Termination of Narrowing using Dependency Pairs*

María Alpuente, Santiago Escobar, and José Iborra

Technical University of Valencia (UPV), Spain.
{alpuente,sescobar,jiborra}@dsic.upv.es

Abstract. In this work, we extend the dependency pair approach for automated proofs of termination in order to prove the termination of narrowing. Our extension of the dependency pair approach generalizes the standard notion of dependency pairs by taking specifically into account the dependencies between the left-hand side of a rewrite rule and its own argument subterms. We demonstrate that the new *narrowing dependency pairs* exactly capture the narrowing termination behavior and provide an effective termination criterion which we prove to be sound and complete. Finally, we discuss how the problem of analyzing narrowing chains can be recast as a standard analysis problem for traditional (rewriting) chains, so that the proposed technique can be effectively mechanized by reusing the standard DP infrastructure.

1 Introduction

In recent years, the dependency pair (DP) method for automating the termination proofs of term rewriting has achieved tremendous success, as witnessed by the large number of publications and tools since its introduction in [6] and subsequent reformulation in [9] (see [11, 13] for extensive references thereof).

Narrowing is a generalization of term rewriting that allows free variables in terms (as in logic programming) and replaces pattern matching by syntactic unification so that it subsumes both rewriting and SLD-resolution [12]. Narrowing has many important applications including execution of functional–logic programming languages [12], verification of security policies [15] and cryptographic protocols [8], equational unification [14], and symbolic reachability [16], among others. Termination of narrowing itself is, therefore, of great interest to these applications.

Termination of narrowing is a more restrictive property than termination of rewriting or termination of pure logic programs due to the high degree of nondeterminism caused by the interaction of rule selection, redex selection, and unification. In recent works [2, 4], we identified some non–trivial classes of TRSs where narrowing terminates. The results in [4] generalize previously known criteria for termination of narrowing, which were essentially restricted before to

* This work has been partially supported by the EU (FEDER) and Spanish MEC project TIN2007-68093-C02-02, Integrated Action Hispano-Alemana HA2006-0007, and UPV-VIDI grant 3249 PAID0607.

either confluent term rewriting systems (TRSs) [14] or to left–flat TRSs (i.e., each argument of the left–hand side of a rewrite rule is either a variable or a ground term) that are compatible with a termination ordering [7], among other applicability conditions. Roughly speaking, we proved in [4] that confluence is a superfluous requirement for the termination of narrowing. We also weakened the left–flatness condition required in [7] to the requirement that every non-ground, strict subterm of the left–hand side (lhs) of every rewrite rule must be a *rigid normal form,* i.e., unnarrowable. Finally, in [2] we proved modular termination of a restriction of narrowing, called basic narrowing [14], in several hierarchical combinations of TRSs, which provides new algorithmic criteria to prove termination of narrowing via termination of basic narrowing (cf. [4]).

In [17], the notions of dependency pairs and dependency graphs, which were originally developed for term rewriting, were adapted to the logic programming domain, leading to automated termination analyses that are directly applicable to any definite logic program. Two different adaptations of the DP technique for narrowing have been proposed recently. In [18, 19], the original dependency pair technique of [6] was adapted to the termination of narrowing, whereas [20] adapts the logic programming dependency pair approach of [17] instead, to prove termination of narrowing w.r.t. a given set of queries that are supplemented with *call modes*. Unfortunately, these two methods apply only to two particular classes of TRSs: right–linear TRSs (i.e., no repeated variables occur in the right–hand sides of the rules) or constructor systems (the arguments of the lhs's of the rules are constructor –i.e., data– terms). These two classes are overly restrictive for many practical uses of narrowing, such as the applications mentioned above. In this work, we are interested in developing automatable methods for proving termination of narrowing in TRSs that resist all previous techniques.

Example 1. Consider our running example, which is the non–right–linear, non–constructor–based, non–confluent TRS, adapted from [15], that is shown in Figure[1] 1. This TRS models a security (filtering) and routing policy that allows packets coming from external networks to be analyzed. We do not describe the intended meaning of each symbol since it is not relevant for this work, but note the kind of expressivity that is assumed in the domain of rule–based policy specification, that does not fit in the right–linear restriction or the constructor discipline. Narrowing is terminating for this TRS, but it cannot be proved by using any of the existing methods [2, 4, 18–20]. In this paper, we provide techniques that allow us to prove it automatically.

The main contributions of this paper are as follows:

– We present a new method for proving the termination of narrowing that is based on a suitable extension of the DP technique to narrowing that is applicable to any class of TRSs. Our method generalizes the standard notion of dependency pairs to narrowing by taking the dependencies between the lhs of a rewrite rule and its own argument subterms specifically into account.

[1] In this paper, variables are written in italic font and function symbols are in typewriter font.

$$
\begin{aligned}
&\mathtt{filter}(\mathtt{pckt}(src, dst, \mathtt{established})) && \to \mathtt{accept}\\
&\mathtt{filter}(\mathtt{pckt}(\mathtt{eth0}, dst, \mathtt{new})) && \to \mathtt{accept}\\
&\mathtt{filter}(\mathtt{pckt}(\mathtt{194.179.1.}x\mathtt{:}port, dst, \mathtt{new})) && \to \mathtt{filter}(\mathtt{pckt}(\mathtt{secure}, dst, \mathtt{new}))\\
&\mathtt{filter}(\mathtt{pckt}(\mathtt{158.42.}x\mathtt{.}y\mathtt{:}port, dst, \mathtt{new})) && \to \mathtt{filter}(\mathtt{pckt}(\mathtt{secure}, dst, \mathtt{new}))\\
&\mathtt{filter}(\mathtt{pckt}(\mathtt{secure}, dst\mathtt{:80}, \mathtt{new})) && \to \mathtt{accept}\\
&\mathtt{filter}(\mathtt{pckt}(\mathtt{secure}, dst\mathtt{:other}, \mathtt{new})) && \to \mathtt{drop}\\
&\mathtt{filter}(\mathtt{pckt}(\mathtt{ppp0}, dst, \mathtt{new})) && \to \mathtt{drop}\\
&\mathtt{filter}(\mathtt{pckt}(\mathtt{123.123.1.1}\mathtt{:}port, dst, \mathtt{new})) && \to \mathtt{accept}\\
&\mathtt{pckt}(\mathtt{10.1.1.1}\mathtt{:}port, \mathtt{ppp0}, s) && \to \mathtt{pckt}(\mathtt{123.23.1.1}\mathtt{:}port, \mathtt{ppp0}, s)\\
&\mathtt{pckt}(\mathtt{10.1.1.2}\mathtt{:}port, \mathtt{ppp0}, s) && \to \mathtt{pckt}(\mathtt{123.23.1.1}\mathtt{:}port, \mathtt{ppp0}, s)\\
&\mathtt{pckt}(src, \mathtt{123.123.1.1}\mathtt{:}port, \mathtt{new}) && \to \mathtt{natroute}(\mathtt{pckt}(src, \mathtt{10.1.1.1}\mathtt{:}port, \mathtt{established}),\\
&&& \qquad \mathtt{pckt}(src, \mathtt{10.1.1.2}\mathtt{:}port, \mathtt{established}))\\
&\mathtt{natroute}(a, b) && \to a\\
&\mathtt{natroute}(a, b) && \to b
\end{aligned}
$$

Fig. 1. The *FullPolicy* TRS

- We demonstrate that the new *narrowing dependency pairs* exactly capture the termination of narrowing behavior. We provide a termination criterion based on *narrowing chains* which we demonstrate to be sound and complete.
- This allows us to develop a technique that is more general in all cases and, for general calls (i.e., without considering call modes) strictly subsumes the DP methods for proving termination of narrowing of [18–20], as well as all previous (decidable) termination of narrowing criteria [2, 4, 7, 14].
- We have implemented a tool for proving the termination of narrowing automatically that is based on our technique, and we made it publicly available.

Plan of the paper

After recalling some preliminaries in Section 2, in Section 3 we discuss the problem of *echoing*, which we identify as being ultimately responsible for the non–termination of narrowing. In Section 4, we develop the notion of narrowing dependency pairs and provide a sound and complete criterion for the termination of narrowing that is based on analyzing narrowing chains. In Section 5, we discuss the effective automation of our method, which mainly consists of two steps: DP extraction and argument filtering transformation. Section 6 concludes. More details and proofs of all technical results can be found in [3].

2 Preliminaries

In this section, we briefly recall the essential notions and terminology of term rewriting. For missing notions and definitions on equations, orderings and rewriting, we refer to [21].

$\mathcal{V}$ denotes a countably infinite set of variables, and Σ denotes a set of function symbols, or signature, each of which has a fixed associated arity. Terms are viewed as labelled trees in the usual way, where $\mathcal{T}(\Sigma, \mathcal{V})$ and $\mathcal{T}(\Sigma)$ denote the non-ground term algebra and the ground algebra built on $\Sigma \cup \mathcal{V}$ and Σ, respectively. Positions are defined as sequences of natural numbers used to address subterms of a term, with ϵ as the root (or top) position (i.e., the empty

sequence). Concatenation of positions p and q is denoted by $p.q$, and $p < q$ is the usual prefix ordering. The root symbol of a term is denoted by root(t). Given $S \subseteq \Sigma \cup \mathcal{V}$, $\mathcal{P}os_S(t)$ denotes the set of positions of a term t that are rooted by function symbols or variables in S. $\mathcal{P}os_{\{f\}}(t)$ with $f \in \Sigma \cup \mathcal{V}$ will be simply denoted by $\mathcal{P}os_f(t)$, and $\mathcal{P}os_{\Sigma\cup\mathcal{V}}(t)$ will be simply denoted by $\mathcal{P}os(t)$. $t|_p$ is the subterm at the position p of t. $t[s]_p$ is the term t with the subterm at the position p replaced with term s. By $Var(s)$, we denote the set of variables occurring in the syntactic object s. By $\bar{x}$, we denote a tuple of pairwise distinct variables. A *fresh* variable is a variable that appears nowhere else. A *linear* term is one where every variable occurs only once.

A *substitution* σ is a mapping from the set of variables $\mathcal{V}$ into the set of terms $\mathcal{T}(\Sigma, \mathcal{V})$ with a (possibly infinite) domain $D(\sigma)$, and image $I(\sigma)$. A substitution is represented as $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ for variables $x_1, \ldots, x_n$ and terms $t_1, \ldots, t_n$. The application of a substitution θ to term t is denoted by $t\theta$, using postfix notation. Composition of substitutions is denoted by juxtaposition, i.e., the substitution $\sigma\theta$ denotes $(\theta \circ \sigma)$. We write $\theta_{\restriction Var(s)}$ to denote the restriction of the substitution θ to the set of variables in s; by abuse of notation, we often simply write $\theta_{\restriction s}$. Given a term t, $\theta = \nu\ [t]$ iff $\theta_{\restriction Var(t)} = \nu_{\restriction Var(t)}$, that is, $\forall x \in Var(t)$, $x\theta = x\nu$. A substitution θ is more general than σ, denoted by $\theta \leq \sigma$, if there is a substitution γ such that $\theta\gamma = \sigma$. A *unifier* of terms s and t is a substitution ϑ such that $s\vartheta = t\vartheta$. The *most general unifier* of terms s and t, denoted by $mgu(s, t)$, is a unifier θ such that for any other unifier θ', $\theta \leq \theta'$.

A *term rewriting system* (TRS) $\mathcal{R}$ is a pair (Σ, R), where R is a finite set of rewrite rules of the form $l \to r$ such that $l, r \in \mathcal{T}(\Sigma, \mathcal{V})$, $l \notin \mathcal{V}$, and $Var(r) \subseteq Var(l)$. For TRS $\mathcal{R}$, $l \to r \ll \mathcal{R}$ denotes that $l \to r$ is a new variant of a rule in $\mathcal{R}$ such that $l \to r$ contains only *fresh* variables, i.e., contains no variable previously met during any computation (standardized apart). We will often write just $\mathcal{R}$ or (Σ, R) instead of $\mathcal{R} = (\Sigma, R)$. A TRS $\mathcal{R}$ is called *left–linear* (respectively *right–linear*) if, for every $l \to r \in \mathcal{R}$, l (respectively r) is a linear term. Given a TRS $\mathcal{R} = (\Sigma, R)$, the signature Σ is often partitioned into two disjoint sets $\Sigma = \mathcal{C} \uplus \mathcal{D}$, where $\mathcal{D} = \{f \mid f(t_1, \ldots, t_n) \to r \in R\}$ and $\mathcal{C} = \Sigma \setminus \mathcal{D}$. Symbols in $\mathcal{C}$ are called *constructors*, and symbols in $\mathcal{D}$ are called *defined functions*. The elements of $\mathcal{T}(\mathcal{C}, \mathcal{V})$ are called *constructor terms*. We let $Def(\mathcal{R})$ denote the set of defined symbols in $\mathcal{R}$. A constructor system is a TRS whose lhs's are terms of the form $f(c_1, \ldots, c_k)$ where $f \in \mathcal{D}$ and $c_1, \ldots, c_k$ are constructor terms. A term whose root symbol is a defined function is called *root-defined*.

A rewrite step is the application of a rewrite rule to an expression. A term $s \in \mathcal{T}(\Sigma, \mathcal{V})$ *rewrites* to a term $t \in \mathcal{T}(\Sigma, \mathcal{V})$, denoted by $s \xrightarrow{p}_{\mathcal{R}} t$, if there exist $p \in \mathcal{P}os_\Sigma(s)$, $l \to r \in \mathcal{R}$, and substitution σ such that $s|_p = l\sigma$ and $t = s[r\sigma]_p$. When no confusion can arise, we omit the subscript in $\to_{\mathcal{R}}$. We also omit the reduced position p when it is not relevant. A term s is a *normal form* w.r.t. the relation $\to_{\mathcal{R}}$ (or simply a normal form), if there is no term t such that $s \to_{\mathcal{R}} t$. A term is a reducible expression or *redex* if it is an instance of the left hand side of a rule in $\mathcal{R}$. A term s is a *head normal form* if there are no terms t, t' s.t. $s \to^*_{\mathcal{R}} t' \xrightarrow{\epsilon}_{\mathcal{R}} t$. A term t is said to be terminating w.r.t. R if there is no infinite

reduction sequence $t \to_{\mathcal{R}} t_1 \to_{\mathcal{R}} t_2 \to_{\mathcal{R}} \ldots$. A TRS $\mathcal{R}$ is *(→)-terminating* (also called strongly normalizing or noetherian) if every term is terminating w.r.t. R. A TRS $\mathcal{R}$ is *confluent* if, whenever $t \to^*_{\mathcal{R}} s_1$ and $t \to^*_{\mathcal{R}} s_2$, there exists a term w s.t. $s_1 \to^*_{\mathcal{R}} w$ and $s_2 \to^*_{\mathcal{R}} w$.

A term $s \in \mathcal{T}(\Sigma, \mathcal{V})$ narrows to a term $t \in \mathcal{T}(\Sigma, \mathcal{V})$, denoted by $s \overset{p}{\leadsto}_{\theta,\mathcal{R}} t$, if there exist $p \in \mathcal{P}os_\Sigma(s)$, $l \to r \ll \mathcal{R}$, and substitution θ such that $\theta = mgu(s|_p, l)$ and $t = (s[r]_p)\theta$. We use $\overset{>\epsilon}{\to}_{\mathcal{R}}$ (resp. $\overset{>\epsilon}{\leadsto}_{\theta,\mathcal{R}}$) to denote steps in which the selected redex (resp. *narrex*, i.e. narrowable expression) is below the root.

3 The *echoing* problem

The dependency pair technique [6] is one of the most powerful methods for automated termination analyses. The technique focuses on the dependency relations between defined function symbols, paying particular attention to strongly connected components within a graph of functional dependencies, in order to produce automated termination proofs. The dependency graph is typically extracted by considering the dependencies between the lhs's of a rewrite rule and all proper subterms of the rhs of the rule.

An adaptation of the DP method to narrowing is given in [19] that requires the TRS to have the so-called *Top Reduced Almost Terminating* (TRAT) property, defined as follows. Given a property P on terms, a term t is said to be a *minimal P term* if t satisfies P but none of the proper subterms of t does. Given a TRS $\mathcal{R}$ and a binary relation $\Rightarrow$ (being $\to_{\mathcal{R}}$ or $\leadsto_{\mathcal{R}}$), an infinite derivation $t \Rightarrow t_1 \Rightarrow t_2 \ldots$ is called *almost terminating* if t is a minimal non–terminating term w.r.t. $\Rightarrow$. An almost terminating derivation $t \Rightarrow t_1 \Rightarrow t_2 \ldots$ is called *top reduced* if it contains a derivation step at the root position. We say that $\Rightarrow$ has the TRAT property if, for every non-terminating term t, there exists a top reduced almost-terminating sequence stemming from one subterm of t.

Let us briefly recall the notion of *context*. A context is a term with several occurrences of a fresh symbol $\Box$. If $C[\,]$ contains k occurrences of symbol $\Box$ at positions $p_1, \ldots, p_k$, we write $C[t_1, \ldots, t_k]$ to denote the term $(C[t_1]_{p_1}) \cdots [t_k]_{p_k}$.

In [19] it is proved that every monotone relation has the TRAT property. Since the rewriting relation is monotone (i.e., $t \to_{\mathcal{R}} s$ implies $C[t] \to_{\mathcal{R}} C[s]$), then it has the TRAT property for every TRS $\mathcal{R}$ (cf. [13, Lemma 1]). In term rewriting this ensures that, in every almost terminating, infinite term rewriting derivation, a rewriting step is given at the root. Unfortunately, the narrowing relation is not monotone: $t \leadsto_{\sigma,\mathcal{R}} s$ does not entail $C[t] \leadsto_{\sigma,\mathcal{R}} C[s]$ but $C[t] \leadsto_{\sigma,\mathcal{R}} (C\sigma)[s]$ instead.

Example 2. [7] Consider the TRS consisting of the rule $\mathtt{f}(\mathtt{f}(x)) \to x$, and the non–linear term $\mathtt{c}(\mathtt{f}(x), x)$. Then there does not exist an infinite narrowing derivation for the subterms, $\mathtt{f}(x)$ and x, whereas $\mathtt{c}(\mathtt{f}(x), x)$ is infinitely narrowed without ever performing a narrowing step at the root:

$$\mathtt{c}(\underline{\mathtt{f}(x)}, x) \leadsto_{\{x \mapsto \mathtt{f}(x')\}} \mathtt{c}(x', \underline{\mathtt{f}(x')}) \leadsto_{\{x' \mapsto \mathtt{f}(x'')\}} \mathtt{c}(\underline{\mathtt{f}(x'')}, x'') \ldots$$

As shown by the above example, in the presence of non linearity the non–monotony of narrowing has undesirable effects for its termination, since *narrexes* can be brought into the context by the substitution computed at the preceding narrowing step, thus causing other terms in the context to grow. This *echoing* effect plays a fatal role in the (non–) termination of narrowing.

There are some classes of TRSs in which narrowing exhibits a monotone or monotone–like behaviour and thus enjoys the TRAT property. [19] considers two such classes: right–linear TRS (w.r.t. linear goals), and constructor systems. We note that these two classes are a particular case of a larger characterization of narrowing termination that we formalized in [4] by the QSRNC (*Quasi stable rigidly normalized condition*), though in [4] we do not make the relation with TRAT explicit.

The inspiration for this work comes from realizing that monotonicity is not really a *necessary* condition for the termination of narrowing, provided the partially computed substitutions do not *echo*, i.e., they do not bring narrexes into the context that might either introduce a term that does not terminate or *echo* again. Let us introduce the idea by means of one example.

Example 3. Consider the non–linear input call $\mathtt{c}(\mathtt{f}(x), x)$ in the non–constructor TRS consisting of rules $\mathtt{f}(\mathtt{g}(x)) \to x$ and $\mathtt{g}(x) \to x$. The only possible derivation for this term is finite, whereas the TRS, together with the considered non–linear input term, do not fit in any of the characterizations given for TRAT [18–20] or any decidable criteria for the termination of narrowing [2, 4, 7, 14]. Note that the argument $\mathtt{g}(x)$ of the lhs of the first rule is a narrex.

Let us start with some lessons learnt from the termination of rewriting that would be good to transfer to the termination of narrowing. In rewriting (and narrowing), if a TRS is not terminating then there must be a minimal non-terminating term. In rewriting such a minimal non-terminating term is rooted by a defined symbol but this is not true for narrowing. As in [13], let us denote the set of all minimal non-terminating terms w.r.t. rewriting (resp. narrowing) by $\mathcal{T}^\infty$ (resp. $\mathcal{T}^\infty_{\leadsto}$). The following definition is crucial.

Definition 1 (Echoing terms). *Let $\mathcal{R}$ be a TRS. We define the set of* minimal echoing terms *w.r.t. $\mathcal{R}$, denoted by $\mathcal{T}^{\circlearrowleft}$, as follows: $s \in \mathcal{T}^{\circlearrowleft}$ if, given a fresh binary symbol c and a variable $x \in Var(s)$, then $c(s, x) \in \mathcal{T}^\infty_{\leadsto}$ but $s \notin \mathcal{T}^\infty_{\leadsto}$, and there is no proper subterm s' of s such that $s' \in \mathcal{T}^{\circlearrowleft}$.*

Now, we provide our key result for the termination of narrowing. We write $s \trianglerighteq t$ to denote that t is a subterm of s, and $s \triangleright t$ if t is a proper subterm of s.

Lemma 1. *Let $\mathcal{R}$ be a TRS. For every term $t \in \mathcal{T}^\infty_{\leadsto}$, we have that either*

1. (TOP) *there exists a rewrite rule $l \to r \in \mathcal{R}$, substitutions σ, ρ, a term t', and a non-variable subterm u of r such that $t \overset{\geq\epsilon}{\leadsto}{}^*_{\rho,\mathcal{R}} t' \overset{\epsilon}{\leadsto}_{\sigma,l\to r} r\sigma \trianglerighteq u$ and $u \in \mathcal{T}^\infty_{\leadsto}$;*
2. (HYBRID) *there are terms t', t'', u, substitutions ρ, σ, a position p, and a variable x such that $t \overset{\geq\epsilon}{\leadsto}{}^*_{\rho,\mathcal{R}} t' \overset{p}{\leadsto}_{\sigma,\mathcal{R}} t''$, $x \in Var(t'|_p)$, $x\sigma \trianglerighteq u$, and $u \in \mathcal{T}^\infty_{\leadsto}$;*

3. (ECHOING) *there are terms* t', t'', u, *substitutions* ρ, σ, *a position* p, *and a variable* x *such that* $t \overset{\geq\epsilon}{\leadsto}{}^*_{\rho,\mathcal{R}} t' \overset{p}{\leadsto}_{\sigma,\mathcal{R}} t''$, $x \in Var(t'|_p)$, $x\sigma \trianglerighteq u$, *and* $t'|_p, u \in \mathcal{T}^{\circlearrowleft}$.

Informally, the lemma above distinguishes three different kinds of minimal non–terminating terms. The TOP case is the usual one shared by rewriting and narrowing non–termination; the other two cases are due to non–monotonicity and thus unique to narrowing. In the pure ECHOING case, the narrowing of an echoing subterm introduces into the context a new echoing subterm that reproduces the process again, as in Example 2. In the HYBRID echoing case, the reduction of an echoing subterm introduces into the context a minimal non–terminating narrex that spawns an infinite narrowing derivation, as in Example 4 below.

Example 4. Consider the following TRS:

$$\mathtt{f}(\mathtt{g}(x)) \to \mathtt{a} \qquad\qquad \mathtt{g}(x) \to \mathtt{g}(x)$$

$\mathtt{g}(x) \in \mathcal{T}^{\infty}_{\rightsquigarrow}$ is a minimal non–terminating term for rewriting. $\mathtt{f}(x) \notin \mathcal{T}^{\infty}_{\rightsquigarrow}$, since only the derivation $\mathtt{f}(x) \rightsquigarrow_{\{x \mapsto \mathtt{g}(x')\}} \mathtt{a}$ can be proven. However, given a fresh symbol $\mathtt{c}$, there is a HYBRID infinite narrowing derivation stemming from the term $\mathtt{c}(\mathtt{f}(x), x) \in \mathcal{T}^{\infty}_{\rightsquigarrow}$. Therefore, $\mathtt{f}(x) \in \mathcal{T}^{\circlearrowleft}$.

4 Narrowing Dependency Pairs

In this section, we develop the notion of narrowing dependency pairs, and provide a sound and complete criterion for the termination of narrowing that is based on analyzing narrowing chains.

The intuitive idea behind our method is as follows. In order to construct the set of dependency pairs, we not only relate the lhs of each rule with the root–defined subterms occurring in the corresponding rhs, as in standard rewriting DP, but also with its *own* root–defined subterms, i.e., those terms whose root symbol is a defined function. The resulting set of dependency pairs faithfully captures the behaviour of infinite narrowing derivations which incrementally compute an infinite substitution, or more precisely, where the substitution computed by narrowing contains an infinite term.

Suppose we split the substitution σ computed by a narrowing step $t \rightsquigarrow_{l \to r, \sigma} s$ into two pieces, $\sigma \equiv \sigma_{\restriction l} \uplus \sigma_{\restriction t}$. The $\sigma_{\restriction l}$ part of the substitution has the usual effect of propagating narrexes from the left hand side to the right hand side of the rule. On the other hand, the $\sigma_{\restriction t}$ part is responsible for the echoing of *narrexes* to the context that can fire a new narrowing step. These narrexes come from the subterms of the left hand side of the rule, as in Example 2 above, or from the term being narrowed itself, e.g. when $\mathtt{c}(z, \mathtt{h}(\mathtt{g}(x), z))$ is narrowed to $\mathtt{c}(\mathtt{g}(x), 0)$ by using the rule $\mathtt{h}(y, y) \to 0$ and most general unifier $\{z \mapsto \mathtt{g}(x), y \mapsto \mathtt{g}(x)\}$.

Although the narrexes coming from proper subterms of the narrex selected at the preceding step might cause non–termination, standard (rewriting) termination analyses already cope with them. However, narrexes coming from proper

(1) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{194.179.1}.x\mathtt{:}p, dst, \mathtt{new})) \to \mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{secure}, dst, \mathtt{new}))$
(2) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{194.179.1}.x\mathtt{:}p, dst, \mathtt{new})) \to \mathtt{pckt}^{\#}(\mathtt{secure}, dst, \mathtt{new})$
(3) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{158.42}.x.y\mathtt{:}p, dst, \mathtt{new})) \to \mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{secure}, dst, \mathtt{new}))$
(4) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{158.42}.x.y\mathtt{:}p, dst, \mathtt{new})) \to \mathtt{pckt}^{\#}(\mathtt{secure}, dst, \mathtt{new})$
(5) $\mathtt{pckt}^{\#}(\mathtt{10.1.1.1}\mathtt{:}p, \mathtt{ppp0}, s) \to \mathtt{pckt}^{\#}(\mathtt{123.23.1.1}\mathtt{:}p, \mathtt{ppp0}, s)$
(6) $\mathtt{pckt}^{\#}(\mathtt{10.1.1.2}\mathtt{:}p, \mathtt{ppp0}, s) \to \mathtt{pckt}^{\#}(\mathtt{123.23.1.1}\mathtt{:}p, \mathtt{ppp0}, s)$
(7) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{123.123.1.1}\mathtt{:}p, dst, \mathtt{new})) \to \mathtt{pckt}^{\#}(\mathtt{123.123.1.1}\mathtt{:}p, dst, \mathtt{new})$
(8) $\mathtt{pckt}^{\#}(src, \mathtt{123.123.1.1}\mathtt{:}p, \mathtt{new}) \to \mathtt{pckt}^{\#}(src, \mathtt{10.1.1.1}\mathtt{:}p, \mathtt{established})$
(9) $\mathtt{pckt}^{\#}(src, \mathtt{123.123.1.1}\mathtt{:}p, \mathtt{new}) \to \mathtt{pckt}^{\#}(src, \mathtt{10.1.1.2}\mathtt{:}p, \mathtt{established})$
(10) $\mathtt{filter}^{\#}(\mathtt{pckt}(src, dst, \mathtt{established})) \to \mathtt{pckt}^{\#}(src, dst, \mathtt{established})$
(11) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{eth0}, dst, \mathtt{new})) \to \mathtt{pckt}^{\#}(\mathtt{eth0}, dst, \mathtt{new})$
(12) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{194.179.1}.x\mathtt{:}p, dst, \mathtt{new})) \to \mathtt{pckt}^{\#}(\mathtt{194.179.1}.x\mathtt{:}p, dst, \mathtt{new})$
(13) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{158.42}.x.y\mathtt{:}p, dst, \mathtt{new})) \to \mathtt{pckt}^{\#}(\mathtt{158.42}.x.y\mathtt{:}p, dst, \mathtt{new})$
(14) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{secure}, dst\mathtt{:80}, \mathtt{new})) \to \mathtt{pckt}^{\#}(\mathtt{secure}, dst\mathtt{:80}, \mathtt{new})$
(15) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{secure}, dst\mathtt{:other}, \mathtt{new})) \to \mathtt{pckt}^{\#}(\mathtt{secure}, dst\mathtt{:other}, \mathtt{new})$
(16) $\mathtt{filter}^{\#}(\mathtt{pckt}(\mathtt{ppp0}, dst, \mathtt{new})) \to \mathtt{pckt}^{\#}(\mathtt{ppp0}, dst, \mathtt{new})$
(17) $\mathtt{pckt}^{\#}(src, \mathtt{123.123.1.1}\mathtt{;}p, \mathtt{new}) \to \mathtt{natroute}^{\#}(\mathtt{pckt}(src, \mathtt{10.1.1.1}\mathtt{:}p, \mathtt{established}), \mathtt{pckt}(src, \mathtt{10.1.1.2}\mathtt{:}p, \mathtt{established}))$

Fig. 2. Dependency pairs of *FullPolicy*

subterms of the lhs of the rules are specific to narrowing, and thus we focus on them in our notion of narrowing dependency pairs.

Notation Let $\mathcal{R}$ be a TRS defined over a signature $\Sigma = \mathcal{D} \uplus \mathcal{C}$. Let $\Sigma^{\#}$ denote the extension of Σ with $\{f^{\#} \mid f \in \mathcal{D}\}$, where $f^{\#}$ is a fresh symbol with the same arity as f. If $t \in \mathcal{T}(\Sigma, \mathcal{V})$ is of the form $\mathtt{f}(s_1, \ldots, s_n)$ with $\mathtt{f}$ a defined symbol, then $\mathrm{t}^{\#}$ denotes the term $\mathtt{f}^{\#}(s_1, \ldots, s_n)$.

The following definition extends the traditional, vanilla DPs with a novel kind of dependency pairs, which we call *ll–dependency pairs*.

Definition 2 (Narrowing Dependency Pair). *Given a TRS $\mathcal{R}$, we have two types of narrowing dependency pairs:*

- *a* lr–dependency pair *(or* standard[2] *DP) of $\mathcal{R}$ is a pair $l^{\#} \to t^{\#}$ where $l \to r \in \mathcal{R}$, $r \trianglerighteq t$, and $root(t) \in \mathcal{D}$.*
- *a* ll–dependency pair *(ll-DP) of $\mathcal{R}$ is a pair $l^{\#} \to u^{\#}$ where $l \to r \in \mathcal{R}$, $l \triangleright u$, and $root(u) \in \mathcal{D}$.*

The set of all (narrowing) dependency pairs of $\mathcal{R}$ is denoted by $NDP_{\mathcal{R}}$.

Example 5. The TRS $\mathtt{f}(\mathtt{f}(x)) \to x$ of Example 2 has no lr–dependency pairs and the single ll–dependency pair $\mathtt{f}^{\#}(\mathtt{f}(x)) \to \mathtt{f}^{\#}(x)$.

Example 6. For the TRS of Example 1 we obtain the narrowing dependency pairs shown in Figure 2.

[2] Modern formulations exclude pairs $l^{\#} \to u^{\#}$ when $l \triangleright u$. This refinement could be applied to lr-DPs in our definition, but the pair would not be actually discarded, since it is also computed as a ll-DP.

Recall that our purpose is to prove that there are no infinite narrowing derivations. Since dependency pairs model all function calls in $\mathcal{R}$, this is equivalent to proving that there are no infinite *chains* of narrowing dependency pairs.

For narrowing we consider suitable the following definition of chain. As in [11, 19], we assume that different occurrences of dependency pairs are variable disjoint. In the following, $\mathcal{P}$ is usually a set of dependency pairs.

Definition 3 (Narrowing Chain). *Let $\mathcal{P}, \mathcal{R}$ be two TRS's. A (possibly infinite) sequence of narrowing dependency pairs $s_1 \to t_1, s_2 \to t_2, \ldots, s_n \to t_n \in \mathcal{P}$ is called a ($\mathcal{P}$,$\mathcal{R}$)-*narrowing chain *if there exist terms $u_1, u_2, \ldots, u_n$ and substitutions $\sigma_1, \rho_1, \sigma_2, \rho_2, \ldots, \sigma_n, \rho_n$ s.t. $u_1 \overset{\epsilon}{\leadsto}_{\sigma_1, s_1 \to t_1} t_1\sigma_1 \overset{>\epsilon *}{\leadsto}_{\rho_1,\mathcal{R}} u_2 \overset{\epsilon}{\leadsto}_{\sigma_2, s_2 \to t_2} t_2\sigma_2 \overset{>\epsilon *}{\leadsto}_{\rho_2,\mathcal{R}} u_3 \cdots u_n \overset{\epsilon}{\leadsto}_{\sigma_n, s_n \to t_n} t_n\sigma_n$.*

We often omit the ($\mathcal{P}$,$\mathcal{R}$) prefix when referring to narrowing chains when it is clear from the context. The following result establishes the soundness of analyzing narrowing chains.

Lemma 2. *Let $\mathcal{R}$ be a TRS. For every (NDP$_\mathcal{R}$,$\mathcal{R}$)–narrowing chain $s_1 \to t_1$, $\ldots, s_n \to t_n$, there exists a narrowing derivation in $\mathcal{R}$ which gives at least one reduction step for every pair in the chain.*

Namely, there are contexts $C_1[\,], \ldots, C_{n+1}[\,]$, positions $p_1, \ldots, p_{n+1}$, terms $u_1, \ldots, u_n$, and substitutions $\tau_1, \ldots, \tau_n, \rho_1, \ldots, \rho_{n-1}$ s.t. $\tau_i = mgu(u_i, s_i)$ for $i \in \{1, \ldots, n\}$, and $C_1[u_1]_{p_1} \overset{p_1}{\leadsto}_{\tau_1,\mathcal{R}} C_2[t_1\tau_1]_{p_2} \overset{>p_2 *}{\leadsto}_{\rho_1,\mathcal{R}} C_2\rho_1[u_2]_{p_2} \overset{p_2}{\leadsto}_{\tau_2,\mathcal{R}} C_3[t_2\tau_2]_{p_3} \overset{>p_3 *}{\leadsto}_{\rho_2,\mathcal{R}} C_3\rho_2[u_3]_{p_3} \cdots C_n\rho_{n-1}[u_n]_{p_n} \overset{p_n}{\leadsto}_{\tau_n,\mathcal{R}} C_{n+1}[t_n\tau_n]_{p_{n+1}}$.

Now we are able to show that, whenever there are no infinite narrowing chains, narrowing does terminate.

Theorem 1 (Termination Criterion). *A TRS $\mathcal{R}$ is terminating for narrowing if and only if no infinite (NDP$_\mathcal{R}$,$\mathcal{R}$)–narrowing chain exists.*

Example 7. Consider the ll-DP $d \equiv \mathtt{f}^\#(f(x)) \to \mathtt{f}^\#(x)$ of Example 5. There is a narrowing chain $\mathtt{f}^\#(x) \rightsquigarrow_{\{x \mapsto \mathtt{f}(x')\},d} \mathtt{f}^\#(x') \rightsquigarrow_{\{x' \mapsto \mathtt{f}(x'')\},d} \mathtt{f}^\#(x'') \cdots$.

5 Automating the method

In order to automate the task of proving the absence of narrowing chains, it would be very convenient to reformulate the problem using only rewriting chains, as it is done e.g. in [19, 20, 17], since this allows us to reuse existing tools and techniques of the rewriting DP literature. We develop our method inspired by [18] but we provide all results without requiring TRAT, which is the main novel contribution of this section. Let us recall the notion of argument filtering.

Definition 4 (Argument Filtering). [6] *An argument filtering (AF) for a signature Σ is a mapping π that assigns to every n-ary function symbol $f \in \Sigma$ an argument position $i \in \{1, \ldots, n\}$, or a (possibly empty) list $[i_1, \ldots, i_m]$ of*

argument positions with $1 \leq i_i < \ldots < i_m \leq n$. *The signature* Σ_π *consists of all function symbols* $\mathtt{f}$ *s.t.* $\pi(\mathtt{f})$ *is some list* $[i_1, \ldots, i_m]$, *where in* Σ_π *the arity of* $\mathtt{f}$ *is* m. *Every AF* π *induces a mapping from* $\mathcal{T}(\Sigma, \mathcal{V})$ *to* $\mathcal{T}(\Sigma_\pi, \mathcal{V})$:

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

We extend π *to a TRS* $\mathcal{R}$ *as* $\pi(\mathcal{R}) = \{\pi(l) \to \pi(r) \mid l \to r \in \mathcal{R} \text{ and } \pi(l) \neq \pi(r)\}$. *For any argument filtering* π *and ordering* $>$, *we define* $s \geq_\pi t \iff \pi(s) > \pi(t)$ *or* $\pi(s) \equiv \pi(t)$. *We also define the filtering of a position* p *w.r.t. a term* t *as follows. Given a* n*-ary symbol* $f \in \Sigma$ *and* $i \in \{1, \ldots, n\}$, $\pi(i, f) = j$ *if* $\pi(f) = [i_1, \ldots, i_j, \ldots, i_k]$, $i_j = i$. *Given a term* t *and a position* $p \in \mathcal{P}os(t)$, *the filtering of* p *w.r.t.* t *is defined as follows:*

$$\pi(p, t) = \begin{cases} \epsilon & \text{if } p = \epsilon \\ \pi(q, t) & \text{if } p = q.i, i \in \mathbb{N}, \pi(root(t|_q)) = i \\ \pi(q, t).\pi(i, root(t|_q)) & \text{if } p = q.i, i \in \mathbb{N}, \pi(root(t|_q)) = [i_1, .., i, .., i_k] \end{cases}$$

Example 8. Consider the TRS of Example 1 and the argument filtering $\pi(\mathtt{pckt}) = [1, 3]$ and $\pi(f) = [1, \ldots, ar(f)]$ for any other $f \in \Sigma$. Let us consider the term $t = \mathtt{filter}(\mathtt{pckt}(\mathtt{secure}, dst, \mathtt{new}))$, its filtered version is $\pi(t) = \mathtt{filter}(\mathtt{pckt}(\mathtt{secure}, \mathtt{new}))$ and the filtering of position 1.3 is $\pi(1.3, \pi(t)) = 1.2$ where $\pi(1.2, \pi(t))$ is undefined.

Definition 5. *Given a TRS* $\mathcal{R}$ *and an AF* π, *we say that* π *is a sound AF for* $\mathcal{R}$ *iff* $\pi(\mathcal{R})$ *is a TRS, i.e., the rhs's of the rules do not contain extra variables not appearing in the corresponding lhs.*

Our main result in this section is Theorem 2 below that relates infinite narrowing $(\mathcal{P}, \mathcal{R})$–chains to infinite rewriting $(\pi(\mathcal{P}), \pi(\mathcal{R}))$–chains. In order to prove this result, we first need two auxiliary lemmata. The first one establishes a correspondence between rewriting derivations in $\mathcal{R}$ and derivations in the filtered TRS $\pi(\mathcal{R})$.

Lemma 3. *Given a TRS* $\mathcal{R}$, *a sound AF* π, *and terms* s *and* t, $s \to^*_{\mathcal{R}} t$ *implies* $\pi(s) \to^*_{\pi(\mathcal{R})} \pi(t)$. *Moreover, the derivation in* $\pi(\mathcal{R})$ *uses the same rules in the same order at the corresponding filtered positions (whenever the filtered position exists).*

The next lemma extends the correspondence established in Lemma 3 to narrowing, which can be done only when the original filtered term is ground. The key point is that the correspondence holds *regardless* of the substitution computed by narrowing. It is in fact a (one-way) lifting lemma from narrowing derivations in $\mathcal{R}$ to rewriting sequences in $\pi(\mathcal{R})$.

Lemma 4. *Given a TRS* $\mathcal{R}$ *and a sound AF* π, *let* s *and* t *be terms s.t.* $\pi(s)$ *is ground. Then* $s \leadsto^*_{\sigma,\mathcal{R}} t$ *implies* $\pi(s) \to^*_{\pi(\mathcal{R})} \pi(t)$. *Moreover, the derivation in* $\pi(\mathcal{R})$ *uses the same rules in the same order at the corresponding filtered positions (whenever the filtered position exists).*

Let us recall here the standard definition of chain for rewriting.

Definition 6 (Chain). [6, 11] *Let $\mathcal{P}, \mathcal{R}$ be two TRS's. A (posibly infinite) sequence of pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ from $\mathcal{P}$ is a $(\mathcal{P}, \mathcal{R})$–chain if there exists a substitution σ with $t_i\sigma \to_{\mathcal{R}} s_{i+1}$ for all i.*

The following result allows us to prove the absence of narrowing chains by analyzing standard rewriting chains. This is very useful because it means that we can reuse all the DP infrastructure available for rewriting.

Theorem 2. *Let $\mathcal{R}$ be a TRS over a signature Σ, $\mathcal{P}$ be a TRS over a signature $\Sigma^\#$, and π a sound AF over $\Sigma^\#$ s.t. $\pi(t)$ is ground for at least one pair $s \to t \in \mathcal{P}$ in every ($\mathcal{P}$,$\mathcal{R}$)–narrowing chain. If there exists no infinite $(\pi(\mathcal{P}), \pi(\mathcal{R}))$–chain, then there exists no infinite ($\mathcal{P}$,$\mathcal{R}$)–narrowing chain.*

The following straightforward consequence of Theorems 1 and 2 characterizes termination of narrowing as a rewriting problem.

Corollary 1. *Let $\mathcal{R}$ be a TRS over a signature Σ, and π a sound AF over $\Sigma^\#$ s.t. $\pi(t)$ is ground for at least one pair $s \to t \in NDP_{\mathcal{R}}$ in every $(NDP_{\mathcal{R}}, \mathcal{R})$–narrowing chain. If there exists no infinite $(\pi(NDP_{\mathcal{R}}), \pi(\mathcal{R}))$–chain, then narrowing terminates in $\mathcal{R}$.*

5.1 Extending the DP framework to narrowing

By means of Theorem 2, it is possible now to recast the problem of termination of narrowing in the DP *framework* of [9]. In this framework a DP *problem* is a tuple $(\mathcal{P}, \mathcal{R})$ of two TRSs, $\mathcal{R}$ and $\mathcal{P}$, where initially $\mathcal{P} = NDP_{\mathcal{R}}$. If there is no associated infinite narrowing chain, we say that the problem is *finite*. Termination methods are then formulated as *DP processors* that take a DP problem and return a new set of DP problems. A DP processor is *sound* if the input problem is finite whenever all the output problems are. We speak of *narrowing DP problems* to distinguish them from the standard ones.

In the usual style [20, 17], we show here how to adapt a few of the most important DP processors, and then give one that transforms a narrowing DP problem into a rewriting one, which allows us to use any of the existing DP processors for termination of rewriting.

The following definition adapts the standard notion of dependency graph to our setting by simply considering narrowing dependency pairs instead of vanilla DPs.

Definition 7 (Dependency Graph). *Given a (narrowing) DP problem $\langle \mathcal{P}, \mathcal{R} \rangle$ its (resp. narrowing) dependency graph is the directed graph where the nodes are the elements of $\mathcal{P}$, and there is an edge from $s \to t \in \mathcal{P}$ to $u \to v \in \mathcal{P}$ if $s \to t$, $u \to v$ is a (resp. narrowing) chain from $\mathcal{P}$.*

The theorem below establishes that the narrowing dependency graph of a narrowing DP problem is equal to the dependency graph of the rewriting DP problem defined by the same TRS and DP set.

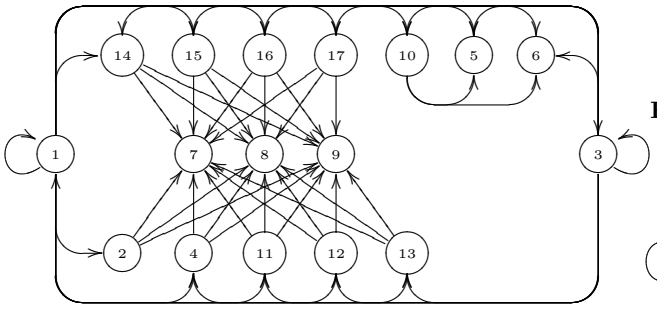

Fig. 3. Estimated dependency graph of *FullPolicy*

Fig. 4. Filtered *FullPolicy*

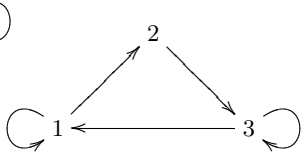

Fig. 5. Dependency Graph

Theorem 3. *Given a narrowing DP problem $\langle \mathcal{P}, \mathcal{R} \rangle$, its narrowing dependency graph is the same as the dependency graph of the rewriting DP problem defined by $\langle \mathcal{P}, \mathcal{R} \rangle$.*

It is well known that computing the exact dependency graph is undecidable and thus several approximations [11] are used to compute an *estimated* dependency graph which includes the exact graph. The following approximation is commonly used.

Definition 8 (Estimated Dependency Graph). [9] *Let $\langle \mathcal{P}, \mathcal{R} \rangle$ be a DP problem. Let $\mathsf{CAP}_{\mathcal{R}}(t)$ be the result of replacing* [3] *all the proper subterms of t with a defined root symbol by a fresh variable, and $\mathsf{REN}(t)$ the linearization of t (replacing all ocurrences of a non linear variable with independent fresh variables). The nodes of the estimated dependency graph (EDG) are the pairs of $\mathcal{P}$ and there is an edge from $s^{\#} \to t^{\#}$ to $u^{\#} \to v^{\#}$ iff $\mathsf{REN}(\mathsf{CAP}_{\mathcal{R}}(t))$ and u are unifiable.*

Example 9. For the problem of Example 1 and the set of DPs obtained in Example 6, the EDG is shown in Figure 3.

For finite TRSs, infinite chains show up as cycles in the dependency graph[4]. We can analyze separately every chain, that is, every cycle in the dependency graph. This is accomplished by the following DP processor.

Theorem 4 (Dependency Graph Processor). [9] *For a DP problem $\langle \mathcal{P}, \mathcal{R} \rangle$, let Proc be the processor that returns problems $\{\langle \mathcal{P}_1, \mathcal{R} \rangle, \ldots, \langle \mathcal{P}_n, \mathcal{R} \rangle\}$, where $\mathcal{P}_1, \ldots, \mathcal{P}_n$ are the sets of nodes of every cycle in the estimated dependency graph. Proc is sound.*

[3] This function was first defined for approximating loops in dependency graphs in [5], where it is called $\overset{\circ}{t}$.

[4] The converse does not hold, not every cycle corresponds to an infinite chain.

Example 10. In the graph obtained in the EDG of Example 9, the only cycle consists of (1) and (3). Thus the dependency graph processor deletes all the other dependency pairs, and returns the problems { ({(1),(3)}, $\mathcal{R}$), ({(1)}, $\mathcal{R}$), ({(3)}, $\mathcal{R}$)} corresponding to the graph shown in Figure 4.

The next processor we adapt is the standard reduction pair processor. The following is the standard notion of reduction pair.

Definition 9 (Reduction Pair). *A reduction pair ($\succeq$, $\succ$) consists of a quasi-rewrite ordering $\succeq$ and an ordering $\succ$ with the following properties: (i) $\succ$ is closed under substitutions and well founded, and (ii) $(\succ \circ \succeq) \subseteq \succ$.*

For a narrowing DP problem $\langle \mathcal{P}, \mathcal{R} \rangle$, this processor tries to find a reduction pair ($\succeq$, $\succ$) and a suitable filtering π s.t. all the filtered $\mathcal{R}$-rules are weakly decreasing w.r.t. $\succeq$, and all filtered $\mathcal{P}$ pairs are weakly or strictly decreasing. For any TRS $\mathcal{P}$ and relation $\succ$, let $\mathcal{P}_\succ = \{s \to t \mid s \succ t\}$.

Theorem 5 (Reduction Pair processor). *Let ($\mathcal{P}$,$\mathcal{R}$) be a narrowing DP problem s.t. $\mathcal{P}$ is a cycle*[5]*, ($\succeq$, $\succ$) be a reduction pair, and π be an argument filtering s.t. $\pi(t)$ is ground for at least one pair $s \to t \in \pi(\mathcal{P})$. Then $Proc_\pi(\mathcal{P}, \mathcal{R})$ returns $\{(\mathcal{P} \setminus \mathcal{P}_{\succ_\pi}, \mathcal{R})\}$ if $\mathcal{P}_{\succ_\pi} \cup \mathcal{P}_{\succeq_\pi} = \mathcal{P}$, $\mathcal{P}_{\succ_\pi}$ is not empty, and $\mathcal{R}_{\succeq_\pi} = \mathcal{R}$; $\{(\mathcal{P}, \mathcal{R})\}$ otherwise.*

Note that it is not enough to consider all the pairs in a strongly connected component (SCC) at once, as it is commonly done in rewriting, and that we consider cycles instead. The reason is that the condition of Theorem 2, groundness of one DP rhs per chain (cycle), would not be ensured when working with SCCs instead.

Example 11. Consider a TRS $\mathcal{R}$ with the Dependency Graph of Figure 5. Our dependency graph processor decomposes this problem into three subproblems corresponding to the cycles {1}, {3} and {1,2,3}. A SCC problem would consider only the last of these three. Suppose we did indeed use SCCs. The reduction pair processor defined above can synthetize a filtering π_2 s.t. the rhs of (2) is ground and an ordering s.t. (3) can be oriented strictly; upon doing so it will remove (3) of the DP problem, thus leaving only {1,2}. This eliminates two cycles at once, {3} and {1,2,3}. But this is unsound, since we cannot eliminate the cycle in {3} unless we find an argument filtering π_3 s.t. the rhs of (3) is ground and there is a suitable ordering.

We claim that it is straightforward to adapt most of the standard DP processors in order to deal with the grounding AF requirement, and due to lack of space we will present only one more processor, which can be used to transform a narrowing DP problem into an ordinary one. Afterwards, any existing DP processor for rewriting becomes applicable.

[5] Note that this requirement is easily fulfilled by running the dependency graph processor first.

Theorem 6 (Argument Filtering Processor). *Let $(\mathcal{P}, \mathcal{R})$ be a narrowing DP problem s.t. $\mathcal{P}$ is a cycle, and π be an argument filtering s.t. $\pi(t)$ is ground for at least one pair $s \to t \in \pi(\mathcal{P})$. Then, $Proc_\pi(\mathcal{P}, \mathcal{R}) = \{(\mathcal{P}_\pi, \pi(\mathcal{R})\}$, where $\mathcal{P}_\pi$ is defined as $\mathcal{P}_\pi = \{\pi(l) \to \pi(r) \mid l \to r \in \mathcal{P}, l \not\trianglerighteq r\}$. $Proc_\pi$ is a sound narrowing DP processor.*

Finally, we include the subterm refinement in the AF processor as it can be the case that the rhs of a DP becomes a subterm of the lhs after the filtering.

Example 12. The set of narrowing DP problems resulting of Example 10 can be solved by using the AF processor to transform them into rewriting problems.

- $(\{(1)\}, \mathcal{R})$ For this problem soundness requires that $\pi(\mathtt{pckt}) = [1, 3]$. Using the identity for all other symbols, we get the following (rewriting) DP problem that is finite, as one can easily check with a modern termination tool implementing the DP method such as Aprove [10], or Mu-Term [1]: $(\{\mathtt{filter}^\#(\mathtt{pckt}(\mathtt{194.179.1}.x\!:\!p, \mathtt{new})) \to \mathtt{filter}^\#(\mathtt{pckt}(\mathtt{secure}, \mathtt{new})\}, \mathcal{R})$
- $(\{(3)\}, \mathcal{R})$ In this case, we proceed in a similar way, and the same AF π allows us to transform the current subproblem into a finite (rewriting) DP problem.
- $(\{(1),(3)\}, \mathcal{R})$ Finally, by using the same AF π, we get a finite DP problem.

This finally proves that the FullPolicy TRS is terminating for narrowing.

6 Conclusion

We have introduced a new technique for termination proofs of narrowing via termination of rewriting that is based on a suitable generalization of dependency pairs. Although several refinements of the notion of dependency pairs such as [11, 13] had been proposed previously for termination analysis of TRSs, this is the first time that the notion of dependency pair has been *extended* to deal with narrowing on arbitrary TRSs and queries. This is possible because we first identified the problem of *echoing*, which is ultimately responsible for narrowing non–termination. Our contribution is threefold: 1) we ascertained the suitable notions that allow us to detect when the terms in a narrowing derivation actually do echo; 2) our approach leads to much weaker conditions for verifying the termination of narrowing that subsume all previously known termination of narrowing criteria; 3) the resulting method can be effectively mechanized. We have implemented our technique in a tool that is publicly available[6]. and satisfactorily evaluated this tool on large example sets.

References

1. B. Alarcón, R. Gutiérrez, J. Iborra, and S. Lucas. Proving termination of context-sensitive rewriting with Mu–Term. *ENTCS*, 188:105–115, 2007.

[6] http://www.dsic.upv.es/users/elp/soft/narradar

2. M. Alpuente, S. Escobar, and J. Iborra. Modular Termination of Basic Narrowing. In *19th Int'l Conf. on Rewriting Techniques and Applications, RTA'08*, LNCS 5117:1–16, 2008.
3. M. Alpuente, S. Escobar, and J. Iborra. Dependency Pairs for the Termination of Narrowing. Technical Report DSIC-II/08/08, DSIC-UPV, 2008.
4. M. Alpuente, S. Escobar, and J. Iborra. Termination of Narrowing revisited. *Theor. Comput. Sci.*, 2008. To appear.
5. M. Alpuente, M. Falaschi, and G. Vidal. Compositional Analysis for Equational Horn Programs. In *4th Int'l Conf. on Algebraic and Logic Programming, ALP'94*, LNCS 850:77-94, 1994.
6. T. Arts and J. Giesl. Termination of Term Rewriting using Dependency Pairs. *Theor. Comput. Sci.*, 236(1-2):133–178, 2000.
7. J. Christian. Some termination criteria for narrowing and e-narrowing. In *11th Int'l Conf. on Automated Deduction CADE'92*, LNCS 607: 582–588, 1992.
8. S. Escobar, C. Meadows, and J. Meseguer. A Rewriting-Based Inference System for the NRL Protocol Analyzer and its Meta-Logical Properties. *Theor. Comput. Sci.*, 367(1-2):162–202, 2006.
9. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The dependency pair framework: Combining techniques for automated termination proofs. In *11th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2005*, LNCS 3452:301–331, 2005.
10. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with AProVe. In *15th Int'l Conf. on Rewriting Techniques and Applications, RTA'04*, LNCS 3091:210–220, 2004.
11. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and Improving Dependency Pairs. *J. Autom. Reasoning*, 37(3):155–203, 2006.
12. M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *J. Log. Program.*, 19&20:583–628, 1994.
13. N. Hirokawa and A. Middeldorp. Dependency pairs revisited. In *15th Int'l Conf. on Rewriting Techniques and Applications, RTA'04*, LNCS 3091:249–268, 2004.
14. J.-M. Hullot. Canonical Forms and Unification. In *5th Int'l Conf. on Automated Deduction CADE'80*, LNCS 87: 318–334, 1980.
15. C. Kirchner, H. Kirchner, and A. Santana de Oliveira. Analysis of Rewrite-Based Access Control Policies. In *3rd Int'l Workshop on Security and Rewriting Techniques, SecreT 2008*. ENTCS, 2008, to appear.
16. J. Meseguer and P. Thati. Symbolic reachability analysis using narrowing and its application to verification of cryptographic protocols. *Higher-Order and Symbolic Computation*, 20(1-2):123–160, 2007.
17. M. T. Nguyen, P. Schneider-Kamp, D. de Schreye, and J. Giesl. Termination Analysis of Logic Programs based on Dependency Graphs. In *17th Int'l Sym. on Logic-Based Program Synthesis and Transformation, LOPSTR'07*, LNCS 4915:8–22, 2007.
18. N. Nishida and K. Miura. Dependency graph method for proving termination of narrowing. In *8th Int'l Workshop on Termination, WST'06*, 2006.
19. N. Nishida, M. Sakai, and T. Sakabe. Narrowing-based simulation of term rewriting systems with extra variables. *ENTCS*, 86(3), 2003.
20. N. Nishida and G. Vidal. Termination of Narrowing via Termination of Rewriting. 2008. Available at http://www.dsic.upv.es/ gvidal.
21. TeReSe, editor. *Term Rewriting Systems*. Cambridge University Press, Cambridge, UK, 2003.