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 logic 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 functional logic languages). Surprisingly, the termination of narrowing has been mostly overlooked. In this paper, we present a new approach for analyzing the termination of narrowing in left-linear constructor systems—a widely accepted class of systems—that allows us to reuse existing methods in the literature on termination of rewriting.

1 Introduction

The narrowing principle [52] generalizes term rewriting by allowing logic variables in terms—as in logic programming [39]—and by replacing pattern matching with unification in order to (non-deterministically) reduce them. Unrestricted narrowing (i.e., not following any particular strategy for selecting reducible expressions) may have a huge—often infinite—search space, mainly because one can freely select any reducible expression *and* applicable rewrite rule at each narrowing step. 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 [31, 49]. Examples of such languages based on narrowing are, e.g., LPG [12], SLOG [25], ALF [30], Babel [45], and the most recent Curry [22] and Toy [40]. Currently, narrowing is regaining popularity in a number of other areas, like protocol verification [15, 23, 35, 42], model checking [24], partial evaluation [1, 48], refining methods for proving the termination of rewriting [8, 9], type checking in the language Ωmega [51], etc.

Termination is a fundamental problem in term rewriting, as witnessed by the extensive literature on the subject (see, e.g., [19, 53] and references therein). Surprisingly, the termination of narrowing has been mostly overlooked so far. To the best of our knowledge, no termination analysis tool has ever been developed for proving the termination of narrowing. Indeed, only a few approaches to this subject can be found in the literature (see a detailed account in Sect. 6).

* This work has been partially supported by the EU (FEDER) and the Spanish MEC under grants TIN2005-09207-C03-02 and *Acción Integrada* HA2006-0008.

In this work, we introduce a new approach to analyze the termination of narrowing by reusing existing results and tools for analyzing the termination of rewriting. The key idea is to consider variables as *data generators* in the context of rewriting. This means that one can analyze the termination of narrowing for the term $\mathsf{add}(x, \mathsf{z})$, where add is a defined function, x is a logic variable, and z is a constructor constant, by analyzing the termination of *rewriting* for the terms $\mathsf{add}(t, \mathsf{z})$, where t stands for an arbitrary—possibly infinite—ground (i.e., without variables) term. Intuitively speaking, we want t to take any possible value that could be computed by narrowing for the logic variable x in any derivation issuing from $\mathsf{add}(x, \mathsf{z})$, even if it goes on infinitely.

This relation between logic variables and (possibly infinite) terms has been pointed out by Dershowitz [20], 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 [6, 17]. The closest approach, though, is the termination analysis for logic programs by Schneider-Kamp *et al* [50], where logic programs are transformed to rewrite systems and logic variables are replaced with infinite terms (cf. Sect. 6).

Since data generators are, by definition, nonterminating, we introduce the use of *argument filterings* in Sect. 4 in order to get rid of data generators in rewrite derivations. Essentially, we consider two alternative approaches:

- The first technique is based on the well-known dependency pair framework [8, 29] for proving the termination of rewriting. We will show that only some slight modifications are required in order to be applicable in our setting.
- The second technique is based on the argument filtering transformation of Kusakari *et al* [38] and, given a TRS $\mathcal{R}$, produces a new rewrite system $\mathcal{R}'$, so that the termination of rewriting in $\mathcal{R}'$ implies the termination of narrowing in $\mathcal{R}$. Therefore, any method or termination tool for rewrite systems can directly be applied to prove the termination of narrowing.

Then, Sect. 5 presents a technique for inferring appropriate argument filterings and reports on a prototype implementation of a termination tool, TNT, that follows the second approach above. Roughly speaking, the user introduces a rewrite system and an *abstract call* indicating the entry function to the program and its *modes*.[1] The tool first computes an argument filtering from the abstract call and, then, transforms the input system using this argument filtering according to the second approach above. The termination of the transformed system is currently checked by using the AProVE tool [27].

The main contributions of this work can be summarized as follows: i) we introduce a sufficient and necessary condition for the termination of narrowing over left-linear constructor systems, a widely accepted class of systems in functional logic programming; ii) we introduce two alternative approaches for analyzing the termination of narrowing w.r.t. a given argument filtering; and iii) we present an automatic tool for proving the termination of narrowing.

Finally, Sect. 6 includes a comparison to related work and Sect. 7 concludes.

[1] We follow the terminology from logic programming, where *modes* are used to specify the degree of instantiation of the arguments of a predicate.

2 Preliminaries

We assume familiarity with basic concepts of term rewriting and narrowing. We refer the reader to, e.g., [10] and [31] for further details.

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

A *substitution* $\sigma : \mathcal{V} \mapsto \mathcal{T}(\mathcal{F}, \mathcal{V})$ is a mapping from variables to terms such that $\mathcal{D}\mathsf{om}(\sigma) = \{x \in \mathcal{V} \mid x \neq \sigma(x)\}$ is its domain. The set of variables introduced by a substitution σ is denoted by $\mathcal{R}\mathsf{an}(\sigma) = \cup_{x \in \mathcal{D}\mathsf{om}(\sigma)} \mathcal{V}\mathsf{ar}(x\sigma)$. Substitutions are extended to morphisms from $\mathcal{T}(\mathcal{F}, \mathcal{V})$ to $\mathcal{T}(\mathcal{F}, \mathcal{V})$ in the natural way. We denote the application of a substitution σ to a term t by $t\sigma$ (rather than $\sigma(t)$). The identity substitution is denoted by *id*. A *variable renaming* is a substitution that is a bijection on $\mathcal{V}$. A substitution σ is *more general* than a substitution θ, denoted by $\sigma \leqslant \theta$, if there is 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* $\theta\restriction_V$ of a substitution θ to a set of variables V is defined as follows: $x\theta\restriction_V = x\theta$ if $x \in V$ and $x\theta\restriction_V = x$ otherwise. We say that $\theta = \sigma\ [V]$ if $\theta\restriction_V = \sigma\restriction_V$.

A term t_2 is an *instance* of a term t_1 (or, equivalently, t_1 is *more general* than t_2), in symbols $t_1 \leqslant t_2$, 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 $\mathsf{mgu}(t_1, t_2)$ if, for every other unifier θ of t_1 and t_2, we have that $\sigma \leqslant \theta$.

TRSs and Rewriting. A set of rewrite rules $l \to r$ 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 $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$.

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

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

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

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

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

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

and $t = (s[r]_p)\sigma$. We often write $s \leadsto_{p,R,\theta} t$ (or simply $s \leadsto_\theta t$) to make explicit the position, rule, and substitution of the narrowing step, where $\theta = \sigma\restriction_{\mathcal{V}\mathsf{ar}(s)}$ (i.e., we label the narrowing step only with the bindings for the narrowed term). A *narrowing derivation* $t_0 \leadsto^*_\sigma t_n$ denotes a sequence of narrowing steps $t_0 \leadsto_{\sigma_1} \ldots \leadsto_{\sigma_n} t_n$ with $\sigma = \sigma_n \circ \cdots \circ \sigma_1$ (if $n = 0$ then $\sigma = id$). Given a narrowing derivation $s \leadsto^*_\sigma t$, we say that σ is a computed *answer* for s.

Example 1. Consider the following TRS $\mathcal{R}$ defining the addition $\mathsf{add}/2$ on natural numbers built from $\mathsf{z}/0$ and $\mathsf{s}/1$:

$$\begin{array}{rcll} \mathsf{add}(\mathsf{z},y) & \to & y & (R_1) \\ \mathsf{add}(\mathsf{s}(x),y) & \to & \mathsf{s}(\mathsf{add}(x,y)) & (R_2) \end{array}$$

Given the term $\mathsf{add}(x,\mathsf{s}(\mathsf{z}))$, we have infinitely many narrowing derivations issuing from $\mathsf{add}(x,\mathsf{s}(\mathsf{z}))$, e.g.:

$$\begin{array}{l} \mathsf{add}(x,\mathsf{s}(\mathsf{z})) \leadsto_{\epsilon,R_1,\{x\mapsto\mathsf{z}\}} \mathsf{s}(\mathsf{z}) \\ \mathsf{add}(x,\mathsf{s}(\mathsf{z})) \leadsto_{\epsilon,R_2,\{x\mapsto\mathsf{s}(y_1)\}} \mathsf{s}(\mathsf{add}(y_1,\mathsf{s}(\mathsf{z}))) \leadsto_{1,R_1,\{y_1\mapsto\mathsf{z}\}} \mathsf{s}(\mathsf{s}(\mathsf{z})) \\ \ldots \end{array}$$

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

3 Termination of Narrowing via Termination of Rewriting

In this section, we present a sufficient and necessary condition 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 set of terms. Given a binary relation $\propto$ on terms, we say that T is $\propto$-terminating iff there is no term $t_1 \in T$ such that there exists an infinite sequence of the form $t_1 \propto t_2 \propto t_3 \propto \ldots$*

We say that a term t is $\propto$-terminating iff the set $\{t\}$ is $\propto$-terminating.

The usual notion of termination can then be formulated as follows: a TRS is *terminating* iff $\mathcal{T}(\mathcal{F})$ is $\to_{\mathcal{R}}$-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 $\leadsto_{\mathcal{R}}$-terminating.

In general, however, only rather trivial TRSs are terminating w.r.t. narrowing. Consider, for instance, the following simple TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{s}(x), y) \to \mathsf{f}(x, y)\}$. Although every term of the form $\mathsf{f}(t_1, t_2)$ has a finite rewrite derivation, we can easily find a term, e.g., $\mathsf{f}(w, \mathsf{z})$, such that an infinite narrowing derivation exists:

$$\mathsf{f}(w,\mathsf{z}) \leadsto_{\{w \mapsto \mathsf{s}(x_1)\}} \mathsf{f}(x_1,\mathsf{z}) \leadsto_{\{x_1 \mapsto \mathsf{s}(x_2)\}} \mathsf{f}(x_2,\mathsf{z}) \leadsto_{\{x_2 \mapsto \mathsf{s}(x_3)\}} \ldots$$

Therefore, in the remainder of the paper, we focus on the termination of narrowing w.r.t. a *given set of terms*, which explains our formulation of termination in Def. 1 above.

It must be clear that, since rewriting is a particular case of narrowing,[2] the termination of narrowing implies the termination of rewriting, i.e., if there is no infinite narrowing derivation issuing from a term t then all rewrite derivations issuing from t are also finite (clearly, the opposite is not true). The following result provides a first—sufficient but not 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 $T^* = \{t\sigma \mid t \in T \text{ and } t \leadsto^*_\sigma s \text{ in } \mathcal{R}\}$. T is $\leadsto_{\mathcal{R}}$-terminating if T^* is finite (modulo variable renaming) and $\to_{\mathcal{R}}$-terminating.*

Proof. We prove the claim by contradiction. Assume that T^* is finite (modulo variable renaming) and $\to_{\mathcal{R}}$-terminating but T is not $\leadsto_{\mathcal{R}}$-terminating. Then, there exists a term $t \in T$ such that there exists an infinite derivation of the form $t \leadsto_{\sigma_1} t_1 \leadsto_{\sigma_2} t_2 \leadsto_{\sigma_3} \ldots$ Then, we have two possibilities. First, if the term $t\sigma_1\sigma_2\ldots$ grows infinitely with the application of every σ_i, then the set T^* must be infinite, which contradicts our initial assumption.

Otherwise, there must be some finite $j > 0$ such that $\sigma_k = id$ for all $k > j$; note that this is possible because every σ_k is restricted to $\mathcal{V}\mathsf{ar}(t_k)$ and because we can always choose a matching substitution that binds the fresh variables of the applied rules to the variables of the reduced term. Then, we can write

[2] Note that, when the considered term is ground, unification reduces to matching and, thus, the definitions of rewriting and narrowing become essentially equivalent.

the infinite narrowing derivation as $t \leadsto_{\sigma_1} t_1 \leadsto_{\sigma_2} \ldots \leadsto_{\sigma_j} t_j \leadsto_{id} t_{j+1} \leadsto_{id} t_{j+2} \leadsto_{id} \ldots$ By the soundness of narrowing (see, e.g., [43, Lemma 3.3]), we have $t\sigma_1 \ldots \sigma_j \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} t_j$. Finally, since $t_k \leadsto_{id} t_{k+1}$ iff $t_k \to_{\mathcal{R}} t_{k+1}$, the previous rewrite derivation goes on infinitely: $t_j \to_{\mathcal{R}} t_{j+1} \to_{\mathcal{R}} t_{j+1} \to_{\mathcal{R}} \ldots$, which contradicts our initial assumption. □

The following example illustrates why the above condition is not necessary:

Example 2. Consider the following TRS: $\mathcal{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{b},\ \mathsf{a} \to \mathsf{a}\}$. Given the set of terms $T = \{\mathsf{f}(x)\}$, we have that T is $\leadsto_{\mathcal{R}}$-terminating since the only narrowing derivation is $\mathsf{f}(x) \leadsto_{\{x \mapsto \mathsf{a}\}} \mathsf{b}$. However, $T^* = \{\mathsf{f}(\mathsf{a})\}$ is finite but not $\to_{\mathcal{R}}$-terminating: $\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}) \to \ldots$

Verifying the finiteness and $\to_{\mathcal{R}}$-termination of T^* is generally, not only undecidable, but also rather difficult to approximate since one should approximate all possible narrowing derivations issuing from the terms in T. Therefore, we now introduce an alternative—easier to check—condition.

For this purpose, a key observation is that variables in narrowing can be seen as *generators* of possibly infinite terms from the point of view of rewriting. The basic idea is then to replace in $T^* = \{t\sigma \mid t \in T \text{ and } t \leadsto^*_\sigma s \text{ in } \mathcal{R}\}$ every answer σ computed by narrowing with any possible substitution mapping variables to possibly infinite terms:

Example 3. Consider again the TRS $\mathcal{R}$ of Ex. 1 and the term $\mathsf{add}(x, \mathsf{z})$. Clearly, $\mathsf{add}(x, \mathsf{z})\sigma$ is $\to_{\mathcal{R}}$-terminating for any substitution σ mapping x to a finite term. However, if σ maps x to an infinite term of the form $\mathsf{s}(\mathsf{s}(\ldots))$, then the derivation for $\mathsf{add}(x, \mathsf{z})\sigma$ is now infinite: $\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)), \mathsf{z}) \to_{\mathcal{R}} \mathsf{s}(\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)), \mathsf{z})) \to_{\mathcal{R}} \mathsf{s}(\mathsf{s}(\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)), \mathsf{z}))) \to_{\mathcal{R}} \ldots$ Indeed, $\mathsf{add}(x, \mathsf{z})$ is not $\leadsto_{\mathcal{R}}$-terminating.

Unfortunately, proving that the set

$$T^* = \{t\sigma \mid t \in T \text{ and } \sigma \text{ maps variables to possibly infinite terms }\}$$

is $\to_{\mathcal{R}}$-terminating is often an unnecessarily strong condition in order to prove that T is $\leadsto_{\mathcal{R}}$-terminating:

Example 4. Consider the following TRS: $\mathcal{R} = \{\mathsf{a} \to \mathsf{a},\ \ \mathsf{f}(x) \to x\}$. While the term $\mathsf{f}(x)$ is clearly $\leadsto_{\mathcal{R}}$-terminating, there exists a substitution $\sigma = \{x \mapsto \mathsf{a}\}$ such that $\mathsf{f}(x)\sigma$ is not $\to_{\mathcal{R}}$-terminating. Here, an infinite computation $\mathsf{f}(\mathsf{a}) \to_{\mathcal{R}} \mathsf{f}(\mathsf{a}) \to_{\mathcal{R}} \ldots$ 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 5. Consider the following TRS: $\mathcal{R} = \{\mathsf{a} \to \mathsf{a},\ \ \mathsf{f}(\mathsf{a}) \to \mathsf{c}(\mathsf{b}, \mathsf{b})\}$. Given the term $t = \mathsf{c}(y, \mathsf{f}(y))$, we have that $t\sigma$ is $\to_{\mathcal{R}}$-terminating if the reduction of the terms introduced by σ (and their descendants) is forbidden. However, t is not $\leadsto_{\mathcal{R}}$-terminating since there exists an infinite narrowing derivation: $\mathsf{c}(y, \mathsf{f}(y)) \leadsto_{\{y \mapsto \mathsf{a}\}} \mathsf{c}(\mathsf{a}, \mathsf{c}(\mathsf{b}, \mathsf{b})) \leadsto_{id} \mathsf{c}(\mathsf{a}, \mathsf{c}(\mathsf{b}, \mathsf{b})) \leadsto_{id} \ldots$

(decomposition)

$$\{\mathsf{g}(a_1,\ldots,a_k) = \mathsf{g}(b_1,\ldots,b_k)\} \cup P; S \Longrightarrow \{a_1 = b_1,\ldots,a_k = b_k\} \cup P; S$$

(term binding)

$$\{t = x\} \cup P; S \Longrightarrow P\{x \mapsto t\}; S\{x \mapsto t\} \cup \{x = t\} \quad \text{if } x \notin \mathcal{V}\mathsf{ar}(t)$$

(rule binding)

$$\{x = s\} \cup P; S \Longrightarrow P\{x \mapsto s\}; S\{x \mapsto s\} \quad \text{if } x \notin \mathcal{V}\mathsf{ar}(s) \text{ and } x \neq s$$

Fig. 1. Simplified unification rules

Clearly, the problem comes from the fact that narrowing does allow the reduction of terms introduced by instantiation.

These problems, though, can be overcome by considering a narrowing strategy over a class of TRSs in which the terms introduced by instantiation cannot be narrowed. Many useful narrowing strategies fulfill this condition, e.g., basic narrowing [34] and innermost basic narrowing [32] over arbitrary TRSs, lazy narrowing [16, 44, 49] and needed narrowing [5] over left-linear constructor TRSs, etc. Actually, any narrowing strategy over left-linear constructor systems fulfills the following well-known property (see, e.g., [45, Theorem 4.1], where it is stated without proof):

Proposition 1. *Let $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ be a signature and $t = \mathsf{f}(t_1,\ldots,t_n)$ be a linear term with $\mathsf{f} \in \mathcal{D}$ and $t_1,\ldots,t_n \in \mathcal{T}(\mathcal{C},\mathcal{V})$. Given a term $s = \mathsf{f}(s_1,\ldots,s_n) \in \mathcal{T}(\mathcal{F},\mathcal{V})$ with $\mathcal{V}\mathsf{ar}(t) \cap \mathcal{V}\mathsf{ar}(s) = \varnothing$, we have that $\mathsf{mgu}(t,s)\restriction_{\mathcal{V}\mathsf{ar}(s)}$ is a constructor substitution.*

Proof. In order to prove this claim, we consider a rule-based unification algorithm [11] (basically, a variant of the Martelli & Montanari unification algorithm [41]). We let $\mathsf{mgu}(\mathsf{f}(t_1,\ldots,t_n),\mathsf{f}(s_1,\ldots,s_n)) = \sigma$ iff

$$\{t_1 = s_1,\ldots,t_n = s_n\}; \{\,\} \Longrightarrow^* \{\,\}; \{x_1 = u_1,\ldots,x_m = u_m\}$$

and $\sigma = \{x_1 \mapsto u_1,\ldots,x_m \mapsto u_m\}$, where the unification relation $\Longrightarrow$ is defined by the rules shown in Fig. 1. We note that, in these rules, we ignore the failure cases and only return the bindings for the variables in $\mathsf{f}(s_1,\ldots,s_n)$ (the *term* bindings) since these are the only relevant bindings for our proof.

Now, we prove a slightly more general claim. For any (possibly incomplete) derivation $\{t_1 = s_1,\ldots,t_n = s_n\}; \{\,\} \Longrightarrow^* P; S$ the following invariants hold:

(I1) for all $x = s' \in S$, we have that s' is a constructor term;
(I2) for all $t' = s', t'' = s'' \in P$ (not necessarily distinct), we have $\mathcal{V}\mathsf{ar}(t') \cap \mathcal{V}\mathsf{ar}(s'') = \varnothing$;
(I3) for all $t' = s' \in P$, we have that t' is a linear constructor term;
(I4) for all $t' = s', t'' = s'' \in P$, if $t' \neq t''$ then $\mathcal{V}\mathsf{ar}(t') \cap \mathcal{V}\mathsf{ar}(t'') = \varnothing$.

Clearly, invariant **I1** implies the desired claim when $P = \{\,\}$. We proceed by induction on the number k of rules applied in the considered derivation.

Base case ($k = 0$). Then, the claim follows trivially since S is empty, $\mathsf{f}(t_1, \ldots, t_n)$ is linear, $t_1, \ldots, t_n$ are constructor terms, and $\mathsf{f}(t_1, \ldots, t_n)$ and $\mathsf{f}(s_1, \ldots, s_n)$ do not share variables.

Inductive case ($k > 0$). Assume a derivation of the form

$$\{t_1 = s_1, \ldots, t_n = s_n\}; \{\,\} \Longrightarrow^{k-1} P'; S' \Longrightarrow P; S$$

By the inductive hypothesis, we have that the above invariants hold in $P'; S'$. Now, we prove that they hold in $P; S$ too. We distinguish the following cases depending on the selected equation of P':

- Let $P' = \{\mathsf{g}(a_1, \ldots, a_j) = \mathsf{g}(b_1, \ldots, b_j)\} \cup P''$, where the equation selected to be reduced in the next step is $\mathsf{g}(a_1, \ldots, a_j) = \mathsf{g}(b_1, \ldots, b_j)$.
  Then, we have $P = \{a_1 = b_1, \ldots, a_j = b_j\} \cup P''$ and $S = S'$, and all invariants hold trivially in $P; S$ since they hold in $P'; S'$.
- Let $P' = \{t = x\} \cup P''$, where the equation selected to be reduced in the next step is $t = x$.
  Then, we have $P = P''\{x \mapsto t\}$ and $S = S'\{x \mapsto t\} \cup \{x = t\}$. Now, we prove that the desired invariants hold in $P; S$:
  - Since **I3** holds in $P'; S'$, we have that t is a constructor term; therefore, since invariant **I1** holds in $P'; S'$, it also holds in $P; S$.
  - Since invariants **I2** and **I4** hold in $P'; S'$, we have that the variables of t occur only once in P'. Therefore, invariants **I2** and **I4** also hold in P.
  - Since invariant **I2** holds in $P'; S'$, we have that variable x does not occur in the left-hand side of any equation of P'. Therefore, invariant **I3** trivially holds in P since it holds in $P'; S'$.
- Let $P' = \{x = s\} \cup P''$, where the equation selected to be reduced in the next step is $x = s$.
  Then, we have $P = P''\{x \mapsto s\}$ and $S = S'\{x \mapsto s\}$. Since invariants **I2** and **I4** hold in $P'; S'$, we have that $P = P''$. Therefore, invariants **I2**, **I3**, and **I4** trivially hold in $P; S$. Finally, since invariant **I3** holds in $P'; S'$, we have that s is a linear constructor term and, thus, invariant **I1** holds in $P; S$. □

Intuitively speaking, this lemma ensures that, given a left-linear constructor system $\mathcal{R}$ and an arbitrary term t, the substitution labeling every narrowing step issuing from t (i.e., the restriction of the computed mgu to the variables of the narrowed term) is a constructor substitution.[3] Thus we restrict ourselves to *left-linear constructor systems* in the remainder of this paper, a large class of TRSs which forms the basis of current functional logic programming languages.

Unfortunately, an important drawback of replacing variables with infinite terms is that existing results relating rewriting and narrowing derivations (e.g.,

[3] Although needed narrowing [5] does not compute mgu's (basically, some bindings are anticipated to ensure that all narrowing steps are *needed*), it computes constructor substitutions (see [3, Lemma 11]) and, thus, our forthcoming results also apply.

the *lifting* lemma) are no longer applicable. Therefore, we follow a different (and simpler) approach: we replace logic variables by *data generators* (as in [6]).

For this purpose, we assume a fixed fresh function symbol "gen" which does not appear in the signature of any TRS. The following definition is a simplified version of the original notion of a *generator* in [6]:

Definition 2 (data generator, gen). *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$. We denote by $\mathcal{R}_{\mathsf{gen}}$ a TRS over $\mathcal{F} \uplus \{\mathsf{gen}\}$ resulting from augmenting $\mathcal{R}$ with the following set of rewrite rules:*

$$\{\mathsf{gen} \to \mathsf{c} \mid \mathsf{c}/0 \in \mathcal{C}\} \cup \{\mathsf{gen} \to \mathsf{c}(\overbrace{\mathsf{gen}, \ldots, \mathsf{gen}}^{n\ times}) \mid \mathsf{c}/n \in \mathcal{C},\ n > 0\}$$

Example 6. For instance, for the TRS $\mathcal{R}$ of Ex. 1 with $\mathcal{C} = \{\mathsf{z}/0, \mathsf{s}/1\}$, we have $\mathcal{R}_{\mathsf{gen}} = \mathcal{R} \cup \{\mathsf{gen} \to \mathsf{z},\ \mathsf{gen} \to \mathsf{s}(\mathsf{gen})\}$.

Trivially, the function gen can be (non-deterministically) reduced to any ground constructor term by using the constructor symbols of $\mathcal{C}$.

Variables are replaced by generators in the obvious way:

Definition 3 (variable elimination, $\widehat{t}$, $\widehat{T}$). *Given a term $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ over a signature $\mathcal{F}$, we let $\widehat{t} = t\sigma$, with $\sigma = \{x \mapsto \mathsf{gen} \mid x \in \mathcal{V}\mathsf{ar}(t)\}$. Analogously, given a set of terms $T \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$, we let $\widehat{T} = \{\widehat{t} \mid t \in T\} \subseteq \mathcal{T}(\mathcal{F} \uplus \{\mathsf{gen}\})$.*

Note that $\widehat{t}$ is ground for any term t since all variables occurring in t are replaced by function gen.

Now, we state the correctness of the variable elimination. Although it is an easy consequence of the results in [6], we provide detailed proofs for completeness. Our first result shows that every narrowing computation can be mimicked by a rewrite derivation if logic variables are replaced with gen in the initial term:

Lemma 1 (completeness). *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and $s \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a term. If $s \leadsto_{p,R,\sigma} t$ in $\mathcal{R}$, then $\widehat{s} \to^* \widehat{s\sigma} \to_{p,R} \widehat{t}$ in $\mathcal{R}_{\mathsf{gen}}$.*

In order to prove this lemma, we first need an auxiliary result:

Lemma 2. *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$, $x \in \mathcal{V}$ be a variable, and $t \in \mathcal{T}(\mathcal{C})$ be a ground constructor term. Then, $\widehat{x} \to^*_{\mathcal{R}_{\mathsf{gen}}} t$.*

Proof. Immediate by definition of function gen. □

Now, we proceed with the proof of Lemma 1:

Proof. By Prop. 1, σ is a constructor substitution. By Lemma 2, since rewriting is closed under context, we have $\widehat{s} \to^* \widehat{s\sigma}$ in $\mathcal{R}_{\mathsf{gen}}$. Since $s \leadsto_{p,R,\sigma} t$, we have that $R = (l \to r) \in \mathcal{R}$ and $\theta = \mathsf{mgu}(l, s|_p)$, with $\sigma = \theta\restriction_{\mathcal{V}\mathsf{ar}(s)}$, $\delta = \theta\restriction_{\mathcal{V}\mathsf{ar}(l)}$, $s|_p\sigma = l\delta$, and $t = s\sigma[r\delta]_p$. Therefore, we have $s\sigma \to_{p,R} s\sigma[r\delta]_p$. Since rewriting is closed under instantiation, we have $\widehat{s\sigma} \to_{p,R} \widehat{s\sigma[r\delta]_p} = \widehat{t}$ and the claim follows. □

Unfortunately, variable elimination is not generally sound because repeated variables must have the same value in a narrowing computation, while different occurrences of gen, though arising from the replacement of the same variable, can be reduced to different terms:

Example 7. Consider again the TRS $\mathcal{R}$ of Ex. 1 and the term $t = \mathsf{add}(x, x)$. Clearly, it can only be narrowed to an even number: z, $\mathsf{s}(\mathsf{s}(\mathsf{z}))$, ... However, $\widehat{t}$ can also be reduced to an odd number, e.g., $\widehat{t} = \mathsf{add}(\mathsf{gen}, \mathsf{gen}) \to \mathsf{add}(\mathsf{z}, \mathsf{gen}) \to \mathsf{gen} \to \mathsf{s}(\mathsf{gen}) \to \mathsf{s}(\mathsf{z})$.

In order to avoid such derivations, we introduce the notion of *admissible* derivation (a straightforward adaptation of the same notion in [6]):

Definition 4 (admissible derivation). *Let $\mathcal{R}$ be a TRS over $\mathcal{F}$ and $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ a term. A derivation for $\widehat{t}$ in $\mathcal{R}_{\mathsf{gen}}$ is called admissible iff all the occurrences of* gen *originating from the replacement of the same variable are reduced to the same term in this derivation.*

A formalism for ensuring that only admissible derivations are possible, based on representing terms by means of pairs term/substitution, can be found in [6].

Now, we can already state the soundness of variable elimination:

Lemma 3 (soundness). *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and $s' \in \mathcal{T}(\mathcal{F} \cup \{\mathsf{gen}\}, \mathcal{V})$ be a term. If $s' \to^* s'' \to_{p,R} t'$ is an admissible derivation in $\mathcal{R}_{\mathsf{gen}}$ and $R \in \mathcal{R}$, then $s \leadsto^*_{\mathcal{R}} t$ with $\widehat{s} = s'$ and $\widehat{t\sigma} = t'$ for some constructor substitution σ.*

In order to prove this lemma, we first need two auxiliary results:

Lemma 4. *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$ be a term. If $\widehat{t} \to^+_{\mathcal{R}_{\mathsf{gen}}} s$ is an admissible derivation where only occurrences of function* gen *are reduced, then there is a constructor substitution σ such that $\widehat{t\sigma} = s$.*

Proof. Since the considered derivation is admissible, w.l.o.g., we consider that all occurrences of gen associated to the same variable are reduced consecutively. We prove the claim by induction on the number k of steps in $\widehat{t} \to^+_{\mathcal{R}_{\mathsf{gen}}} s$.

Base case ($k = 1$). Then, $\widehat{t} \to_{\mathcal{R}_{\mathsf{gen}}} s$ and we have $t|_p = x \in \mathcal{V}$ for some position p, $(\mathsf{gen} \to \mathsf{c}(\mathsf{gen}, \ldots, \mathsf{gen})) \in \mathcal{R}_{\mathsf{gen}}$, and $s = \widehat{t}[\mathsf{c}(\mathsf{gen}, \ldots, \mathsf{gen})]_p$. Since the derivation is admissible, x occurs only once in t. Therefore, the claim follows with $\sigma = \mathsf{c}(y, \ldots, y)$, with y a fresh variable, since $t\sigma = t[\mathsf{c}(y, \ldots, y)]_p$ and, trivially, $\widehat{t\sigma} = s$.

Inductive case ($k > 1$). In this case, we consider a prefix $\widehat{t} \to^+_{\mathcal{R}_{\mathsf{gen}}} t'$ of the derivation $\widehat{t} \to^+_{\mathcal{R}_{\mathsf{gen}}} s$ such that in $\widehat{t} \to^+_{\mathcal{R}_{\mathsf{gen}}} t'$ all occurrences of gen associated to the same variable are reduced one step. Therefore, there exists positions $p_1, \ldots, p_n$, $n > 0$, such that $t|_{p_i} = x \in \mathcal{V}$, $i = 1, \ldots, n$, $(\mathsf{gen} \to \mathsf{c}(\mathsf{gen}, \ldots, \mathsf{gen})) \in \mathcal{R}_{\mathsf{gen}}$, and $t' = \widehat{t}[\mathsf{c}(\mathsf{gen}, \ldots, \mathsf{gen})]_{p_1} \ldots [\mathsf{c}(\mathsf{gen}, \ldots, \mathsf{gen})]_{p_n}$. Similarly to the base case, we consider a constructor substitution of the form $\sigma = \mathsf{c}(y,\ldots,y)$, with y a fresh variable. Hence, $t\sigma = t[\mathsf{c}(y,\ldots,y)]_{p_1}\ldots[\mathsf{c}(y,\ldots,y)]_{p_n}$ and, trivially, $\widehat{t\sigma} = t'$. The claim follows by applying the inductive hypothesis to $t' \to^* s$. □

Lemma 5. *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and $t \in \mathcal{T}(\mathcal{F},\mathcal{V})$ a term. If $\widehat{t} \to_{p,R} s'$ with $R \in \mathcal{R}$, then $t \to_{p,R} s$ such that $\widehat{s} = s'$.*

Proof. Since $\widehat{t} \to_{p,R} s'$, by definition of rewrite step, we have $R = (l \to r) \in \mathcal{R}$, $t|_p = l\sigma$ for some substitution σ, and $s' = \widehat{t}[r\sigma]_p$. The fact that there are no occurrences of gen in R implies that $t|_p$ is also an instance of l. Let σ' be a substitution such that $t|_p = l\sigma'$. Then, $t \to_{p,R} t[r\sigma']_p = s$. Moreover, since σ and σ' only differ in the replacement of some variables by gen, we have that $\widehat{s} = \widehat{t[r\sigma']_p} = \widehat{t}[r\sigma]_p = s'$, and the claim follows. □

Now, we can already proceed with the proof of Lemma 3:

Proof. We prove a slightly more general claim: Let $s' \to^* s'' \to_{p,R} t'$ be an admissible derivation in $\mathcal{R}_{\mathsf{gen}}$ with $R \in \mathcal{R}$ and let $\widehat{s\vartheta} = s'$ for some constructor substitution ϑ. Then, we have $s \leadsto^*_{\mathcal{R}} t$ with $\widehat{t\theta} = t'$ for some constructor substitution θ. Note that the original lemma is an instance of this claim by considering ϑ the empty substitution.

We prove the claim by induction on the number k of steps in $s' \to^* s'' \to_{p,R} t'$ in which the reduced function is not gen.

Base case ($k = 1$). In this case, only occurrences of gen are reduced in $s' \to^* s''$. Therefore, by Lemma 4, there exists a constructor substitution σ such that $\widehat{s\vartheta\sigma} = s''$. Moreover, by Lemma 5, since $s'' \to_{p,R} t'$, there exists a term t such that $s\vartheta\sigma \to_{p,R} t$ in $\mathcal{R}$ with $\widehat{t} = t'$. Since both ϑ and σ are constructor substitutions, $s|_p$ is not a variable and, thus, by definition of narrowing (see, e.g., the completeness result of [43, Lemma 3.4]), we have $s \leadsto_{\mathcal{R}} u$ with $u\theta = t$ for some constructor substitution θ.

Inductive case ($k > 1$). Let $s' \to^* v'$ be a strict prefix of $s' \to^* s''$ such that all steps but the last one are reductions of function gen. By following a similar argument as in the base case, we have that there is a term v such that $s \leadsto_{\mathcal{R}} v$ with $\widehat{s\vartheta\sigma} = s'$ and $\widehat{v\theta} = v'$, where σ and θ are constructor substitutions. By applying the inductive hypothesis to the remaining derivation $v' \to^* s'' \to_{p,R} t'$ with $\widehat{v\theta} = v'$, we have $v \leadsto^*_{\mathcal{R}} u$ with $\widehat{u\theta'} = t'$ for some constructor substitution θ'. Putting all pieces together, we have $s \leadsto^*_{\mathcal{R}} u$ with $\widehat{u\theta'} = t'$. □

Obviously, given a TRS $\mathcal{R}$, no set of terms containing occurrences of gen is generally $\to_{\mathcal{R}_{\mathsf{gen}}}$-terminating because of the nonterminating definition of function gen. Luckily, we are interested in a weaker property: we may allow infinite derivations in $\mathcal{R}_{\mathsf{gen}}$, as long as the number of functions different from gen reduced in these derivations is kept finite. This idea is formalized by using the notion of *relative termination* [37]:

Definition 5 (relative termination). *Let $\mathcal{R}$ and $\mathcal{Q}$ be rewrite systems. Let T be a set of terms. T is relatively $\to_{\mathcal{R}\cup\mathcal{Q}}$-terminating to $\mathcal{R}$ if every infinite derivation $t_0 \to_{\mathcal{R}\cup\mathcal{Q}} t_1 \to_{\mathcal{R}\cup\mathcal{Q}} \ldots$ contains only finitely many $\to_{\mathcal{R}}$-steps.*

The following theorem states one of the main results of this paper:[4]

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 set of terms. Then, T is $\leadsto_\mathcal{R}$-terminating iff $\widehat{T}$ is relatively $\to_{\mathcal{R}_{\mathsf{gen}}}$-terminating to $\mathcal{R}$.*

Proof. ($\Leftarrow$) We prove the claim by contradiction. Assume that $\widehat{T}$ is relatively $\to_{\mathcal{R}_{\mathsf{gen}}}$-terminating to $\mathcal{R}$ but T is not $\leadsto_\mathcal{R}$-terminating. Then, there must be a term $t \in T$ with an associated infinite narrowing derivation $t \leadsto_{p_1,R_1,\sigma_1} t_1 \leadsto_{p_2,R_2,\sigma_2} t_2 \leadsto_{p_3,R_3,\sigma_3} \ldots$ By Lemma 1, there exists an infinite rewrite sequence of the form $\widehat{t} \to^* \widehat{t\sigma_1} \to_{p_1,R_1} \widehat{t_1} \to^* \widehat{t_1\sigma_2} \to_{p_2,R_2} \widehat{t_2} \to^* \widehat{t_2\sigma_2} \to_{p_2,R_2} \ldots$ which is trivially admissible. Therefore, $\widehat{T}$ is not relatively $\to_{\mathcal{R}_{\mathsf{gen}}}$-terminating to $\mathcal{R}$, which contradicts our initial assumption.

($\Rightarrow$) By contradiction. Assume that T is $\leadsto_\mathcal{R}$-terminating but $\widehat{T}$ is not relatively $\to_{\mathcal{R}_{\mathsf{gen}}}$-terminating to $\mathcal{R}$. Therefore, there exists an infinite admissible sequence for some $\widehat{t_1} \in \widehat{T}$. By definition of relative termination, we can denote this infinite sequence as follows: $\widehat{t_1} \to^* t'_1 \to_{p_1,R_1} \widehat{t_2} \to^* t'_2 \to_{p_2,R_2} \widehat{t_3} \to^* \ldots$, where every subsequence $\widehat{t_i} \to^* t'_i$ only reduces occurrences of function gen and $R_i \in \mathcal{R}$, $i > 0$. By Lemma 3, we can construct an associated infinite narrowing derivation of the form $s_1 \leadsto_\mathcal{R} s_2 \leadsto_\mathcal{R} s_3 \leadsto_\mathcal{R} \ldots$ with $s_1 = t_1$ and $s_i\theta_i = t_i$, $i > 1$, which contradicts the initial assumption. □

The above result lays the ground for analyzing the termination of narrowing by reusing existing techniques for proving the termination of rewriting. The next section presents two such approaches.

4 Automating the Termination Analysis

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

4.1 From Abstract Terms to Argument Filterings

In general, we are not interested in providing a set of terms T for proving that T is $\leadsto$-terminating. Rather, it is much more convenient to allow the user to provide a higher-level specification of the function calls in which she is interested in. For this purpose, we introduce the notion of an *abstract term*, which is inspired by the mode declarations of logic programming [18].

Definition 6 (abstract term). *Let $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ be a signature. An abstract term over $\mathcal{F}$ has the form $\mathsf{f}(m_1, \ldots, m_n)$, where $\mathsf{f} \in \mathcal{D}$ and m_i is either g (definitely **g**round) or v (possibly **v**ariable), for all $i = 1, \ldots, n$.*

[4] We note that the use of relative termination does not add an additional complexity since the techniques presented in the next section filter away the occurrences of gen and, thus, ordinary rewrite derivations suffice.

Any abstract term implicitly induces a (possibly infinite) set of terms:

Definition 7 (concretization, γ). *Let $\mathcal{F}$ be a signature and t^α an abstract term over $\mathcal{F}$. The concretization of t^α, in symbols $\gamma(t^\alpha)$, is obtained as follows:*

$$\gamma(\mathsf{f}(m_1,\ldots,m_n)) = \{\mathsf{f}(t_1,\ldots,t_n) \in \mathcal{T}(\mathcal{F},\mathcal{V}) \mid t_i \in \mathcal{T}(\mathcal{C}) \text{ if } m_i = g,\ i = 1,\ldots,n\}$$

Given a set of abstract terms T^α, we let $\gamma(T^\alpha) = \{\gamma(t^\alpha) \mid t^\alpha \in T^\alpha\}$.

Consider the TRS of Ex. 1 and the abstract term $\mathsf{add}(g,v)$. Then, $\gamma(\mathsf{add}(g,v)) = \{\mathsf{add}(\mathsf{z},x), \mathsf{add}(\mathsf{z},\mathsf{z}), \mathsf{add}(\mathsf{s}(\mathsf{z}),x), \mathsf{add}(\mathsf{s}(\mathsf{z}),\mathsf{z}), \mathsf{add}(\mathsf{s}(\mathsf{z}),\mathsf{s}(x)), \mathsf{add}(\mathsf{s}(\mathsf{z}),\mathsf{s}(\mathsf{z})),\ldots\}$.

Thanks to Theorem 2, given a set of abstract terms T^α, we can prove that $\gamma(T^\alpha)$ is $\rightsquigarrow_{\mathcal{R}}$-terminating by proving that $\widehat{\gamma(T^\alpha)}$ is relatively $\rightarrow_{\mathcal{R}_{\mathsf{gen}}}$-terminating to $\mathcal{R}$. This approach, however, presents two drawbacks:

- the set $\gamma(T^\alpha)$ is generally infinite and
- checking relative termination require non-standard techniques and tools.

In order to overcome these drawbacks, we introduce the use of (a simplified version of) argument filterings:

Definition 8 (argument filtering, π). *An argument filtering over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ is a function π such that, for every defined function $\mathsf{f}/n \in \mathcal{D}$, we have $\pi(\mathsf{f}) \subseteq \{1,\ldots,n\}$. Argument filterings are extended to terms as follows:*[5]

- *$\pi(x) = x$ for all $x \in \mathcal{V}$,*
- *$\pi(\mathsf{c}(t_1,\ldots,t_n)) = \mathsf{c}(\pi(t_1),\ldots,\pi(t_n))$ for all $\mathsf{c}/n \in \mathcal{C}$, $n \geq 0$, and*
- *$\pi(\mathsf{f}(t_1,\ldots,t_n)) = \mathsf{f}(\pi(t_{i_1}),\ldots,\pi(t_{i_m}))$ for all $\mathsf{f}/n \in \mathcal{F}$, $n \geq 0$, where $\pi(\mathsf{f}) = \{i_1,\ldots,i_m\}$ and $1 \leq i_1 < \ldots < i_m \leq n$.*

Given a TRS $\mathcal{R}$, we let $\pi(\mathcal{R}) = \{\pi(l) \to \pi_{rhs}(r) \mid l \to r \in \mathcal{R}\}$, where function π_{rhs} is defined as follows:

- *$\pi_{rhs}(x) = \bot$ for all $x \in \mathcal{V}$,*
- *$\pi_{rhs}(\mathsf{c}(t_1,\ldots,t_n)) = \mathsf{c}(\pi_{rhs}(t_1),\ldots,\pi_{rhs}(t_n))$ for all $\mathsf{c}/n \in \mathcal{C}$, $n \geq 0$, and*
- *$\pi_{rhs}(\mathsf{f}(t_1,\ldots,t_n)) = \mathsf{f}(\pi(t_{i_1}),\ldots,\pi(t_{i_m}))$ for all $\mathsf{f}/n \in \mathcal{F}$, $n \geq 0$, where $\pi(\mathsf{f}) = \{i_1,\ldots,i_m\}$ and $1 \leq i_1 < \ldots < i_m \leq n$*

where $\bot$ is a fresh constant constructor not appearing in $\mathcal{C}$.

The original notion of *argument filtering* in [8, 38] may return a single argument position so that $\pi(f(t_1,\ldots,t_n)) = \pi(t_i)$ if $\pi(f) = i$; furthermore, it applies to both constructor and defined function symbols. We consider a simpler definition because our argument filterings will be automatically derived from a set of abstract terms (cf. Sect. 5), where only defined function symbols occur.

On the other hand, our argument filterings replace those variables that are not below a defined function symbol by a fresh constant $\bot$. This is done in

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

order to avoid the introduction of *extra* variables (i.e., variables that appear in the right-hand side of a rule but not in its left-hand side). Consider, e.g., the rule $\mathsf{add}(\mathsf{z}, y) \to y$ and the argument filtering $\pi = \{\mathsf{add} \mapsto \{1\}\}$. Then, $(\pi(\mathsf{add}(\mathsf{z}, y)) \to \pi(y)) = (\mathsf{add}(\mathsf{z}) \to y)$ that contains an extra variable y. Our definition above returns instead $(\pi(\mathsf{add}(\mathsf{z}, y)) \to \pi_{rhs}(y)) = (\mathsf{add}(\mathsf{z}) \to \bot)$. Lemma 6 below states this property.

In the following, though, we are not interested in arbitrary argument filterings but only in what we call *safe* argument filterings.

Definition 9 (safe argument filtering). *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and let T^α be a finite set of abstract terms. We say that an argument filtering π is safe for T^α in $\mathcal{R}$ iff*

- *for all $t^\alpha \in T^\alpha$, if $\pi(t^\alpha) = \mathsf{f}(m_1, \ldots, m_n)$, then $m_i = g$, $i = 1, \ldots, n$;*
- *for all narrowing step $s_1 \leadsto_{\mathcal{R}} s_2$, if $\pi(s_1|_p) \in \mathcal{T}(\mathcal{F})$ for all subterm $s_1|_p$ with $\mathsf{root}(s_1|_p) \in \mathcal{D}$, then $\pi(s_2|_q) \in \mathcal{T}(\mathcal{F})$ for all subterm $s_2|_q$ with $\mathsf{root}(s_2|_q) \in \mathcal{D}$.*

Intuitively speaking, an argument filtering π is safe for a set of abstract terms T^α if π filters away all non-ground arguments of the terms in $\gamma(T^\alpha)$ as well as the non-ground arguments of any function call that can be obtained by narrowing.

Example 8. Consider the TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{s}(x), y) \to \mathsf{f}(y, x)\}$ and the set $T^\alpha = \{\mathsf{f}(g, v)\}$. Given the argument filtering $\pi = \{\mathsf{f} \mapsto \{1\}\}$, although $\pi(\mathsf{f}(g, v)) = \mathsf{f}(g)$ holds (the first condition in Def. 9), this argument filtering is not safe because there exists a term $\mathsf{f}(\mathsf{s}(\mathsf{z}), x) \in \gamma(T^\alpha)$ and a narrowing step $\mathsf{f}(\mathsf{s}(\mathsf{z}), x) \leadsto \mathsf{f}(x, \mathsf{z})$ such that $\pi(\mathsf{f}(\mathsf{s}(\mathsf{z}), x)) = \mathsf{f}(\mathsf{s}(\mathsf{z}))$ is ground but $\pi(\mathsf{f}(x, \mathsf{z})) = \mathsf{f}(x)$ is not.

A useful property is that the filtered form of a TRS does not contain extra variables when the argument filtering is safe.

Lemma 6. *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and let T^α be a finite set of abstract terms. If π is a safe argument filtering for T^α in $\mathcal{R}$ then we have that $\mathcal{V}\mathsf{ar}(\pi_{rhs}(r)) \subseteq \mathcal{V}\mathsf{ar}(\pi(l))$ for all rule $l \to r \in \mathcal{R}$.*

Proof. We prove the claim by contradiction. Assume that π is a safe argument filtering for T^α in $\mathcal{R}$ but there exists a rule $l \to r \in \mathcal{R}$ such that $\mathcal{V}\mathsf{ar}(\pi_{rhs}(r)) \not\subseteq \mathcal{V}\mathsf{ar}(\pi(l))$. Let us consider that $l = \mathsf{f}(l_1, \ldots, l_n)$ for some $\mathsf{f} \in \mathcal{D}$.

Since π is a safe argument filtering, by Def. 9, we have that, for any narrowing step $s_1 \leadsto_{p,l\to r,\sigma} s_2$, if $\pi(s_1|_p) \in \mathcal{T}(\mathcal{F})$ for all subterm $s_1|_p$ with $\mathsf{root}(s_1|_p) \in \mathcal{D}$, then $\pi(s_2|_p) \in \mathcal{T}(\mathcal{F})$ for all subterm $s_2|_p$ with $\mathsf{root}(s_2|_p) \in \mathcal{D}$.

We choose $s_1 = \mathsf{f}(u_1, \ldots, u_n)$ such that $u_i \in \mathcal{T}(\mathcal{F})$ if $i \in \pi(\mathsf{f})$ and $u_i \in \mathcal{V}$ if $i \notin \pi(\mathsf{f})$, and $\mathsf{f}(u_1, \ldots, u_n) \leadsto_{\epsilon,l\to r,\sigma} s_2$, with $s_1\sigma = \mathsf{f}(u_1, \ldots, u_n)\sigma = l\theta$ for some substitution θ, and $s_2 = r\theta$. Now, we consider the following cases:

- If $r \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ is a constructor term, then $\pi_{rhs}(r)$ does not contain variables by definition, and hence we get a contradiction.
- Otherwise, consider a subterm $r|_q = \mathsf{g}(r_1, \ldots, r_m)$ rooted by a defined function symbol $\mathsf{g} \in \mathcal{D}$ such that there exists $x \in \mathcal{V}\mathsf{ar}(\pi(r|_q))$ and $x \notin \mathcal{V}\mathsf{ar}(\pi(l))$. Assume that x appears in argument r_j, $1 \le j \le m$. Now, we have two possibilities:

- If $r_j\theta$ is not ground, then there exists a subterm $r\theta|_q$ of s_2 rooted by a defined function symbol such that $\pi(r\theta|_q)$ is not ground, which contradicts the fact that π is safe.
- Otherwise ($r_j\theta$ is ground), x is bound to a ground term by θ. Assume $x \mapsto t_x \in \theta$ and $t_x \in \mathcal{T}(\mathcal{F})$. Since $\mathcal{R}$ does not contain extra variables, x must occur in one of the arguments of $\mathsf{f}(l_1, \ldots, l_n)$. Assume that $x \in \mathcal{V}\mathsf{ar}(l_k)$ with $k \in \{1, \ldots, n\}$. Since $x \notin \mathcal{V}\mathsf{ar}(\pi(l))$, we have $k \notin \pi(\mathsf{f})$ and, thus, u_k is a variable, which contradicts the fact that t_x is ground. $\square$

The following lemma shows a fundamental property of safe argument filterings:

Lemma 7. *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and let T^α be a finite set of abstract terms. If π is a safe argument filtering for T^α in $\mathcal{R}$ then, for all term $t \in \gamma(T^\alpha)$ and narrowing derivation $t \leadsto^*_{\mathcal{R}} s$, we have $\pi(s|_p) \in \mathcal{T}(\mathcal{F})$ for all subterm $s|_p$ with $\mathsf{root}(s|_p) \in \mathcal{D}$.*

Proof. We prove the claim by induction on the length k of the narrowing derivation $t \leadsto^*_{\mathcal{R}} s$.

Base case ($k = 0$). In this case, we have $t = s$. Assume that $t \in \gamma(t^\alpha)$ for some $t^\alpha \in T^\alpha$. By Def. 9, we have $\pi(t^\alpha) = \mathsf{f}(m_1, \ldots, m_n)$ with $m_1 = \ldots = m_n = g$ for some $\mathsf{f} \in \mathcal{D}$. Hence, by Def. 7, $\pi(t) = \mathsf{f}(t_1, \ldots, t_m)$ with $t_1, \ldots, t_m \in \mathcal{T}(\mathcal{C})$, and the claim follows.

Inductive case ($k > 0$). Consider a narrowing derivation of the form $t \leadsto^*_{\mathcal{R}} u \leadsto_{\mathcal{R}} s$. By the inductive hypothesis, we have that $\pi(u|_p) \in \mathcal{T}(\mathcal{F})$ for all subterm $u|_p$ with $\mathsf{root}(u|_p) \in \mathcal{D}$. Since π is a safe argument filtering, by Def. 9, we have that $\pi(s|_p) \in \mathcal{T}(\mathcal{F})$ for all subterm $s|_p$ with $\mathsf{root}(s|_p) \in \mathcal{D}$, and the claim follows. $\square$

The following lemma relates safe argument filterings and rewriting derivations:

Lemma 8. *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and let T^α be a finite set of abstract terms. If π is a safe argument filtering for T^α in $\mathcal{R}$ then, for all term $t \in \gamma(T^\alpha)$ and admissible derivation $\widehat{t} \to^*_{\mathcal{R}_{\mathsf{gen}}} s' \to_{\mathcal{R}} s$, we have that $\pi(s|_p) \in \mathcal{T}(\mathcal{F})$ (i.e., there is no occurrence of* gen *in $\pi(s|_p)$) for all subterm $s|_p$ with $\mathsf{root}(s|_p) \in \mathcal{D}$.*

Proof. By contradiction. Assume that π is a safe argument filtering for T^α but there exists a term $t \in \gamma(T^\alpha)$ and an admissible derivation $\widehat{t} \to^*_{\mathcal{R}_{\mathsf{gen}}} s' \to_{\mathcal{R}} s$ such that $\pi(s|_p) \notin \mathcal{T}(\mathcal{F})$ for some subterm $s|_p$ with $\mathsf{root}(s|_p) \in \mathcal{D}$. Assume that $\pi(s|_p)|_q = \mathsf{gen}$ for some position q. By Lemma 3, there exists a narrowing derivation $t \leadsto^*_{\mathcal{R}} r$ such that $s = \widehat{r\vartheta}$ with ϑ a constructor substitution. Hence $r|_p$ is rooted by a defined function symbol but $\pi(r|_p)|_q \in \mathcal{V}$. Therefore, $\pi(r|_p) \notin \mathcal{T}(\mathcal{F})$. However, since π is safe, by Lemma 7, we have that $\pi(r|_p) \in \mathcal{T}(\mathcal{F})$, and we get a contradiction. $\square$

In the following, we consider that the input for the termination analysis is a left-linear TRS together with a safe argument filtering for some set of abstract terms. An algorithm for generating safe argument filterings from abstract terms can be found in Sect. 5.

4.2 A Direct Approach to Termination Analysis

In this section, we present a direct approach for proving the termination of narrowing by extending the well-known *dependency pair* technique [8]. Actually, only a slight extension is required to deal with the occurrences of data generators.

The remainder of this section adapts and extends some of the developments in [8]. Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, for each $\mathsf{f}/n \in \mathcal{F}$, we let $\mathsf{f}^\sharp/n$ be a fresh *tuple symbol*; we often write F instead of $\mathsf{f}^\sharp$. Given a term $\mathsf{f}(t_1,\ldots,t_n)$ with $\mathsf{f} \in \mathcal{D}$, we let $t^\sharp$ denote $\mathsf{f}^\sharp(t_1,\ldots,t_n)$.

Definition 10 (dependency pair [8]). *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:*[6]

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

Example 9. Consider the following TRS $\mathcal{R}$ defining the functions append and reverse over lists built from nil (the empty list) and cons:

$$\begin{array}{rcl}
\mathsf{append}(\mathsf{nil}, y) & \to & y \\
\mathsf{append}(\mathsf{cons}(x, xs), y) & \to & \mathsf{cons}(x, \mathsf{append}(xs, y)) \\
\mathsf{reverse}(\mathsf{nil}) & \to & \mathsf{nil} \\
\mathsf{reverse}(\mathsf{cons}(x, xs)) & \to & \mathsf{append}(\mathsf{reverse}(xs), \mathsf{cons}(x, \mathsf{nil}))
\end{array}$$

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

$$\begin{array}{lll}
\mathsf{APPEND}(\mathsf{cons}(x, xs), y) & \to \mathsf{APPEND}(xs, y) & (1) \\
\mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to \mathsf{REVERSE}(xs) & (2) \\
\mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to \mathsf{APPEND}(\mathsf{reverse}(xs), \mathsf{cons}(x, \mathsf{nil})) & (3)
\end{array}$$

In order to prove termination, we should try to prove that there are no infinite chains of dependency pairs. The standard notion of *chain* in [8], however, cannot be used because we are interested in the termination of narrowing (i.e., the relative termination of rewrite sequences in which variables are replaced by gen).

Definition 11 (chain). *Let $\mathcal{R}$ be a TRS over a signature $\mathcal{F}$ and let π be an argument filtering over $\mathcal{F}$ that is extended over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ for all $f \in \mathcal{D}$. A (possibly infinite) sequence of pairs $s_1 \to t_1$, $s_2 \to t_2$, $\ldots$ from $DP(\mathcal{R})$ is a $(DP(\mathcal{R}), \mathcal{R}, \pi)$-chain if the following conditions hold:*[7]

- *there exists a constructor substitution σ such that $\widehat{t_i\sigma} \to^*_{\mathcal{R}_{\mathsf{gen}}} \widehat{s_{i+1}\sigma}$ for every two consecutive pairs in the sequence;*
- *we have $\pi(\widehat{s_i\sigma}), \pi(\widehat{t_i\sigma}) \in \mathcal{T}(\mathcal{F})$ for all $i > 0$ (i.e., π filters away all occurrences of gen).*

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

[7] As in [8], we assume fresh variables in every (occurrence of a) dependency pair and that the domain of substitutions may be infinite.

Example 10. Consider the TRS $\mathcal{R}$ of Example 9 and its dependency pairs $DP(\mathcal{R})$. Here, $\mathcal{R}_{\mathsf{gen}} = \mathcal{R} \cup \{\mathsf{gen} \to \mathsf{nil},\ \mathsf{gen} \to \mathsf{cons}(\mathsf{gen},\mathsf{gen}),\ \mathsf{gen} \to \mathsf{z},\ \mathsf{gen} \mapsto \mathsf{s}(\mathsf{gen})\}$. Then, we have that "$(1),(1),\ldots$" is an infinite $(DP(\mathcal{R}),\mathcal{R},\pi)$-chain for any argument filtering in which $\pi(\mathsf{APPEND}) = \{2\}$ since there exists a substitution $\sigma = \{y \mapsto \mathsf{nil}\}$ such that (we denote the dependency pair (1) by $l_1 \to t_1$)

$$\widehat{t_1\sigma} = \mathsf{APPEND}(\mathsf{gen},\mathsf{nil}) \to_{\mathcal{R}_{\mathsf{gen}}} \mathsf{APPEND}(\mathsf{cons}(\mathsf{gen},\mathsf{gen}),\mathsf{nil}) = \widehat{l_1\sigma}$$

and $\pi(\mathsf{APPEND}(\mathsf{gen},\mathsf{nil})) = \pi(\mathsf{APPEND}(\mathsf{cons}(\mathsf{gen},\mathsf{gen}),\mathsf{nil})) = \mathsf{nil} \in \mathcal{T}(\mathcal{F})$. Note that it would be not a chain in the standard dependency pair framework.

The following result states the soundness of our approach:

Theorem 3. *Let $\mathcal{R}$ be a left-linear constructor TRS over a signature $\mathcal{F} = \mathcal{D} \uplus \mathcal{C}$ and let T^α be a finite set of abstract terms. Let π be a safe argument filtering for T^α in $\mathcal{R}$ that is extended over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ for all $f \in \mathcal{D}$. If there is no infinite $(DP(\mathcal{R}),\mathcal{R},\pi)$-chain, then $\gamma(T^\alpha)$ is $\leadsto_{\mathcal{R}}$-terminating.*

Proof. We proceed by contradiction. Let us assume that there exists no infinite $(DP(\mathcal{R}),\mathcal{R},\pi)$-chain, but $\gamma(T^\alpha)$ is not $\leadsto_{\mathcal{R}}$-terminating. By Theorem 2, we have that $\widehat{\gamma(T^\alpha)}$ is not relatively $\to_{\mathcal{R}_{\mathsf{gen}}}$-terminating to $\mathcal{R}$. Therefore, there exists an infinite admissible rewrite derivation for some ground $\widehat{t}$, with $t \in \gamma(T^\alpha)$, in which an infinite number of rules from $\mathcal{R}$ are used. In the following, we denote by $\overline{s}$ a finite sequence of terms of the form $s_1,\ldots,s_n$.

Let $\mathsf{f}_1(\overline{s_1}) \in \mathcal{T}(\mathcal{F} \cup \{\mathsf{gen}\})$, with $\mathsf{f}_1 \in \mathcal{D}$, be a subterm of $\widehat{t}$ that starts an infinite admissible derivation in which infinitely many rules from $\mathcal{R}$ are used, but none of the terms in $\overline{s_1}$ starts such an infinite derivation.

Let us consider an infinite reduction starting with $\mathsf{f}_1(\overline{s_1})$. First, the arguments $\overline{s_1}$ are reduced in a finite number of steps to some terms $\overline{u_1} \in \mathcal{T}(\mathcal{C} \cup \{\mathsf{gen}\},\mathcal{V})$, $\mathsf{f}_1(\overline{s_1}) \to^*_{\mathcal{R}_{\mathsf{gen}}} \mathsf{f}_1(\overline{u_1})$, so that a rewrite rule $R_1 = (\mathsf{f}_1(\overline{w_1}) \to r_1) \in \mathcal{R}$ is applied: $\mathsf{f}_1(\overline{u_1}) \to_{\epsilon,R_1} r_1\sigma_1$, where $\mathsf{f}_1(\overline{w_1})\sigma_1 = \mathsf{f}_1(\overline{u_1})$. Now, the infinite reduction continues with $r_1\sigma_1$, i.e., the term $r_1\sigma_1$ starts an infinite derivation too. Observe that for each binding $x \mapsto t_x \in \sigma_1$, we have that t_x is either a constructor term or gen by construction.

By assumption, there exists no infinite reduction (using infinitely many rules from $\mathcal{R}$) starting with one of the terms $\overline{u_1} = \overline{w_1\sigma_1}$. Hence, for all $x \in \mathcal{V}\mathsf{ar}(\mathsf{f}_1(\overline{w_1}))$ the term $x\sigma_1$ does not start an infinite derivation in which infinitely many rules from $\mathcal{R}$ are used. Thus, since $r_1\sigma_1$ starts an infinite derivation, there must be a subterm $\mathsf{f}_2(\overline{s_2})$ in r_1 such that $\mathsf{f}_2(\overline{s_2})\sigma_1$ starts an infinite derivation using infinitely many rules from $\mathcal{R}$, but none of the terms in $\overline{s_2\sigma_1}$ starts an infinite derivation in which infinitely many rules from $\mathcal{R}$ are used.

The first dependency pair of the infinite $(DP(\mathcal{R}),\mathcal{R},\pi)$-chain that we construct is $\mathsf{f}_1^\sharp(\overline{w_1}) \to \mathsf{f}_2^\sharp(\overline{s_2})$ corresponding to the rewrite rule $\mathsf{f}_1(\overline{w_1}) \to r_1 \in \mathcal{R}$. Since π is a safe argument filtering for T^α in $\mathcal{R}$, by Def. 9, we have that $\pi(\mathsf{f}_1(\overline{u_1}))$ has no occurrences of gen. Also, by Lemma 8, we have that $\pi(\mathsf{f}_2(\overline{s_2})\sigma_1)$ has no occurrences of gen. Furthermore, by Lemma 6, we have that $\pi(\mathsf{f}_1(\overline{w_1})) \to \pi_{\mathsf{rhs}}(r_1)$

has no extra variables and, thus, $\pi(\mathsf{f}_2(\overline{s_2})\sigma_1)$ is ground. Now, since $\pi(\mathsf{f}) = \pi(\mathsf{f}^\sharp)$ for all $\mathsf{f} \in \mathcal{D}$, we have that $\pi(\mathsf{f}_1^\sharp(\overline{u_1}))$ and $\pi(\mathsf{f}_2^\sharp(\overline{s_2})\sigma_1)$ are ground and have no occurrences of gen.

The infinite sequence continues by rewriting $\mathsf{f}_2(\overline{s_2})\sigma_1$ repeatedly. Since π is safe, all occurrences of gen are filtered away by 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, which contradicts our initial assumption. □

In order to show the absence of $(DP(\mathcal{R}), \mathcal{R}, \pi)$-chains automatically, we can follow the *DP framework* [28]. In this context, 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 return 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 $\mathsf{Proc}(d)$ 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.

We could adapt most of the standard DP processors in order to deal with the use of data generators and argument filterings following similar ideas as those in [50]. For the sake of brevity, we only present one of such DP processors:

Theorem 4 (argument filtering processor). *Given a DP problem $(\mathcal{P}, \mathcal{R}, \pi)$, let* Proc *return $\{(\pi(\mathcal{P}), \pi(\mathcal{R}), id)\}$, where $id(\mathsf{f}) = \{1, \ldots, n\}$ for all defined function symbol f/n occurring in $\pi(\mathcal{R})$. Then* Proc *is sound.*

Proof. We proceed by contradiction. Let us assume that there exists no infinite $(\pi(\mathcal{P}), \pi(\mathcal{R}), id)$-chain but there is an infinite $(\mathcal{P}, \mathcal{R}, \pi)$-chain. Since there is an infinite $(\mathcal{P}, \mathcal{R}, \pi)$-chain, by Def. 11, there exists a constructor substitution σ and an infinite sequence of pairs $s_1 \to t_1$, $s_2 \to t_2$, ... from $\mathcal{P}$ such that $\widehat{t_i\sigma} \to^*_{\mathcal{R}_{\mathsf{gen}}} \widehat{s_{i+1}\sigma}$ for every two consecutive pairs in the sequence, with $\pi(\widehat{s_i\sigma}), \pi(\widehat{t_i\sigma}) \in \mathcal{T}(\mathcal{F})$ for all $i > 0$. Therefore, we have $\pi(\widehat{t_i})\sigma_\pi \to^*_{\mathcal{R}_{\mathsf{gen}}} \pi(\widehat{s_{i+1}})\sigma_\pi$ for all $i > 0$, with $\sigma_\pi = \{x \mapsto \pi(\widehat{x\sigma}) \mid x \in \mathcal{D}\mathsf{om}(\sigma)\}$. Hence, $\pi(s_1) \to \pi(t_1)$, $\pi(s_2) \to \pi(t_2)$, ... is an infinite $(\pi(\mathcal{P}), \pi(\mathcal{R}), id)$-chain, which contradicts our initial assumption. □

The nice property of this DP processor is that, after its application, all existing DP processors for proving the termination of rewriting [29] can be used to prove the termination of narrowing.

Example 11. Consider the TRS of Example 9, the set of abstract terms $T^\alpha = \{\mathsf{append}(g, v)\}$, and the argument filtering $\pi = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$ which is safe for T^α. Given this DP problem, the argument filtering processor returns a new DP problem that consists of the following elements:

– Dependency pairs:

$$\begin{array}{lll} \mathsf{APPEND}(\mathsf{cons}(x, xs)) & \to \mathsf{APPEND}(xs) & (1) \\ \mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to \mathsf{REVERSE}(xs) & (2) \\ \mathsf{REVERSE}(\mathsf{cons}(x, xs)) & \to \mathsf{APPEND}(\mathsf{reverse}(xs)) & (3) \end{array}$$

– Rewrite system:

$$\begin{aligned} \mathsf{append}(\mathsf{nil}) &\to \bot \\ \mathsf{append}(\mathsf{cons}(x, xs)) &\to \mathsf{cons}(x, \mathsf{append}(xs)) \\ \mathsf{reverse}(\mathsf{nil}) &\to \mathsf{nil} \\ \mathsf{reverse}(\mathsf{cons}(x, xs)) &\to \mathsf{append}(\mathsf{reverse}(xs)) \end{aligned}$$

– Argument filtering:

$$id = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$$

The derived DP problem can be proved terminating using standard techniques.

4.3 A Transformational Approach

Now, we present an alternative approach for proving the termination of narrowing. The basic idea is similar to that in the previous section: using an argument filtering to eliminate those subterms that might be bound to a data generator.

Now, however, our aim is to transform the original TRS $\mathcal{R}$ into a new TRS $\mathcal{R}'$ so that narrowing terminates in $\mathcal{R}$ if rewriting terminates 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 easily 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 and, moreover, it is only defined over defined function symbols. Roughly speaking our program transformation generates, for every rule $l \to r$ of the original program,

– a filtered rule $\pi(l) \to \pi_{rhs}(r)$ and
– an additional rule $\pi(l) \to \pi(t)$, where t is a subterm of r that is filtered away in $\pi_{rhs}(r)$ and $\pi(t)$ is not a constructor term.

Definition 12 (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:*

$$\mathsf{AFT}_\pi(\mathcal{R}) = \pi(\mathcal{R}) \cup \{\pi(l) \to \pi(r') \mid l \to r \in \mathcal{R},\ r' \in \mathsf{dec}_\pi(r),\ \pi(r') \notin \mathcal{T}(\mathcal{C}, \mathcal{V})\}$$

where the auxiliary function dec_π is defined inductively as follows:

$$\begin{aligned} \mathsf{dec}_\pi(x) &= \varnothing & (x \in \mathcal{V}) \\ \mathsf{dec}_\pi(c(t_1, \ldots, t_n)) &= \textstyle\bigcup_{i=1}^n \mathsf{dec}_\pi(t_i) & (c \in \mathcal{C}) \\ \mathsf{dec}_\pi(f(t_1, \ldots, t_n)) &= \textstyle\bigcup_{i \notin \pi(f)} \{t_i\} \ \cup\ \bigcup_{i=1}^n \mathsf{dec}_\pi(t_i) & (f \in \mathcal{D}) \end{aligned}$$

Example 12. Consider the TRS $\mathcal{R}$ of Ex. 9. If we consider the argument filtering $\pi_1 = \{\mathsf{append} \mapsto \{1\},\ \mathsf{reverse} \mapsto \{1\}\}$ of Ex. 11, then $\mathsf{AFT}_{\pi_1}(\mathcal{R})$ returns the same filtered rewrite system of Ex. 11.

Consider now the argument filtering $\pi_2 = \{\mathsf{append} \mapsto \{2\},\ \mathsf{reverse} \mapsto \{1\}\}$. Then, $\mathsf{AFT}_{\pi_2}(\mathcal{R})$ returns the following TRS:

$$\begin{aligned}
\mathsf{append}(y) &\to y\\
\mathsf{append}(y) &\to \mathsf{cons}(\bot, \mathsf{append}(y))\\
\mathsf{reverse}(\mathsf{nil}) &\to \mathsf{nil}\\
\mathsf{reverse}(\mathsf{cons}(x, xs)) &\to \mathsf{append}(\mathsf{cons}(x, \mathsf{nil}))\\
\mathsf{reverse}(\mathsf{cons}(x, xs)) &\to \mathsf{reverse}(xs)
\end{aligned}$$

Note that the last rule is introduced because we have

$$\mathsf{dec}_{\pi_2}(\mathsf{append}(\mathsf{reverse}(xs), \mathsf{cons}(x, \mathsf{nil}))) = \{\mathsf{reverse}(xs)\}$$

The next result is the main contribution of this section, since it allows one to reduce the termination of narrowing to the termination of rewriting:

Theorem 5. *Let $\mathcal{R}$ be a left-linear constructor TRS and T^α be a finite set of abstract terms, with $T = \gamma(T^\alpha)$. Let π be a safe argument filtering for T^α in $\mathcal{R}$. If $\mathsf{AFT}_\pi(\mathcal{R})$ is terminating, then T is $\leadsto_{\mathcal{R}}$-terminating.*

Proof. We prove the claim by contradiction. Assume that $\mathsf{AFT}_\pi(\mathcal{R})$ is terminating, but T is not $\leadsto_{\mathcal{R}}$-terminating. By Theorem 3, there exists an infinite $(DP(\mathcal{R}), \mathcal{R}, \pi)$-chain. By Theorem 4, there is an infinite $(\pi(DP(\mathcal{R})), \pi(\mathcal{R}), id)$-chain, where $id(\mathsf{f}) = \{1, \ldots, n\}$ for all defined function symbol f/n occurring in $\pi(\mathcal{R})$. Now, we construct an infinite derivation in $\mathsf{AFT}_\pi(\mathcal{R})$ which will contradict our initial assumption.

Since there exists an infinite $(\pi(DP(\mathcal{R})), \pi(\mathcal{R}), id)$-chain, we have an infinite sequence of the form $\mathsf{f}_1^\sharp(\overline{s_1}) \to \mathsf{f}_2^\sharp(\overline{t_2})$, $\mathsf{f}_2^\sharp(\overline{s_2}) \to \mathsf{f}_3^\sharp(\overline{t_3})$, $\mathsf{f}_3^\sharp(\overline{s_3}) \to \mathsf{f}_4^\sharp(\overline{t_4})$, ... Hence, by Def. 11, there exists a constructor substitution σ such that $\widehat{t_i\sigma} \to^*_{\pi(\mathcal{R})_{\mathsf{gen}}} \widehat{s_{i+1}\sigma}$ for every two consecutive pairs $s_i \to t_i, s_{i+1} \to t_{i+1}$ in the sequence, with $\pi(\widehat{s_i\sigma}), \pi(\widehat{t_i\sigma}) \in \mathcal{T}(\mathcal{F})$ for all $i > 0$ (i.e., π filters away all occurrences of gen). Thus, we have $\mathsf{f}_2^\sharp(\overline{t_2})\sigma \to_{\pi(\mathcal{R})} \mathsf{f}_2^\sharp(\overline{s_2})\sigma$, $\mathsf{f}_3^\sharp(\overline{t_3})\sigma \to_{\pi(\mathcal{R})} \mathsf{f}_3^\sharp(\overline{s_3})\sigma$, ... Furthermore, since $\mathsf{f}_i^\sharp$ are not defined in $\pi(\mathcal{R})$, we have $\mathsf{f}_2(\overline{t_2})\sigma \to_{\pi(\mathcal{R})} \mathsf{f}_2(\overline{s_2})\sigma$, $\mathsf{f}_3(\overline{t_3})\sigma \to_{\pi(\mathcal{R})} \mathsf{f}_3(\overline{s_3})\sigma$, ...

Since $\pi(\mathcal{R}) \subseteq \mathsf{AFT}_\pi(\mathcal{R})$, we only need to prove that, for all $u \to v \in \pi(DP(\mathcal{R}))$, there exists a rule $u \to c[v]_p \in \mathsf{AFC}_\pi(\mathcal{R})$ for some position p. Consider some $\pi(l) \to \pi(r|_p) \in \pi(DP(\mathcal{R}))$, where $l \to r \in \mathcal{R}$ and $\mathsf{root}(r|_p) \in \mathcal{D}$. Now, we distinguish two cases:

- If $\pi(r|_p)$ occurs in the right-hand side of some rule of $\pi(\mathcal{R})$, then the claim follows trivially.
- Otherwise, we have that there exists a subterm $r|_q$ such that $\mathsf{root}(r|_q) \in \mathcal{D}$, $i \notin \pi(\mathsf{root}(r|_q))$, and $r|_p = r|_{q.i.w}$ for some position w (i.e., $r|_p$ is a subterm of the i-th argument of $r|_q$), with $\pi(r|_{q.i.w}) \notin \mathcal{T}(\mathcal{C}, \mathcal{V})$. By definition, we have that $\pi(l) \to \pi(r|_q.i) \in \mathsf{AFT}_\pi(\mathcal{R})$.

Therefore, we have that for all rule $s \to t \in (\pi(DP(\mathcal{R})) \cup \pi(\mathcal{R}))$, there exists a rule $s \to c[t]_p \in \mathsf{AFT}_\pi(\mathcal{R})$ for some position p. Hence, we can construct an infinite derivation in $\mathsf{AFT}_\pi(\mathcal{R})$ as follows:

$$\begin{array}{lll}
\mathsf{f}_1(\overline{s_1})\sigma \to c_1[\mathsf{f}_2(\overline{t_2})]_{p_1}\sigma & & \\
\qquad\qquad\qquad\qquad \downarrow_* & & \\
\qquad\qquad\quad c_1[\mathsf{f}_2(\overline{s_2})]_{p_1}\sigma \to c_1[c_2[\mathsf{f}_3(\overline{t_3})]_{p_2}]_{p_1}\sigma & & \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \downarrow_* & & \\
\qquad\qquad\qquad\qquad\qquad\quad c_1[c_2[\mathsf{f}_3(\overline{s_3})]_{p_2}]_{p_1}\sigma \to \ldots & &
\end{array}$$

which contradicts the termination of $\mathsf{AFT}_\pi(\mathcal{R})$. □

The significance of Theorem 5 is that $\mathsf{AFT}_\pi(\mathcal{R})$ can be analyzed using standard techniques and tools for proving the termination of TRSs since no data generator is involved in the derivations of $\mathsf{AFT}_\pi(\mathcal{R})$.

Let us note that [26] proves that the AFT transformation is subsumed by the DP method regarding simple termination (i.e., termination based on simplification orderings). In our case, however, the approach of this section is not directly subsumed by that of Sect. 4.2 because we consider termination rather than simple termination. Also, the AFT transformation deals with TRSs rather than DP problems, which could be more convenient in some cases. On the other hand, the AFT transformation can be seen as a preprocessing stage so that standard techniques (e.g., the DP method, but not only this method) can be applied to the transformed program, as we will see in the next section.

5 The Termination Tool TNT

In this section, we describe the implementation of a program transformation that follows the approach presented in Sect. 4.3. The tool, called TNT, is publicly available from `http://german.dsic.upv.es/filtering.html`.

The tool is written in Prolog (around 650 lines of code) and includes a parser for TRSs (which accepts the TRS format of the *Termination Problem Data Base*, TPDB, see `http://www.lri.fr/~marche/tpdb/`), a static analysis to infer a safe argument filtering from an abstract term—we consider a single abstract term rather than a set of abstract terms for simplicity—and the AFT_π transformation of Sect. 4.3. The tool is available through a web interface, whose input data are

- a left-linear constructor TRS $\mathcal{R}$ (the user can either write it down or choose it from a selection of TRSs from the TPDB) and
- an initial abstract term t^α that describes a (possibly infinite) set of initial terms $\gamma(t^\alpha)$.

The tool returns a transformed TRS $\mathcal{R}'$ whose termination w.r.t. standard rewriting implies the termination of narrowing for $\gamma(t^\alpha)$ in the original TRS $\mathcal{R}$. The termination of $\mathcal{R}'$ can be analyzed using any tool for proving the termination of rewriting. In particular, the web interface allows the user to check the termination of the transformed TRS using the AProVE tool [27].

Regarding the generation of a safe argument filtering for a given set of abstract terms, we have adapted a simple *binding-time analysis* [36], which is often used in partial evaluation to propagate static (i.e., ground) and dynamic (i.e., possibly nonground) values through a program. Observe that we use g (ground) and v (possibly variable) as binding-times, rather than the more traditional S (static) and D (dynamic). The output of the binding-time analysis is a *division* which includes a mapping $\mathsf{f}/n \mapsto (m_1, \ldots, m_n)$ for every defined function $\mathsf{f}/n \in \mathcal{D}$, where each m_i is a binding-time. A binding-time *environment* is a substitution mapping variables to binding-times. The least upper bound over binding-times is defined as follows:

$$g \sqcup g = g \qquad g \sqcup v = v \qquad v \sqcup g = v \qquad v \sqcup v = v$$

The least upper bound operation can be extended to sequences of binding-times and divisions in the natural way, e.g.,

$$(g, v, g) \sqcup (g, g, v) = (g, v, v)$$
$$\{\mathsf{f} \mapsto (g, v),\ \mathsf{g} \mapsto (g, v)\} \sqcup \{\mathsf{f} \mapsto (g, g),\ \mathsf{g} \mapsto (v, g)\} = \{\mathsf{f} \mapsto (g, v),\ \mathsf{g} \mapsto (v, v)\}$$

Following [36], our binding-time analysis includes two auxiliary functions, B_v and B_e, which are defined in our context as follows:

$$\begin{array}{lll}
B_v[[x]]\ \mathsf{g}/n\ \rho & = (\overbrace{g, \ldots, g}^{n\ times}) & (\text{if } x \in \mathcal{V}) \\
B_v[[\mathsf{c}(t_1, \ldots, t_n)]]\ \mathsf{g}/n\ \rho & = B_v[[t_1]]\ \mathsf{g}/n\ \rho \sqcup \ldots \sqcup B_v[[t_n]]\ \mathsf{g}/n\ \rho & (\text{if } \mathsf{c} \in \mathcal{C}) \\
B_v[[\mathsf{f}(t_1, \ldots, t_n)]]\ \mathsf{g}/n\ \rho & = bt \sqcup (B_e[[t_1]]\ \rho, \ldots, B_e[[t_n]]\ \rho) & (\text{if } \mathsf{f} = \mathsf{g},\ \mathsf{f} \in \mathcal{D}) \\
 & \ bt & (\text{if } \mathsf{f} \neq \mathsf{g},\ \mathsf{f} \in \mathcal{D}) \\
 & \ \text{where } bt = B_v[[t_1]]\ \mathsf{g}/n\ \rho \sqcup \ldots \sqcup B_v[[t_n]]\ \mathsf{g}/n\ \rho &
\end{array}$$

$$\begin{array}{lll}
B_e[[x]]\ \rho & = x\rho & (\text{if } x \in \mathcal{V}) \\
B_e[[\mathsf{h}(t_1, \ldots, t_n)]]\ \rho & = B_e[[t_1]]\ \rho \sqcup \ldots \sqcup B_e[[t_n]]\ \rho & (\text{if } \mathsf{h} \in \mathcal{C} \cup \mathcal{D})
\end{array}$$

Roughly speaking, an expression $(B_v[[t]]\ \mathsf{g}/n\ \rho)$ returns a sequence of n binding-times that denote the (least upper bound of the) binding-times of the arguments of the calls to g/n that occur in t in the context of the binding-time environment ρ. An expression $(B_e[[t]]\ \rho)$ then returns g if t contains no variable which is bound to v in ρ, and v otherwise.

The binding-time analysis is computed as the fixpoint of an iterative process. Assuming that the input abstract term is $\mathsf{f}_1(m_1, \ldots, m_{n_1})$, the initial division is

$$div_0 = \{\mathsf{f}_1 \mapsto (m_1, \ldots, m_{n_1}),\ \mathsf{f}_2 \mapsto (g, \ldots, g),\ \ldots,\ \mathsf{f}_k \mapsto (g, \ldots, g)\}$$

where $\mathsf{f}_1/n_1, \ldots, \mathsf{f}_k/n_k$ are the defined functions of the TRS. Then, given a division $div_i = \{\mathsf{f}_1 \mapsto b_1,\ \ldots,\ \mathsf{f}_k \mapsto b_k\}$, the next division in the sequence is obtained as

$$\begin{array}{rl}
div_{i+1} = \{ & \mathsf{f}_1 \mapsto b_1 \sqcup B_v[[r_1]]\ \mathsf{f}_1/n_1\ e(b_1, l_1) \sqcup \ldots \sqcup B_v[[r_j]]\ \mathsf{f}_1/n_1\ e(b_j, l_j), \\
 & \ldots, \\
 & \mathsf{f}_k \mapsto b_k \sqcup B_v[[r_1]]\ \mathsf{f}_k/n_k\ e(b_1, l_1) \sqcup \ldots \sqcup B_v[[r_j]]\ \mathsf{f}_k/n_k\ e(b_j, l_j)\ \}
\end{array}$$

where $l_1 \to r_1, \ldots, l_j \to r_j$, $j \geq k$, are the rules of $\mathcal{R}$ and the auxiliary function $e(b, l)$ for computing a binding-time environment from a sequence of binding-times and the left-hand side of a rule is defined as follows:

$$
\begin{aligned}
e((m_1, \ldots, m_n), \mathsf{f}(t_1, \ldots, t_n)) = & \{x \mapsto m_1 \mid x \in \mathcal{V}\mathsf{ar}(t_1)\} \\
& \cup \ldots \\
& \cup \{x \mapsto m_n \mid x \in \mathcal{V}\mathsf{ar}(t_n)\}
\end{aligned}
$$

Once we get a fixpoint, i.e., $div_{i+1} = div_i$ for some $i \geq 0$, the corresponding safe argument filtering π is easily obtained by filtering away the positions of nonground arguments. For instance, if the computed division is $div = \{\mathsf{f}_1 \mapsto (m^1_1, \ldots, m^1_{n_1}), \ldots, \mathsf{f}_k \mapsto (m^k_1, \ldots, m^k_{n_k})\}$, we have

$$\pi(div) = \{\mathsf{f}_1 \mapsto \{i \mid m^1_i = g\}, \ldots, \mathsf{f}_k \mapsto \{i \mid m^k_i = g\}\}$$

The fact that $\pi(div)$ is a safe argument filtering is a trivial consequence of the fact that the computed division div is *congruent* [36], i.e., of the fact that an argument is classified as g only when every call to this function in the TRS has a ground term in this argument position.

6 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 [21] defined a narrowing procedure that incorporates pruning of some unsatisfiable goals. Similar approaches have been presented by Chabin and Réty [13], where narrowing is directed by a graph of terms, and by Alpuente *et al* [2], where the notion of *loop-check* is introduced to detect some unsatisfiable equations. Also, Antoy and Ariola [4] 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 [14] 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, 7]. 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.

Nishida and Miura [46] adapted the dependency graph method for proving the termination of narrowing. The presented dependency pair method (an extension of that introduced in [47]) is, in principle, not comparable with ours (Sect. 4.2), since we do not allow extra variables in TRSs and they do not remove some (unnecessary) extra-variables of right-hand sides as we do with π_{rhs}.

The closest approach is that of Schneider-Kamp *et al* [50], who present an automated termination analysis for logic programs. In their approach, logic programs are first translated into TRSs and, then, logic variables are replaced by possibly infinite terms. An extension of the dependency pair framework for dealing with argument filterings is presented, which is similar to our extension in Sect. 4.2. Besides considering a different target (proving termination of SLD resolution vs proving termination of narrowing), there are a number of differences between both approaches. First, [50] considers the replacement of logic variables by infinite terms, while we use data generators (so that we could reuse existing results relating narrowing and standard finitary rewriting). Also, they consider arbitrary argument filterings but require the *variable condition* (i.e., that the filtered TRS contains no extra variables). In our case, argument filterings must be safe which, in principle, do not always imply that the variable condition holds in filtered TRSs. Actually, we allow extra variables above the defined functions of the right-hand sides of the filtered rules (which are then replaced by $\perp$ since they play no role for termination in our context). Furthermore, we introduce a simple binding-time analysis in order to automate the generation of safe argument filterings from higher-level abstract terms. Finally, we also present a transformation approach to proving termination, while [50] only introduces a direct approach based on the dependency pair framework.

7 Conclusions

In this paper, we have presented new techniques for proving the termination of narrowing in left-linear constructor systems. Our approach allows one to analyze the termination of narrowing by analyzing the termination of rewriting, so that one can reuse existing methods and tools in the extensive literature on termination of rewriting.

Regarding future work, we find it interesting to investigate the application of our results in order to improve the precision of narrowing-driven partial evaluation [48]. Also, it would be useful to extend our approach in order to accept source Curry programs rather than TRSs.

Acknowledgements

We thank Naoki Nishida and the anonymous referees for their many suggestions for improving this paper. We also thank the developers of the AProVE tool for allowing us to interface the TNT tool with the web interface of AProVE.

References

1. E. Albert and G. Vidal. The Narrowing-Driven Approach to Functional Logic Program Specialization. *New Generation Computing*, 20(1):3–26, 2002.
2. M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Narrowing Approximations as an Optimization for Equational Logic Programs. In *Proc. of PLILP'93*, pages 391–409. Springer LNCS 714, 1993.

3. S. Antoy. Optimal non-deterministic functional logic computations. In *Proc. of the Int'l Conference on Algebraic and Logic Programming, ALP'97*, pages 16–30. Springer LNCS 1298, 1997.
4. S. Antoy and Z. Ariola. Narrowing the Narrowing Space. In *Proc. of the 9th Int'l Symposium on Programming Languages: Implementations, Logics, and Programs (PLILP'97)*, pages 1–15. Springer LNCS 1292, 1997.
5. S. Antoy, R. Echahed, and M. Hanus. A Needed Narrowing Strategy. *Journal of the ACM*, 47(4):776–822, 2000.
6. S. Antoy and M. Hanus. Overlapping Rules and Logic Variables in Functional Logic Programs. In *Proc. of the 22snd Int'l Conf. on Logic Programming (ICLP'06)*, pages 87–101. Springer LNCS 4079, 2006.
7. G. Arroyo, J.G. Ramos, J. Silva, and G. Vidal. Improving Offline Narrowing-Driven Partial Evaluation using Size-Change Graphs. In *Proc. of LOPSTR'06*, pages 60–76. Springer LNCS 4407, 2007.
8. T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
9. 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.
10. F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.
11. F. Baader and W. Snyder. Unification Theory. In *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.
12. 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.
13. J. Chabin and P. Réty. Narrowing Directed by a Graph of Terms. In *Proc. of RTA'91*, pages 112–123. Springer LNCS 488, 1991.
14. J. Christian. Some Termination Criteria for Narrowing and E-narrowing. In *Proc. of CADE-11*, pages 582–588. Springer LNCS 607, 1992.
15. 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.
16. J. Darlington and Y. Guo. Narrowing and Unification in Functional Programming: an Evaluation Mechanism for Absolute Set Abstraction. In *Proc. of the 3rd Int'l Conf. on Rewriting Techniques and Applications (RTA'89)*, pages 92–108. Springer LNCS 355, 1989.
17. J. de Dios-Castro and F. López-Fraguas. Extra Variables Can Be Eliminated from Functional Logic Programs. In *Proc. of the 6th Spanish Conf. on Programming and Languages (PROLE'06)*, pages 3–19. ENTCS 188, 2007.
18. S. Debray and D.S. Warren. Automatic Mode Inference for Logic Programs. *Journal of Logic Programming*, 5:207–230, 1988.
19. N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1&2):69–115, 1987.
20. N. Dershowitz. Goal Solving as Operational Semantics. In *Proc. of ILPS'95*, pages 3–17. The MIT Press, Cambridge, MA, 1995.
21. N. Dershowitz and G. Sivakumar. Goal-Directed Equation Solving. In *Proc. of 7th National Conf. on Artificial Intelligence*, pages 166–170. Morgan Kaufmann, 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. S. Escobar and J. Meseguer. Symbolic Model Checking of Infinite-State Systems Using Narrowing. In *Proc. of RTA'07*, pages 153–168. Springer LNCS 4533, 2007.
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. J. Giesl and A. Middeldorp. Eliminating Dummy Elimination. In *Proc. of CADE 2000*, pages 309–323. Springer LNCS 1831, 2000.
27. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework. In *Proc. of 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. J.W. Klop. Term Rewriting Systems: A Tutorial. *Bulletin of the European Association for Theoretical Computer Science*, 32:143–183, 1987.
38. K. Kusakari, M. Nakamura, and Y. Toyama. Argument Filtering Transformation. In *Proc. of 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 RTA'99*, pages 244–247. Springer LNCS 1631, 1999.
41. A. Martelli and U. Montanari. An Efficient Unification Algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.
42. 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.
43. A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.

44. J.J. Moreno-Navarro, H. Kuchen, R. Loogen, and M. Rodriguez-Artalejo. Lazy Narrowing in a Graph Machine. In *Proc. of the 2nd Int'l Conf. on Algebraic and Logic Programming (ALP'90)*, pages 298–317. Springer LNCS 463, 1990.
45. 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.
46. N. Nishida and K. Miura. Dependency Graph Method for Proving Termination of Narrowing. In *Proc. of WST'06*, pages 12–16, 2006.
47. N. Nishida, M. Sakai, and T. Sakabe. Narrowing-Based Simulation of Term Rewriting Systems with Extra Variables. *ENTCS*, 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 Int'l Conf. on Functional Programming (ICFP'05)*, 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 LOPSTR'06*, pages 177–193. Springer LNCS 4407, 2007.
51. T. Sheard. Type-Level Computation Using Narrowing in Ωmega. In *Proc. of the Workshop on Programming Languages meets Program Verification (PLPV'06)*, pages 105–128. ENTCS 174, 2007.
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: Histrory of Results. *Fundamenta Informaticae*, 24(1/2):47–87, 1995.