

Termination of Narrowing in Left-Linear Constructor Systems

Germán Vidal

Technical University of Valencia, Spain
gvidal@dsic.upv.es

Abstract. Narrowing extends rewriting with logic capabilities by allowing free variables in terms and replacing matching with unification. Narrowing has been widely used in different contexts, ranging from theorem proving (e.g., protocol verification) to language design (e.g., it forms the basis of so called functional logic languages). Surprisingly, the termination of narrowing has been mostly overlooked. In this paper, we present new techniques for proving the termination of narrowing in left-linear constructor systems—a widely accepted class of systems in functional and functional logic programming—that allow us to reuse existing methods in the extensive literature on termination of rewriting.

1 Introduction

The narrowing principle [52] generalizes term rewriting by allowing free variables in terms—as in logic programming [39]—and replacing matching with unification in order to (non-deterministically) reduce these terms. Since unrestricted narrowing can have a huge search space—because one can freely select any redex *and* applicable rewrite rule—great effort has been devoted in the last two decades to design sophisticated narrowing *strategies* that prune the narrowing search space without losing completeness. Some well established narrowing strategies are, e.g., *basic* narrowing [34], *innermost* narrowing [25], *lazy* narrowing [15, 43, 49], and *needed* narrowing [3]. A survey of results about the completeness of narrowing strategies can be found in [11, 21, 31].

Narrowing, originally introduced as an *E*-unification mechanism in equational theories, has been mostly used as the operational semantics of so called *functional logic programming languages* [49] (some languages based on narrowing are, e.g., LPG [9], SLOG [25], ALF [30], Babel [44], and the most recent Curry [22] and Toy [40]). Narrowing has also been used in a number of other areas, e.g., protocol verification [13, 23, 35, 41], refining methods for proving the termination of rewriting [6, 7], type checking in the language Ω mega [51], etc.

Termination is a fundamental problem in term rewriting, as witnessed by the extensive literature on the subject (see, e.g., the surveys of Dershowitz [18] and Steinbach [53]). Surprisingly, the termination of narrowing has been mostly overlooked so far. Actually, to the best of our knowledge, no termination analysis tool has ever been developed for proving the termination of narrowing. Indeed, only some rather simple or very specific approaches can be found in the literature (see a detailed account in Section 5).

Our main contribution in this work is the definition of sufficient conditions for the termination of narrowing that can be checked by using existing techniques and tools for the termination analysis of standard rewriting.

The paper is organized as follows. After providing some preliminary definitions in Sect. 2, we present our basic results that relate the termination of narrowing to that of rewriting in Sect. 3. Then, Sect. 4 considers the automated termination analysis of narrowing and presents two approaches: a direct approach based on an adaptation of the dependency pair framework [28] and a transformational approach based on the argument filtering transformation [38]. Finally, Sect. 5 includes a comparison to related work and Sect. 6 concludes.

2 Preliminaries

We assume familiarity with basic concepts of term rewriting and narrowing. We refer the reader to, e.g., [8] and [31] for further details.

Terms and substitutions. A *signature* \mathcal{F} is a set of function symbols. We often write $f/n \in \mathcal{F}$ 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 $f/0$. 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 $\text{root}(t)$. 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 . $\text{Var}(t)$ denotes the set of variables appearing in t . A term t is *ground* if $\text{Var}(t) = \emptyset$. 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 its domain $\text{Dom}(\sigma) = \{x \in \mathcal{V} \mid x \neq \sigma(x)\}$ is finite. Substitutions are extended to morphisms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ in the natural way. We often 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*. A *variable renaming* is a substitution that is a bijection on \mathcal{V} . We say that a substitution σ is *more general* than a substitution θ , denoted by $\sigma \leq \theta$, if there exists a substitution δ such that $\delta \circ \sigma = \theta$, where “ \circ ” denotes the composition of substitutions (i.e., $\sigma \circ \theta(x) = x\theta\sigma$). The *restriction* $\theta \upharpoonright_V$ of a substitution θ to a set of variables V is defined as follows: $x\theta \upharpoonright_V = x\theta$ if $x \in V$ and $x\theta \upharpoonright_V = x$ otherwise.

A term t_2 is an *instance* of a term t_1 (or, equivalently, t_1 is *more general* than t_2), in symbols $t_1 \leq t_2$, if there is a substitution σ with $t_2 = t_1\sigma$. Two terms t_1 and t_2 are *variants* (or equal up to variable renaming) if $t_1 = t_2\rho$ for some variable renaming ρ . A *unifier* of two terms t_1 and t_2 is a substitution σ with $t_1\sigma = t_2\sigma$; furthermore, σ is the *most general unifier* of t_1 and t_2 , denoted by $\text{mgu}(t_1, t_2)$ if, for every other unifier θ of t_1 and t_2 , we have that $\sigma \leq \theta$.

TRSs and rewriting. A set of rewrite rules $l \rightarrow r$ such that l is a nonvariable 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 $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. 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 $x \in \text{Dom}(\sigma)$.

A TRS \mathcal{R} is a *constructor* system if the left-hand sides of its rules have the form $f(s_1, \dots, s_n)$ where s_i are constructor terms, i.e., $s_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$, for all $i = 1, \dots, n$. 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 all rules $l \rightarrow r \in \mathcal{R}$.

For a TRS \mathcal{R} , we define the associated rewrite relation $\rightarrow_{\mathcal{R}}$ as follows: given terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have $s \rightarrow_{\mathcal{R}} t$ iff there exists a position p in s , a rewrite rule $(l \rightarrow r) \in \mathcal{R}$ and a substitution σ with $s|_p = l\sigma$ and $t = s[r\sigma]_p$; the rewrite step is often denoted by $s \rightarrow_{p, l \rightarrow r} t$ 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 $t \rightarrow_{\mathcal{R}} s$. A substitution σ is *normalized* in a TRS \mathcal{R} iff the terms $x\sigma$ are irreducible in \mathcal{R} for all $x \in \text{Dom}(\sigma)$. A *derivation* is a (possibly empty) sequence of rewrite steps. Given a binary relation \rightarrow , we denote by \rightarrow^+ the transitive closure of \rightarrow and by \rightarrow^* its reflexive and transitive closure. Thus $t \rightarrow_{\mathcal{R}}^* s$ means that t can be reduced to s in \mathcal{R} in zero or more steps; we also use $t \rightarrow_{\mathcal{R}}^n s$ to denote that t can be reduced to s in exactly n steps.

Narrowing. The *narrowing* mechanism [52] mainly extends rewriting by replacing matching with unification, so that terms containing free variables can also be (non-deterministically) reduced. Formally, given a TRS \mathcal{R} and two terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have that $s \rightsquigarrow_{\mathcal{R}} t$ is a *narrowing step* iff

- there exist a nonvariable position p of t ,
- a variant $R = (l \rightarrow r)$ of a rule in \mathcal{R} ,
- a substitution $\sigma = \text{mgu}(s|_p, l)$ which is the most general unifier of $s|_p$ and l ,
- and $t = (s[r]_p)\sigma$.

We often write $s \rightsquigarrow_{p, R, \sigma} t$ (or simply $s \rightsquigarrow_{\sigma} t$) to make explicit the position, rule, and substitution of the narrowing step. Furthermore, in the remainder of this paper, we always label the narrowing relation with $\sigma \upharpoonright_{\text{Var}(s)}$ rather than σ . A *narrowing derivation* $t_0 \rightsquigarrow_{\sigma}^* t_n$ denotes a sequence of narrowing steps $t_0 \rightsquigarrow_{\sigma_1} \dots \rightsquigarrow_{\sigma_n} t_n$ with $\sigma = \sigma_n \circ \dots \circ \sigma_1$ (if $n = 0$ then $\sigma = \text{id}$). Given a narrowing derivation $s \rightsquigarrow_{\sigma}^* t$, we say that σ is a computed *answer* for s .

Example 1. Consider the following TRS \mathcal{R} defining the addition `add` on natural numbers built from `z/0` and `s/1`:

$$\begin{aligned} \text{add}(z, y) &\rightarrow y && (R_1) \\ \text{add}(s(x), y) &\rightarrow s(\text{add}(x, y)) && (R_2) \end{aligned}$$

Given the term `add(x, s(z))`, we have infinitely many narrowing derivations issuing from `add(x, s(z))`, e.g.:

$$\begin{aligned} \text{add}(x, s(z)) &\rightsquigarrow_{\epsilon, R_1, \{x \mapsto z\}} s(z) \\ \text{add}(x, s(z)) &\rightsquigarrow_{\epsilon, R_2, \{x \mapsto s(y_1)\}} s(\text{add}(y_1, z)) \rightsquigarrow_{1, R_1, \{y_1 \mapsto z\}} s(s(z)) \\ &\dots \end{aligned}$$

with computed answers $\{x \mapsto z\}$, $\{x \mapsto s(z)\}$, etc.

Since unrestricted narrowing has quite a large search space, several strategies to control the selection of redexes have been defined. A *narrowing strategy* is often given by a mapping that takes a term and returns a set of triples (p, R, σ) so that only the narrowing steps $t \rightsquigarrow_{p,R,\sigma} t'$ are allowed. Some well established narrowing strategies are, e.g., *basic* narrowing [34], *innermost* narrowing [25], *lazy* narrowing [15, 43, 49], and *needed* narrowing [3]. A survey of results about the completeness of narrowing strategies can be found in [11, 21, 31].

We note that, in our definition of a narrowing step $s \rightsquigarrow_{p,l \rightarrow r,\sigma} t$, the substitution σ should always be the most general unifier of $s|_p$ and l (restricted to the variables of s). A notable exception to this definition is needed narrowing [3], where σ needs not be $\text{mgu}(s|_p, l)$ but only a unifier of $s|_p$ and l . In our definition of narrowing, we required σ to be $\text{mgu}(s|_p, l)$ in order to ensure that all symbols in σ come from either $s|_p$ or l . Since this property also holds for needed narrowing unifiers, we could straightforwardly generalize our development in this paper to the case of needed narrowing. We keep however our previous definition of narrowing step in order to ease the understanding and simplify the technical notations.

3 Termination of Narrowing via Termination of Rewriting

In this section, we present two sufficient—and necessary—conditions for the termination of narrowing in terms of the termination of rewriting. First, let us introduce our notion of termination, which is parameterized by a given binary relation:

Definition 1 (termination). *Let T be a finite set of terms. Given a binary relation α on terms, we say that T is α -terminating iff there is no term $t \in T$ such that there exists an infinite sequence of the form $t \alpha t_1 \alpha t_2 \alpha \dots$*

The usual notion of termination can then be formulated as follows: a TRS is *terminating* iff $\mathcal{T}(\mathcal{F})$ is $\rightarrow_{\mathcal{R}}$ -terminating. As for narrowing, we say that a TRS \mathcal{R} is *terminating w.r.t. narrowing* iff $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is $\rightsquigarrow_{\mathcal{R}}$ -terminating.

In general, however, only rather trivial TRSs are terminating w.r.t. narrowing. For instance, given the following simple TRS:

$$f(s(x)) \rightarrow f(x)$$

we can easily find a term, e.g., $f(y)$, such that an infinite narrowing derivation exists: $f(y) \rightsquigarrow_{\{y \mapsto s(x_1)\}} f(x_1) \rightsquigarrow_{\{x_1 \mapsto s(x_2)\}} f(x_2) \rightsquigarrow_{\{x_2 \mapsto s(x_3)\}} \dots$. Therefore, in the remainder of the paper, we only consider the termination of narrowing w.r.t. a *given set of terms*, as introduced in Definition 1.

In narrowing, the reduction of a term can be non-terminating either because there exists an infinite narrowing derivation or because there are infinitely many (possibly finite) narrowing derivations issuing from this term. By the soundness of narrowing (see, e.g., [42, Lemma 3.3]), for every narrowing derivation

$$t \rightsquigarrow_{p_1, R_1, \sigma_1} t_1 \rightsquigarrow_{p_2, R_2, \sigma_2} \dots \rightsquigarrow_{p_n, R_n, \sigma_n} t_n$$

there exists a corresponding rewriting derivation of the form

$$t\sigma_1\sigma_2 \dots \sigma_n \rightarrow_{p'_1, R'_1} t'_1 \rightarrow_{p'_2, R'_2} \dots \rightarrow_{p'_n, R'_n} t'_n$$

with $p_i = p'_i$, $R_i = R'_i$, and $t_i = t'_i$ for all $i = 1, \dots, n$. Hence, non-termination of narrowing implies non-termination of rewriting (the opposite is not true, however).

The following result provides a simple—sufficient and necessary—condition for the termination of narrowing in terms of the termination of rewriting:

Theorem 1. *Let \mathcal{R} be a TRS and T be a finite set of terms. Let $T^* = \{t\sigma \mid t \in T \text{ and } t \rightsquigarrow_\sigma^* s \text{ in } \mathcal{R}\}$. T is $\rightsquigarrow_{\mathcal{R}}$ -terminating iff T^* is finite and $\rightarrow_{\mathcal{R}}$ -terminating.*

Proof. We prove the claim by contradiction.

(\Rightarrow) Assume that T is $\rightsquigarrow_{\mathcal{R}}$ -terminating but either T^* is infinite or T^* is not $\rightarrow_{\mathcal{R}}$ -terminating. Consider first that T^* contains an infinite number of terms. Then, there is some term $t \in T$ such that there exist infinite narrowing derivations issuing from t . Since the signature is finite and we only consider finite terms over this signature, one of these derivations must be infinite, which contradicts our assumption. Otherwise, consider that T^* is not $\rightarrow_{\mathcal{R}}$ -terminating. Then, there is some term $s \in T^*$ such that there exists an infinite sequence of the form $s \rightarrow_{\mathcal{R}} s_1 \rightarrow_{\mathcal{R}} s_2 \rightarrow_{\mathcal{R}} \dots$. By construction, there must be a term $t \in T$ such that $t \rightsquigarrow_{\sigma_1} t_1 \rightsquigarrow_{\sigma_2} \dots$ is a narrowing derivation with $t\sigma_1\sigma_2\dots = s$. Since both derivations employ the same rules at the same positions, then $t \rightsquigarrow_{\sigma_1} t_1 \rightsquigarrow_{\sigma_2} \dots$ must also be infinite, thus contradicting our assumption.

(\Leftarrow) Assume now that T^* is finite and $\rightarrow_{\mathcal{R}}$ -terminating but T is not $\rightsquigarrow_{\mathcal{R}}$ -terminating. Then, there must be some $t \in T$ such that there exists an infinite derivation of the form $t \rightsquigarrow_{\sigma_1} t_1 \rightsquigarrow_{\sigma_2} t_2 \rightsquigarrow_{\sigma_3} \dots$. Then, we have two possibilities. First, if the term $t\sigma_1\sigma_2\dots$ grows infinitely with the application of every σ_i , then the set T^* is infinite, which contradicts our initial assumption. Otherwise, there must be some finite $j > 0$ such that $\sigma_k = id$ (recall that every σ_k is restricted to $\text{Var}(t_k)$) for all $k > j$. Then, we can write the infinite narrowing derivation as $t \rightsquigarrow_{\sigma_1} t_1 \rightsquigarrow_{\sigma_2} \dots \rightsquigarrow_{\sigma_j} t_j \rightsquigarrow_{id} t_{j+1} \rightsquigarrow_{id} t_{j+2} \rightsquigarrow_{id} \dots$. By the soundness of narrowing, we have $t\sigma_1\dots\sigma_j \rightarrow_{\mathcal{R}} \dots \rightarrow_{\mathcal{R}} t_j$. Finally, since $t_k \rightsquigarrow_{id} t_{k+1}$ iff $t_k \rightarrow_{\mathcal{R}} t_{k+1}$, the previous rewrite derivation goes on infinitely as follows: $t_j \rightarrow_{\mathcal{R}} t_{j+1} \rightarrow_{\mathcal{R}} t_{j+1} \rightarrow_{\mathcal{R}} \dots$, which contradicts our initial assumption. \square

A key observation is that variables in narrowing can be seen as (possibly) *infinite* terms from the point of view of rewriting. This relation between logic variables and infinite terms has been pointed out by Dershowitz [19], who advocated a form of *stream programming* based on logic variables. A similar idea has taken up recently in order to eliminate logic variables from functional logic computations [4, 16] and, more closely related to our purposes, to analyze the termination of logic programs by translating them to TRSs [50].

Following, e.g., [37], we only consider infinite terms which, when considered as trees, have the property that each node is at finite distance from the root, i.e., an infinite term of the form $f(s(s(\dots)))$ is allowed but $f(s(s(\dots(z))))$ is not. The set of (possibly infinite) terms is denoted by $\mathcal{T}^\infty(\mathcal{F}, \mathcal{V})$. The only difference between standard finitary term rewriting and the kind of infinitary rewriting that we consider in this paper is that the set of terms over a signature is extended by the infinite terms over that signature.¹ Then, the notions of substitution, redex, rewrite step, normal form, etc., generalize naturally to the set of infinite terms.

Unfortunately, the condition of Theorem 1, i.e., verifying the $\rightarrow_{\mathcal{R}}$ -termination of $T^* = \{t\sigma \mid t \in T \text{ and } t \rightsquigarrow_\sigma s \text{ in } \mathcal{R}\}$, is not only undecidable but also rather difficult to approximate (since one should approximate all possible narrowing derivations

¹ This contrasts to [37], where the right-hand sides of rules can be infinite terms.

issuing from the terms in T^*). Therefore, we now consider an alternative condition which is based on approximating T^* with the set $\{t\sigma \mid t \in T, \sigma : \mathcal{V} \mapsto \mathcal{T}^\infty(\mathcal{F}, \mathcal{V})\}$, i.e., we replace the answers computed by narrowing by arbitrary (possibly infinite) substitutions.

Example 2. Consider again the TRS \mathcal{R} of Example 1 and the term $\text{add}(x, z)$. Clearly, $\text{add}(x, z)\sigma$ is $\rightarrow_{\mathcal{R}}$ -terminating for every substitution σ mapping x to a finite term. However, if σ maps x to an infinite term of the form $s(s(\dots))$, then the derivation for $\text{add}(x, z)\sigma$ is now infinite:

$$\text{add}(s(s(\dots)), z) \rightarrow_{\mathcal{R}} s(\text{add}(s(s(\dots)), z)) \rightarrow_{\mathcal{R}} s(s(\text{add}(s(s(\dots)), z))) \rightarrow_{\mathcal{R}} \dots$$

Indeed, $\text{add}(x, z)$ is not $\rightsquigarrow_{\mathcal{R}}$ -terminating.

Unfortunately, proving that the set $\{t\sigma \mid t \in T, \sigma : \mathcal{V} \mapsto \mathcal{T}^\infty(\mathcal{F}, \mathcal{V})\}$ is $\rightarrow_{\mathcal{R}}$ -terminating is often an unnecessarily strong condition in order to prove that T is $\rightsquigarrow_{\mathcal{R}}$ -terminating, as the following example illustrates:

Example 3. Consider the following TRS $\mathcal{R} = \{a \rightarrow a, f(x) \rightarrow x\}$. While the term $f(x)$ is clearly $\rightsquigarrow_{\mathcal{R}}$ -terminating, there exists a substitution $\sigma = \{x \mapsto a\}$ such that $f(x)\sigma$ is not $\rightarrow_{\mathcal{R}}$ -terminating. Here, the infinite computation $f(a) \rightarrow_{\mathcal{R}} f(a) \rightarrow_{\mathcal{R}} \dots$ has been introduced by σ .

In order to avoid this problem, one could forbid the reduction of redexes introduced by σ (as well as their *descendants* [33]). However, such a restriction on rewriting derivations would make the previous condition unsound:

Example 4. Consider the following TRS $\mathcal{R} = \{a \rightarrow a, f(x, a) \rightarrow b\}$. Given the term $t = c(y, f(y, y))$, we have that $t\sigma$ is $\rightarrow_{\mathcal{R}}$ -terminating if the reduction of the terms introduced by σ (and their descendants) is forbidden. However, t is not $\rightsquigarrow_{\mathcal{R}}$ -terminating since there exists an infinite narrowing derivation: $c(y, f(y, y)) \rightsquigarrow_{\{y \mapsto a\}} c(a, b) \rightsquigarrow_{id} c(a, b) \rightsquigarrow_{id} \dots$

These problems, though, can be overcome by considering a narrowing strategy over a class of TRSs in which the terms introduced by instantiation are not narrowed. Actually, many useful narrowing strategies fulfill this condition, e.g., basic narrowing [34] and innermost basic narrowing [32] over arbitrary TRSs, lazy narrowing [15, 43, 49] and needed narrowing [3] over left-linear constructor TRSs, etc. Indeed, any narrowing strategy over left-linear constructor systems fulfill this condition, since the following trivial property holds:

Lemma 1. *Let $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ be a signature.² Let $f(t_1, \dots, t_n)$ be a linear term with $f \in \mathcal{D}$ and $t_1, \dots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{V})$. Given an arbitrary term $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ with $\text{Var}(f(t_1, \dots, t_n)) \cap \text{Var}(s) = \emptyset$, we have that $\text{mgu}(f(t_1, \dots, t_n), s)|_{\text{Var}(s)}$ is a constructor substitution.*

For simplicity, we restrict ourselves to left-linear constructor systems in our following result which states a more useful sufficient (and necessary) condition:

² 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} .

Theorem 2. *Let \mathcal{R} be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and let $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a finite set of terms. Then, T is $\rightsquigarrow_{\mathcal{R}}$ -terminating iff $\{t\sigma \mid t \in T, \sigma : \mathcal{V} \mapsto \mathcal{T}^\infty(\mathcal{C}, \mathcal{V})\}$ is $\rightarrow_{\mathcal{R}}$ -terminating.*

Proof. We proceed by contradiction.

(\Rightarrow) Assume that T is $\rightsquigarrow_{\mathcal{R}}$ -terminating but $s \rightarrow_{\mathcal{R}} s_1 \rightarrow_{\mathcal{R}} s_2 \rightarrow_{\mathcal{R}} \dots$ is infinite for some $s \in \{t\sigma \mid t \in T, \sigma : \mathcal{V} \mapsto \mathcal{T}^\infty(\mathcal{C}, \mathcal{V})\}$. Let t be a term $t \in T$ such that $s = t\sigma$ with $\sigma : \mathcal{V} \mapsto \mathcal{T}^\infty(\mathcal{C}, \mathcal{V})$. Trivially, since σ is a constructor substitution, it is *normalized* (i.e., $s\sigma$ is in normal form for all $x \in \text{Dom}(\sigma)$). Therefore, the *lifting lemma* of [34] (see the corrected version of [42, Lemma 3.4]) applies. Hence, there exists an infinite narrowing derivation $t \rightsquigarrow_{\sigma_1} t_1 \rightsquigarrow_{\sigma_2} t_2 \rightsquigarrow_{\sigma_3} \dots$ that employs the same rewrite rules at the same positions, with $\sigma_1\sigma_2\sigma_3 \dots \leq \sigma$, which contradicts the initial assumption.

(\Leftarrow) This case proceeds analogously to the proof of Theorem 1. □

The restriction to constructor (though possibly infinite) substitutions in the previous theorem greatly simplifies proving the termination of narrowing computations. The next section will introduce an automated termination analysis for narrowing which is based on this result.

4 Automated Termination Analysis

In this section, we present two approaches to the termination analysis of narrowing that can be fully automated.

In the following, rather than analyzing the termination of narrowing for a set of terms T in a TRS \mathcal{R} , we analyze the termination of the narrowing computations that fulfill a given specification of how every function can be called. This is similar to the notion of *modes* [17] in logic programming which specify, for each predicate, which are the *input* arguments—which are ground every time this predicate is called—and which are the *output* arguments. In our context, the counterpart of mode declarations are given by means of an argument filtering:

Definition 2 (argument filtering). *An argument filtering over a signature \mathcal{F} is a function π such that, for every function $f/n \in \mathcal{F}$, we have $\pi(f) \subseteq \{1, \dots, n\}$. Argument filterings are extended to terms and TRSs in the natural way: $\pi(x) = x$ for all $x \in \mathcal{V}$ and $\pi(f(t_1, \dots, t_n)) = f(\pi(t_{i_1}), \dots, \pi(t_{i_m}))$ if $f \in \mathcal{F}$, $\pi(f) = \{i_1, \dots, i_m\}$, and $i_1 < \dots < i_m$; $\pi(\mathcal{R}) = \{\pi(l) \rightarrow \pi(r) \mid l \rightarrow r \in \mathcal{R}\}$.*

This definition is similar to that in [50] and a slight simplification of the original notion in [6, 38] where $\pi(f)$ may return a single argument position so that $\pi(f(t_1, \dots, t_n)) = \pi(t_i)$ if $\pi(f) = i$. This case is meaningless in our context because we use π to specify the subset of the arguments of a function that are ground in every call.

Example 5. Consider the TRS of Example 1 and the set of terms $T = \{\text{add}(z, x), \text{add}(s(z), x)\}$. Here, we specify the behavior of narrowing for this set of terms by means of the following argument filtering:

$$\pi(z) = \{\} \quad \pi(s) = \{1\} \quad \pi(\text{add}) = \{1\}$$

i.e., we are interested in those narrowing computations in which the first argument of `add` is always a finite ground term.

We are interested in *safe* argument filterings, in the sense that they correctly describe the possible narrowing derivations from a given set of terms. This notion is formalized as follows:

Definition 3 (safe argument filtering). *Let \mathcal{R} be a TRS over a signature \mathcal{F} and let T be a finite set of terms. We say that an argument filtering π is safe for T in \mathcal{R} if $\pi(s) \in \mathcal{T}(\mathcal{F})$ for every $t \in T$ and term s such that $t \rightarrow_{\mathcal{R}}^* s$, and $\pi(\mathcal{R})$ has no extra-variables, i.e., $\text{Var}(r) \subseteq \text{Var}(l)$ for all $l \rightarrow r \in \pi(\mathcal{R})$.³*

The relevance of safe argument filterings is explained by the following result:

Lemma 2. *Let \mathcal{R} be a TRS over a signature \mathcal{F} and let T be a finite set of terms. If the argument filtering π is safe for T in \mathcal{R} , then $\pi(s) \in \mathcal{T}(\mathcal{F})$ for every $t \in T$ and term s such that $t\sigma \rightarrow_{\mathcal{R}}^* s$ with $\sigma : \mathcal{T}^\infty(\mathcal{C}, \mathcal{V})$.*

Proof. By contradiction. Assume that $t\sigma \rightarrow_{\mathcal{R}}^* s$ and $\pi(s) \notin \mathcal{T}(\mathcal{F})$. First, arbitrary values cannot be introduced by instantiation of extra variables since π is safe. Also, since σ is a constructor substitution (and thus normalized), we can apply the lifting lemma of [42, Lemma 3.4] so that the following narrowing derivation exists: $t \rightarrow_{\theta}^* s'$ with $s' \leq s$. Therefore, if $\pi(s) \notin \mathcal{T}(\mathcal{F})$ then $\pi(s') \notin \mathcal{T}(\mathcal{F})$, and we get a contradiction with the initial assumption. \square

Regarding the generation of a safe argument filtering for a given set of terms T in a TRS \mathcal{R} , we consider several possibilities:

- It can be provided by the user as a high-level specification of the termination problem that she wants to analyze.
- One can use a *reachability analysis* like that of [24], where an approximation of the \mathcal{R} -reachable⁴ terms for a given set of *ground* terms is obtained by completion of a tree automaton. In our setting, since the terms in T are not necessarily ground, we should extend the automaton in order to include a *generic* element that represents any (possibly infinite) ground constructor term. The interactive tool Timbuk [26] for reachability analysis may help in this process.
- Other possibilities include the use of a *binding-time analysis* [36], often used in partial evaluation to propagate static (i.e., ground) and dynamic (i.e., possibly nonground) values through a program, as well as the use of standard techniques for abstract interpretation [14] or data-flow analysis [45].

The inference of safe argument filterings is an interesting topic for further work that is outside the scope of this paper.

4.1 A Direct Approach to Termination Analysis

In this section, we present a direct approach for proving the termination of narrowing w.r.t. a given argument filtering by extending the *dependency pair* technique [6]. As we will see, only a slight extension is required—in order to deal with possibly infinite constructor terms—and, indeed, a similar extension has recently been proposed in [50] for the automated termination analysis of logic programs and is already implemented in the AProVE tool [27].

³ This second condition is called the “variable condition” in [50].

⁴ A term s is \mathcal{R} -reachable from t if $t \rightarrow_{\mathcal{R}}^* s$.

The main difference with [50] is that they restrict rewriting so that $s \rightarrow_{p,l \rightarrow r} t$ is only allowed if there exists a *constructor* substitution $\sigma : \mathcal{V} \mapsto \mathcal{T}^\infty(\mathcal{C}, \mathcal{V})$ such that $s|_p = l\sigma$. This is reasonable in their context since they want to simulate the computations of a logic program but it is not acceptable in our case.

The following definitions are slight extensions of those in [6] and [50].

Given a TRS \mathcal{R} over a signature \mathcal{F} , for each $f/n \in \mathcal{F}$, we let f^\sharp/n be a fresh *tuple symbol*. We often write F instead of f^\sharp ; given a term $f(t_1, \dots, t_n)$ with $f \in \mathcal{D}$, we let t^\sharp denote $f^\sharp(t_1, \dots, t_n)$.

Definition 4 (dependency pair). *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:⁵*

$$DP(\mathcal{R}) = \{l^\sharp \rightarrow t^\sharp \mid l \rightarrow r \in \mathcal{R}, t \text{ is a subterm of } r, \text{root}(t) \in \mathcal{D}\}$$

Example 6. Consider the following TRS \mathcal{R} defining the functions `append` and `reverse` over lists built from `nil` (the empty list) and `cons`:

$$\begin{aligned} \text{append}(\text{nil}, y) &\rightarrow y \\ \text{append}(\text{cons}(x, xs), y) &\rightarrow \text{cons}(x, \text{append}(xs, y)) \\ \text{reverse}(\text{nil}) &\rightarrow \text{nil} \\ \text{reverse}(\text{cons}(x, xs)) &\rightarrow \text{append}(\text{reverse}(xs), \text{cons}(x, \text{nil})) \end{aligned}$$

Here, we have the following dependency pairs $DP(\mathcal{R})$:

$$\begin{aligned} \text{APPEND}(\text{cons}(x, xs), y) &\rightarrow \text{APPEND}(xs, y) & (1) \\ \text{REVERSE}(\text{cons}(x, xs)) &\rightarrow \text{REVERSE}(xs) & (2) \\ \text{REVERSE}(\text{cons}(x, xs)) &\rightarrow \text{APPEND}(\text{reverse}(xs), \text{cons}(x, \text{nil})) & (3) \end{aligned}$$

Sequences of function calls are then defined from dependency pairs by using the notion of *chain*.

Definition 5 (chain). *Let \mathcal{R} be a TRS over a signature \mathcal{F} and let π be an argument filtering over \mathcal{F} . A (possibly infinite) sequence of pairs $s_1 \rightarrow t_1, s_2 \rightarrow t_2, \dots$ from $DP(\mathcal{R})$ is a $(DP(\mathcal{R}), \mathcal{R}, \pi)$ -chain iff the following conditions hold:*

- There are substitutions $\sigma_i : \mathcal{V} \mapsto \mathcal{T}^\infty(\mathcal{F}, \mathcal{V})$ such that $t_i\sigma_i \xrightarrow{*}_{\mathcal{R}} s_{i+1}\sigma_{i+1}$.
- $\pi(s_i\sigma_i), \pi(t_i\sigma_i) \in \mathcal{T}(\mathcal{F})$ and for all terms u in the rewrite sequence from $t_i\sigma_i$ to $s_{i+1}\sigma_{i+1}$ we have $\pi(u) \in \mathcal{T}(\mathcal{F})$ as well. So all terms in the sequence have finite ground terms on those positions which are not filtered away by π .

We note that, in contrast to [50], we do not require the matchers σ_i to be constructor substitutions.

Example 7. Consider the TRS \mathcal{R} of Example 6 and its dependency pairs $DP(\mathcal{R})$. Then, for instance, “(1), (1), ...” is an infinite $(DP(\mathcal{R}), \mathcal{R}, \pi)$ -chain for any argument filtering π in which $\pi(\text{APPEND}) \cap \{1\} = \emptyset$. Observe that this would not be possible if variable x in (1) could not be bound to an infinite term.

The following result states the soundness of our approach:

⁵ Note that if \mathcal{R} is a TRS, so is $DP(\mathcal{R})$.

Theorem 3. *Let \mathcal{R} be a TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and let $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a finite set of terms. Let π be a safe argument filtering for T in \mathcal{R} that is extended over tuple symbols so that $\pi(f^\#) = \pi(f)$ for all $f \in \mathcal{D}$. If there is no infinite $(DP(\mathcal{R}), \mathcal{R}, \pi)$ -chain, then $T^* = \{t\sigma \mid t \in T, \sigma : \mathcal{V} \mapsto \mathcal{T}^\infty(\mathcal{C}, \mathcal{V})\}$ is $\rightarrow_{\mathcal{R}}$ -terminating.*

Proof. We proceed by contradiction. We denote by \bar{s} a finite sequence of the form s_1, \dots, s_n . Assume that there is some $t\sigma \in T^*$ such that there exists an infinite derivation $t\sigma \rightarrow_{\mathcal{R}} t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} \dots$. Let $f_1(\bar{s}_1)$ be a subterm of $t\sigma$ that starts an infinite derivation, but none of the terms in \bar{s}_1 starts an infinite derivation (i.e., \bar{s}_1 are $\rightarrow_{\mathcal{R}}$ -terminating).

Let us consider an infinite derivation for $f_1(\bar{s}_1)$. First, the arguments \bar{s}_1 are reduced in a finite number of steps to terms \bar{u}_1 so that a rewrite rule $f_1(\bar{w}_1) \rightarrow r_1$ is applied to $f_1(\bar{u}_1)$, i.e., a (possibly infinite) substitution σ_1 exists such that $f_1(\bar{w}_1)\sigma_1 = f_1(\bar{u}_1) \rightarrow_{\mathcal{R}} r_1\sigma_1$. Now, the infinite reduction continues with $r_1\sigma_1$, i.e., the term $r_1\sigma_1$ starts an infinite derivation too.

By assumption, there exists no infinite reduction beginning with one of the terms $\bar{u}_1 = \bar{w}_1\sigma_1$. Hence, for all $x \in \text{Var}(f_1(\bar{w}_1))$ the term $x\sigma_1$ is $\rightarrow_{\mathcal{R}}$ -terminating. Thus, since $r_1\sigma_1$ starts an infinite derivation, there must be a subterm $f_2(\bar{s}_2)$ in r_1 such that $f_2(\bar{s}_2)\sigma_1$ starts an infinite derivation and $\bar{s}_2\sigma_1$ are $\rightarrow_{\mathcal{R}}$ -terminating.

The first dependency pair of the infinite $(DP(\mathcal{R}), \mathcal{R}, \pi)$ -chain that we construct is $f_1^\#(\bar{w}_1) \rightarrow f_2^\#(\bar{s}_2)$ corresponding to the rewrite rule $f_1^\#(\bar{w}_1) \rightarrow r_1 \in \mathcal{R}$. Since π is a safe argument filtering for T in \mathcal{R} , by Lemma 2, we have that both $\pi(f_1(\bar{u}_1))$ and $\pi(r_1\sigma_1)$ are finite ground terms and, thus, $\pi(f_2(\bar{s}_2)\sigma_1)$ is a finite ground term as well. Also, since $\pi(f) = \pi(f^\#)$ for all $f \in \mathcal{D}$, we have that $\pi(f_1^\#(\bar{u}_1))$ and $\pi(f_2^\#(\bar{s}_2)\sigma_1)$ are finite ground terms.

The infinite sequence continues by rewriting $f_2(\bar{s}_2)\sigma_1$ repeatedly. Since π is safe, the terms remain finite and ground when applying the argument filtering π . Eventually, a rewrite step at the root position is performed again. Repeating this construction infinitely many times results in an infinite $(DP(\mathcal{R}), \mathcal{R}, \pi)$ -chain. \square

In order to show the absence of $(DP(\mathcal{R}), \mathcal{R}, \pi)$ -chains automatically, we follow the *DP framework* [28] as extended in [50] to consider argument filterings and infinitary rewriting.

A *DP problem* is a tuple $(\mathcal{P}, \mathcal{R}, \pi)$ where \mathcal{P} and \mathcal{R} are TRSs and π is an argument filtering. If there is no associated infinite $(\mathcal{P}, \mathcal{R}, \pi)$ -chain, we say that the DP problem is *finite*. Termination methods are then formulated as *DP processors* that take a DP problem and returns a new set of DP problems that should be solved instead. A DP processor *Proc* is sound if, for all DP problems d , we have that d is finite if all DP problems in $\text{Proc}(d)$ are finite. Therefore, a termination proof starts with the initial DP problem $(DP(\mathcal{R}), \mathcal{R}, \pi)$ and applies sound DP processors until an empty set of DP problems is obtained.

In order to reduce the search space, we first introduce the notion of *estimated dependency graph*:⁶

Definition 6 (estimated dependency graph). *Let $(\mathcal{P}, \mathcal{R}, \pi)$ be a DP problem. The nodes of the estimated $(\mathcal{P}, \mathcal{R}, \pi)$ -dependency graph are the pairs of \mathcal{P} and there*

⁶ We observe that a similar notion, called *loop-check*, was introduced in [1, Def. 16] as a method of loop detection for narrowing.

is an arc from $s \rightarrow t$ to $u \rightarrow v$ iff $\text{CAP}(t)$ and a variant u' of u unify with $\delta = \text{mgu}(\text{CAP}(t), u')$ where $\pi(\text{CAP}(t)\delta) = \pi(u'\delta)$ is a finite term. Here, $\text{CAP}(t)$ replaces all subterms of t with $\text{root}(t) \in \mathcal{D}$ by different fresh variables.

Note that argument filterings are used to detect potentially infinite arguments but they are not removed since they can still be useful in the termination proof. If $s \rightarrow t, u \rightarrow v$ is a $(\mathcal{P}, \mathcal{R}, \pi)$ -chain then there is an arc from $s \rightarrow t$ to $u \rightarrow v$ in the estimated dependency graph. Therefore, absence of infinite chains can be proved separately for each maximal strongly connected component (SCC) of the graph.

The following two DP processors can be proved sound as an easy extension of those in [50].

Theorem 4 (dependency graph processor). *Given a DP problem $(\mathcal{P}, \mathcal{R}, \pi)$, let Proc return $\{(\mathcal{P}_1, \mathcal{R}, \pi), \dots, (\mathcal{P}_n, \mathcal{R}, \pi)\}$, where $\mathcal{P}_1, \dots, \mathcal{P}_n$ are the nodes of the SCCs in the estimated dependency graph. Then Proc is sound.*

Theorem 5 (argument filtering processor). *Given a DP problem $(\mathcal{P}, \mathcal{R}, \pi)$, let Proc return $\{(\pi(\mathcal{P}), \pi(\mathcal{R}), id)\}$, where $id(f) = \{1, \dots, n\}$ for all f/n . Then Proc is sound.*

A nice consequence of the last result is that the DP problem $(\pi(\mathcal{P}), \pi(\mathcal{R}), id)$ only involves ordinary rewriting over finite terms and, thus, all existing DP processors [29] can be used to prove the termination of narrowing.

4.2 A Transformational Approach

In this section, we present a more general approach in which the original TRS \mathcal{R} is transformed into a new TRS \mathcal{R}' so that narrowing derivations are finite in \mathcal{R} if *rewriting* derivations are finite in \mathcal{R}' . As a consequence, *every* termination technique for rewrite systems can be applied to prove the termination of narrowing. This allows one to reuse the extensive literature on termination of rewriting as well as the associated termination tools.

Our transformation is based on the *argument filtering transformation* of [38], that we simplify because in our case an argument filtering never returns a single argument position.

Definition 7 (argument filtering transformation). *Let \mathcal{R} be a TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and let π be an argument filtering over \mathcal{F} . The argument filtering transformation AFT_π is defined as follows:*

$$\begin{aligned} \text{AFT}_\pi(\mathcal{R}) = & \pi(\mathcal{R}) \\ & \cup \{ \pi(l) \rightarrow \pi(r') \mid l \rightarrow r \in \mathcal{R}, r' \in \text{dec}_\pi(r), \pi(r') \notin \mathcal{T}(\mathcal{C}, \mathcal{V}) \} \end{aligned}$$

where the auxiliary function dec_π is defined inductively as follows:

$$\begin{aligned} \text{dec}_\pi(x) &= \emptyset \\ \text{dec}_\pi(f(t_1, \dots, t_n)) &= \bigcup_{i \notin \pi(f)} \{t_i\} \cup \bigcup_{i=1}^n \text{dec}_\pi(t_i) \end{aligned}$$

Example 8. Consider the TRS \mathcal{R} of Ex. 6 and the argument filtering given by $\pi(\text{append}) = \{2\}$, $\pi(\text{reverse}) = \{1\}$, $\pi(\text{cons}) = \{2\}$, and $\pi(\text{nil}) = \{\}$. Then, $\text{AFT}_\pi(\mathcal{R})$ returns the following TRS:

$$\begin{aligned} \text{append}(y) &\rightarrow y \\ \text{append}(y) &\rightarrow \text{cons}(\text{append}(y)) \\ \text{reverse}(\text{nil}) &\rightarrow \text{nil} \\ \text{reverse}(\text{cons}(xs)) &\rightarrow \text{append}(\text{cons}(\text{nil})) \\ \text{reverse}(\text{cons}(xs)) &\rightarrow \text{reverse}(xs) \end{aligned}$$

Note that the last rule is introduced because we have

$$\text{dec}_\pi(\text{append}(\text{reverse}(xs), \text{cons}(x, \text{nil}))) = \{\text{reverse}(xs)\}$$

The argument filtering π is safe, e.g., for $T = \{\text{app}(x, \text{nil})\}$. Trivially, $\text{AFT}_\pi(\mathcal{R})$ is not terminating and, thus, T is not $\sim_{\mathcal{R}}$ -terminating as well.

The soundness of the argument filtering transformation is stated as follows:

Theorem 6. *Let \mathcal{R} be a TRS over a signature \mathcal{F} and let T be a set of terms. Let π be a safe argument filtering for T in \mathcal{R} . Let $T_\pi = \{\pi(t) \mid t \in T\}$. If T_π is $\rightarrow_{\text{AFT}_\pi(\mathcal{R})}$ -terminating, then T is $\rightarrow_{\mathcal{R}}$ -terminating.*

Proof. It is a straightforward extension of the proof of Theorem 4.4 in [38] since π is a safe argument filtering and, thus, there are no infinite terms involved in the rewrite derivations in $\text{AFT}_\pi(\mathcal{R})$. \square

The next result is perhaps the main contribution of our work, since it allows one to reduce the termination of narrowing to the termination of rewriting:

Theorem 7. *Let \mathcal{R} be a left-linear constructor TRS and let T be a finite set of terms. Let π be a safe argument filtering for T in \mathcal{R} . Then, T is $\sim_{\mathcal{R}}$ -terminating if $\{\pi(t) \mid t \in T\}$ is $\rightarrow_{\text{AFT}_\pi(\mathcal{R})}$ -terminating.*

Proof. By Theorem 2, we have that T is $\sim_{\mathcal{R}}$ -terminating iff $T^* = \{t\sigma \mid t \in T, \sigma : \mathcal{T}^\infty(\mathcal{C}, \mathcal{V})\}$ is $\rightarrow_{\mathcal{R}}$ -terminating. By Theorem 6, T^* is $\rightarrow_{\mathcal{R}}$ -terminating if $T_\pi = \{\pi(t) \mid t \in T\}$ is $\rightarrow_{\text{AFT}_\pi(\mathcal{R})}$ -terminating, which concludes the proof. \square

The significance of Theorem 7 is that $\text{AFT}_\pi(\mathcal{R})$ can be analyzed using standard techniques and tools for proving the termination of TRSs since no infinite term appears in the derivations of $\pi(T^*)$ in $\text{AFT}_\pi(\mathcal{R})$. Furthermore, one can simply prove the termination of $\text{AFT}_\pi(\mathcal{R})$ (i.e., the fact that $\mathcal{T}(\mathcal{F}, \mathcal{V})$ is $\rightarrow_{\text{AFT}_\pi(\mathcal{R})}$ -terminating), which is trivially sound.

5 Related Work

Despite the relevance of narrowing as a symbolic computation mechanism, we find in the literature only a few works devoted to analyze its termination. For instance, Dershowitz and Sivakumar [20] defined a narrowing procedure that incorporates pruning of some unsatisfiable goals. Similar approaches have been presented by Chabin and Réty [10], where narrowing is directed by a graph of terms, and by Alpuente et al. [1], where the notion of *loop-check* is introduced to detect some unsatisfiable equations. Also, Antoy and Ariola [2] introduced a sort of memoization

technique for functional logic languages so that, in some cases, a finite representation of an infinite narrowing space can be achieved. All these approaches are basically related with pruning the narrowing search space rather than analyzing the termination of narrowing.

On the other hand, Christian [12] introduced a characterization of TRSs for which narrowing terminates. Basically, he requires the left-hand sides to be *flat*, i.e., all arguments are either variables or ground terms. Unfortunately, as we discussed at the beginning of Sect. 3, the termination of narrowing for arbitrary terms is quite a strong property that almost no TRS fulfills.

Recent approaches include [48, 5]. However, both of them consider a form of *quasi-termination* analysis, i.e., they analyze whether only finitely many different function calls are reachable. Moreover, only needed narrowing is considered. Perhaps the closest approaches are [47, 46]. However, in contrast to us, they focus on TRSs with extra-variables and extend the notion of *chain* so that narrowing is used instead of rewriting. Unfortunately, this makes their technique less practical than ours since standard techniques for the termination of rewriting are not directly applicable. Indeed, we think that our approach could also be applied in their context by treating extra variables as (possibly infinite) terms.

Of course, although the goals are different, we share many similarities with [50], as witnessed by the different remarks given throughout the paper.

6 Conclusions

In this paper, we have presented new techniques for proving the termination of narrowing in left-linear constructor systems. Our methods allow one to reduce the termination analysis for narrowing to a termination analysis for rewriting, so that one can reuse existing methods in the extensive literature on termination of rewriting.

Regarding future work, we find it interesting the problem of inferring safe argument filterings and the extension of our developments to deal with TRSs with extra variables. In particular, regarding the first topic, we consider the extension of the approach in [24] as a promising direction for future work.

References

1. M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Narrowing Approximations as an Optimization for Equational Logic Programs. In *Proc. of the 5th Int'l Symp. on Programming Language Implementation and Logic Programming (PLILP'93)*, pages 391–409. Springer LNCS 714, 1993.
2. S. Antoy and Z.M. Ariola. Narrowing the Narrowing Space. In *In Proc. of the 9th Int'l Symposium on Programming Languages: Implementations, Logics, and Programs (PLILP'97)*, pages 1–15. Springer LNCS 1292, 1997.
3. S. Antoy, R. Echahed, and M. Hanus. A Needed Narrowing Strategy. *Journal of the ACM*, 47(4):776–822, 2000.
4. S. Antoy and M. Hanus. Overlapping Rules and Logic Variables in Functional Logic Programs. In *Proc. of the 22nd Int'l Conf. on Logic Programming (ICLP'06)*, pages 87–101. Springer LNCS 4079, 2006.
5. G. Arroyo, J.G. Ramos, J. Silva, and G. Vidal. Improving Offline Narrowing-Driven Partial Evaluation using Size-Change Graphs. In *Proc. of the 16th Int'l Symp. on*

- Logic-based Program Synthesis and Transformation (LOPSTR'06)*, pages 55–61. Università Ca' Foscari di Venezia, 2006. Extended version to appear in Springer LNCS.
6. T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
 7. T. Arts and H. Zantema. Termination of Logic Programs Using Semantic Unification. In *Proc. of the 5th Int'l Workshop on Logic Programming Synthesis and Transformation (LOPSTR'95)*, pages 219–233. Springer LNCS 1048, 1996.
 8. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
 9. D. Bert and R. Echahed. Design and Implementation of a Generic, Logic and Functional Programming Language. In *Proc. of the 1st European Symp. on Programming (ESOP'86)*, pages 119–132. Springer LNCS 213, 1986.
 10. J. Chabin and P. Réty. Narrowing directed by a graph of terms. In *Proc. of the 4th Int'l Conf. on Rewriting Techniques and Applications (RTA'91)*, pages 112–123. Springer LNCS 488, 1991.
 11. P.H. Cheong and L. Fribourg. A survey of the implementations of narrowing. In J. Darlington and R. Dietrich, editors, *Declarative Programming. Workshops in Computing*, pages 177–187. Springer-Verlag and BCS, 1992.
 12. J. Christian. Some termination criteria for narrowing and E-narrowing. In *Proc. of CADE-11*, pages 582–588. Springer LNCS 607, 1992.
 13. H. Comon-Lundh and S. Delaune. The Finite Variant Property: How to Get Rid of Some Algebraic Properties. In *Proc. of the 16th Int'l Conf. on Term Rewriting and Applications (RTA'05)*, pages 294–307. Springer LNCS 3467, 2005.
 14. P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. of 4th ACM Symp. on Principles of Programming Languages (POPL'77)*, pages 238–252, 1977.
 15. J. Darlington and Y. Guo. Narrowing and unification in functional programming: an evaluation mechanism for absolute set abstraction. In *Proc. of the Conf. on Rewrite Techniques and Applications, RTA'89*, pages 92–108. Springer LNCS 355, 1989.
 16. J. de Dios-Castro and F. López-Fraguas. Elimination of Extra-Variables in Functional Logic Programs. In *Proc. of the 6th Spanish Conf. on Programming and Languages (PROLE'06)*. To appear in ENTCS, 2007.
 17. S.K. Debray and D.S. Warren. Automatic Mode Inference for Logic Programs. *Journal of Logic Programming*, 5(3):207–229, 1988.
 18. N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1&2):69–115, 1987.
 19. N. Dershowitz. Goal Solving as Operational Semantics. In *Proc. of ILPS'95*, pages 3–17. The MIT Press, Cambridge, MA, 1995.
 20. N. Dershowitz and G. Sivakumar. Goal-Directed Equation Solving. In *Proc. of 7th National Conf. on Artificial Intelligence*, pages 166–170. Morgan Kaufmann, 1988.
 21. R. Echahed. On completeness of narrowing strategies. In *Proc. of CAAP'88*, pages 89–101. Springer LNCS 299, 1988.
 22. M. Hanus (ed.). Curry: An Integrated Functional Logic Language. Available at: <http://www.informatik.uni-kiel.de/~mh/curry/>.
 23. S. Escobar, C. Meadows, and J. Meseguer. A Rewriting-Based Inference System for the NRL Protocol Analyzer and its Meta-Logical Properties. *Theoretical Computer Science*, 367(1-2):162–202, 2006.
 24. G. Feuillade, T. Genet, and V. Viet Triem Tong. Reachability Analysis over Term Rewriting Systems. *Journal of Automated Reasoning*, 33(3-4):341–383, 2004.
 25. L. Fribourg. SLOG: a Logic Programming Language Interpreter Based on Clausal Superposition and Rewriting. In *Proc. of the Symposium on Logic Programming (SLP'85)*, pages 172–185. IEEE Press, 1985.

26. T. Genet and V. Viet Triem Tong. Reachability Analysis of Term Rewriting Systems with Timbuk. In *Proc. of the 8th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'01)*, pages 695–706. Springer LNCS 2250, 2001.
27. J. Giesl, P. Schneider-Kamp, and R. Thiemann. Automatic Termination Proofs in the Dependency Pair Framework. In *Proc. of the 3rd Int'l Joint Conf. on Automated Reasoning (IJCAR'06)*, pages 281–286. Springer LNCS 4130, 2006.
28. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In *Proc. of the 11th Int'l Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'04)*, pages 301–331. Springer LNCS 3452, 2005.
29. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and Improving Dependency Pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
30. M. Hanus. Compiling Logic Programs with Equality. In *Proc. of the 2nd Int'l Workshop on Programming Language Implementation and Logic Programming (PLILP'90)*, pages 387–401. Springer LNCS 456, 1990.
31. M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 19&20:583–628, 1994.
32. S. Hölldobler. *Foundations of Equational Logic Programming*. Springer LNAI 353, 1989.
33. G. Huet and J.J. Lévy. Computations in orthogonal rewriting systems, Part I + II. In J.L. Lassez and G.D. Plotkin, editors, *Computational Logic – Essays in Honor of Alan Robinson*, pages 395–443, 1992.
34. J.M. Hullot. Canonical Forms and Unification. In *Proc of 5th Int'l Conf. on Automated Deduction*, pages 318–334. Springer LNCS 87, 1980.
35. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proc. of the 7th Int'l Conf. on Logic for Programming and Automated Reasoning (LPAR'00)*, pages 131–160. Springer LNAI 1955, 2000.
36. N.D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Englewood Cliffs, NJ, 1993.
37. R. Kennaway, J.W. Klop, M.R. Sleep, and F.-J. de Vries. Transfinite Reductions in Orthogonal Term Rewriting Systems. *Information and Computation*, 119(1):18–38, 1995.
38. K. Kusakari, M. Nakamura, and Y. Toyama. Argument Filtering Transformation. In *Proc. of the 1st Int'l Conf. on Principles and Practice of Declarative Programming (PPDP'99)*, pages 48–62. Springer LNCS 1702, 1999.
39. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987. Second edition.
40. F. López-Fraguas and J. Sánchez-Hernández. TOY: A Multiparadigm Declarative System. In *Proc. of the 10th Int'l Conf. on Rewriting Techniques and Applications (RTA'99)*, pages 244–247. Springer LNCS 1631, 1999.
41. J. Meseguer and P. Thati. Symbolic Reachability Analysis Using Narrowing and its Application to Verification of Cryptographic Protocols. *Electronic Notes in Theoretical Computer Science*, 117:153–182, 2005.
42. A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.
43. J. J. Moreno-Navarro, H. Kuchen, R. Loogen, and M. Rodríguez-Artalejo. Lazy narrowing in a graph machine. In *Proceedings of the 2nd International Conference on Algebraic and Logic Programming*, volume 463 of LNCS, pages 298–317. Springer-Verlag, 1990.
44. J.J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic Programming with Functions and Predicates: The language Babel. *J. Logic Programming*, 12(3):191–224, 1992.
45. F. Nielson, H.R. Nielson, and C. Hankin. *Principles of Program Analysis*. Springer-Verlag, New York, Inc., 1999.

46. N. Nishida and K. Miura. Dependency Graph Method for Proving Termination of Narrowing. In *Proc. of the 8th Int'l Workshop on Termination (WST'06)*, pages 12–16, 2006.
47. N. Nishida, M. Sakai, and T. Sakabe. Narrowing-Based Simulation of Term Rewriting Systems with Extra Variables. *Electronic Notes in Theoretical Computer Science*, 86(3), 2003.
48. J.G. Ramos, J. Silva, and G. Vidal. Fast Narrowing-Driven Partial Evaluation for Inductively Sequential Systems. In *Proc. of the 10th ACM SIGPLAN International Conference on Functional Programming (ICFP 2005)*, pages 228–239. ACM Press, 2005.
49. U.S. Reddy. Narrowing as the Operational Semantics of Functional Languages. In *Proc. of the Symposium on Logic Programming (SLP'85)*, pages 138–151. IEEE Press, 1985.
50. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated Termination Analysis for Logic Programs by Term Rewriting. In *Proc. of the 16th Int'l Symp. on Logic-based Program Synthesis and Transformation (LOPSTR'06)*. Springer LNCS, 2007. To appear.
51. T. Sheard. Type-level computation using narrowing in ω mega. In *Proc. of the Workshop on Programming Languages meets Program Verification (PLPV'06)*. Electronic Notes in Theoretical Computer Science, 2007. To appear.
52. J.R. Slagle. Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity. *Journal of the ACM*, 21(4):622–642, 1974.
53. J. Steinbach. Simplification Orderings: History of Results. *Fundamenta Informaticae*, 24(1/2):47–87, 1995.