thus $\pi(s) \trianglelefteq \pi(t)$ as desired. □

Lemma 6. *Let* r *be a term. For every subterm* t *of* r *with a defined root symbol there exists a term* $u \in \mathrm{AFT}_\pi(r)$ *such that* $\pi(t) \trianglelefteq u$.

Proof. We use induction on the structure of r. In the base case we must have $t = r$ and we take $u = \pi(r)$. Note that $\pi(r) \in \mathrm{AFT}_\pi(r)$ because $\mathrm{root}(\bar{\pi}(r)) = \mathrm{root}(r)$ is defined. In the induction step we distinguish two cases. If $t \trianglelefteq_\pi r$ then we also have $t \trianglelefteq_{\bar{\pi}} r$ and hence $\bar{\pi}(t) \trianglelefteq \bar{\pi}(r)$ by Lemma 5. As $\mathrm{root}(\bar{\pi}(t)) = \mathrm{root}(t)$ is defined, the term $\bar{\pi}(r)$ contains a defined symbol. Hence $\pi(r) \in \mathrm{AFT}_\pi(r)$ by definition and thus we can take $u = \pi(r)$. In the other case t is not a preserved subterm of r. This implies that $t \trianglelefteq s$ for some outermost non-preserved subterm s of r. The induction hypothesis, applied to $t \trianglelefteq s$, yields a term $u \in \mathrm{AFT}_\pi(s)$ such that $\pi(t) \trianglelefteq u$. We have $\mathrm{AFT}_\pi(s) \subseteq \mathrm{AFT}_\pi(r)$ and hence u satisfies the requirements. □

Theorem 7. *Let* $\mathcal{R}$ *be a TRS and* π *an argument filtering. If* $\mathrm{AFT}_\pi(\mathcal{R})$ *is simply terminating then* $\mathcal{R}$ *is DP simply terminating.*

Proof. Like in the proof of Theorem 4 there is no need to consider the dependency graph. Let $\succ$ be a simplification order that shows the (simple) termination of $\mathrm{AFT}_\pi(\mathcal{R})$. We claim that the dependency pair constraints are satisfied by π and $\succ$, where π and $\succ$ are extended to $\mathcal{F}^\sharp$ by treating each marked symbol F in the same way as the corresponding unmarked f. For rewrite rules $l \to r \in \mathcal{R}$ we have $\pi(l) \succ \pi(r)$ as $\pi(l) \to \pi(r) \in \mathrm{AFT}_\pi(\mathcal{R})$. Let $l^\sharp \to t^\sharp$ be a dependency pair of $\mathcal{R}$, originating from the rewrite rule $l \to r$. We show that $\pi(l) \succ \pi(t)$ and hence, $\pi(l^\sharp) \succ \pi(t^\sharp)$ as well. We have $t \trianglelefteq r$. Since $\mathrm{root}(t)$ is a defined function symbol

[1] Argumentations similar to the proofs of Lemma 6 and Theorem 7 can also be found in [16, Lemma 4.3 and Theorem 4.4]. However, [16] contains neither Theorem 7 nor our main Theorem 8, since the authors do not compare the argument filtering transformation with the dependency pair approach.

by the definition of dependency pairs, we can apply Lemma 6. This yields a term $u \in \mathrm{AFT}_\pi(r)$ such that $\pi(t) \trianglelefteq u$. The subterm property of $\succ$ yields $u \succeq \pi(t)$. By definition, $\pi(l) \to u \in \mathrm{AFT}_\pi(\mathcal{R})$ and thus $\pi(l) \succ u$ by compatibility of $\succ$ with $\mathrm{AFT}_\pi(\mathcal{R})$. Hence $\pi(l) \succ \pi(t)$ as desired. □

Note that in the above proof we did not make use of the possibility to treat marked symbols differently from unmarked ones. This clearly shows why the dependency pair technique is much more powerful than the argument filtering transformation; there are numerous DP simply terminating TRSs which are no longer DP simply terminating if we are forced to interpret a defined function symbol and its marked version in the same way. As a simple example, consider

$$\mathcal{R}_1 = \left\{ \begin{array}{ll} x - 0 \to x & 0 \div \mathsf{s}(y) \to 0 \\ x - \mathsf{s}(y) \to \mathsf{p}(x - y) & \mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x - y) \div \mathsf{s}(y)) \\ \mathsf{p}(\mathsf{s}(x)) \to x & \end{array} \right\}.$$

Note that $\mathcal{R}_1$ is not simply terminating as the rewrite step $\mathsf{s}(x) \div \mathsf{s}(\mathsf{s}(x)) \to \mathsf{s}((x-\mathsf{s}(x)) \div \mathsf{s}(\mathsf{s}(x)))$ is self-embedding. To obtain a terminating TRS $\mathrm{AFT}_\pi(\mathcal{R}_1)$, the rule $\mathsf{p}(\mathsf{s}(x)) \to x$ enforces $\mathsf{p} \not\sim_\pi 1$ and $\mathsf{s} \not\sim_\pi 1$. From $\mathsf{p} \not\sim_\pi 1$ and the rules for $-$ we infer that $\pi(-) = [1, 2]$. But then, for all choices of $\pi(\div)$, the rule $\mathsf{s}(x) \div \mathsf{s}(y) \to \mathsf{s}((x-y) \div \mathsf{s}(y))$ is transformed into one that is incompatible with a simplification order. So $\mathrm{AFT}_\pi(\mathcal{R}_1)$ is not simply terminating for any π. (Similarly, dummy elimination cannot transform this TRS into a simply terminating one either.) On the other hand, DP simple termination of $\mathcal{R}_1$ is easily shown by the argument filtering $\pi(\mathsf{p}) = 1$, $\pi(-) = 1$, $\pi(-^\sharp) = [1, 2]$, and $\pi(f) = [1, \ldots, \mathrm{arity}(f)]$ for every other function symbol f in combination with the recursive path order. This example illustrates that treating defined symbols and their marked versions differently is often required in order to benefit from the fact that the dependency pair approach only requires *weak* decreasingness for the rules of $\mathcal{R}_1$.

The next question we address is whether the argument filtering transformation can be useful as a preprocessing step for the dependency pair technique. Surprisingly, the answer to this question is yes. Consider the TRS

$$\mathcal{R}_2 = \left\{ \begin{array}{lll} \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{c}(\mathsf{a})) & \mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{d}(\mathsf{a})) & \mathsf{e}(\mathsf{g}(x)) \to \mathsf{e}(x) \\ \mathsf{f}(\mathsf{c}(x)) \to x & \mathsf{f}(\mathsf{d}(x)) \to x & \\ \mathsf{f}(\mathsf{c}(\mathsf{a})) \to \mathsf{f}(\mathsf{d}(\mathsf{b})) & \mathsf{f}(\mathsf{c}(\mathsf{b})) \to \mathsf{f}(\mathsf{d}(\mathsf{a})) & \end{array} \right\}.$$

This TRS is not DP simply terminating which can be seen as follows. The dependency pair $\mathsf{E}(\mathsf{g}(x)) \to \mathsf{E}(x)$ constitutes a cluster in the dependency graph of $\mathcal{R}_2$. Hence, if $\mathcal{R}_2$ were DP simply terminating, there would be an argument filtering π and a simplification order $\succ$ such that (amongst others)

$$\begin{array}{ll} \pi(\mathsf{f}(\mathsf{a})) \succeq \pi(\mathsf{f}(\mathsf{c}(\mathsf{a}))) & \pi(\mathsf{f}(\mathsf{a})) \succeq \pi(\mathsf{f}(\mathsf{d}(\mathsf{a}))) \\ \pi(\mathsf{f}(\mathsf{c}(x))) \succeq x & \pi(\mathsf{f}(\mathsf{d}(x))) \succeq x \\ \pi(\mathsf{f}(\mathsf{c}(\mathsf{a}))) \succeq \pi(\mathsf{f}(\mathsf{d}(\mathsf{b}))) & \pi(\mathsf{f}(\mathsf{c}(\mathsf{b}))) \succeq \pi(\mathsf{f}(\mathsf{d}(\mathsf{a}))) \end{array}$$

From $\pi(\mathsf{f}(\mathsf{c}(x))) \succeq x$ and $\pi(\mathsf{f}(\mathsf{d}(x))) \succeq x$ we infer that $\mathsf{f} \not\sim_\pi 1$, $\mathsf{c} \not\sim_\pi 1$, and $\mathsf{d} \not\sim_\pi 1$. Hence $\pi(\mathsf{f}(\mathsf{a})) \succeq \pi(\mathsf{f}(\mathsf{c}(\mathsf{a})))$ and $\pi(\mathsf{f}(\mathsf{a})) \succeq \pi(\mathsf{f}(\mathsf{d}(\mathsf{a})))$ can only be satisfied

if $\pi(\mathsf{c}) = \pi(\mathsf{d}) = 1$. But then $\pi(\mathsf{f}(\mathsf{c}(\mathsf{a}))) \succeq \pi(\mathsf{f}(\mathsf{d}(\mathsf{b})))$ and $\pi(\mathsf{f}(\mathsf{c}(\mathsf{b}))) \succeq \pi(\mathsf{f}(\mathsf{d}(\mathsf{a})))$ amount to either $\mathsf{f}(\mathsf{a}) \succeq \mathsf{f}(\mathsf{b})$ and $\mathsf{f}(\mathsf{b}) \succeq \mathsf{f}(\mathsf{a})$ (if $\pi(\mathsf{f}) = [1]$) or $\mathsf{a} \succeq \mathsf{b}$ and $\mathsf{b} \succeq \mathsf{a}$ (if $\pi(\mathsf{f}) = 1$). Since $\mathsf{f}(\mathsf{a}) \neq \mathsf{f}(\mathsf{b})$ and $\mathsf{a} \neq \mathsf{b}$ the required simplification order does not exist.

On the other hand, if $\pi(\mathsf{e}) = 1$ then $\mathrm{AFT}_\pi(\mathcal{R}_2)$ consists of the first six rewrite rules of $\mathcal{R}$ together with $\mathsf{g}(x) \to x$. One easily verifies that there are no clusters in $\mathrm{DG}(\mathrm{AFT}_\pi(\mathcal{R}_2))$ and hence $\mathrm{AFT}_\pi(\mathcal{R}_2)$ is trivially DP simply terminating.

Definition 6. *An argument filtering π is called* collapsing *if $\pi(f) = i$ for some defined function symbol f.*

The argument filtering in the previous example is collapsing. In the remainder of this section we show that for non-collapsing argument filterings the implication "$\mathrm{AFT}_\pi(\mathcal{R})$ is DP simply terminating $\Rightarrow$ $\mathcal{R}$ is DP simply terminating" is valid. Thus, using the argument filtering transformation with a non-collapsing π as a preprocessing step to the dependency pair technique has no advantages.

First we prove a lemma to relate the dependency pairs of $\mathcal{R}$ and $\mathrm{AFT}_\pi(\mathcal{R})$.

Lemma 7. *Let π be a non-collapsing argument filtering. If $l^\sharp \to t^\sharp \in \mathrm{DP}(\mathcal{R})$ then $\pi(l)^\sharp \to \pi(t)^\sharp \in \mathrm{DP}(\mathrm{AFT}_\pi(\mathcal{R}))$.*

Proof. By definition there is a rewrite rule $l \to r \in \mathcal{R}$ and a subterm $t \trianglelefteq r$ with defined root symbol. According to Lemma 6 there exists a term $u \in \mathrm{AFT}_\pi(r)$ such that $\pi(t) \trianglelefteq u$. Thus, $\pi(l) \to u \in \mathrm{AFT}_\pi(\mathcal{R})$. Since π is non-collapsing, $\mathrm{root}(\pi(t)) = \mathrm{root}(t)$. Hence, as $\mathrm{root}(t)$ is defined, $\pi(l)^\sharp \to \pi(t)^\sharp$ is a dependency pair of $\mathrm{AFT}_\pi(\mathcal{R})$. □

Example $\mathcal{R}_2$ shows that the above lemma is not true for arbitrary argument filterings. The reason is that $\mathsf{e}(\mathsf{g}(x))^\sharp \to \mathsf{e}(x)^\sharp$ is a dependency pair of $\mathcal{R}$, but with $\pi(\mathsf{e}) = 1$ there is no corresponding dependency pair in $\mathrm{AFT}_\pi(\mathcal{R})$.

The next three lemmata will be used to show that clusters in $\mathrm{DG}(\mathcal{R})$ correspond to clusters in $\mathrm{DG}(\mathrm{AFT}_\pi(\mathcal{R}))$.

Definition 7. *Given an argument filtering π and a substitution σ, the substitution σ_π is defined as $\pi \circ \sigma$ (i.e., σ is applied first).*

Lemma 8. *For all terms t, argument filterings π, and substitutions σ, $\pi(t\sigma) = \pi(t)\sigma_\pi$.*

Proof. Easy induction on the structure of t. □

Lemma 9. *Let $\mathcal{R}$ be a TRS and π a non-collapsing argument filtering. If $s \to_\mathcal{R}^* t$ then $\pi(s) \to_{\mathrm{AFT}_\pi(\mathcal{R})}^* \pi(t)$.*

Proof. It suffices to show that $\pi(s) \to_{\mathrm{AFT}_\pi(\mathcal{R})}^* \pi(t)$ whenever $s \to_\mathcal{R}^* t$ consists of a single rewrite step. Let $s = C[l\sigma]$ and $t = C[r\sigma]$ for some context C, rewrite rule $l \to r \in \mathcal{R}$, and substitution σ. We use induction on C. If C is the empty context, then $\pi(s) = \pi(l\sigma) = \pi(l)\sigma_\pi$ and $\pi(t) = \pi(r\sigma) = \pi(r)\sigma_\pi$ according to

Lemma 8. As $\pi(l) \to \pi(r) \in \mathrm{AFT}_\pi(\mathcal{R})$, we have $\pi(s) \to_{\mathrm{AFT}_\pi(\mathcal{R})} \pi(t)$. Suppose $C = f(s_1, \ldots, C', \ldots, s_n)$ where C' is the i-th argument of C. If $f \perp_\pi i$ then $\pi(s) = \pi(t)$. If $\pi(f) = i$ (which is possible for constructors f) then $\pi(s) = \pi(C'[l\sigma])$ and $\pi(t) = \pi(C'[r\sigma])$, and thus we obtain $\pi(s) \to^*_{\mathrm{AFT}_\pi(\mathcal{R})} \pi(t)$ from the induction hypothesis. In the remaining case we have $\pi(f) = [i_1, \ldots, i_m]$ with $i_j = i$ for some j and hence $\pi(s) = f(\pi(s_{i_1}), \ldots, \pi(C'[l\sigma]), \ldots, \pi(s_{i_m}))$ and $\pi(t) = f(\pi(s_{i_1}), \ldots, \pi(C'[r\sigma]), \ldots, \pi(s_{i_m}))$. In this case we obtain $\pi(s) \to^*_{\mathrm{AFT}_\pi(\mathcal{R})} \pi(t)$ from the induction hypothesis as well. □

The following lemma states that if two dependency pairs are connected in $\mathcal{R}$'s dependency graph, then the corresponding pairs are connected in the dependency graph of $\mathrm{AFT}_\pi(\mathcal{R})$ as well.

Lemma 10. *Let $\mathcal{R}$ be a TRS, π a non-collapsing argument filtering, and s, t be terms with defined root symbols. If $s^\sharp\sigma \to^*_{\mathcal{R}} t^\sharp\sigma$ for some substitution σ then $\pi(s)^\sharp\sigma_\pi \to^*_{\mathrm{AFT}_\pi(\mathcal{R})} \pi(t)^\sharp\sigma_\pi$.*

Proof. We have $s = f(s_1, \ldots, s_n)$ and $t = f(t_1, \ldots, t_n)$ for some n-ary defined function symbol f with $s_i\sigma \to^*_{\mathcal{R}} t_i\sigma$ for all $1 \leqslant i \leqslant n$. Let $\pi(f) = [i_1, \ldots, i_m]$. This implies $\pi(s\sigma)^\sharp = f^\sharp(\pi(s_{i_1}\sigma), \ldots, \pi(s_{i_m}\sigma))$ and $\pi(t\sigma)^\sharp = f^\sharp(\pi(t_{i_1}\sigma), \ldots, \pi(t_{i_m}\sigma))$. From the preceding lemma we know that $\pi(s_{i_j}\sigma) \to^*_{\mathrm{AFT}_\pi(\mathcal{R})} \pi(t_{i_j}\sigma)$ for all $1 \leqslant j \leqslant m$. Hence, using Lemma 8, $\pi(s)^\sharp\sigma_\pi = \pi(s\sigma)^\sharp \to^*_{\mathrm{AFT}_\pi(\mathcal{R})} \pi(t\sigma)^\sharp = \pi(t)^\sharp\sigma_\pi$. □

Now we can finally prove the main theorem of this section.

Theorem 8. *Let $\mathcal{R}$ be a TRS and π a non-collapsing argument filtering. If $\mathrm{AFT}_\pi(\mathcal{R})$ is DP simply terminating then $\mathcal{R}$ is DP simply terminating.*

Proof. Let $\mathcal{C}$ be a cluster in $\mathrm{DG}(\mathcal{R})$. According to Lemmata 7 and 10, there is a corresponding cluster in $\mathrm{DG}(\mathrm{AFT}_\pi(\mathcal{R}))$, which we denote by $\pi(\mathcal{C})$. By assumption, there exist an argument filtering π' and a simplification order $\succ$ such that $\pi'(\mathrm{AFT}_\pi(\mathcal{R}) \cup \pi(\mathcal{C})) \subseteq \succeq$ and $\pi'(\pi(\mathcal{C})) \cap \succ \neq \varnothing$. We define an argument filtering π'' for $\mathcal{R}$ as the composition of π and π'. For a precise definition, let $\flat$ denote the unmarking operation, i.e., $f^\flat = f$ and $(f^\sharp)^\flat = f$ for all $f \in \mathcal{F}$. Then for all $f \in \mathcal{F}^\sharp$ we define

$$\pi''(f) = \begin{cases} [i_{j_1}, \ldots, i_{j_k}] & \text{if } \pi(f^\flat) = [i_1, \ldots, i_m] \text{ and } \pi'(f) = [j_1, \ldots, j_k], \\ i_j & \text{if } \pi(f^\flat) = [i_1, \ldots, i_m] \text{ and } \pi'(f) = j, \\ i & \text{if } \pi(f) = i. \end{cases}$$

It is not difficult to show that $\pi''(t) = \pi'(\pi(t))$ and $\pi''(t^\sharp) = \pi'(\pi(t)^\sharp)$ for all terms t without marked symbols. We claim that π'' and $\succ$ satisfy the constraints for $\mathcal{C}$, i.e., $\pi''(\mathcal{R} \cup \mathcal{C}) \subseteq \succeq$ and $\pi''(\mathcal{C}) \cap \succ \neq \varnothing$. These two properties follow from the two assumptions $\pi'(\mathrm{AFT}_\pi(\mathcal{R}) \cup \pi(\mathcal{C})) \subseteq \succeq$ and $\pi'(\pi(\mathcal{C})) \cap \succ \neq \varnothing$ in conjunction with the obvious inclusion $\pi(\mathcal{R}) \subseteq \mathrm{AFT}_\pi(\mathcal{R})$. □

Theorem 8 also holds for the *estimated* dependency graph instead of the real dependency graph.

5 Conclusion

In this paper, we have compared two transformational techniques for termination proofs, viz. dummy elimination [11] and the argument filtering transformation [16], with the dependency pair technique of Arts and Giesl [1–3]. Essentially, all these techniques transform a given TRS into new inequalities or rewrite systems which then have to be oriented by suitable well-founded orders. Virtually all well-founded orders which can be generated automatically are simplification orders. As our focus was on *automated* termination proofs, we therefore investigated the strengths of these three techniques when combined with simplification orders.

To that end, we showed that whenever an automated termination proof is possible using dummy elimination or the argument filtering transformation, then a corresponding termination proof can also be obtained by dependency pairs. Thus, the dependency pair technique is more powerful than dummy elimination or the argument filtering transformation on their own.

Moreover, we examined whether dummy elimination or the argument filtering transformation would at least be helpful as a preprocessing step to the dependency pair technique. We proved that for dummy elimination and for an argument filtering transformation with a non-collapsing argument filtering, this is not the case. In fact, whenever there is a (pre)order satisfying the dependency pair constraints for the rewrite system resulting from dummy elimination or a non-collapsing argument filtering transformation, then the *same* (pre)order also satisfies the dependency pair constraints for the original TRS.

As can be seen from the proofs of our main theorems, this latter result even holds for arbitrary (i.e., non-simplification) (pre)orders. Thus, in particular, Theorems 5 and 8 also hold for *DP quasi-simple termination* [13]. This notion captures those TRSs where the dependency pair constraints are satisfied by an arbitrary simplification preorder $\succsim$ (instead of just a preorder $\succeq$ where the equivalence relation is syntactic equality as in *DP simple termination*).

Future work will include a further investigation on the usefulness of collapsing argument filtering transformations as a preprocessing step to dependency pairs. Note that our counterexample $\mathcal{R}_2$ is DP quasi-simply terminating (but not DP simply terminating). In other words, at present it is not clear whether the argument filtering transformation is useful as a preprocessing step to the dependency pair technique if one admits arbitrary simplification *pre*orders to solve the generated constraints. However, an extension of Theorem 8 to DP quasi-simple termination and to *collapsing* argument filterings π is not straightforward, since clusters of dependency pairs in $\mathcal{R}$ may disappear in $\mathrm{AFT}_\pi(\mathcal{R})$ (i.e., Lemma 7 does not hold for collapsing argument filterings). We also intend to examine the relationship between dependency pairs and other transformation techniques such as "freezing" [20].

Acknowledgements. Jürgen Giesl is supported by the DFG under grant GI 274/4-1. Aart Middeldorp is partially supported by the Grant-in-Aid for Scientific Research C(2) 11680338 of the Ministry of Education, Science, Sports and Culture of Japan.

References

1. T. Arts and J. Giesl, *Automatically Proving Termination where Simplification Orderings Fail*, Proc. 7th TAPSOFT, Lille, France, LNCS 1214, pp. 261–273, 1997.
2. T. Arts and J. Giesl, *Termination of Term Rewriting Using Dependency Pairs*, Theoretical Computer Science 236, pp. 133–178, 2000. Long version available at `www.inferenzsysteme.informatik.tu-darmstadt.de/~reports/ibn-97-46.ps`.
3. T. Arts and J. Giesl, *Modularity of Termination Using Dependency Pairs*, Proc. 9th RTA, Tsukuba, Japan, LNCS 1379, pp. 226–240, 1998.
4. F. Baader and T. Nipkow, *Term Rewriting and All That*, Cambridge University Press, 1998.
5. F. Bellegarde and P. Lescanne, *Termination by Completion*, Applicable Algebra in Engineering, Communication and Computing 1, pp. 79–96, 1990.
6. A. Ben Cherifa and P. Lescanne, *Termination of Rewriting Systems by Polynomial Interpretations and its Implementation*, Science of Computer Programming 9, pp. 137–159, 1987.
7. N. Dershowitz, *Orderings for Term-Rewriting Systems*, Theoretical Computer Science 17, pp. 279–301, 1982.
8. N. Dershowitz, *Termination of Rewriting*, Journal of Symbolic Computation 3, pp. 69–116, 1987.
9. J. Dick, J. Kalmus, and U. Martin, *Automating the Knuth Bendix Ordering*, Acta Informatica 28, pp. 95–119, 1990.
10. M.C.F. Ferreira, *Termination of Term Rewriting: Well-foundedness, Totality and Transformations*, Ph.D. thesis, Utrecht University, The Netherlands, 1995.
11. M.C.F. Ferreira and H. Zantema, *Dummy Elimination: Making Termination Easier*, Proc. 10th FCT, Dresden, Germany, LNCS 965, pp. 243–252, 1995.
12. J. Giesl, *Generating Polynomial Orderings for Termination Proofs*, Proc. 6th RTA, Kaiserslautern, Germany, LNCS 914, pp. 426–431, 1995.
13. J. Giesl and E. Ohlebusch, *Pushing the Frontiers of Combining Rewrite Systems Farther Outwards*, Proc. 2nd FROCOS, 1998, Amsterdam, The Netherlands, Studies in Logic and Computation 7, Research Studies Press, Wiley, pp. 141–160, 2000.
14. S. Kamin and J.J. Lévy, *Two Generalizations of the Recursive Path Ordering*, unpublished manuscript, University of Illinois, USA, 1980.
15. D.E. Knuth and P. Bendix, *Simple Word Problems in Universal Algebras*, in: Computational Problems in Abstract Algebra (ed. J. Leech), Pergamon Press, pp. 263–297, 1970.
16. K. Kusakari, M. Nakamura, and Y. Toyama, *Argument Filtering Transformation*, Proc. 1st PPDP, Paris, France, LNCS 1702, pp. 48–62, 1999.
17. D. Lankford, *On Proving Term Rewriting Systems are Noetherian*, Report MTP-3, Louisiana Technical University, Ruston, USA, 1979.
18. A. Middeldorp, H. Ohsaki, and H. Zantema, *Transforming Termination by Self-Labelling*, Proc. 13th CADE, New Brunswick (New Jersey), USA, LNAI 1104, pp. 373–387, 1996.
19. J. Steinbach, *Simplification Orderings: History of Results*, Fundamenta Informaticae 24, pp. 47–87, 1995.
20. H. Xi, *Towards Automated Termination Proofs Through "Freezing"*, Proc. 9th RTA, Tsukuba, Japan, LNCS 1379, pp. 271–285, 1998.
21. H. Zantema, *Termination of Term Rewriting: Interpretation and Type Elimination*, Journal of Symbolic Computation 17, pp. 23–50, 1994.
22. H. Zantema, *Termination of Term Rewriting by Semantic Labelling*, Fundamenta Informaticae 24, pp. 89–105, 1995.