Goal-directed and Relative Dependency Pairs for Proving the Termination of Narrowing*

José Iborra[1], Naoki Nishida[2], German Vidal[1]

[1] DSIC, Universidad Politécnica de Valencia, Spain
{jiborra,gvidal}@dsic.upv.es
[2] Graduate School of Information Science, Nagoya University, Nagoya, Japan
nishida@is.nagoya-u.ac.jp

Abstract. In this work, we first consider a *goal-oriented* extension of the dependency pair framework for proving termination w.r.t. a given set of initial terms. Then, we introduce a new result for proving *relative* termination in terms of a dependency pair problem. Both contributions put together allow us to define a simple and powerful approach to analyzing the termination of *narrowing*, an extension of rewriting that replaces matching with unification in order to deal with logic variables. Our approach could also be useful in other contexts where considering termination w.r.t. a given set of terms is also natural (e.g., proving the termination of functional programs).

1 Introduction

Proving that a program terminates is a fundamental problem that has been extensively studied in almost all programming paradigms. In *term rewriting*, where termination analysis has attracted considerable attention (see, e.g., the surveys of Dershowitz [8] and Steinbach [23]), the termination of a rewrite system is usually proved for all possible reduction sequences.

In some cases, however, one is only interested in those sequences that start from a distinguished set of terms. This case has been already considered in some previous works, e.g., for proving the termination of logic programs [20], for proving the termination of Haskell programs [10], and for proving the termination of *narrowing* [25], an extension of rewriting to deal with logic variables. Unfortunately, these works do not focus on proving termination from an initial set of terms—only consider this problem to some extent—and are difficult to generalize.

In this paper, we first extend the well-known *dependency pair* framework [3, 11] for proving the termination of rewriting in order to only consider derivations

* This work has been partially supported by the Spanish *Ministerio de Ciencia e Innovación* under grant TIN2008-06622-C03-02, by *Generalitat Valenciana* under grant ACOMP/2009/017, and by *UPV* (programs PAID-05-08 and PAID-06-08). The second author has been partially supported by a grant from Nagoya Industrial Science Research Institute.

from a given initial set of terms. The fundamental improvements are twofold: firstly, we introduce a notion of chain which considers only reachable loops, thus reducing the number of pairs to consider; secondly, we also present a notion of usable rules that regards as usable only those rules which occur in the derivation from an initial term, allowing us to reduce the number of rules.

As a second contribution of this paper, we study a direct application of the dependency pair approach to the solution of relative termination problems when the involved TRSs form a hierarchical combination. Roughly speaking, we can study whether $\mathcal{R}$ terminates relative to $\mathcal{B}$ (i.e., whether all $\to_{\mathcal{R}} \cup \to_{\mathcal{B}}$ reductions contain only finitely many $\to_{\mathcal{R}}$ steps) in those cases where $\mathcal{B}$ does not make calls to functions defined in $\mathcal{R}$. Although this application is arguably folklore in the literature (see, e.g., the work of [24]), to our knowledge this is the first time that the necessary conditions have been ascertained and proved in a formal publication.

Finally, we illustrate the usefulness of our developments by applying them to proving the termination of narrowing starting from initial terms. Our results are more general and potentially more accurate than previous approaches (e.g., [25]). Moreover, our approach could also be useful in other contexts where considering termination w.r.t. a given set of terms is also a natural requirement, like the approach to proving the termination of Haskell programs of [10] or that to proving the termination of logic programs by translating them to rewrite systems of [20].

The paper is organized as follows. After introducing some preliminaries in the next section, we present the goal-directed dependency pair framework in Section 3. Then, Section 4 first states a useful result for proving the relative termination of a rewrite system and, then, presents a new approach for proving the termination of narrowing. Finally, Section 5 reports on the implementation of a termination prover based on the ideas of this paper and concludes. An extended version including proofs of technical results can be found in [15].

2 Preliminaries

We assume familiarity with basic concepts of term rewriting and narrowing. We refer the reader to, e.g., [4] and [13] for further details.

Terms and Substitutions. A *signature* $\mathcal{F}$ is a set of function symbols. We often write $\mathsf{f}/n \in \mathcal{F}$ to denote that the arity of function f is n. Given a set of variables $\mathcal{V}$ with $\mathcal{F} \cap \mathcal{V} = \emptyset$, we denote the domain of *terms* by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We assume that $\mathcal{F}$ always contains at least one constant $\mathsf{f}/0$. We use $\mathsf{f}, \mathsf{g}, \ldots$ to denote functions and $x, y, \ldots$ to denote variables. A *position* p in a term t is represented by a finite sequence of natural numbers, where ϵ denotes the root position. Positions are used to address the nodes of a term viewed as a tree. The root symbol of a term t is denoted by $\mathsf{root}(t)$. We let $t|_p$ denote the *subterm* of t at position p and $t[s]_p$ the result of *replacing the subterm* $t|_p$ by the term s. $\mathcal{V}\mathsf{ar}(t)$

denotes the set of variables appearing in t. A term t is *ground* if $\mathcal{V}\mathsf{ar}(t) = \emptyset$. We write $\mathcal{T}(\mathcal{F})$ as a shorthand for the set of ground terms $\mathcal{T}(\mathcal{F}, \emptyset)$.

A *substitution* $\sigma : \mathcal{V} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{V})$ is a mapping from variables to terms such that $\mathcal{D}\mathsf{om}(\sigma) = \{x \in \mathcal{V} \mid x \neq \sigma(x)\}$ is its domain. The set of variables introduced by a substitution σ is denoted by $\mathcal{R}\mathsf{an}(\sigma) = \cup_{x \in \mathcal{D}\mathsf{om}(\sigma)} \mathcal{V}\mathsf{ar}(x\sigma)$. Substitutions are extended to morphisms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ in the natural way. We denote the application of a substitution σ to a term t by $t\sigma$ (rather than $\sigma(t)$). The identity substitution is denoted by *id*.

TRSs and Rewriting. A set of rewrite rules $l \to r$ such that l is a non-variable term and r is a term whose variables appear in l is called a *term rewriting system* (TRS for short); terms l and r are called the left-hand side and the right-hand side of the rule, respectively. We restrict ourselves to finite signatures and TRSs. Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined* symbols $\mathcal{D}$ are the root symbols of the left-hand sides of the rules and the *constructors* are $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$.

We use the notation $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ to point out that $\mathcal{D}$ are the defined function symbols and $\mathcal{C}$ are the constructors of a signature $\mathcal{F}$, with $\mathcal{D} \cap \mathcal{C} = \emptyset$. The domains $\mathcal{T}(\mathcal{C}, \mathcal{V})$ and $\mathcal{T}(\mathcal{C})$ denote the sets of *constructor terms* and *ground constructor terms*, respectively. A substitution σ is (ground) *constructor*, if $x\sigma$ is a (ground) constructor term for all $x \in \mathcal{D}\mathsf{om}(\sigma)$.

A TRS $\mathcal{R}$ is a *constructor system* if the left-hand sides of its rules have the form $\mathsf{f}(s_1, \ldots, s_n)$ where s_i are constructor terms, i.e., $s_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$, for all $i = 1, \ldots, n$. A term t is *linear* if every variable of $\mathcal{V}$ occurs at most once in t. A TRS $\mathcal{R}$ is *left-linear* if l is linear for every rule $l \to r \in \mathcal{R}$.

For a TRS $\mathcal{R}$, we define the associated rewrite relation $\to_{\mathcal{R}}$ as follows: given terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have $s \to_{\mathcal{R}} t$ iff there exists a position p in s, a rewrite rule $l \to r \in \mathcal{R}$ and a substitution σ with $s|_p = l\sigma$ and $t = s[r\sigma]_p$; the rewrite step is often denoted by $s \to_{p, l \to r} t$ to make explicit the position and rule used in this step. The instantiated left-hand side $l\sigma$ is called a *redex*.

A term t is called *irreducible* or in *normal form* in a TRS $\mathcal{R}$ if there is no term s with $t \to_{\mathcal{R}} s$. A *derivation* is a (possibly empty) sequence of rewrite steps. Given a binary relation $\to$, we denote by $\to^+$ the transitive closure of $\to$ and by $\to^*$ its reflexive and transitive closure. Thus $t \to_{\mathcal{R}}^* s$ means that t can be reduced to s in $\mathcal{R}$ in zero or more steps; we also use $t \to_{\mathcal{R}}^n s$ to denote that t can be reduced to s in exactly n rewrite steps.

Narrowing. Given a TRS $\mathcal{R}$ and two terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have that $s \leadsto_{\mathcal{R}} t$ is a *narrowing step* iff there exist

- a non-variable position p of s,
- a variant $R = (l \to r)$ of a rule in $\mathcal{R}$,
- a substitution $\sigma = \mathsf{mgu}(s|_p, l)$ which is the most general unifier of $s|_p$ and l,

and $t = (s[r]_p)\sigma$. We often write $s \leadsto_{p,R,\theta} t$ (or simply $s \leadsto_\theta t$) to make explicit the position, rule, and substitution of the narrowing step, where $\theta = \sigma\restriction_{\mathcal{V}\mathsf{ar}(s)}$. A *narrowing derivation* $t_0 \leadsto_\sigma^* t_n$ denotes a sequence of narrowing steps $t_0 \leadsto_{\sigma_1} \ldots \leadsto_{\sigma_n} t_n$ with $\sigma = \sigma_n \circ \cdots \circ \sigma_1$ (if $n = 0$ then $\sigma = id$).

3 Goal-directed Dependency Pairs

In this section, we present a goal-directed extension of the well-known *dependency pair* (DP) framework [3, 11]. Our framework is *goal-directed* since only derivations starting from a given set of terms, denoted by means of an *initial goal*, are considered.

Definition 1 (initial goal). *Let $\mathcal{R}$ be a TRS over $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and $t_0 = \mathsf{f}(x_1, \ldots, x_n)$ be a term. We say that t_0 is an initial goal for $\mathcal{R}$ if $\mathsf{f} \in \mathcal{D}$ is a defined function symbol and $x_1, \ldots, x_n \in \mathcal{V}$ are distinct variables.*

Intuitively speaking, an initial goal t_0 represents the set $\lceil t_0 \rceil$ of (non necessarily ground) *constructor* instances of the term t_0, i.e.,

$$\lceil t_0 \rceil \ = \ \{ \ t_0\sigma \mid \sigma \text{ is a constructor substitution } \}$$

For instance, given the signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ with $\mathsf{f}/1 \in \mathcal{D}$ and $\mathsf{z}/0, \mathsf{s}/1 \in \mathcal{C}$, the initial goal $\mathsf{f}(x)$ represents the set $\lceil \mathsf{f}(x) \rceil = \{\mathsf{f}(x), \mathsf{f}(\mathsf{z}), \mathsf{f}(\mathsf{s}(x)), \mathsf{f}(\mathsf{s}(\mathsf{z})), \ldots\}$. In the following, we say that a set of terms T is terminating if there is no term $t_1 \in T$ such that an infinite sequence of the form $t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \cdots$ exists.

It is worthwhile to observe that there is no loss of generality in our notion of initial goal since any arbitrary term t could be used as an initial goal by just adding a new rule, $\mathsf{goal}(x_1, \ldots, x_n) \to t$, where goal is a fresh function symbol with $\mathcal{V}\mathsf{ar}(t) = \{x_1, \ldots, x_n\}$, and then considering $\mathsf{goal}(x_1, \ldots, x_n)$ as initial goal.

Two key ingredients of the DP approach are the notion of *dependency pairs* and that of *chains* of dependency pairs. The first notion remains unchanged in our setting. Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, for each $\mathsf{f}/n \in \mathcal{F}$, we let $\mathsf{f}^\sharp/n$ be a fresh *tuple symbol*; we often write F instead of $\mathsf{f}^\sharp$ in the examples. Given a term $\mathsf{f}(t_1, \ldots, t_n)$ with $\mathsf{f} \in \mathcal{D}$, we let $t^\sharp$ denote $\mathsf{f}^\sharp(t_1, \ldots, t_n)$.

Definition 2 (dependency pair [3]). *Given a TRS $\mathcal{R}$ over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$, the associated set of dependency pairs, $DP(\mathcal{R})$, is defined as follows:*[3]

$$DP(\mathcal{R}) = \{l^\sharp \to t^\sharp \mid l \to r \in \mathcal{R},\ r|_p = t,\ \textit{and } \mathsf{root}(t) \in \mathcal{D}\}$$

In order to formalize our definition of chains, we first introduce the notion of *reachable calls* from a given term. Formally, given a TRS $\mathcal{R}$ and a term t, we define the set of reachable calls, $calls_{\mathcal{R}}(t)$, from t in $\mathcal{R}$ as follows:

$$calls_{\mathcal{R}}(t) = \{ \ s|_p \ \mid \ t \to^*_{\mathcal{R}} s, \text{ with } \mathsf{root}(s|_p) \in \mathcal{D} \text{ for some position } p \ \}$$

Also, given a set of terms T, we let $calls_{\mathcal{R}}(T) = \bigcup_{t \in T} calls_{\mathcal{R}}(t)$.

Definition 3 (chain). *Let $\mathcal{R}$ and $\mathcal{P}$ be TRSs over the signatures $\mathcal{F}$ and $\mathcal{F}^\sharp$, respectively. Let t_0 be an initial goal. A (possibly infinite) sequence of pairs $s_1 \to t_1$, $s_2 \to t_2$, ... from $\mathcal{P}$ is a $(t_0, \mathcal{P}, \mathcal{R})$-chain if there is a substitution $\sigma : \mathcal{V} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{V})$ such that the following conditions hold:*[4]

[3] Note that if $\mathcal{R}$ is a TRS, so is $DP(\mathcal{R})$.

[4] As in [3], we assume fresh variables in every (occurrence of a) dependency pair and that the domain of substitutions may be infinite.

- *there exists a term $s \in calls_{\mathcal{R}}(\lceil t_0 \rceil)$ such that $s^\sharp = s_1\sigma$ and*
- *$t_i\sigma \to^*_{\mathcal{R}} s_{i+1}\sigma$ for every two consecutive pairs in the sequence.*

The chain is minimal iff all $t_i\sigma$ are terminating w.r.t. $\mathcal{R}$.

Note that the only difference with the standard notion of chains is that only chains which are *reachable* from (an instance of) the initial goal are considered.

Now, without further ado, we introduce our termination criterion:

Theorem 1 (termination criterion). *Let $\mathcal{R}$ be a TRS and t_0 be an initial goal. All derivations starting from a term in $\lceil t_0 \rceil$ in $\mathcal{R}$ are finite iff there are no infinite minimal $(t_0, DP(\mathcal{R}), \mathcal{R})$-chains.*

As in the standard DP framework [11], and in order to ease the automation of the proof search, we introduce a *goal-directed* DP (GDP) framework as follows:

Definition 4 (GDP problems and processors). *A GDP problem is a tuple $(t_0, \mathcal{P}, \mathcal{R}, f)$ consisting of two TRSs $\mathcal{R}$ and $\mathcal{P}$ over the signatures $\mathcal{F}$ and $\mathcal{F}^\sharp$, respectively, an initial goal t_0 for $\mathcal{R}$, and a minimality flag $f \in \{\mathbf{m}, \mathbf{a}\}$ where* **m** *and* **a** *stand for "minimal" and "arbitrary", respectively. A GDP problem is finite if there is no associated infinite (minimal if f is* **m***) $(t_0, \mathcal{P}, \mathcal{R})$-chain, and infinite if it is not finite or if $\lceil t_0 \rceil$ does not terminate in $\mathcal{R}$. A (standard) DP problem is a tuple $(\mathcal{P}, \mathcal{R}, f)$ consisting of $\mathcal{P}$, $\mathcal{R}$ and f described above.*

A GDP processor *is a function Proc which takes a GDP problem and returns either a new set of GDP problems or fails. Proc is sound if for any GDP problem $\mathcal{M}$, $\mathcal{M}$ is finite whenever all GDP problems in $Proc(\mathcal{M})$ are finite. Proc is complete if for any GDP problem $\mathcal{M}$, $\mathcal{M}$ is infinite whenever $Proc(\mathcal{M})$ fails or contains an infinite GDP problem.*

Following [11], one can construct a tree whose root is labeled with the problem $(t_0, DP(\mathcal{R}), \mathcal{R}, \mathbf{m})$ and whose nodes are produced by application of sound GDP processors. If no leaf of the tree is a failure, then $\lceil t_0 \rceil$ is terminating in $\mathcal{R}$. Otherwise, if all the processors used on the path from the root to the failure node are complete, then $\lceil t_0 \rceil$ is not terminating in $\mathcal{R}$.

3.1 Dependency Graphs

The auxiliary notion of *initial pairs* denotes the pairs that match an initial goal:

Definition 5 (initial pairs). *Let $(t_0, \mathcal{P}, \mathcal{R}, f)$ be a GDP problem. The associated set of initial pairs is given by $\{s \to t \in DP(\mathcal{R}) \mid t_0^\# \sigma = s$ for some subst. $\sigma\}$.*

Note that the set of initial pairs associated to a GDP problem $(t_0, \mathcal{P}, \mathcal{R}, f)$ need not belong to the current set of pairs $\mathcal{P}$.

We now recall the standard notion of dependency graph [11]:

Definition 6 (dependency graph). *Given a GDP problem $(t_0, \mathcal{P}, \mathcal{R}, f)$, its dependency graph is a directed graph where the nodes are the pairs of $\mathcal{P}$, and there is an edge from $s \to t \in \mathcal{P}$ to $u \to v \in \mathcal{P}$ iff $s \to t, u \to v$ is a $(t_0, \mathcal{P}, \mathcal{R})$-chain.*

Although we consider the standard definition of dependency graph, since our notion of chain is different, the dependency graph of a GDP problem may contain less pairs than the dependency graph of the corresponding standard DP problem; in our case, all the pairs which are not reachable from the initial goal are removed from the dependency graph.

Dependency graphs are not generally computable and, thus, several approximations have been defined. Instead of adapting one of these approximations, we show how any arbitrary approach can easily be reused in our context:

Theorem 2 (estimated dependency graph). *Let $(t_0, \mathcal{P}, \mathcal{R}, f)$ be a GDP problem. Let $\mathcal{G}_0$ and $\mathcal{G}$ be estimated dependency graphs (according to [11]) for the DP problems $(DP(\mathcal{R}), \mathcal{R}, f)$ and $(\mathcal{P}, \mathcal{R}, f)$, respectively.*

Assuming that the nodes of $\mathcal{G}_0$ and $\mathcal{G}$ are shared in the obvious way, the estimated dependency graph of the GDP problem $(t_0, \mathcal{P}, \mathcal{R}, f)$ is the restriction of $\mathcal{G}$ to those nodes which are reachable from an initial pair in $\mathcal{G}_0$.

The graph obtained is an over-estimation of the dependency graph.

Example 1. Consider the following dependency graph $\mathcal{G}_0$ whose nodes are labeled with $(0), (1), \ldots, (7)$:

$$\begin{tikzcd}
\boxed{1} \arrow[r, bend left] & \boxed{2} \arrow[l, bend left] & \boxed{3} \arrow[r, bend left] & \boxed{4} \arrow[l, bend left] & \boxed{0} \arrow[l] \arrow[r] & \boxed{5} \arrow[r] & \boxed{6} \arrow[r, bend left] & \boxed{7} \arrow[l, bend left]
\end{tikzcd}$$

Given a GDP problem where the only initial pair is (0), we have that pairs (1) and (2) do not belong to its dependency graph since they are not reachable.

By using the estimated dependency graph, it is immediate to define a sound GDP processor that takes a GDP problem $(t_0, \mathcal{P}, \mathcal{R}, f)$ and divides the problem into its strongly connected components (SCC) as usual. Note that, in contrast to the standard processor, we remove those SCCs which are not reachable from the initial pairs.

3.2 Usable Rules

Another way for removing pairs from $\mathcal{P}$ is based on the notion of *reduction pair* $(\succsim, \succ)$.[5] For this purpose, we first need to introduce the notion of *argument filtering* [16]. An argument filtering over a signature $\mathcal{F}$ is a function π such that, for every symbol $\mathsf{f}/n \in \mathcal{F}$, we have either $\pi(\mathsf{f}) \in \{1, \ldots, n\}$ or $\pi(\mathsf{f}) \subseteq \{1, \ldots, n\}$. Argument filterings are extended to terms as follows:[6]

- $\pi(x) = x$ for all $x \in \mathcal{V}$;

[5] A pair of orders $(\succsim, \succ)$ is a reduction pair if $\succsim$ is a quasi-order and $\succ$ is a well-founded order where $\succsim$ is closed under contexts, and both $\succsim$ and $\succ$ are closed under substitutions and compatible (i.e., $\succsim \circ \succ \subseteq \succ$ and $\succ \circ \succsim \subseteq \succ$ but $\succsim \subseteq \succ$ is not necessary) [16].

[6] By abuse of notation, we keep the same symbol for the original function and the filtered function with a possibly different arity.

- $\pi(\mathsf{f}(t_1,\ldots,t_n)) = \pi(t_i)$ if $\pi(\mathsf{f}) = i$;
- $\pi(\mathsf{f}(t_1,\ldots,t_n)) = \mathsf{f}(\pi(t_{i_1}),\ldots,\pi(t_{i_m}))$ if $\pi(\mathsf{f}) = \{i_1,\ldots,i_m\}$ and $1 \le i_i \le n$.

Given a TRS $\mathcal{R}$, we let $\pi(\mathcal{R}) = \{\pi(l) \to \pi(r) \mid l \to r \in \mathcal{R}\}$.

For any relation $\succ$, we let $\succ_\pi$ be the relation where $t \succ_\pi u$ holds iff $\pi(t) \succ \pi(u)$. For any TRS $\mathcal{P}$ and any relation $\succ$, we 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. $\succ$.

Theorem 3 (reduction pair processor). *Let $(\succsim, \succ)$ be a reduction pair and π be an argument filtering. Given a GDP problem $(t_0, \mathcal{P}, \mathcal{R}, f)$, if Proc returns:*

- *$(t_0, \mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R}, f)$, if $\mathcal{P}_{\succ_\pi} \cup \mathcal{P}_{\succsim_\pi} = \mathcal{P}$, $\mathcal{P}_{\succ_\pi} \neq \emptyset$, and $\mathcal{R}_{\succsim_\pi} = \mathcal{R}$;*
- *$(t_0, \mathcal{P}, \mathcal{R}, f)$, otherwise;*

then Proc is sound and complete.

Basically, this processor can be used to remove the strictly decreasing pairs of $\mathcal{P}$ when the remaining pairs of $\mathcal{P}$ and all rules of $\mathcal{R}$ are weakly decreasing. In fact, a weak decrease is not required for all the rules but only for the *usable* rules [12]. These rules are a superset of the rules that may be used to connect dependency pairs in a chain. For GDP problems, a notion of usable rules also removes those rules which are not reachable from the initial goal.

There are several approaches for approximating the usable rules of a problem. In this section we show how any of them can be adapted to our goal-directed setting, using as an example the usable rules of [12]. For this purpose, we first need the following auxiliary notion: given an argument filtering π and a term t, we let $RegPos_\pi(t)$ denote the *regarded positions* of t w.r.t. π; formally, $RegPos_\pi(t) = \{\epsilon\} \cup \{i.p \mid t = \mathsf{f}(t_1, \cdots, t_n), p \in RegPos_\pi(t_i)$, and $i \in \pi(f)\}$. In essence, the regarded positions of t w.r.t. π are those positions of t which are not dropped by the filtering π.

In the following, given a TRS $\mathcal{R}$ and symbol $\mathsf{f} \in \mathcal{F}$, we let $\mathsf{Def}_\mathcal{R}(\mathsf{f}) = \{l \to r \in \mathcal{R} \mid \mathsf{root}(l) = \mathsf{f}\}$ and $\mathcal{R}'_\mathsf{f} = \mathcal{R} \setminus \mathsf{Def}_\mathcal{R}(\mathsf{f})$.

Definition 7 (estimated usable rules w.r.t. an argument filtering [12]). *For any TRS $\mathcal{R}$ and any argument filtering π, we define*

- *$\mathcal{U}^\pi_\mathcal{R}(x) = \emptyset$ for $x \in \mathcal{V}$.*
- *$\mathcal{U}^\pi_\mathcal{R}(\mathsf{f}(t_1, \cdots, t_n)) = \mathsf{Def}_\mathcal{R}(\mathsf{f}) \cup \bigcup_{l \to r \in \mathsf{Def}_\mathcal{R}(\mathsf{f})} \mathcal{U}^\pi_{\mathcal{R}'_\mathsf{f}}(r) \cup \bigcup_{i \in RegPos_\pi(\mathsf{f})} \mathcal{U}^\pi_{\mathcal{R}'_\mathsf{f}}(t_i)$*

For any set of rules $\mathcal{P}$ we define $\mathcal{U}^\pi_\mathcal{R}(\mathcal{P}) = \bigcup_{l \to r \in \mathcal{P}} \mathcal{U}^\pi_\mathcal{R}(r)$.

Definition 8 (estimated goal-directed usable rules w.r.t. an argument filtering). *For a graph $\mathcal{G}$ and two set of nodes $\mathcal{I}$ and $\mathcal{P}$, let $PATH_\mathcal{G}(\mathcal{I}, \mathcal{P})$ denote the smallest set of nodes of $\mathcal{G}$ which contains both $\mathcal{I}$ and $\mathcal{P}$ and all the nodes which are in a path from some node in $\mathcal{I}$ to some node in $\mathcal{P}$ in $\mathcal{G}$.*

Let $(t_0, \mathcal{P}, \mathcal{R}, f)$ be a GDP problem, $\mathcal{P}_0$ its initial pairs, π an argument filtering, and $\mathcal{G}_0$ a (standard) estimated dependency graph for the DP problem

$(DP(\mathcal{R}), \mathcal{R}, f)$. *The goal-directed usable rules of* $\mathcal{P}$ *in* $\mathcal{R}$ *w.r.t.* π *and* t_0 *are defined as follows:*

$$\mathcal{GU}(t_0, \mathcal{P}, \mathcal{R}, \pi) = \begin{cases} \mathcal{U}^{\pi}_{\mathcal{R}}(PATH_{\mathcal{G}_0}(\mathcal{P}_0, \mathcal{P})) & \text{if } \mathcal{V}\mathsf{ar}(\pi(t)) \subseteq \mathcal{V}\mathsf{ar}(\pi(s)) \\ & \text{for all } s \to t \in PATH_{\mathcal{G}_0}(\mathcal{P}_0, \mathcal{P}) \\ \mathcal{R} & \text{otherwise} \end{cases}$$

Example 2. Let us consider again the dependency graph $\mathcal{G}_0$ from Example 1, together with the set $\mathcal{I} = \{(0)\}$. Then,

- if $\mathcal{P}$ is the SCC $\{(6), (7)\}$ then $PATH_{\mathcal{G}_0}(\mathcal{I}, \mathcal{P}) = \{(0), (5), (6), (7)\}$;
- if $\mathcal{P} = \{(3), (4)\}$, then $PATH_{\mathcal{G}_0}(\mathcal{I}, \mathcal{P}) = \{(0), (3), (4)\}$.

The goal-directed usable rules coincide with the usable rules for all the chains from an initial pair to the pairs in $\mathcal{P}$. While this means that they are a superset of the usable rules of $\mathcal{P}$, they are still advantageous as they are applicable in cases where the usable rules are not. In particular, *minimality* is not required. This is critical in Section 4 where we consider the termination of narrowing as a problem of relative termination, since in this context minimality does not generally hold.

Theorem 4 (reduction pair processor with goal-directed usable rules w.r.t. argument filterings). *Let* $(\succsim, \succ)$ *be a reduction pair and* π *be an argument filtering. Given a GDP problem* $(t_0, \mathcal{P}, \mathcal{R}, f)$*, if Proc returns:*

- $(t_0, \mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R}, \mathbf{m})$*, if* f *is* $\mathbf{m}$*,* $\mathcal{P}_{\succ_\pi} \cup \mathcal{P}_{\succsim_\pi} = \mathcal{P}$*,* $\mathcal{R}_{\succsim_\pi} \supseteq \mathcal{U}^{\pi}_{\mathcal{R}}(\mathcal{P})$*, and* $\succsim$ *is* $\mathcal{C}_{\mathcal{E}}$*-compatible;*[7]
- $(t_0, \mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R}, \mathbf{a})$*, if* f *is* $\mathbf{a}$*,* $\mathcal{P}_{\succ_\pi} \cup \mathcal{P}_{\succsim_\pi} = \mathcal{P}$*, and* $\mathcal{R}_{\succsim_\pi} \supseteq \mathcal{GU}^{\pi}(t_0, \mathcal{P}, \mathcal{R}, \pi)$*;*
- $(t_0, \mathcal{P}, \mathcal{R}, f)$*, otherwise;*

then Proc is sound and complete.

Example 3. Consider the following GDP problem

$$(\mathsf{goal}(x), \{\mathsf{ADD}(\mathsf{s}(x), y) \to \mathsf{ADD}(x, y)\}, \mathcal{R}_{add}, \mathbf{a})$$

where:

$$\mathcal{R}_{add} = \left\{ \begin{array}{r} \mathsf{goal}(x) \to \mathsf{add}(x, \mathsf{gen}) \\ \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y)) \\ \mathsf{add}(\mathsf{zero}, y) \to y \\ \mathsf{gen} \to \mathsf{s}(\mathsf{gen}) \\ \mathsf{gen} \to \mathsf{zero} \end{array} \right\} \quad DP(\mathcal{R}_{add}) = \left\{ \begin{array}{c} \mathsf{GOAL}(x) \to \mathsf{ADD}(x, \mathsf{gen}) \\ \mathsf{GOAL}(x) \to \mathsf{GEN} \\ \mathsf{ADD}(\mathsf{s}(x), y) \to \mathsf{ADD}(x, y) \\ \mathsf{GEN} \to \mathsf{GEN} \end{array} \right\}$$

Using the default argument filtering which filters nothing, we have that

- the usable rules w.r.t. $\mathsf{goal}(x)$ include only the rules for gen.

[7] A quasi-rewrite order $\succsim$ is $\mathcal{C}_{\mathcal{E}}$*-compatible* if for a new function symbol c, $\mathsf{c}(x, y) \succsim x$ and $\mathsf{c}(x, y) \succsim y$.

Using the argument filtering defined as $\pi(\mathsf{add}) = \{1\}$ (*i.e.*, $\pi(\mathsf{add}(x, y)) = \mathsf{add}(x)$ and the identity otherwise), we have that

- the usable rules w.r.t. $\mathsf{goal}(x)$ are the empty set.

Since the rules of gen are increasing, the finiteness of the GDP problem can only be proved in the second case.

As this section has shown, using this notion of usable rules it is straightforward to adapt an existing reduction pair processor to the goal-directed setting. Adapting other processors to the framework is straightforward too, since every GDP chain is a DP chain and, thus, soundness is preserved as long as the processor does not introduce new pairs in the graph. In any case, discussing the details is out of the scope of this paper.

4 Goal-directed Termination of Narrowing

In this section, we consider the termination of *narrowing* [22] and show how this problem can be reduced to proving the *relative* termination (see below) of a TRS from an initial set of terms, so that the GDP framework introduced in the previous section can be steadily applied.

4.1 Relative Termination

First, we show that it is possible to cast a relative termination problem as a standard DP problem as long as the systems involved satisfy the condition that they form hierarchical combinations.

Definition 9 (hierarchical combination [19]). *A system $\mathcal{R}_0 \cup \mathcal{R}_1$ is the* hierarchical combination *(HC) of a base $\mathcal{R}_0$ over $\mathcal{F}_0 = \mathcal{D}_0 \uplus \mathcal{C}_0$ and an extension $\mathcal{R}_1$ over $\mathcal{F}_1 = \mathcal{D}_1 \uplus \mathcal{C}_0$ if and only if $\mathcal{D}_0 \cap \mathcal{D}_1 = \varnothing$ and $\mathcal{C}_0 \cap \mathcal{D}_1 = \varnothing$.*

Let us now recall the notion of relative termination:

Definition 10 (relative termination). *Given two relations $\to_R$ and $\to_E$ we define the compound relation $\to_R / \to_E$ as $\to_E^* \cdot \to_R \cdot \to_E^*$.*

Given two TRSs $\mathcal{R}_1$ and $\mathcal{R}_0$, we say that $\mathcal{R}_1$ terminates w.r.t. $\mathcal{R}_0$ if the relation $\to_{\mathcal{R}_1} / \to_{\mathcal{R}_0}$ is terminating, i.e., if every (possibly infinite) $\to_{\mathcal{R}_0} \cup \to_{\mathcal{R}_1}$ derivation contains only finitely many $\to_{\mathcal{R}_0}$ steps.

Note that sequences of $\to_{\mathcal{R}_0}$ steps are “collapsed” and seen as a single $\to_{\mathcal{R}_1} / \to_{\mathcal{R}_0}$ step. Hence, an infinite $\to_{\mathcal{R}_1} / \to_{\mathcal{R}_0}$ derivation must contain an infinite number of $\to_{\mathcal{R}_1}$ steps, and thus by assumption only finite $\to_{\mathcal{R}_0}$ subderivations.

We say that a term t is $\to_{\mathcal{R}}$-terminating w.r.t. $\mathcal{B}$ if there is no infinite $\to_{\mathcal{R}} / \to_{\mathcal{B}}$ derivation issuing from t. Similarly, we say that T is $\to_{\mathcal{R}}$-terminating w.r.t. $\mathcal{B}$ if every term in T is $\to_{\mathcal{R}}$-terminating w.r.t. $\mathcal{B}$.

We make use of the standard notion of minimal (*non-terminating*) term, i.e., a term which starts an infinite derivation while all its proper subterms are terminating. We say that a term that is not $\to_{\mathcal{R}}$-terminating w.r.t. $\mathcal{B}$ is $\to_{\mathcal{R}} / \to_{\mathcal{B}}$-*minimal* if all its proper subterms are $\to_{\mathcal{R}}$-terminating w.r.t. $\mathcal{B}$.

Lemma 1. *Let $\mathcal{R}$ and $\mathcal{B}$ be two TRSs over $\mathcal{F}_{\mathcal{R}}$ and $\mathcal{F}_{\mathcal{B}}$ respectively, such that $\mathcal{R} \cup \mathcal{B}$ is the HC of the base $\mathcal{B}$ and the extension $\mathcal{R}$. Every $\rightarrow_{\mathcal{R}}/\rightarrow_{\mathcal{B}}$-minimal term $t_0 \in \mathcal{T}(\mathcal{F}_{\mathcal{B}} \cup \mathcal{F}_{\mathcal{R}}, \mathcal{V})$ starting an infinite $\rightarrow_{\mathcal{R}}/\rightarrow_{\mathcal{B}}$ derivation is of the form $t_0 = \mathsf{f}(\overline{u})$, where f is a defined symbol from $\mathcal{F}_{\mathcal{R}}$.*

Now we state the main result of this section. In order to prove relative termination of a TRS $\mathcal{R}$ w.r.t. a TRS $\mathcal{B}$, as long as they form an HC, one only needs to prove that the pairs of $\mathcal{R}$ are strongly decreasing, while the pairs of $\mathcal{B}$ can be ignored, even if $\mathcal{B}$ is not terminating.

Theorem 5 (relative termination criterion). *Let $\mathcal{R}$ and $\mathcal{B}$ be two TRSs such that $\mathcal{R} \cup \mathcal{B}$ is the HC of the base $\mathcal{B}$ and the extension $\mathcal{R}$. Then, $\mathcal{R}$ terminates w.r.t. $\mathcal{B}$ if and only if there are no infinite $(DP(\mathcal{R}), \mathcal{R} \cup \mathcal{B})$-chains.*

Example 4. Let $\mathcal{R} = \{\mathsf{f} \rightarrow \mathsf{gen}\}$ and $\mathcal{B} = \{\mathsf{gen} \rightarrow \mathsf{f}\}$ be TRSs, which are trivially terminating. Moreover, the DP Problem $(DP(\mathcal{R}), \mathcal{R} \cup \mathcal{B}) = (\varnothing, \mathcal{R} \cup \mathcal{B})$ is trivially finite (here we assume that gen is a constructor symbol in $\mathcal{R}$, hence the set of dependency pairs $DP(\mathcal{R})$ is empty). However, $\mathcal{R}$ is not terminating w.r.t. $\mathcal{B}$ since we have the following infinite derivation: $\mathsf{f} \rightarrow \mathsf{gen} \rightarrow \mathsf{f} \rightarrow \ldots$

In contrast to the termination criterion in [9] for relative termination, the HC property is required in Theorem 5, i.e., the HC property is necessary to extend the DP framework for proving relative termination. Note also that it does not suffice to prove the absence of *minimal* chains, as the following example shows:

Example 5. Let $\mathcal{R} = \{\mathsf{f}(\mathsf{s}(x)) \rightarrow \mathsf{f}(x)\}$ and $\mathcal{B} = \{\mathsf{gen} \rightarrow \mathsf{s}(\mathsf{gen})\}$. We have that $DP(\mathcal{R}) = \{\mathsf{F}(\mathsf{s}(x)) \rightarrow \mathsf{F}(x)\}$ and there are no infinite minimal chains. However there is an infinite chain with $\sigma = \{x \mapsto \mathsf{gen}\}$.

The relative termination criterion can be combined with Theorem 1 for relative termination from an initial goal.

Corollary 1 (goal-directed relative termination criterion). *Let $\mathcal{R}$ and $\mathcal{B}$ be two TRSs such that $\mathcal{R} \cup \mathcal{B}$ is the HC of the base $\mathcal{B}$ and the extension $\mathcal{R}$. Then, all derivations starting from a term in $\lceil t_0 \rceil$ in $\mathcal{R}$ terminate w.r.t. $\mathcal{B}$ if and only if there are no infinite $(t_0, DP(\mathcal{R}), \mathcal{R} \cup \mathcal{B})$-chains.*

4.2 Termination of Narrowing via Relative Termination

Recently, [18, 25] introduced a termination analysis for narrowing which is roughly based on the following process[8]. First, following [2, 7], logic variables are replaced with a fresh function, called gen, which can be seen as a *data generator* that can be non-deterministically reduced to any ground (constructor) term. A first result relates the termination of narrowing in the original TRS and the *relative* termination of rewriting using occurrences of gen to replace logic variables. However,

[8] The termination analysis of logic programs of [20] follows a similar pattern but logic variables are replaced with *infinite* terms (the net effect, though, is similar).

in order to avoid dealing with relative termination, [25] considers the use of an argument filtering to filter away occurrences of gen in the considered computations so that relative termination and termination coincide. Finally, termination is analyzed using the DP framework [11] for proving the termination of rewriting over the filtered terms.

This approach has several problems all related to the use of an argument filtering to filter the occurrences of gen. The most important one is that the search for an argument filtering that allows to prove termination is exponential in the arities of the signature, and even worse, this search cannot be casted as an optimization problem. This leads to the application of complex heuristics (as in [20]) which complicate the approach and diminish the effectiveness of the automation. Another issue is related to collapsing rules. Consider, for instance, a *collapsing* rule, i.e., a rule of the form $\mathsf{f}(x, y) \to y$, together with the argument filtering $\pi(\mathsf{f}) = \{1\}$. The filtered rule $\mathsf{f}(x) \to y$ contains an extra variable, y, and no refinement[9] of π will be able to eliminate it.[10]

Here, we argue that there is a better way to approach this problem. Instead of using a global argument filtering to filter away occurrences of gen, we propose to not filter them at all, and instead use the GDP framework developed in Section 3. As we show next this effectively solves the mentioned issues.

In order to formalize our approach, we first need to recall some existing notation and terminology from the literature.

Given a left-linear constructor TRS $\mathcal{R}$ over the signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$, we define the generator of $\mathcal{R}$, $\mathsf{GEN}(\mathcal{R})$, as the following set of rules:

$$\mathsf{GEN}(\mathcal{R}) \;=\; \{\;\; \mathsf{gen} \to \mathsf{c}(\overbrace{\mathsf{gen}, \ldots, \mathsf{gen}}^{n \text{ times}}) \;\mid\; \mathsf{c}/n \in \mathcal{C},\; n \geqslant 0 \;\;\}$$

Given a left-linear constructor TRS $\mathcal{R}$, we denote by $\mathcal{R}_{\mathsf{gen}}$ the set of rules resulting from augmenting $\mathcal{R}$ with $\mathsf{GEN}(\mathcal{R})$, in symbols $\mathcal{R}_{\mathsf{gen}} = \mathcal{R} \cup \mathsf{GEN}(\mathcal{R})$.

Following [25], variables are then replaced by generators in the obvious way: given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we let $\widehat{t} = t\sigma$, with $\sigma = \{x \mapsto \mathsf{gen} \mid x \in \mathcal{V}\mathsf{ar}(t)\}$. Also, given a TRS $\mathcal{R}$, possibly with *extra variables*,[11] we denote by $\widehat{\mathcal{R}}$ the result of replacing every extra variable in $\mathcal{R}$ (if any) with gen.

Note that $\widehat{t}$ is ground for any term t since all variables occurring in t are replaced by the function gen. As for $\widehat{\mathcal{R}}$, we note that it contains no extra variables by definition.

The completeness of replacing logic variables by generators is stated in [2, 7]:

Lemma 2 (completeness). *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a term. If $s \leadsto_{p,R,\sigma} t$ in $\mathcal{R}$, then $\widehat{s} \to^*_{\mathsf{GEN}(\mathcal{R})} \widehat{s}\sigma \to_{p,R} \widehat{t}$ in $\mathcal{R}_{\mathsf{gen}}$.*

[9] An argument filtering π' is a refinement of another argument filtering π if it filters the same or more arguments, i.e., either $\pi'(\mathsf{f}) = \pi(\mathsf{f})$ or $\pi'(\mathsf{f}) \subseteq \pi(\mathsf{f})$ for every f.

[10] This is not a limitation of [20] since the considered rewrite systems that are produced from the translation of logic programs never have collapsing rules.

[11] Extra variables are variables that appear in the rhs of a rule but not in its lhs.

In the following, we say that a set of terms T is $\leadsto_{\mathcal{R}}$-terminating if there is no term $t_1 \in T$ such that an infinite sequence of the form $t_1 \leadsto_{\mathcal{R}} t_2 \leadsto_{\mathcal{R}} \ldots$ exists.

In [25], the (possibly infinite) set of initial terms T was described by means of an *abstract* term $\mathsf{f}(m_1, \ldots, m_n)$, where f is a defined function symbol and $m_1, \ldots, m_n$ are either $\mathbf{g}$ (a definitely **g**round constructor term) or $\mathbf{v}$ (a possibly **v**ariable constructor term). Given an abstract term t^α, we let $\gamma(t^\alpha)$ denote the set of terms that can be obtained by replacing every argument $\mathbf{g}$ with a ground constructor term and every argument $\mathbf{v}$ with any arbitrary constructor term. Then, [25] shows that the termination of narrowing can be recast in terms of the relative termination of rewriting as follows:

Theorem 6 (termination of narrowing). *Let $\mathcal{R}$ be a left-linear constructor TRS and t^α an abstract term. Then, $\gamma(t^\alpha)$ is $\leadsto_{\mathcal{R}}$-terminating if $\widehat{\gamma(t^\alpha)}$ is $\to_{\mathcal{R}}$-terminating relative to $\mathsf{GEN}(\mathcal{R})$.*

In the next result we apply the framework of goal-directed dependency pairs to solve this kind of problems. First, let us recall that our GDP problems consider an initial goal rather than an abstract term. So we embed the abstract goal into the TRS by means of an additional rule. To be precise, given an abstract term $t^\alpha = \mathsf{f}(t_1, \ldots, t_n)$ with m occurrences of $\mathbf{g}$, we let $goal(t^\alpha)$ be the rule

$$\mathsf{goal}(x_{j_1}, \ldots, x_{j_m}) \to \mathsf{f}(x_1, \ldots, x_n)$$

where $x_1, \ldots, x_n$ are (fresh) distinct variables and $j_1, \ldots, j_m$ are the positions of the $\mathbf{g}$ arguments of t^α. Given a TRS $\mathcal{R}$ and an abstract term t^α, we denote by $\mathcal{R}_{t^\alpha}$ the TRS that extends $\mathcal{R}$ with this rule; formally, $\mathcal{R}_{t^\alpha} = \mathcal{R} \cup \{goal(t^\alpha)\}$.

In other words, we replace $\mathbf{v}$ arguments of the abstract term t^α by extra variables. Extra variables occur very naturally in the context of narrowing since they behave as *free* variables which can only be instantiated to finite constructor terms. In our context they are simply replaced by occurrences of gen in $\widehat{\mathcal{R}}$.

The following result combining Theorems 1 and 6 is the basis of our termination proving method.

Theorem 7. *Let $\mathcal{R}$ be a left-linear constructor TRS (possibly with extra variables) and t^α an abstract term. Then, $\gamma(t^\alpha)$ is $\leadsto_{\mathcal{R}}$-terminating if the GDP problem $(\mathsf{goal}(x_1, \ldots, x_n), DP(\widehat{\mathcal{R}_{t^\alpha}}), \widehat{\mathcal{R}_{t^\alpha}} \cup \mathsf{GEN}(\mathcal{R}), \mathbf{a})$ is finite, where $\mathsf{goal}(x_1, \ldots, x_n)$ is the left-hand side of $goal(t^\alpha)$.*

Example 6. Consider the following TRS that is part of Example 3:

$$\mathcal{R} = \left\{ \begin{array}{l} \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y)) \\ \mathsf{add}(\mathsf{zero}, y) \to y \end{array} \right\}$$

For the abstract term $t^\alpha = \mathsf{add}(\mathbf{g}, \mathbf{v})$, the initial GDP problem $(\mathsf{goal}(x), DP(\widehat{\mathcal{R}_{t_\alpha}}), \widehat{\mathcal{R}_{t_\alpha}} \cup \mathsf{GEN}(\mathcal{R}), \mathbf{a})$ is reduced to the finite GDP problem in Example 3. Therefore, $\gamma(\mathsf{add}(\mathbf{g}, \mathbf{v}))$ is $\leadsto_{\mathcal{R}}$-terminating.

Example 7 ([20]). Consider the TRS and its associated set of pairs:

$$\left(\begin{array}{rl} \mathsf{p_{in}}(\mathsf{g}(X)) & \to \mathsf{u_3}(\mathsf{p_{in}}(X), X) \\ \mathsf{p_{in}}(X) & \to \mathsf{u_1}(\mathsf{q_{in}}(\mathsf{f}(Y)), X) \\ \mathsf{q_{in}}(\mathsf{g}(Y)) & \to \mathsf{q_{out}}(\mathsf{g}(Y)) \\ \mathsf{u_1}(\mathsf{q_{out}}(\mathsf{f}(Y)), X) & \to \mathsf{u_2}(\mathsf{p_{in}}(Y), X, Y) \\ \mathsf{u_2}(\mathsf{p_{out}}(Y), X, Y) & \to \mathsf{p_{out}}(X) \\ \mathsf{u_3}(\mathsf{p_{out}}(X), X) & \to \mathsf{p_{out}}(\mathsf{g}(X) \end{array} \quad \begin{array}{rl} \mathsf{P_{in}}(\mathsf{g}(X)) & \to \mathsf{U_3}(\mathsf{p_{in}}(X), X) \\ \mathsf{P_{in}}(\mathsf{g}(X)) & \to \mathsf{P_{in}}(X) \\ \mathsf{P_{in}}(X) & \to \mathsf{U_1}(\mathsf{q_{in}}(\mathsf{f}(Y)), X) \\ \mathsf{P_{in}}(X) & \to \mathsf{Q_{in}}(\mathsf{f}(Y)) \\ \mathsf{U_1}(\mathsf{q_{out}}(\mathsf{f}(Y)), X) & \to \mathsf{U_2}(\mathsf{p_{in}}(Y), X, Y) \\ \mathsf{U_1}(\mathsf{q_{out}}(\mathsf{f}(Y)), X) & \to \mathsf{P_{in}}(Y) \\ \mathsf{GOAL}(X) & \to \mathsf{P_{in}}(X) \end{array}\right)$$

where the last pair is the initial pair, added to model the abstract goal $\mathsf{p_{in}}(\mathbf{g})$. A similar rule is implicitly added to $\mathcal{R}$, together with the rules for gen. Also, note the presence of extra variables, which in our approach would be replaced by calls to gen. An estimation of the dependency graph can detect that the only SCC is the pair $\mathsf{P_{in}}(\mathsf{g}(X)) \to \mathsf{P_{in}}(X)$, and that there is only the trivial path from the initial pair to this pair which includes both. As the goal-directed usable rules are the empty set, it is trivial to find an RPO that orients this pair and solves the termination problem. On the other hand, the technique of [20] needs a global argument filtering that removes every extra variable, which ultimately has to filter either the argument of $\mathsf{P_{in}}$ or g, precluding a successful termination proof. The same remark applies to [18, 25] after filtering out the extra variables.

5 Results and Discussion

The technique for proving termination of narrowing introduced in the previous section is not directly comparable to [18, 25] due to the loss of minimality. This means that many desirable techniques, such as the subterm criterion of [14], cannot be applied without restrictions. The same remarks apply when comparing our new approach to the infinitary rewriting framework of [20], which also employs a global argument filtering and heuristics.

In order to see how well the new approach behaves, we benchmark it versus [25, 18] and [20]. We emply a set of examples generated from the LP category of the Termination Problem Database (TPDB) 5.0 [17] using the transform of [20], minus those examples containing cuts, impure primitives or arithmetic. A huge number of techniques for termination have been developed in the recent years. In order to provide a fair comparison we focus on a small set of these: the dependency graph processor, the RPO reduction pair implemented by means of the SAT encoding of [5, 21, 6] extended to account for our notion of goal-directed usable rules, the subterm criterion of [14] and the narrowing and instantiation graph refinement processors. The (very fast) subterm criterion processor is included to illustrate the shortcomings of losing minimality.

We have implemented our approach in the termination tool *Narradar*, available at `http://safe-tools.dsic.upv.es/narradar`. The tool recognizes the TPDB format [17] with extensions for expressing initial goals and narrowing. In order to perform the test Narradar was extended to implement the approach of

[18, 25] and [20]. For [18, 25] Narradar uses a simple heuristic which always filters the innermost position. Also, although this approach does not consider extra variables, we filter them from the initial problem assuming that this is safe. For [20] Narradar uses the unbounded positions heuristic which does a type analysis of the logic program and computes an optimal heuristic.

All the problems were run on a 2.5Ghz Intel CPU with a 60 seconds timeout. The results are displayed in the table below, where the new approach is the best performer overall, even though the average success time is higher. It is easy to see why: the RPO constraints generated are slightly more complex, and the fast subterm criterion processor cannot be employed. There were four examples which Narradar failed to solve where the other techniques succeeded. These fall into two categories: the incompleteness of the generator approach due to the problem of admissible derivations [25] (e.g. `SGST06/toyama.pl`), or the inability of the RPO processor to solve a given problem where the subterm criterion succeeds easily (e.g. `SGST06/prime.pl`). While the former is an intrinsic limitation of the approach, the latter can be fixed by means of more powerful termination processors. The full results, including the proofs generated by Narradar, are available at `http://www.dsic.upv.es/~gvidal/lopstr09`. Let us remark that the results for [20] must be regarded as orientative only, for it is in fact a technique for the termination of logic programs.

	Successes	Failures	Timeouts	Average success time
Narradar	183	113	9	1.67 seconds
[25]	167	122	16	0.33 seconds
[20]	178	111	16	0.29 seconds

Future work. Although the new technique outperforms the state of the art, there is still ample room for improvement ahead. We could pursue the approach of [2] and replace every extra variable by the generator of the terms it can get instantiated to, by means of a previous static analysis step. Moreover, minimality could be recovered under some conditions. Finally, although we have defined our method for left-linear constructor systems, it remains to be seen whether this restriction can be lifted using the LL-DPs of [1].

Acknowledgements. The first author is in debt to Peter Schneider-Kamp for many insightful discussions during a research visit to South Denmark. We also thank Raúl Gutiérrez for his numerous relevant remarks on a draft of this paper, and Nao Hirokawa for his insights on relative termination.

References

1. M. Alpuente, S. Escobar, and J. Iborra. Termination of Narrowing Using Dependency Pairs. In *Proc. of the 24th Int'l Conf. on Logic Programming (ICLP 2008)*, pages 317–331. LNCS 5366, 2008.
2. S. Antoy and M. Hanus. Overlapping Rules and Logic Variables in Functional Logic Programs. In *Proc. of ICLP'06*, *LNCS* 4079, pp 87–101. Springer, 2006.
3. T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.

4. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
5. M. Codish, V. Lagoon, and P. J. Stuckey. Solving partial order constraints for lpo termination. *CoRR*, abs/cs/0512067, 2005.
6. M. Codish, P. Schneider-Kamp, V. Lagoon, R. Thiemann, and J. Giesl. Sat solving for argument filterings. In Miki Hermann and Andrei Voronkov, editors, *LPAR*, volume 4246 of *LNCS*, pages 30–44. Springer, 2006.
7. J. de Dios-Castro and F. López-Fraguas. Extra Variables Can Be Eliminated from Functional Logic Programs. In *Proc. of the 6th Spanish Conf. on Programming and Languages (PROLE'06)*, volume of 188 of *ENTCS*, pages 3–19. 2007.
8. N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1&2):69–115, 1987.
9. A. Geser. *Relative termination.* Dissertation, Fakultät für Mathematik und Informatik, Universität Passau, Germany, 1990.
10. J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated Termination Analysis for Haskell: From Term Rewriting to Programming Languages. In *Proc. of RTA 2006*, volume 4098 of *LNCS*, pages 297–312. Springer, 2006.
11. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In *Proc. of LPAR'04*, volume of 3452 of *LNCS*, pages 301–331. Springer, 2005.
12. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and Improving Dependency Pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
13. M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 19&20:583–628, 1994.
14. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In Vincent van Oostrom, editor, *RTA*, volume 3091 of *LNCS*, pages 249–268. Springer, 2004.
15. J. Iborra, N. Nishida and G. Vidal. Goal-directed Dependency Pairs and its Application to Proving the Termination of Narrowing, 2010. Available from `http://users.dsic.upv.es/~gvidal/german/papers.html`.
16. K. Kusakari, M. Nakamura, and Y. Toyama. Argument Filtering Transformation. In *Proc. of PPDP'99*, volume 1702 of *LNCS*, pages 48–62. Springer, 1999.
17. C. Marche and H. Zantema. The termination competition. In *Term Rewriting and Applications*, volume 4533 of *LNCS*, pages 303–313. Springer, 2007.
18. N. Nishida and G. Vidal. Termination of Narrowing via Termination of Rewriting, 2009. Submitted for publication.
19. E Ohlebusch. *Advanced topics in term rewriting.* Springer-Verlag, UK, 2002.
20. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated Termination Analysis for Logic Programs by Term Rewriting. In *Proc. of LOPSTR'06*, volume of 4407 of *LNCS*, pages 177–193. Springer, 2007.
21. P. Schneider-Kamp, R. Thiemann, E. Annov, M. Codish, and J. Giesl. Proving termination using recursive path orders and sat solving. In Boris Konev and Frank Wolter, editors, *FroCos*, volume 4720 of *LNCS*, pages 267–282. Springer, 2007.
22. J.R. Slagle. Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity. *Journal of the ACM*, 21(4):622–642, 1974.
23. J. Steinbach. Simplification Orderings: Histrory of Results. *Fundamenta Informaticae*, 24(1/2):47–87, 1995.
24. X. Urbain. Modular & incremental automated termination proofs. *Int. Journal of Approx. Reasoning*, 32(4):315–355, 2004.
25. G. Vidal. Termination of Narrowing in Left-Linear Constructor Systems. In J. Garrigue and M. Hermenegildo, editors, *Proc. of FLOPS 2008*, volume of 4989 of *LNCS*, pages 113–129. Springer, 2008.