AAECC manuscript No.
(will be inserted by the editor)

Termination of Narrowing via Termination of Rewriting

Naoki Nishida · **Germán Vidal**

the date of receipt and acceptance should be inserted later

Abstract Narrowing extends rewriting with logic capabilities by allowing logic variables in terms and by 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 work, we present a novel 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.

Keywords narrowing, term rewriting, termination

Mathematics Subject Classification (2000) 68N17 · 68Q42

1 Introduction

The narrowing principle [62] generalizes term rewriting by allowing logic variables in terms—as in logic programming [49]—and by replacing pattern matching with unification in order to (non-deterministically) reduce them. 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

A preliminary short version of this paper appeared in the Proceedings of FLOPS 2008 [65].
This work has been partially supported by the Spanish *Ministerio de Ciencia e Innovación* under grant TIN2008-06622-C03-02, by *Generalitat Valenciana* under grant ACOMP/2009/017, and by *UPV* (programs PAID-05-08 and PAID-06-08).

Naoki Nishida
Graduate School of Information Science, Nagoya University, Nagoya, Japan,
E-mail: nishida@is.nagoya-u.ac.jp

Germán Vidal
MiST, DSIC, Universidad Politécnica de Valencia, Camino de Vera, S/N, 46022 Valencia, Spain,
E-mail: gvidal@dsic.upv.es
Tel.: +34-96-3877350, Fax: +34-96-3877359

[37,59]. Examples of such languages based on narrowing are, e.g., LPG [15], SLOG [31], ALF [36], Babel [55], and the most recent Curry [27] and Toy [50]. Currently, narrowing is regaining popularity in a number of other areas, like protocol verification [19,29,43,52], model checking [30], partial evaluation [1,58], refining methods for proving the termination of rewriting [11,12], type checking in the language Ωmega [61], etc.

As witnessed by the extensive literature on the subject, termination is a fundamental problem in both term rewriting (see, e.g., the surveys of Dershowitz [24] and Steinbach [63]) and in logic programming (see, e.g., [18,22,48,64], and references therein). Surprisingly, the termination of narrowing has been mostly overlooked so far. We note that, although functional logic programs combine a functional syntax for programs (namely, term rewrite systems are usually considered) with a logic programming-like evaluation mechanism, no existing technique for proving the termination of either rewrite systems or logic programs is applicable to proving the termination of functional logic programs. Indeed, only a few approaches to this subject can be found in the literature (see a detailed account in Section 7). Furthermore, only very recently some implementations of a termination prover for narrowing have been introduced (the first one being our own tool, TNT, originally introduced in [65]).

Analyzing the termination of narrowing is not only of theoretical interest but has a number of useful application domains. On the one hand, it can be used to check the termination of functional logic programs. We note that, although current functional logic languages like Curry [27] and Toy [50] consider some narrowing *strategy* (i.e., some variant of *lazy* narrowing [20,54,59] like *needed* narrowing [8]), our termination analysis techniques provide a *sufficient* condition for the termination of these programs. Moreover, it would not be difficult to extend our approach following the ideas in [33] in order to get more accurate results. Termination analysis is also essential for ensuring the finiteness of *partial evaluation* [44]. For instance, they can be very useful within the so-called *narrowing-driven* partial evaluation [1], a partial evaluation scheme for both functional and functional logic programs where narrowing is used to perform symbolic computations at partial evaluation time. In general, termination analysis can be useful in almost every transformation technique in order to guarantee the finiteness of the process.

In this work, we are primarily interested in analyzing the termination of narrowing by reusing the large body of techniques 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 any 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 [25], 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 [9,21]. The closest approach, though, is the

termination analysis for logic programs by Schneider-Kamp *et al* [60], where logic programs are transformed to rewrite systems and logic variables are replaced with infinite terms (cf. Section 7).

In contrast to [60], we model data generators by means of rules (as in [9,21]). For instance, a data generator for natural numbers built from z (zero) and s (successor) is defined as follows:

$$\mathsf{gen} \to \mathsf{z}$$
$$\mathsf{gen} \to \mathsf{s}(\mathsf{gen})$$

Note that data generators are, by definition, nonterminating. In particular, one of our main contributions is a result that relates the termination of narrowing for a term t to the *relative termination* [45] of rewriting for a term $\widehat{t}$, where $\widehat{t}$ is obtained from t by replacing its variables with occurrences of the data generator gen. Here, the relative termination of rewriting implies that only finitely many steps with the rules of the original rewrite system are performed (but the occurrences of gen could be reduced infinitely).

However, a main drawback of this approach is that analyzing the relative termination of rewriting is a difficult problem that requires non-standard techniques and tools. In order to overcome this problem, we introduce the use of *argument filterings* to get rid of data generators in rewrite derivations. In this way, we are able to reduce the analysis of relative termination in a rewrite system with data generators to the analysis of termination in this rewrite system without data generators. Essentially, we consider two alternative approaches:

- The first technique is based on the well-known dependency pair framework [11, 35] for proving the termination of rewriting. We will show that only a few 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* [47] and, given a rewrite system $\mathscr{R}$, produces a new rewrite system $\mathscr{R}'$, so that the termination of rewriting in $\mathscr{R}'$ implies the termination of narrowing in $\mathscr{R}$. Therefore, any method or termination tool for rewrite systems can directly be applied to prove the termination of narrowing.

In order to make our approach useful in practice, we also introduce an algorithm for inferring appropriate argument filterings. Finally, we present a number of extensions and refinements to the basic technique and report 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 [32].

The main contributions of this work can be summarized as follows:

1. We introduce, in Section 3, a sufficient and necessary condition for the termination of narrowing over left-linear constructor systems, a widely accepted class of

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

systems in functional and functional logic programming. The condition is formulated in terms of the (relative) termination of rewriting.
2. We then present two alternative approaches for analyzing the termination of narrowing w.r.t. a given argument filtering. The first approach (cf. Section 4.2) is based on adapting the dependency pair method [11] and the second one (cf. Section 4.3) on applying a program transformation similar to that in [47]. Furthermore, both approaches can be easily automated.
3. Section 5 introduces an algorithm for computing appropriate argument filterings so that the previous methods can be applied in practice. Finally, Section 6 presents several applications and refinements, together with a description of an automatic tool (called TNT) for proving the termination of narrowing which follows the transformational approach. To the best of our knowledge, TNT is the first fully automatic tool for proving the termination of narrowing.

2 Preliminaries

In this section, we introduce some basic concepts of term rewriting and narrowing. We refer the reader to, e.g., [13] and [37] for further details.

2.1 Terms and Substitutions

A *signature* $\mathscr{F}$ is a set of function symbols. We often write $\mathsf{f}/n \in \mathscr{F}$ to denote that the arity of function f is n. Given a set of variables $\mathscr{V}$ with $\mathscr{F} \cap \mathscr{V} = \varnothing$, we denote the domain of *terms* by $\mathscr{T}(\mathscr{F},\mathscr{V})$. We assume that $\mathscr{F}$ 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. As it is common practice, a *position* p in a term t is represented by a finite sequence of natural numbers, where ε denotes the root position; we denote by $p.q$ the concatenation of positions p and q. Positions are used to address the nodes of a term viewed as a tree. Positions are partially ordered by the prefix ordering $\leqslant$, i.e., $p \leqslant q$ if there exists an r such that $p.r = q$. We write $p < q$ if $p \leqslant q$ and $p \neq q$. 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. $\mathscr{V}\mathsf{ar}(t)$ denotes the set of variables appearing in t. A term t is *ground* if $\mathscr{V}\mathsf{ar}(t) = \varnothing$. We write $\mathscr{T}(\mathscr{F})$ as a shorthand for the set of ground terms $\mathscr{T}(\mathscr{F},\varnothing)$. We only consider *finite* terms in this paper, i.e., terms with a finite number of symbols (equivalently, trees with finitely many branches).

A *substitution* $\sigma : \mathscr{V} \mapsto \mathscr{T}(\mathscr{F},\mathscr{V})$ is a mapping from variables to terms such that $\mathscr{D}\mathsf{om}(\sigma) = \{x \in \mathscr{V} \mid x \neq \sigma(x)\}$ is its domain. Substitutions are extended to morphisms from $\mathscr{T}(\mathscr{F},\mathscr{V})$ to $\mathscr{T}(\mathscr{F},\mathscr{V})$ 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 $\mathscr{V}$. 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* $\theta|_V$ 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 $\sigma \leqslant \theta$.

2.2 Term Rewriting

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

We use the notation $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ to point out that $\mathscr{D}$ are the defined function symbols and $\mathscr{C}$ are the constructors of a signature $\mathscr{F}$, with $\mathscr{D} \cap \mathscr{C} = \varnothing$. The domains $\mathscr{T}(\mathscr{C},\mathscr{V})$ and $\mathscr{T}(\mathscr{C})$ 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 $x \in \mathscr{D}\mathsf{om}(\sigma)$.

A TRS $\mathscr{R}$ 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., $s_i \in \mathscr{T}(\mathscr{C},\mathscr{V})$, for all $i = 1,\ldots,n$. A term t is *linear* if every variable of $\mathscr{V}$ occurs at most once in t. A TRS $\mathscr{R}$ is *left-linear* (resp. right-linear) if l (resp. r) is linear for every rule $l \to r \in \mathscr{R}$.

For a TRS $\mathscr{R}$, we define the associated rewrite relation $\to_{\mathscr{R}}$ as follows: given terms $s,t \in \mathscr{T}(\mathscr{F},\mathscr{V})$, we have $s \to_{\mathscr{R}} t$ iff there exists a position p in s, a rewrite rule $l \to r \in \mathscr{R}$ 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* w.r.t. a TRS $\mathscr{R}$ if there is no term s with $t \to_{\mathscr{R}} s$. A *derivation* is a (possibly empty) sequence of rewrite steps. Given a binary relation $\to$, we denote by $\to^+$ the transitive closure of $\to$ and by $\to^*$ its reflexive and transitive closure. Thus $t \to^*_{\mathscr{R}} s$ means that t can be reduced to s in zero or more steps; we also use $t \to^n_{\mathscr{R}} s$ to denote that t can be reduced to s in exactly n rewrite steps.

2.3 Narrowing

The *narrowing* principle [62] 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 (analogously to SLD resolution in logic programming). Formally, given a TRS $\mathscr{R}$ and two terms $s,t \in \mathscr{T}(\mathscr{F},\mathscr{V})$, we have that $s \leadsto_{\mathscr{R}} t$ is a *narrowing step* iff there exist

– a non-variable position p of s,
– a variant $R = (l \to r)$ of a rule in $\mathscr{R}$ such that $\mathscr{V}\mathsf{ar}(l) \cap \mathscr{V}\mathsf{ar}(s) = \varnothing$,
– 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 $s \leadsto_\theta t$ in $\mathscr{R}$) to make explicit the position, rule, and substitution of the narrowing step, where $\theta = \sigma|_{\mathscr{V}\mathsf{ar}(s)}$ (i.e., we label the narrowing step only with the bindings for the variables in 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 $\sigma = id$). Due to the presence of free variables, a term may be reduced to different values after instantiating these variables to different terms. Given a narrowing derivation $s \leadsto^*_\sigma t$, we say that σ is a computed *answer* for s.

Example 1 Consider the following TRS $\mathscr{R}$ 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.:

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

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 introduce a precise characterization of the termination of narrowing in terms of the termination of rewriting.

3.1 A Simple Characterization of Narrowing Termination

Let us first 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 an infinite sequence of the form $t_1 \propto t_2 \propto t_3 \propto \ldots$ exists.

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 $\mathscr{T}(\mathscr{F})$ is $\to_{\mathscr{R}}$-terminating. As for narrowing, we say that a TRS $\mathscr{R}$ is terminating *w.r.t. narrowing* iff $\mathscr{T}(\mathscr{F}, \mathscr{V})$ is $\leadsto_{\mathscr{R}}$-terminating.

In general, however, only rather trivial TRSs are terminating w.r.t. narrowing.[2] Consider, for instance, the TRS of Example 1 defining the addition on natural numbers: $\mathscr{R} = \{\mathsf{add}(\mathsf{z}, y) \to y,\ \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y))\}$. Although every ground term of the form $\mathsf{add}(t_1, t_2)$ has a finite rewrite sequence, we can easily find a term, e.g., $\mathsf{add}(x, \mathsf{z})$, such that an infinite narrowing derivation exists:

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

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 Definition 1 above.

It must be clear that, since rewriting is a particular case of narrowing,[3] 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 $\mathscr{R}$ be a TRS and T be a finite set of terms. Let $T^* = \{t\sigma \mid t \in T \text{ and } t \leadsto_\sigma^* s \text{ in } \mathscr{R}\}$. T is $\leadsto_{\mathscr{R}}$-terminating if T^* is finite (modulo variable renaming) and $\to_{\mathscr{R}}$-terminating.*

Proof We prove the claim by contradiction. Assume that T^* is finite (modulo variable renaming) and $\to_{\mathscr{R}}$-terminating but T is not $\leadsto_{\mathscr{R}}$-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 $\mathscr{V}\mathsf{ar}(t_k)$ 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 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., [53, Lemma 3.3]), we have $t\sigma_1\ldots\sigma_j \to_{\mathscr{R}} \ldots \to_{\mathscr{R}} t_j$. Finally, since $t_k \leadsto_{id} t_{k+1}$ iff $t_k \to_{\mathscr{R}} t_{k+1}$, the previous rewrite derivation goes on infinitely: $t_j \to_{\mathscr{R}} t_{j+1} \to_{\mathscr{R}} t_{j+1} \to_{\mathscr{R}} \ldots$, which contradicts our initial assumption. □

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

Example 2 Consider the following TRS: $\mathscr{R} = \{\mathsf{f}(\mathsf{a}) \to \mathsf{b},\ \mathsf{a} \to \mathsf{a}\}$. Given the set of terms $T = \{\mathsf{f}(x)\}$, we have that T is $\leadsto_{\mathscr{R}}$-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 $\to_{\mathscr{R}}$-terminating: $\mathsf{f}(\mathsf{a}) \to \mathsf{f}(\mathsf{a}) \to \ldots$

[2] Actually, we are not aware of any termination analysis for logic programs—where logic variables are also allowed, as in narrowing—that does not consider some restriction on the possible calls.

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

Verifying the finiteness and $\to_{\mathscr{R}}$-termination of T^* is generally, not only undecidable, but also rather difficult to approximate since one should consider 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 $T^* = \{t\sigma \mid t \in T \text{ and } t \leadsto^*_\sigma s \text{ in } \mathscr{R}\}$ every answer σ computed by narrowing with any possible substitution mapping variables to possibly infinite terms:

Example 3 Consider again the TRS $\mathscr{R}$ of Example 1 and the term $\mathsf{add}(x,\mathsf{z})$. Clearly, $\mathsf{add}(x,\mathsf{z})\sigma$ is $\to_{\mathscr{R}}$-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:

$$\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)),\mathsf{z}) \to_{\mathscr{R}} \mathsf{s}(\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)),\mathsf{z})) \to_{\mathscr{R}} \mathsf{s}(\mathsf{s}(\mathsf{add}(\mathsf{s}(\mathsf{s}(\ldots)),\mathsf{z}))) \to_{\mathscr{R}} \ldots$$

Indeed, $\mathsf{add}(x,\mathsf{z})$ is not $\leadsto_{\mathscr{R}}$-terminating.

Unfortunately, proving that the set

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

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

Example 4 Consider the following TRS: $\mathscr{R} = \{\mathsf{a} \to \mathsf{a},\ \ \mathsf{f}(x) \to x\}$. While the term $\mathsf{f}(x)$ is clearly $\leadsto_{\mathscr{R}}$-terminating, there exists a substitution $\sigma = \{x \mapsto \mathsf{a}\}$ such that $\mathsf{f}(x)\sigma$ is not $\to_{\mathscr{R}}$-terminating. Here, an infinite computation $\mathsf{f}(\mathsf{a}) \to_{\mathscr{R}} \mathsf{f}(\mathsf{a}) \to_{\mathscr{R}} \ldots$ 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* [40]). However, such a restriction on rewriting derivations would make the previous condition unsound:

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

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 [41] and innermost basic narrowing [39] over arbitrary TRSs, lazy narrowing [20, 54, 59] and needed narrowing [8] over left-linear constructor TRSs, etc. Actually, any narrowing strategy over left-linear constructor systems fulfills the following well-known property (see, e.g., [55, 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 $\mathsf{mgu}(t, s)|_{\mathcal{V}\mathsf{ar}(s)}$ is a constructor substitution.*

The proof can be found in the appendix.

Intuitively speaking, this proposition 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.[4] Thus we restrict ourselves to *left-linear constructor systems* in the remainder of this paper, a large class of TRSs which forms the (first-order) basis of many functional and functional logic programming languages.

With a similar aim, [56, 57] introduced the *TRAT property* w.r.t. narrowing. Basically, a TRS has the TRAT property w.r.t. narrowing when, for every infinite narrowing derivation where all strict subterms are terminating w.r.t. narrowing, at least one reduction at the topmost position is performed. Since this property is undecidable, several sufficient conditions are given: i) every constructor TRS has the TRAT property, and ii) every right-linear TRS has the TRAT property when the initial term is linear. The interest in TRSs with the TRAT property comes from the fact that, in these TRSs, no *infinite* rewrite sequence can be introduced by instantiation. Therefore, their condition is slightly weaker than requiring left-linear constructor TRSs.

3.2 Using Data Generators to Characterize Narrowing Termination

As mentioned before, we can relate the termination of narrowing and the termination of rewriting by replacing variables with possibly infinite terms. Unfortunately, an important drawback of this approach is that existing results relating rewriting and narrowing derivations (e.g., the *lifting* lemma) are no longer applicable to rewrite derivations with infinite terms. Therefore, we follow a different (and simpler) approach: we replace logic variables by *data generators* (as in [9]).

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 [9]:

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

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

where constants $\mathsf{c}()$ are simply denoted by c.

We also denote by $\mathcal{R}_{\mathsf{gen}}$ a TRS over $\mathcal{F} \uplus \{\mathsf{gen}\}$ resulting from augmenting $\mathcal{R}$ with $GEN(\mathcal{R})$, in symbols $\mathcal{R}_{\mathsf{gen}} = \mathcal{R} \cup GEN(\mathcal{R})$.

[4] Although needed narrowing [8] does not compute mgu's (basically, some bindings are anticipated to ensure that all narrowing steps are *needed*), it computes constructor substitutions (see [6, Lemma 11]) and, thus, our forthcoming results also apply.

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

Trivially, the function gen can be (non-deterministically) reduced to any ground constructor term by using the constructor symbols of $\mathscr{C}$. We note that, in contrast to [9], we do not consider a different function gen for every type. This decision does not affect the correctness of the approach but only introduces some useless derivations. Nevertheless, these generators are only used as a theoretical device but the actual techniques for proving the termination of narrowing (see Section 4) do not use them and, thus, there is no loss of efficiency involved in their use.

Variables are replaced by generators in the obvious way:

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

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

Now, we state the correctness of the variable elimination. Although it is an easy consequence of the results in [9], we provide detailed proofs for completeness. Basically, the main differences come from the fact that we consider that each narrowing step computes a most general unifier between the selected subterm and the left-hand side of a rule, while [9] considers that narrowing can compute arbitrary unifiers (i.e., according to [9], $s \leadsto_{p,R,\sigma} t$ is a narrowing step if $s\sigma \to_{p,R} t$).

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

Proof By Proposition 1, σ is a constructor substitution. By the definition of function gen, we have that $\widehat{x} \to^*_{GEN(\mathscr{R})} c$ holds for every variable $x \in \mathscr{V}$ and ground constructor term $c \in \mathscr{T}(\mathscr{C})$. Now, since rewriting is closed under contexts, we have $\widehat{s} \to^*_{GEN(\mathscr{R})} \widehat{s\sigma}$. Also, since $s \leadsto_{p,R,\sigma} t$, we have that $R = (l \to r) \in \mathscr{R}$ and $\theta = \mathsf{mgu}(l, s|_p)$, with $\sigma = \theta\restriction_{\mathscr{V}\mathsf{ar}(s)}$, $\delta = \theta\restriction_{\mathscr{V}\mathsf{ar}(l)}$, $s|_p\sigma = l\delta$, 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 also 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[5] 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:

[5] Here, by *sound* we mean that every rewrite derivation issuing from the term $\widehat{t}$ can be mimicked with a narrowing derivation issuing from t.

Example 7 Consider again the TRS $\mathscr{R}$ of Example 1 and the term $t = \mathsf{add}(x,x)$. Clearly, it can only be narrowed to an even number: $\mathsf{z},\ \mathsf{s}(\mathsf{s}(\mathsf{z})),\ \ldots$ 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 use the notion of *admissible* derivation:

Definition 4 (admissible derivation [9]) Let $\mathscr{R}$ be a TRS over $\mathscr{F}$ and $t \in \mathscr{T}(\mathscr{F},\mathscr{V})$ a term. A derivation for $\widehat{t}$ in $\mathscr{R}_{\mathsf{gen}}$ 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 [9].

When the considered derivation is admissible, w.l.o.g., we consider that all occurrences of gen associated to the same variable are reduced to the same term consecutively. As a trivial consequence we have that, if $s \to^* t \to_{p,R} u \to^* v$ is an admissible derivation in $\mathscr{R}_{\mathsf{gen}}$, with $R \in \mathscr{R}$, then $s \to^* u$ and $u \to^* v$ are also admissible derivations in $\mathscr{R}_{\mathsf{gen}}$.

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

Lemma 2 (soundness) *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F}$ and let $s' \in \mathscr{T}(\mathscr{F} \uplus \{\mathsf{gen}\},\mathscr{V})$ be a term. If $s' \to^*_{\mathscr{R}_{\mathsf{gen}}} s'' \to_{p,R} t'$ is an admissible derivation and $R \in \mathscr{R}$, then $s \rightsquigarrow^+_{\mathscr{R}} t$ 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 (their proofs can be found in the appendix):

Lemma 3 *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F}$ and let $t \in \mathscr{T}(\mathscr{F},\mathscr{V})$ be a term. If $\widehat{t} \to^+_{GEN(\mathscr{R})} s$ is an admissible derivation, then there is a constructor substitution σ such that $\widehat{t\sigma} = s$.*

Lemma 4 *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and let $t \in \mathscr{T}(\mathscr{F},\mathscr{V})$ be a term. If $\widehat{t} \to_{p,R} s'$ with $R \in \mathscr{R}$, then $t \to_{p,R} s$ such that $\widehat{s} = s'$.*

Now, we can already proceed with the proof of Lemma 2:

Proof We prove a slightly more general claim: Let $s' \to^*_{\mathscr{R}_{\mathsf{gen}}} s'' \to_{p,R} t'$ be an admissible derivation with $R \in \mathscr{R}$ and let $\widehat{s\vartheta} = s'$ for some constructor substitution ϑ. Then, we have $s \rightsquigarrow^+_{\mathscr{R}} t$ 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 $s' \to^*_{\mathscr{R}_{\mathsf{gen}}} s'' \to_{p,R} t'$ in which the reduced function is not gen.

Base case ($k = 1$). In this case, only occurrences of gen are reduced in $s' \to^*_{\mathscr{R}_{\mathsf{gen}}} s''$. Therefore, by Lemma 3, there exists a constructor substitution σ such that $\widehat{s\vartheta\sigma} = s''$. Moreover, by Lemma 4, since $s'' \to_{p,R} t'$, there exists a term t such that $s\vartheta\sigma \to_{p,R}$

t in $\mathscr{R}$ 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 [53, Lemma 3.4]), we have $s \leadsto_{\mathscr{R}} u$ with $u\theta = t$ for some constructor substitution θ.

Inductive case ($k > 1$). Let $s' \to^+_{\mathscr{R}_{\mathsf{gen}}} v'$ be a strict prefix of $s' \to^*_{\mathscr{R}_{\mathsf{gen}}} s''$ such that all steps but the last one are reductions of function gen. Note that, since $s' \to^*_{\mathscr{R}_{\mathsf{gen}}} s'' \to_{p,R} t'$ is admissible, so are $s' \to^+_{\mathscr{R}_{\mathsf{gen}}} v'$ and $v' \to^*_{\mathscr{R}_{\mathsf{gen}}} s''$. By following a similar argument as in the base case, we have that there is a term v such that $s \leadsto_{\mathscr{R}} v$ 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 $v' \to^*_{\mathscr{R}_{\mathsf{gen}}} s'' \to_{p,R} t'$ with $\widehat{v\theta} = v'$, we have $v \leadsto^+_{\mathscr{R}} u$ with $\widehat{u\theta'} = t'$ for some constructor substitution θ'. Putting all pieces together, we have $s \leadsto^+_{\mathscr{R}} u$ with $\widehat{u\theta'} = t'$. □

Obviously, given a TRS $\mathscr{R}$, no set of terms containing occurrences of gen is generally $\to_{\mathscr{R}_{\mathsf{gen}}}$-terminating because of the nonterminating definition of function gen. Luckily, we are interested in a weaker property: we may allow infinite derivations in $\mathscr{R}_{\mathsf{gen}}$, 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* [45]:

Definition 5 (relative termination) Given two relations $\to_{\mathscr{R}}$ and $\to_{\mathscr{P}}$ we define the compound relation $\to_{\mathscr{R}}/\to_{\mathscr{P}}$ as $\to^*_{\mathscr{P}} \cdot \to_{\mathscr{R}} \cdot \to^*_{\mathscr{P}}$.

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

The following theorem states one of the main results of this paper:[6]

Theorem 2 *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and let $T \subseteq \mathscr{T}(\mathscr{F}, \mathscr{V})$ be a set of terms. Then, T is $\leadsto_{\mathscr{R}}$-terminating iff $\widehat{T}$ is relatively $\to_{\mathscr{R}}$-terminating w.r.t. $GEN(\mathscr{R})$ when only admissible derivations are considered.*

Proof ($\Leftarrow$) We prove the claim by contradiction. Assume that $\widehat{T}$ is relatively $\to_{\mathscr{R}}$-terminating w.r.t. $GEN(\mathscr{R})$ but T is not $\leadsto_{\mathscr{R}}$-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 admissible. Therefore, $\widehat{T}$ is not relatively $\to_{\mathscr{R}}$-terminating w.r.t. $GEN(\mathscr{R})$, which contradicts our initial assumption.

($\Rightarrow$) By contradiction. Assume that T is $\leadsto_{\mathscr{R}}$-terminating but $\widehat{T}$ is not relatively $\to_{\mathscr{R}}$-terminating w.r.t. $GEN(\mathscr{R})$, so that we can construct an infinite admissible $\to_{\mathscr{R}_{\mathsf{gen}}}$-derivation with infinite $\to_{\mathscr{R}}$ steps for some $\widehat{t_1} \in \widehat{T}$. By definition of relative termination, we can denote this infinite sequence as follows: $\widehat{t_1} \to^*_{GEN(\mathscr{R})} t'_1 \to_{p_1,R_1} \widehat{t_2} \to^*_{GEN(\mathscr{R})} t'_2 \to_{p_2,R_2} \widehat{t_3} \to^*_{GEN(\mathscr{R})} \ldots$ with $R_i \in \mathscr{R}$, $i > 0$. By Lemma 2, we can construct an associated infinite narrowing derivation of the form $s_1 \leadsto^+_{\mathscr{R}} s_2 \leadsto^+_{\mathscr{R}} s_3 \leadsto^+_{\mathscr{R}} \ldots$ with $s_1 = t_1$ and $s_i\theta_i = t_i$, $i > 1$, which contradicts the initial assumption. □

[6] We note that the use of relative termination does not add an additional complexity since the techniques presented in the next section will filter away the occurrences of gen.

The following corollary introduces a (simpler) sufficient condition for the termination of narrowing by removing the restriction to admissible rewrite derivations.

Corollary 1 *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and let $T \subseteq \mathscr{T}(\mathscr{F}, \mathscr{V})$ be a set of terms. Then, T is $\leadsto_{\mathscr{R}}$-terminating if $\widehat{T}$ is relatively $\to_{\mathscr{R}}$-terminating w.r.t. GEN$(\mathscr{R})$.*

The above results lay 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 first introduce the use of abstract terms to specify a termination problem, together with the corresponding argument filterings in order to allow the automation of the analysis. Then, Section 4.2 introduces a direct approach to proving the termination of narrowing by adapting the well-known dependency pair method [11]. Finally, Section 4.3 presents a transformational approach based on the so-called argument filtering transformation of [47].

4.1 From Abstract Terms to Argument Filterings

In general, we do not want the user to specify a termination problem by providing a TRS and a set of terms T, since this would be very difficult when T is a large set (or even an infinite set). Rather, it is much more convenient to allow the user to provide a higher-level specification of the function calls 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 [23].

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

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

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

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

Observe that, in the definition above, we consider that g only approximates ground *constructor* terms. In principle, we could relax this condition so that arbitrary ground terms would be allowed as far as these terms have only finite rewrite sequences. Since this is not decidable, we prefer to keep the restriction to constructor terms.

Furthermore, we observe that requiring abstract terms to have only one (outermost) defined function is not a real restriction. In particular, given an arbitrary term

t, one could add the rule $\mathsf{foo}(x_1,\ldots,x_n) \to t$ to the TRS, where foo is a fresh function symbol and $x_1,\ldots,x_n$ are the variables of t. Then, abstract terms of the form $\mathsf{foo}(m_1,\ldots,m_n)$ would actually represent the corresponding concretizations of t.

Consider the TRS of Example 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 Corollary 1, given an abstract term t^α, we can prove that $\gamma(t^\alpha)$ is $\rightsquigarrow_{\mathscr{R}}$-terminating by proving that $\widehat{\gamma(t^\alpha)}$ is relatively $\to_{\mathscr{R}}$-terminating to $GEN(\mathscr{R})$. The main drawback of this approach, however, is that proving relative termination requires non-standard techniques and tools.[7]

In order to overcome this problem, similarly to [60], we introduce the use of *argument filterings* [11,47]. Intuitively speaking, we use argument filterings to get rid of non-ground arguments in a narrowing derivation or, equivalently, occurrences of gen in the corresponding rewrite derivation, so that techniques for standard termination can then be used.

First, we need some preparatory definitions. Let $l \to r$ be a rule. We denote its *extra variables* [8] by $EV(l \to r) = \mathcal{V}\mathsf{ar}(r) \setminus \mathcal{V}\mathsf{ar}(l)$. Also, we denote by $[l \to r]_\perp$ the rule that results from $l \to r$ by instantiating its extra variables with a fresh constant function $\perp$, i.e., $[l \to r]_\perp = l \to r\sigma$, where $\sigma = \{x \mapsto \perp \mid x \in EV(l \to r)\}$.

Definition 8 (argument filtering, π) An argument filtering over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ is a function π such that, for every function or constructor symbol $\mathsf{f}/n \in \mathscr{F}$, we have $\pi(\mathsf{f}) \subseteq \{1,\ldots,n\}$. Argument filterings are extended to terms as follows:[9]

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

Given a TRS $\mathscr{R}$, argument filterings are extended to TRSs as follows:

$$\pi(\mathscr{R}) = \{\ [\pi(l) \to \pi(r)]_\perp \mid l \to r \in \mathscr{R}\ \}$$

We note that the original concept of *argument filtering* in [11,47] may also return a single argument position so that $\pi(f(t_1,\ldots,t_n)) = \pi(t_i)$ if $\pi(f) = i$ (which are usually called *collapsing* argument filterings). We use argument filterings to specify the

[7] We note that there already exist several tools for proving relative termination of TRSs (e.g., Jambox [28] and TPA [46]). Unfortunately, they are not useful in our context since they do not allow us to specify a set of initial terms and, more importantly, some restriction on the values of their arguments. Consider, e.g., $\mathscr{R} = \{\mathsf{add}(\mathsf{z},y) \to y, \mathsf{add}(\mathsf{s}(x),y) \to \mathsf{s}(\mathsf{add}(x,y))\}$ with $\mathscr{R}_{\mathsf{gen}} = \mathscr{R} \cup \{\mathsf{gen} \to \mathsf{z},\ \mathsf{gen} \to \mathsf{s}(\mathsf{gen})\}$. Here, given the abstract term $\mathsf{add}(g,v)$ representing the calls to add with a ground constructor term as a first argument, we have that $\gamma(\mathsf{add}(g,v))$ is relatively $\to_{\mathscr{R}}$-terminating w.r.t. $GEN(\mathscr{R})$. However, $\mathscr{R}$ is not relatively terminating w.r.t. $GEN(\mathscr{R})$ in general (i.e., for any arbitrary term) since we may have infinite derivations like $\mathsf{add}(\mathsf{gen},\mathsf{z}) \to_{\mathscr{R}_{\mathsf{gen}}} \mathsf{add}(\mathsf{s}(\mathsf{gen}),\mathsf{z}) \to_{\mathscr{R}_{\mathsf{gen}}} \mathsf{s}(\mathsf{add}(\mathsf{gen},\mathsf{z})) \to_{\mathscr{R}_{\mathsf{gen}}} \mathsf{s}(\mathsf{add}(\mathsf{s}(\mathsf{gen}),\mathsf{z})) \to_{\mathscr{R}_{\mathsf{gen}}} \ldots$ where the initial argument of add is a "variable" represented by gen.

[8] Our definition of TRS requires $\mathcal{V}\mathsf{ar}(r) \subseteq \mathcal{V}\mathsf{ar}(l)$ for all rules $l \to r$. However, the filtering process may introduce extra variables, thus the need to deal with this situation.

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

function arguments that will definitely be ground in a narrowing derivation (or, equivalently, different from gen in a rewrite derivation). Therefore, an argument filtering with $\pi(f) = i$ would have no meaning in our context.

The main difference, though, comes from the filtering of rewrite rules. Usually, we find the following simpler notion in the literature: $\pi(l \to r) = \pi(l) \to \pi(r)$. Intuitively speaking, our extended definition will become useful when the filtered rules contain extra variables but they are not used in the considered computations. In our approach, all extra variables in the filtered rules are replaced with a special fresh symbol. Then, when the considered argument filtering fulfills some conditions (see Definition 12 below), we can guarantee the correctness of the approach.

Example 8 Consider the following TRS $\mathscr{R}$ defining the function append over lists built from nil (the empty list) and cons:

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

Given the argument filtering $\pi = \{\mathsf{append} \mapsto \{1\}, \mathsf{nil} \mapsto \varnothing, \mathsf{cons} \mapsto \{1,2\}\}$, the filtered TRS $\pi(\mathscr{R})$ is as follows:

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

Note that the right-hand side of the first rule is $\bot$ because $y \in EV(\mathsf{append}(\mathsf{nil}) \to y)$.

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 [11].

The remainder of this section adapts and extends some of the developments in [11]. Given a TRS $\mathscr{R}$ over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$, for each $\mathsf{f}/n \in \mathscr{D}$, we let $\mathsf{f}^\sharp/n$ be a fresh *tuple* (constructor) symbol; we often write F instead of $\mathsf{f}^\sharp$ in the examples. We denote by $\mathscr{F}^\sharp$ the set $\mathscr{F} \cup \{f^\sharp/n \mid f/n \in \mathscr{D}\}$. Given a term $\mathsf{f}(t_1, \ldots, t_n)$ with $\mathsf{f} \in \mathscr{D}$, we let $t^\sharp$ denote $\mathsf{f}^\sharp(t_1, \ldots, t_n)$.

Definition 9 (dependency pair [11]) Given a TRS $\mathscr{R}$ over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$, the associated set of dependency pairs, $DP(\mathscr{R})$, is defined as follows:[10]

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

Example 9 Consider the following TRS $\mathscr{R}$ defining the functions append and reverse:

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

[10] Note that if $\mathscr{R}$ is a TRS, so is $DP(\mathscr{R})$, i.e., $\mathscr{V}\mathsf{ar}(t^\sharp) \subseteq \mathscr{V}\mathsf{ar}(l^\sharp)$ for all rules $l \to t \in DP(\mathscr{R})$.

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

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

In order to prove termination, we should try to prove that there are no infinite chains of dependency pairs. The standard notion of *chain* [11], 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) and, moreover, we are only concerned with those chains that are *reachable* from the initial abstract term.

In order to define our notion of chain, we first need to formalize the following notion of *reachable calls*.

Definition 10 (reachable calls) Let $\mathscr{R}$ be a TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and t be a term. We define the set of reachable calls $calls_{\mathscr{R}}(t)$ from t in $\mathscr{R}$ as follows: $calls_{\mathscr{R}}(t) = \{\ s|_p \mid t \to^*_{\mathscr{R}} s,\ \text{with } \mathsf{root}(s|_p) \in \mathscr{D} \text{ for some position } p\ \}$. We say that a term s is reachable from t in $\mathscr{R}$ if $s \in calls_{\mathscr{R}}(t)$.

Intuitively speaking, the set of reachable calls are the subterms rooted by a defined function symbol (thus the name *call*) that occur in the rewrite derivations issuing from a given term. Note that, in particular, every subterm $t|_p$ rooted by a defined function symbol is reachable from the term t itself.

Definition 11 (chain) Let $\mathscr{R}$ and $\mathscr{P}$ be TRSs over the signatures $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and $\mathscr{F}^\sharp$, respectively. Let π be an argument filtering over $\mathscr{F}$ that is extended over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ for all $f \in \mathscr{D}$ and let t^α be an abstract term. A (possibly infinite) sequence of pairs $s_1 \to t_1$, $s_2 \to t_2$, ... from $\mathscr{P}$ is a $(t^\alpha, \mathscr{P}, \mathscr{R}, \pi)$-chain if there is a substitution $\sigma : \mathscr{V} \mapsto \mathscr{T}(\mathscr{F}, \mathscr{V})$ such that the following conditions hold:[11]

- there exists a term $s \in calls_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t}\,)$ for some $t \in \gamma(t^\alpha)$ such that $s^\sharp = \widehat{s_1\sigma}$ and
- $\widehat{t_i\sigma} \to^*_{\mathscr{R}_{\mathsf{gen}}} \widehat{s_{i+1}\sigma}$ for every two consecutive pairs in the sequence and, moreover, $\pi(\widehat{s_i\sigma}), \pi(\widehat{t_i\sigma}) \in \mathscr{T}(\mathscr{F}^\sharp)$ for all $i > 0$ (i.e., π filters away all occurrences of gen).

The following example illustrates our notion of chain.

Example 10 Consider the TRS $\mathscr{R}$ of Example 9 and its dependency pairs $DP(\mathscr{R})$, where $\mathscr{R}_{\mathsf{gen}} = \mathscr{R} \cup \{\mathsf{gen} \to \mathsf{nil},\ \mathsf{gen} \to \mathsf{cons}(\mathsf{gen},\mathsf{gen})\}$. Given the abstract term $t^\alpha = \mathsf{append}(v,g)$, we have that "(1), (1),..." is an infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chain for any argument filtering in which $\pi(\mathsf{append}) = \pi(\mathsf{APPEND}) = \{2\}$ since there is a substitution $\sigma = \{y \mapsto \mathsf{nil}\}$ such that (we denote the dependency pair (1) by $s_1 \to t_1$)

$$\widehat{t_1\sigma} = \mathsf{APPEND}(\mathsf{gen},\mathsf{nil}) \to_{\mathscr{R}_{\mathsf{gen}}} \mathsf{APPEND}(\mathsf{cons}(\mathsf{gen},\mathsf{gen}),\mathsf{nil}) = \widehat{s_1\sigma}$$

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

[11] As in [11], we assume fresh variables in every (occurrence of a) dependency pair and that the domain of substitutions may be infinite.

Observe that it is not difficult to check the first condition in the definition of chain above, e.g., given the term $\mathsf{append}(\mathsf{nil},y) \in \gamma(t^\alpha)$, we have that $s = \mathsf{append}(\mathsf{nil},\mathsf{gen}) \in \mathit{calls}_{\mathscr{R}_\mathsf{gen}}(\mathsf{append}(\mathsf{nil},\mathsf{gen}))$ (since every subterm rooted by a defined function is reachable from itself) and $s^\sharp = \widehat{s_1\sigma} = \mathsf{APPEND}(\mathsf{nil},\mathsf{gen})$.

Unfortunately, not all argument filterings are useful in our context. In the following, we focus on what we call *safe* argument filterings. We denote by $s \to_{>p,\mathscr{R}} t$ any rewrite step $s \to_{q,\mathscr{R}} t$ with $p < q$, i.e., any rewrite step where a strict subterm $s|_q$ of $s|_p$ is reduced. This notation is extended to rewrite derivations in the natural way. Safe argument filterings are then formalized as follows:

Definition 12 (safe argument filtering) Let $\mathscr{R}$ be a TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$. Let π be an argument filtering over $\mathscr{F}$ that is extended over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ for all $f \in \mathscr{D}$ and let $t^\alpha = \mathsf{f}(m_1,\ldots,m_n)$ be an abstract term. We say that π is safe for t^α in $\mathscr{R}$ if the following conditions hold:

(1) $m_i = g$ for all $i \in \pi(\mathsf{f})$;
(2) $\mathscr{V}\mathsf{ar}(\pi(t)) \subseteq \mathscr{V}\mathsf{ar}(\pi(s))$ for all dependency pairs $s \to t \in \mathit{DP}(\mathscr{R})$;
(3) $s \to^*_{>\varepsilon,\mathscr{R}_\mathsf{gen}} s'$ implies $\pi(s) \to^*_{>\varepsilon,\pi(\mathscr{R})} \pi(s')$ for all $s \in \mathit{calls}_{\mathscr{R}_\mathsf{gen}}(\widehat{t})$ with $t \in \gamma(t^\alpha)$.

The first condition in the definition above should be clear: a safe argument filtering must remove all non-ground arguments of the initial abstract term. The second condition ensures that the ground arguments of filtered terms are correctly propagated from one call to the next one. The third condition, though, is more subtle. Basically, it allows us to ensure that the chains of dependency pairs are correctly preserved by the argument filtering (despite replacing extra variables with $\bot$); this will become clear in the proof of Theorem 4 below, where this condition is essential.

Example 11 Consider the following TRS $\mathscr{R} = \{\mathsf{f}(\mathsf{s}(x),y) \to \mathsf{f}(y,x)\}$ over $\mathscr{F} = \{\mathsf{f}/1\} \uplus \{\mathsf{z}/0,\mathsf{s}/1\}$ and the abstract term $t^\alpha = \mathsf{f}(g,v)$.

- Clearly, the argument filtering $\pi_1 = \{\mathsf{f} \mapsto \{1,2\},\mathsf{s} \mapsto \{1\},\mathsf{z} \mapsto \varnothing\}$ is not safe because it violates the first condition, i.e., $\pi(\mathsf{f}(g,v))|_2 = v$.
- The argument filtering $\pi_2 = \{\mathsf{f} \mapsto \{1\},\mathsf{s} \mapsto \{1\},\mathsf{z} \mapsto \varnothing\}$ is not safe because the filtered dependency pairs contain some extra variable, i.e., we have a dependency pair $\mathsf{f}(\mathsf{s}(x),y) \to \mathsf{f}(y,x) \in \mathit{DP}(\mathscr{R})$ with $\pi_2(\mathsf{f}(\mathsf{s}(x),y)) = \mathsf{f}(\mathsf{s}(x))$, $\pi_2(\mathsf{f}(y,x)) = \mathsf{f}(y)$, and $\mathscr{V}\mathsf{ar}(\mathsf{f}(y)) \not\subseteq \mathscr{V}\mathsf{ar}(\mathsf{f}(\mathsf{s}(x)))$.
- In contrast, the argument filtering $\pi_3 = \{\mathsf{f} \mapsto \varnothing,\mathsf{s} \mapsto \{1\},\mathsf{z} \mapsto \varnothing\}$ is safe.

Example 12 Consider now the following TRS

$$\mathscr{R} = \{\mathsf{b} \to \mathsf{h}(\mathsf{g}(\mathsf{a},\mathsf{a})),\ \mathsf{h}(\mathsf{c}(\mathsf{a})) \to \mathsf{b},\ \mathsf{g}(x,y) \to c(y)\}$$

over $\mathscr{F} = \{\mathsf{b}/0,\mathsf{h}/1,\mathsf{g}/2\} \uplus \{\mathsf{a}/0,\mathsf{c}/1\}$ and the abstract term $t^\alpha = b$. Consider an argument filtering $\pi = \{\mathsf{b} \mapsto \varnothing,\mathsf{h} \mapsto \{1\},\mathsf{g} \mapsto \{1\},\mathsf{a} \mapsto \varnothing,\mathsf{c} \mapsto \{1\}\}$. Then, we have

$$\pi(\mathscr{R}) = \{\mathsf{b} \to \mathsf{h}(\mathsf{g}(\mathsf{a})),\ \mathsf{h}(\mathsf{c}(\mathsf{a})) \to \mathsf{b},\ \mathsf{g}(x) \to c(\bot)\}$$

This argument filtering π is not safe because, although the first and second conditions in the definition of a safe argument filtering hold, the third one does not hold. In particular, we can find a reachable call, namely $\mathsf{h}(\mathsf{g}(\mathsf{a},\mathsf{a}))$, such that $\mathsf{h}(\mathsf{g}(\mathsf{a},\mathsf{a})) \to_{>\varepsilon,\mathscr{R}_\mathsf{gen}}$

$\mathsf{h}(\mathsf{c}(\mathsf{a}))$ but $\pi(\mathsf{h}(\mathsf{g}(\mathsf{a},\mathsf{a}))) = \mathsf{h}(\mathsf{g}(\mathsf{a})) \not\to_{>\varepsilon,\pi(\mathscr{R})} \mathsf{h}(\mathsf{c}(\mathsf{a})) = \pi(\mathsf{h}(\mathsf{c}(\mathsf{a})))$ because the only applicable rule is $[\pi(\mathsf{g}(x,y)) \to \pi(\mathsf{c}(y))]_\perp = \mathsf{g}(x) \to \mathsf{c}(\perp)$. Note that, in this case, we could recover safeness by having $\pi(\mathsf{c}) = \varnothing$ instead of $\pi(\mathsf{c}) = \{1\}$.

In the following, we consider that the input for the termination analysis is a left-linear constructor TRS together with a safe argument filtering for some abstract term. We will discuss the generation of safe argument filterings in Section 5.

Before we proceed with the main result of this section, we need the following auxiliary result for safe argument filterings (the proof can be found in the appendix):

Lemma 5 *Let $\mathscr{R}$ be a TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$. Let π be an argument filtering over $\mathscr{F}$ that is extended over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ for all $f \in \mathscr{D}$, and let t^α be an abstract term. If π is safe for t^α in $\mathscr{R}$, then $\pi(s) \in \mathscr{T}(\mathscr{F})$ for all terms $s \in calls_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$ with $t \in \gamma(t^\alpha)$.*

Intuitively speaking, this result allows us to conclude that safe argument filterings actually remove all occurrences of gen in those rewrite sequences issuing from a concretization of the abstract term.

The following result states the soundness of our approach:

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

Proof We proceed by contradiction.

Let us assume that there exists no infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chain, but $\gamma(t^\alpha)$ is not $\rightsquigarrow_{\mathscr{R}}$-terminating. By Corollary 1, we have that $\widehat{\gamma(t^\alpha)}$ is not relatively $\to_{\mathscr{R}}$-terminating to $GEN(\mathscr{R})$. Therefore, there exists an infinite rewrite derivation for some term $\widehat{t} \in \widehat{\gamma(t^\alpha)}$ in which an infinite number of rules from $\mathscr{R}$ are used. In the following, we denote by $\overline{s}$ a finite sequence of terms of the form $s_1, \ldots, s_n$.

Let $\widehat{t} = \mathsf{f}_1(\overline{s_1})$. Since $\overline{s_1}$ does not contain defined function symbols (apart from gen), the infinite derivation starts as follows: $\mathsf{f}_1(\overline{s_1}) \to^*_{>\varepsilon,GEN(\mathscr{R})} \mathsf{f}_1(\overline{u_1}) \to_{\varepsilon,R_1} r_1\sigma_1$, with $R_1 = (\mathsf{f}_1(\overline{w_1}) \to r_1) \in \mathscr{R}$ and $\mathsf{f}_1(\overline{w_1})\sigma_1 = \mathsf{f}_1(\overline{u_1})$. Since σ_1 cannot introduce defined function symbols from $\mathscr{D}$ (only constructors or occurrences of gen), all defined function symbols of $r_1\sigma_1$ occur on positions of r_1. Therefore, there must be a subterm r_1' of r_1 with a defined root such that $r_1'\sigma_1$ also starts an infinite rewrite derivation. Assume that r_1' is the smallest such subterm (i.e., for all proper subterms r_1'' of r_1', the term $r_1''\sigma_1$ is relatively $\to_{\mathscr{R}}$-terminating w.r.t. $GEN(\mathscr{R})$). Then, $\mathsf{f}_1(\overline{w_1}) \to r_1'$ is the first dependency pair of the infinite chain that we construct.

Since $\widehat{t} = \mathsf{f}_1(\overline{s_1}) \to^*_{>\varepsilon,GEN(\mathscr{R})} \mathsf{f}_1(\overline{u_1}) \to_{\varepsilon,R_1} r_1\sigma_1$, we have $\mathsf{f}_1(\overline{u_1}), r_1'\sigma_1 \in calls_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$. Now, since π is a safe argument filtering, by Lemma 5, we have $\pi(\mathsf{f}_1(\overline{u_1})), \pi(r_1'\sigma_1) \in \mathscr{T}(\mathscr{F})$.

Now, since $r_1'\sigma_1$ is the smallest subterm that starts an infinite rewrite sequence where infinitely many rules of $\mathscr{R}$ are applied, the infinite derivation continues as follows: $r'\sigma_1 \to^*_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} \mathsf{f}_2(\overline{u_2}) \to_{\varepsilon,R_2} r_2\sigma_2$, with $R_2 = (\mathsf{f}_2(\overline{w_2}) \to r_2) \in \mathscr{R}$ and $\mathsf{f}_2(\overline{w_2})\sigma_2 = \mathsf{f}_2(\overline{u_2})$. Here, we have that the terms $\overline{u_2}$ cannot introduce infinite rewrite sequences

where infinitely many rules of $\mathscr{R}$ are applied (which is safe since all proper subterms of $r_1'\sigma_1$ are relatively $\to_{\mathscr{R}}$-terminating w.r.t. $GEN(\mathscr{R})$ by assumption). Therefore, there must be a subterm r_2' of r_2 with a defined root such that $r_2'\sigma_2$ also starts an infinite rewrite derivation. Assume that r_2' is the smallest such subterm. Then, $\mathsf{f}_2(\overline{w_2}) \to r_2'$ is our second dependency pair.

Since $\widehat{t} = \mathsf{f}_1(\overline{s_1}) \to^*_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} \mathsf{f}_1(\overline{u_1}) \to_{\varepsilon,R_1} r_1\sigma_1$ and $r_1'\sigma_1 \to^*_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} \mathsf{f}_2(\overline{u_2}) \to_{\varepsilon,R_2} r_2\sigma_2$, with $r_1'\sigma_1$ a subterm of $r_1\sigma_1$ and $r_2'\sigma_2$ a subterm of $r_2\sigma_2$, we have $\mathsf{f}_2(\overline{u_2}), r_2'\sigma_2 \in calls_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$. Hence, since π is a safe argument filtering, by Lemma 5, we have $\pi(\mathsf{f}_2(\overline{u_2})), \pi(r_2'\sigma_2) \in \mathscr{T}(\mathscr{F})$ too.

The infinite sequence continues by rewriting $r_2'\sigma_2$'s proper subterms 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 $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chain, which contradicts our initial assumption. □

It is worthwhile to observe that the $\rightsquigarrow_{\mathscr{R}}$-termination of $\gamma(t^\alpha)$ does not imply the absence of $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chains because rewrite sequences are not required to be admissible in Definition 11. The following example illustrates this point.

Example 13 Consider the TRS $\mathscr{R} = \{\mathsf{f}(x) \to \mathsf{h}(x,x),\ \mathsf{h}(\mathsf{a},\mathsf{b}) \to \mathsf{h}(\mathsf{a},\mathsf{b})\}$ over $\mathscr{F} = \{\mathsf{f}/1, \mathsf{h}/2\} \uplus \{\mathsf{a}/0, \mathsf{b}/0\}$, together with the abstract term $t^\alpha = \mathsf{f}(v)$. Then, we have $\mathscr{R}_{\mathsf{gen}} = \mathscr{R} \cup \{\mathsf{gen} \to \mathsf{a},\ \mathsf{gen} \to \mathsf{b}\}$.

Here, all narrowing derivations issuing from the terms in $\gamma(t^\alpha) = \{\mathsf{f}(x), \mathsf{f}(\mathsf{a}), \mathsf{f}(\mathsf{b})\}$ are terminating since, given an arbitrary term $\mathsf{f}(t) \in \gamma(t^\alpha)$ with $t \in \mathscr{T}(\mathscr{C}, \mathscr{V})$ a constructor term, we have $\mathsf{f}(t) \rightsquigarrow_{\mathscr{R}} \mathsf{h}(t,t)$ but $\mathsf{h}(t,t)$ does not unify with the left-hand side $\mathsf{h}(\mathsf{a},\mathsf{b})$ of the second rule no matter the value of t.

In contrast, given the argument filtering $\pi = \{\mathsf{f} \mapsto \varnothing, \mathsf{h} \mapsto \varnothing, \mathsf{a} \mapsto \varnothing, \mathsf{b} \mapsto \varnothing\}$, one can easily construct an infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chain since there exists a term $\mathsf{h}(\mathsf{a},\mathsf{b}) \in calls_{\mathscr{R}_{\mathsf{gen}}}(\widehat{\mathsf{f}(x)}) = calls_{\mathscr{R}_{\mathsf{gen}}}(\mathsf{f}(\mathsf{gen}))$, with $\mathsf{f}(x) \in \gamma(t^\alpha)$, such that $\mathsf{h}^\sharp(\mathsf{a},\mathsf{b})$ is the left-hand side of the first dependency pair in the infinite sequence: $\mathsf{h}^\sharp(\mathsf{a},\mathsf{b}) \to \mathsf{h}^\sharp(\mathsf{a},\mathsf{b}), \mathsf{h}^\sharp(\mathsf{a},\mathsf{b}) \to \mathsf{h}^\sharp(\mathsf{a},\mathsf{b}), \ldots$

Our next result further clarifies the relation between the non-termination of narrowing and the existence of infinite chains (the proof can be found in the appendix):

Lemma 6 *Let $\mathscr{R}$ be a TRS over the signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and let $\mathscr{P} \subseteq DP(\mathscr{R})$ be a set of dependency pairs. Let π be an argument filtering over $\mathscr{F}$ that is extended over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ for all $f \in \mathscr{D}$ and let t^α be an abstract term. For every $(t^\alpha, \mathscr{P}, \mathscr{R}, \pi)$-chain of the form $s_1^\sharp \to t_1^\sharp, s_2^\sharp \to t_2^\sharp, \ldots$, there exists a rewrite derivation of the form*

$$\begin{array}{lll}
\widehat{t} & \to^*_{\mathscr{R}_{\mathsf{gen}}} & t'[\widehat{s_1\sigma}]_p \to_{p,\mathscr{R}} t'[r_1\widehat{\sigma[t_1\sigma]}_{p_1}]_p \\
& \to^*_{>p.p_1,\mathscr{R}_{\mathsf{gen}}} & t'[r_1\widehat{\sigma[s_2\sigma]}_{p_1}]_p \to_{p.p_1,\mathscr{R}} t'[r_1\sigma[r_2\widehat{\sigma[t_2\sigma]}_{p_2}]_{p_1}]_p \\
& \to^*_{>p.p_1.p_2,\mathscr{R}_{\mathsf{gen}}} & \cdots
\end{array}$$

for some substitution σ.

As a consequence of this result we have that, for every infinite chain, there exists an associated infinite rewrite sequence in $\mathscr{R}_{\mathsf{gen}}$. Hence, if this sequence is admissible, we can lift it to a corresponding narrowing derivation and, thus, the converse of Theorem 3 holds when only admissible rewrite derivations are considered.

Now, in order to show the absence of $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chains automatically, one could adapt the *DP framework* [34] or, alternatively, reduce the termination problem to a standard dependency pair problem. In this work, we consider the second approach, i.e., we introduce a sufficient condition for the termination of narrowing in terms of a standard dependency pair problem. This approach allows us to reuse all the existing body of techniques and tools for proving the termination of rewriting within the DP framework.

Let us first summarize some basic notions from the DP framework [34]. In this context, a *DP problem* is a tuple $(\mathscr{P}, \mathscr{R})$ where $\mathscr{P}$ and $\mathscr{R}$ are TRSs. A $(\mathscr{P}, \mathscr{R})$-chain is a (possibly infinite) sequence of pairs $s_1 \to t_1, s_2 \to t_2, \ldots$ from $\mathscr{P}$ such that there is a substitution σ with $t_i\sigma \to^*_{\mathscr{R}} s_{i+1}\sigma$ for all $i > 0$. A DP problem $(\mathscr{P}, \mathscr{R})$ is *finite* if there are no infinite $(\mathscr{P}, \mathscr{R})$-chains. 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 $(DP(\mathscr{R}), \mathscr{R})$ and applies sound DP processors until an empty set of DP problems is obtained.

Theorem 4 *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$, let $\mathscr{P} \subseteq DP(\mathscr{R})$ be a set of dependency pairs, and let t^α be an abstract term. Let π be a safe argument filtering for t^α in $\mathscr{R}$ that is extended over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ for all $f \in \mathscr{D}$. If the DP problem $(\pi(\mathscr{P}), \pi(\mathscr{R}))$ is finite, then there are no infinite $(t^\alpha, \mathscr{P}, \mathscr{R}, \pi)$-chains.*

Proof We proceed by contradiction.

Let us assume that there exists no infinite $(\pi(\mathscr{P}), \pi(\mathscr{R}))$-chain but there is an infinite $(t^\alpha, \mathscr{P}, \mathscr{R}, \pi)$-chain of the form $s_1^\sharp \to t_1^\sharp, s_2^\sharp \to t_2^\sharp, \ldots$ By definition, neither $\pi(\mathscr{P})$ nor $\pi(\mathscr{R})$ contain extra variables (since they are all replaced by $\bot$). Moreover, since π is safe, we have that $\mathscr{V}\mathsf{ar}(\pi(t)) \subseteq \mathscr{V}\mathsf{ar}(\pi(s))$ for all $s \to t \in DP(\mathscr{R})$ and, thus, $\pi(\mathscr{P})$ contains no occurrences of $\bot$. Now, by Lemma 6, there exists an infinite rewrite sequence of the form

$$\begin{array}{lll}
\widehat{t} & \to^*_{\mathscr{R}_{\mathsf{gen}}} & t'[\widehat{s_1\sigma}]_p \to_{p,\mathscr{R}} t'[r_1\widehat{\sigma[t_1\sigma}]_{p_1}]_p \\
& \to^*_{>p.p_1,\mathscr{R}_{\mathsf{gen}}} & t'[r_1\widehat{\sigma[s_2\sigma}]_{p_1}]_p \to_{p.p_1,\mathscr{R}} t'[r_1\sigma[r_2\widehat{\sigma[t_2\sigma}]_{p_2}]_{p_1}]_p \\
& \to^*_{>p.p_1.p_2,\mathscr{R}_{\mathsf{gen}}} & \cdots
\end{array}$$

for some substitution σ. By Definition 10, we have $\widehat{s_1\sigma}, \widehat{t_1\sigma}, \widehat{s_2\sigma}, \widehat{t_2\sigma}, \ldots \in calls_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$. Since π is safe, by Lemma 5, we have $\pi(\widehat{s_1\sigma}), \pi(\widehat{t_1\sigma}), \pi(\widehat{s_2\sigma}), \pi(\widehat{t_2\sigma}), \ldots \in \mathscr{T}(\mathscr{F})$. Finally, since π is safe (third condition), we have that $\widehat{t_i\sigma} \to^*_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} \widehat{s_{i+1}\sigma}$ implies that $\pi(t_i\sigma) \to^*_{>\varepsilon,\pi(\mathscr{R})} \pi(s_{i+1}\sigma)$ also holds for all $i > 0$. Therefore, there exists an infinite $(\pi(\mathscr{P}), \pi(\mathscr{R}))$-chain of the form $\pi(s_1^\sharp) \to \pi(t_1^\sharp), \pi(s_2^\sharp) \to \pi(t_2^\sharp), \ldots$, which contradicts our initial assumption. □

Example 14 Consider the TRS of Example 9, the abstract term $t^\alpha = \mathsf{append}(g, v)$, and the argument filtering $\pi = \{\mathsf{append} \mapsto \{1\}, \mathsf{reverse} \mapsto \{1\}, \mathsf{nil} \mapsto \varnothing, \mathsf{cons} \mapsto \{1,2\}\}$ which is safe for t^α. Here, Theorem 4 reduces the termination of narrowing to the termination of the following DP problem:

- Dependency pairs $\pi(DP(\mathscr{R}))$:

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

- Rewrite system $\pi(\mathscr{R})$:

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

The derived DP problem can be proved terminating using standard techniques. It is worthwhile to note that the elimination of the extra variable in the first rule of $\pi(\mathscr{R})$ is crucial to be able to prove termination in this example.

Unfortunately, in some cases, the filtering of both the TRS and its dependency pairs may introduce an over approximation that prevents us from proving the termination of narrowing.

Example 15 Consider the following TRS $\mathscr{R} = \{\mathsf{f}(x) \to \mathsf{g}(\mathsf{a},\mathsf{b}),\ \mathsf{g}(\mathsf{a},y) \to \mathsf{g}(\mathsf{b},y)\}$ and the abstract term $\mathsf{f}(v)$. Here, we have the following dependency pairs: $DP(\mathscr{R}) = \{\mathsf{F}(x) \to \mathsf{G}(\mathsf{a},\mathsf{b}),\ \mathsf{G}(\mathsf{a},y) \to \mathsf{G}(\mathsf{b},y)\}$.

The argument filtering $\pi = \{\mathsf{f} \mapsto \varnothing, \mathsf{g} \mapsto \{2\}, \mathsf{a} \mapsto \varnothing, \mathsf{b} \mapsto \varnothing\}$ is safe for $\mathsf{f}(v)$. Clearly, there are no infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chains since the only possible chain would be $\mathsf{G}(\mathsf{a},y) \to \mathsf{G}(\mathsf{b},y),\ \mathsf{G}(\mathsf{a},y) \to \mathsf{G}(\mathsf{b},y), \ldots$ but we have

$$\widehat{\mathsf{G}(\mathsf{b},y)}\sigma \not\to^*_{\mathscr{R}_{\mathsf{gen}}} \widehat{\mathsf{G}(\mathsf{a},y)}\sigma$$

no matter the value of the substitution σ.

However, by using Theorem 4, we get a DP problem with the following set of dependency pairs: $\pi(DP(\mathscr{R})) = \{\mathsf{F} \to \mathsf{G}(\mathsf{b}),\ \mathsf{G}(y) \to \mathsf{G}(y)\}$. Clearly, the resulting DP problem is infinite (no matter the rules of $\pi(\mathscr{R})$ since the left- and right-hand sides of the dependency pair $\mathsf{G}(y) \to \mathsf{G}(y)$ are equal) and, thus, cannot be used to prove the termination of narrowing in the original TRS.

In order to overcome this drawback, we adapt a well-known DP processor based on the construction of the *estimated dependency graph* [34], a graph that can be used to estimate which dependency pairs can follow each other in chains.[12] By using this graph, some potential chains can be safely ignored. Therefore, it can be applied to the initial TRS and its dependency pairs as a pre-processing stage in order to increase the accuracy of the analysis.

[12] We note that the estimated dependency graph is similar to the notion of *loop-check* in [3], where it is used to remove unfeasible narrowing derivations.

Definition 13 (estimated dependency graph [11]) Let $\mathscr{R}$ be a TRS. The estimated dependency graph of $\mathscr{R}$ is the directed graph whose nodes (vertices) are the dependency pairs of $\mathscr{R}$ and there is an arc (directed edge) from $s \to t$ to $u \to v$ iff $\mathsf{REN}(\mathsf{CAP}(t))$ and u are unifiable, where

- $\mathsf{CAP}(t)$ replaces all proper subterms of t that are rooted by a defined symbol by different fresh variables, and
- $\mathsf{REN}(t')$ replaces all variables in t' by different fresh variables.

The estimated dependency graph of $\mathscr{R}$ is denoted by $EDG(\mathscr{R})$. In the following, we denote by $SCC(EDG(\mathscr{R}))$ the set of strongly connected components (SCC) of $EDG(\mathscr{R})$.

Let t^α be an abstract term and let $\mathscr{P} \in SCC(EDG(\mathscr{R}))$ be a strongly connected component of $EDG(\mathscr{R})$. Then, we say that $\mathscr{P}$ is reachable from t^α if there exists a path in $EDG(\mathscr{R})$ from a node $s^\sharp \to t^\sharp$ with $\mathsf{root}(s) = \mathsf{root}(t^\alpha)$ to a node of $\mathscr{P}$.

Our next result shows that it is enough to consider those chains that are built using the dependency pairs of some strongly connected component of the estimated dependency graph:

Theorem 5 *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and t^α be an abstract term. Let π be a safe argument filtering for t^α in $\mathscr{R}$ that is extended over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ for all $f \in \mathscr{D}$. If there is no infinite $(t^\alpha, \mathscr{P}, \mathscr{R}, \pi)$-chain for any strongly connected component $\mathscr{P} \in SCC(EDG(\mathscr{R}))$ which is reachable from t^α, then there are no infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chains.*

Proof We provide a proof of this result for completeness, but it is essentially equivalent to that of Theorem 21 in [11]. In the following, we prove that, if the dependency pairs $s_1 \to t_1$ and $s_2 \to t_2$ are part of a $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chain, then there is a corresponding arc in $EDG(\mathscr{R})$, i.e., $\mathsf{REN}(\mathsf{CAP}(t_1))$ and s_2 are unifiable. This suffices for this theorem, since then every infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chain corresponds to an infinite path in the graph. This path ends in some strongly connected component $\mathscr{P}$ and, thus, there is also an infinite $(t^\alpha, \mathscr{P}, \mathscr{R}, \pi)$-chain that is a postfix of the infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chain where $\mathscr{P}$ is reachable from t^α. Hence, if there are no infinite $(t^\alpha, \mathscr{P}, \mathscr{R}, \pi)$-chains, then there are no $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chains too.

Let the dependency pairs $s_1 \to t_1$ and $s_2 \to t_2$ be part of a $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chain, i.e., $\widehat{t_1\sigma} \to^*_{\mathscr{R}_{\mathsf{gen}}} \widehat{s_2\sigma}$. for some substitution σ. Let $p_1, \ldots, p_k$ be the top (parallel) positions where t_1 has defined symbols (including gen) or variables. Then, we have $\mathsf{REN}(\mathsf{CAP}(t_1)) = t_1[y_1]_{p_1} \cdots [y_k]_{p_k}$ for some fresh variables $y_1, \ldots, y_k$. Since $\widehat{t_1\sigma} \to^*_{\mathscr{R}_{\mathsf{gen}}} \widehat{s_2\sigma}$ and both t_1 and s_2 are rooted by (constructor) tuple functions, we have that $t_1\mu = \widehat{s_2\sigma}$ for some substitution μ such that $y_i\mu = \widehat{s_2\sigma}|_{p_i}$ for all $i = 1, \ldots, k$. Furthermore, since $\mathsf{REN}(\mathsf{CAP}(t_1))$ only contains fresh variables, then $\mathsf{REN}(\mathsf{CAP}(t_1))$ and s_2 are unifiable with a substitution $\delta = \mu \cup \sigma \cup \{x \in \mathscr{V}\mathsf{ar}(s_2\sigma) \mid x \mapsto \mathsf{gen}\}$. □

The following corollary is a consequence of Theorems 3, 4 and 5 above:

Corollary 2 *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and t^α be an abstract term. Let π be a safe argument filtering for t^α in $\mathscr{R}$ that is extended*

over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ *for all* $f \in \mathscr{D}$. *If* $(\pi(\mathscr{P}), \pi(\mathscr{R}))$ *is finite for every strongly connected component* $\mathscr{P} \in SCC(EDG(\mathscr{R}))$ *which is reachable from* t^α, *then* $\gamma(t^\alpha)$ *is* $\leadsto_{\mathscr{R}}$*-terminating.*

4.3 A Transformational Approach

In this section, we present an alternative approach for proving the termination of narrowing. The basic idea is similar to that in the previous section: reduce the termination of narrowing to the (relative) termination of rewriting using data generators and then use some argument filtering to eliminate those subterms that might be bound to a data generator.

Now, however, our aim is to transform the original TRS $\mathscr{R}$ into a new TRS $\mathscr{R}'$ so that narrowing terminates in $\mathscr{R}$ if rewriting terminates in $\mathscr{R}'$. 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 [47], that we simplify because, in our case, an argument filtering never returns a single argument position (i.e., we do not accept collapsing argument filterings). Roughly speaking our program transformation generates, for every rule $l \to r$ of the original program,

- a filtered rule $\pi(l) \to \pi(r)$ and
- an additional rule $\pi(l) \to \pi(r')$ for each subterm r' of r that is filtered away in $\pi(r)$ and such that $\pi(r')$ is not a constructor term.

Definition 14 (argument filtering transformation) Let $\mathscr{R}$ be a TRS over a signature $\mathscr{F}$ and let π be an argument filtering over $\mathscr{F}$. The argument filtering transformation AFT_π is defined as follows:

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

where the auxiliary function dec_π is defined inductively as follows:

$$\begin{array}{rl} \mathsf{dec}_\pi(x) = \varnothing & (x \in \mathscr{V}) \\ \mathsf{dec}_\pi(f(t_1,\ldots,t_n)) = \bigcup_{i \notin \pi(f)} \{t_i\} \cup \bigcup_{i=1}^n \mathsf{dec}_\pi(t_i) & (f \in \mathscr{F}) \end{array}$$

Example 16 Consider the TRS $\mathscr{R}$ of Example 9. If we consider the argument filtering $\pi_1 = \{\mathsf{append} \mapsto \{1\}, \mathsf{reverse} \mapsto \{1\}, \mathsf{nil} \mapsto \varnothing, \mathsf{cons} \mapsto \{1,2\}\}$ of Example 14, then $\mathsf{AFT}_{\pi_1}(\mathscr{R})$ returns the same filtered rewrite system $\pi(\mathscr{R})$ of Example 14.

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

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

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 following auxiliary results are needed for proving the soundness of the AFT transformation (the proofs can be found in the appendix). Our first result states a basic property of argument filterings and function dec_π:

Lemma 7 *Let π be an argument filtering and let r be a term. Then, for every subterm $r|_p$ of r rooted by a defined symbol, either $\pi(r|_p)$ is a subterm of $\pi(r)$ or there exists a term $r' \in \mathsf{dec}_\pi(r)$ such that $\pi(r|_p)$ is a subterm of $\pi(r')$.*

Our second result is essential to relate the chains in a TRS $\mathscr{R}$ and in $\mathsf{AFT}_\pi(\mathscr{R})$:

Lemma 8 *Let $\mathscr{R}$ be a left-linear constructor TRS, t^α be an abstract term, and π be a safe argument filtering for t^α in $\mathscr{R}$. Then, $\pi(DP(\mathscr{R})) \subseteq DP(\mathsf{AFT}_\pi(\mathscr{R}))$.*

Now we prove the soundness of the AFT transformation, the main contribution of this section:

Theorem 6 *Let $\mathscr{R}$ be a left-linear constructor TRS and let t^α be an abstract term. Let π be a safe argument filtering for t^α in $\mathscr{R}$. If $\mathsf{AFT}_\pi(\mathscr{R})$ is terminating, then there are no infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chains.*

Proof We proceed by contradiction. Assume that $\mathsf{AFT}_\pi(\mathscr{R})$ is terminating but there exists an infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chain. By Theorem 4, we have that the DP problem $(\pi(DP(\mathscr{R})), \pi(\mathscr{R}))$ is not finite and, as a consequence, there must be an infinite $(\pi(DP(\mathscr{R})), \pi(\mathscr{R}))$-chain. Trivially, by definition of the AFT transformation, we have $\pi(\mathscr{R}) \subseteq \mathsf{AFT}_\pi(\mathscr{R})$. Furthermore, by Lemma 8, we have that $\pi(DP(\mathscr{R})) \subseteq DP(\mathsf{AFT}_\pi(\mathscr{R}))$ holds. Hence there must be an infinite $(DP(\mathsf{AFT}_\pi(\mathscr{R})), \mathsf{AFT}_\pi(\mathscr{R}))$-chain, which contradicts our initial assumption. □

A straightforward consequence of the previous result is the following corollary that shows that the termination of narrowing in a TRS $\mathscr{R}$ can be analyzed by using standard techniques and tools over $\mathsf{AFT}_\pi(\mathscr{R})$:

Corollary 3 *Let $\mathscr{R}$ be a left-linear constructor TRS and let t^α be an abstract term. Let π be a safe argument filtering for t^α in $\mathscr{R}$. If $\mathsf{AFT}_\pi(\mathscr{R})$ is terminating, then $\gamma(t^\alpha)$ is $\leadsto_{\mathscr{R}}$-terminating.*

Unfortunately, the AFT transformation is generally less precise than the direct approach that we introduced in the previous section, as the following example illustrates:

Example 17 Consider again the TRS from Example 15:

$$\mathscr{R} = \{\mathsf{f}(x) \to \mathsf{g}(\mathsf{a},\mathsf{b}),\ \mathsf{g}(\mathsf{a},y) \to \mathsf{g}(\mathsf{b},y)\}$$

the abstract term $\mathsf{f}(v)$, and the argument filtering $\pi = \{\mathsf{f} \mapsto \varnothing, \mathsf{g} \mapsto \{2\}, \mathsf{a} \mapsto \varnothing, \mathsf{b} \mapsto \varnothing\}$, which is safe for $\mathsf{f}(v)$. Here, the AFT transformation returns the following TRS:

$$\mathsf{AFT}_\pi(\mathscr{R}) = \{\mathsf{f} \to \mathsf{g}(\mathsf{b}),\ \mathsf{g}(y) \to \mathsf{g}(y)\}$$

which is clearly non-terminating. In contrast, by using the original TRS, one can easily prove that there are no infinite $(t^\alpha, DP(\mathscr{R}), \mathscr{R}, \pi)$-chains (e.g., by using the estimated dependency graph).

Therefore, the only advantage of the AFT transformation over the direct approach of the previous section lies in the fact that it can be more easily implemented and that, moreover, the transformed TRS can be used as the input of any termination tool (not necessarily based on the DP framework).

5 Safe Argument Filterings

In this section, we consider the generation of safe argument filterings that can be used to analyze the termination of narrowing. In general, though, determining if an argument filtering is safe (according to Definition 12) is undecidable. Therefore, in the following we introduce *sufficient*—but not necessary—conditions.

5.1 Sufficient Conditions for Safe Argument Filterings

In the following, we say that an argument filtering π over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ is *trivial* iff $\pi(f) = \varnothing$ for all $f \in \mathscr{D}$. Our first sufficient condition for the safeness of an argument filtering is defined as follows:

Lemma 9 *Let $\mathscr{R}$ be a TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and let t^α be an abstract term. Let π be a non-trivial argument filtering π that fulfills the following conditions:*

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

Then, π is safe for t^α in $\mathscr{R}$.

In order to prove this result, we first prove that it is equivalent to the "variable condition" of [60], i.e., that $\mathscr{V}\mathsf{ar}(\pi(r)) \subseteq \mathscr{V}\mathsf{ar}(\pi(l))$ for all $l \to r \in (DP(\mathscr{R}) \cup \mathscr{R})$.

Lemma 10 *Let $\mathscr{R}$ be a TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and let t^α be an abstract term. Let π be a non-trivial argument filtering π that fulfills the conditions of Lemma 9. Then, we have $\mathscr{V}\mathsf{ar}(\pi(r)) \subseteq \mathscr{V}\mathsf{ar}(\pi(l))$ for all $l \to r \in (DP(\mathscr{R}) \cup \mathscr{R})$.*

Proof We prove the claim by contradiction. Assume that π is a non-trivial argument filtering for t^α in $\mathscr{R}$ that fulfills the conditions of Lemma 9 but there exists a rule $l \to r \in (DP(\mathscr{R}) \cup \mathscr{R})$ such that $\mathscr{V}\mathsf{ar}(\pi(r)) \not\subseteq \mathscr{V}\mathsf{ar}(\pi(l))$. Let us consider that $l = \mathsf{f}(l_1, \ldots, l_n)$ for some $\mathsf{f} \in \mathscr{D}$.

Since π is non-trivial, we assume that there exists some defined symbol $\mathsf{h} \in \mathscr{D}$ such that $\pi(\mathsf{h}) \neq \varnothing$. Let us consider, for simplicity, that $\pi(\mathsf{h}) = \{j\}$.

Since π fulfills the second condition of Lemma 9, we have that, for any narrowing step $s_1 \leadsto_{p, l \to r, \sigma} s_2$, if $\pi(s_1|_p) \in \mathscr{T}(\mathscr{F})$ for all subterm $s_1|_p$ with $\mathsf{root}(s_1|_p) \in \mathscr{D}$, then $\pi(s_2|_p) \in \mathscr{T}(\mathscr{F})$ for all subterm $s_2|_p$ with $\mathsf{root}(s_2|_p) \in \mathscr{D}$.

We choose $s_1 = \mathsf{h}(x_1,\ldots,x_{j-1},\mathsf{f}(u_1,\ldots,u_n),x_{j+1},\ldots,x_m)$ such that $u_i \in \mathscr{T}(\mathscr{F})$ if $i \in \pi(\mathsf{f})$ and $u_i \in \mathscr{V}$ if $i \notin \pi(\mathsf{f})$; we also consider that both $x_1,\ldots,x_{j-1},x_{j+1},\ldots,x_m$ and the variable arguments of f are fresh. Trivially, for all subterms s' of s_1 rooted by a defined function, we have $\pi(s') \in \mathscr{T}(\mathscr{F})$ by construction (since $\pi(\mathsf{h}) = \{j\}$). Consider now the following narrowing step:

$$s_1 = s_1[\mathsf{f}(u_1,\ldots,u_n)]_j \leadsto_{j,l\to r,\sigma} s_1[r\theta]_j = s_2$$

with $\mathsf{f}(u_1,\ldots,u_n)\sigma = l\theta$ for some substitution θ. Observe that σ needs not be applied to $s_1[r\theta]_j$ since the variables of s_1 are fresh and, thus, they only occur once.

Since $\mathscr{V}\mathsf{ar}(\pi(r)) \not\subseteq \mathscr{V}\mathsf{ar}(\pi(l))$, there exists a variable $x \in \mathscr{V}\mathsf{ar}(\pi(r))$ such that $x \notin \mathscr{V}\mathsf{ar}(\pi(l))$. Assume that x occurs at position p of r, i.e., $r|_p = x$. Now, we consider two possibilities:

- If $r|_p\theta$ is not ground, we also have that $\pi(s_2) = \mathsf{h}(\pi(r\theta))$ is not ground, which contradicts the second condition of Lemma 9.
- If $r|_p\theta$ is ground, then x is bound to a ground term by θ. Assume that $x \mapsto t_x \in \theta$ and $t_x \in \mathscr{T}(\mathscr{F})$. Since $\mathscr{R}$ does not contain extra variables, x must occur in one of the arguments of $\mathsf{f}(l_1,\ldots,l_n)$. Assume that $x \in \mathscr{V}\mathsf{ar}(l_k)$ with $k \in \{1,\ldots,n\}$. Since $x \notin \mathscr{V}\mathsf{ar}(\pi(l))$, we have $k \notin \pi(\mathsf{f})$ and, thus, u_k is a variable, which contradicts the fact that t_x is ground. □

Therefore, non-trivial argument filterings fulfilling the conditions of Lemma 9 respect the variable condition. Now, we proceed with the proof of Lemma 9 above:

Proof The first condition of safe argument filtering is trivially implied by the conditions of Lemma 9. Regarding the second condition, it is an immediate consequence of Lemma 10. Finally, we proceed to check the third condition of safe argument filtering. By Lemma 10, we have that $\pi(\mathscr{R}) = \{\pi(l) \to \pi(r) \mid l \to r \in \mathscr{R}\}$ since $\mathscr{V}\mathsf{ar}(\pi(r)) \subseteq \mathscr{V}\mathsf{ar}(\pi(l))$ for all $l \to r \in \mathscr{R}$. Moreover, trivially, we have that $\mathscr{V}\mathsf{ar}(\pi(r)) \subseteq \mathscr{V}\mathsf{ar}(\pi(l))$ for all $l \to r \in \mathscr{R}_{\mathsf{gen}}$ since the rules for the data generators contain no variables in their right-hand sides. Therefore, we have that $s \to^*_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} s'$ implies $\pi(s) \to^*_{>\varepsilon,\pi(\mathscr{R}_{\mathsf{gen}})} \pi(s')$ (see, e.g., Lemma 3 in [2]) for all $s \in \mathit{calls}_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$ with $t \in \gamma(t^\alpha)$. Now, by Lemma 5, we have that s do not contain occurrences of gen and, thus, we have $\pi(s) \to^*_{>\varepsilon,\pi(\mathscr{R})} \pi(s')$, which concludes the proof. □

Our first sufficient condition is similar to the one required in [60]. Unfortunately, it is often too strong in our context:

Example 18 Consider the TRS of Example 9, the abstract term $t^\alpha = \mathsf{append}(g,v)$, and the argument filtering $\pi = \{\mathsf{append} \mapsto \{1\}, \mathsf{reverse} \mapsto \{1\}, \mathsf{nil} \mapsto \varnothing, \mathsf{cons} \mapsto \{1,2\}\}$ of Example 14. Although π is safe for t^α, it does not fulfill the sufficient condition of Lemma 9, since there is a narrowing step like

$$\mathsf{append}(\mathsf{append}(\mathsf{nil},x_1),x_2) \leadsto_{1,\mathsf{append}(\mathsf{nil},y)\to y,id} \mathsf{append}(x_1,x_2)$$

where both

$$\pi(\mathsf{append}(\mathsf{append}(\mathsf{nil},x_1),x_2)) = \mathsf{append}(\mathsf{append}(\mathsf{nil}))$$

and

$$\pi(\mathsf{append}(\mathsf{nil},x_1),x_2)) = \mathsf{append}(\mathsf{nil})$$

are ground, but $\pi(\mathsf{append}(x_1,x_2)) = \mathsf{append}(x_1)$ is not ground.

Alternatively, there is a rule $\mathsf{append}(\mathsf{nil},y) \to y \in \mathscr{R}$ such that $\mathscr{V}\mathsf{ar}(\pi(y)) = \{y\} \not\subseteq \varnothing = \mathscr{V}\mathsf{ar}(\mathsf{app}(\mathsf{nil})) = \mathscr{V}\mathsf{ar}(\pi(\mathsf{append}(\mathsf{nil},y)))$, i.e., the variable condition is not satisfied.

Now, we introduce a weaker sufficient condition for argument filterings that still guarantees its safeness. For this purpose, we first introduce the notion of graph of functional dependencies:

Definition 15 (graph of functional dependencies) Given a TRS $\mathscr{R}$ over $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$, its graph of functional dependencies, in symbols $\mathscr{G}(\mathscr{R})$, contains nodes labeled with the defined symbols in $\mathscr{D}$ and there is an arrow from node f to node g iff there is a subterm rooted by g in the right-hand side of a rule whose left-hand side is rooted by f.

We say that g is *reachable* from f in $\mathscr{R}$ if there is a path from f to g in $\mathscr{G}(\mathscr{R})$.

In the following, we denote by $\mathit{inner}(t)$ the set of defined symbols of t that occur nested inside some other defined symbols, i.e.,

$$\mathit{inner}(t) = \{\ \mathsf{f} \mid \mathsf{root}(t|_p) \in \mathscr{D} \text{ and } \mathsf{root}(t|_{p.q}) = \mathsf{f} \in \mathscr{D} \text{ with } q \neq \varepsilon\ \}$$

Our second sufficient condition is then formalized as follows:

Lemma 11 *Let $\mathscr{R}$ be a TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and let t^α be an abstract term. Let π be an argument filtering π that fulfills the following conditions:*

(1) *if $\pi(t^\alpha) = \mathsf{f}(m_1,\ldots,m_n)$, then $m_i = g$, for all $i = 1,\ldots,n$;*
(2) *$\mathscr{V}\mathsf{ar}(\pi(t)) \subseteq \mathscr{V}\mathsf{ar}(\pi(s))$ for all $s \to t \in DP(\mathscr{R})$;*
(3) *for all rules $l \to r, l' \to r' \in \mathscr{R}$ such that $\mathsf{root}(l)$ is reachable from $\mathsf{root}(t^\alpha)$ and $\mathsf{root}(l')$ is reachable from some defined symbol in $\mathit{inner}(\pi(r))$, we have $\mathscr{V}\mathsf{ar}(\pi(r')) \subseteq \mathscr{V}\mathsf{ar}(\pi(l'))$.*

Then, π is safe for t^α in $\mathscr{R}$.

Proof The first two conditions of the claim are the same as those in the definition of safe argument filtering. Hence we focus only on the third condition, i.e., we prove that $s \to^*_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} s'$ implies $\pi(s) \to^*_{>\varepsilon,\pi(\mathscr{R})} \pi(s')$ for all $s \in \mathit{calls}_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$ with $t \in \gamma(t^\alpha)$.

First, if $s \in \mathit{calls}_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$ with $t \in \gamma(t^\alpha)$, then $\mathsf{root}(s)$ is reachable from $\mathsf{root}(t^\alpha)$. Therefore, by condition (3) we have that, for every step in $s \to^*_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} s'$ either the rewrite rule $l' \to r' \in \mathscr{R}$ used at this step fulfills the condition $\mathscr{V}\mathsf{ar}(\pi(r')) \subseteq \mathscr{V}\mathsf{ar}(\pi(l))$ or this step cannot occur in $\pi(s) \to^*_{>\varepsilon,\pi(\mathscr{R})} \pi(s')$ (by Lemma 3 in [2]), and the claim follows. □

Intuitively speaking, the third condition in Lemma 11 implies the third condition in the definition of safe argument filtering since, for all applicable rules $l \to r$ in the subderivations $s \to^*_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} s'$, we have that the filtered rules $\pi(l) \to \pi(r)$ contain no extra variables and, thus, the corresponding rules in $\pi(\mathscr{R})$ contain no occurrences of $\bot$. Then, one can lift this subderivation to $\pi(\mathscr{R})$ so that $\pi(s) \to^*_{>\varepsilon,\pi(\mathscr{R})} \pi(s')$ holds.

Example 19 Consider again the TRS of Example 9, the abstract term $t^\alpha = \mathsf{append}(g,v)$, and the argument filtering $\pi = \{\mathsf{append} \mapsto \{1\}, \mathsf{reverse} \mapsto \{1\}, \mathsf{nil} \mapsto \varnothing, \mathsf{cons} \mapsto \{1,2\}\}$ of Example 14. Here, one can easily check that π fulfills the conditions of Lemma 11 since only append is reachable from append and the set $\mathit{inner}(r)$ is empty for the right-hand sides of the two rules of append, which means that the third condition of Lemma 11 holds by vacuity.

5.2 An Algorithm for Computing Safe Argument Filterings

In this section, we focus on defining an automatic algorithm to construct a safe argument filtering. For this purpose, we consider the sufficient condition of Lemma 11 in the previous section. Basically, our algorithm proceeds as follows:

1. The input of the algorithm is a left-linear constructor TRS $\mathscr{R}$, together with an abstract term t^α that specifies a (possibly infinite) set of terms.
2. Then, we adapt a simple *binding-time analysis* (BTA) [44], which is often used in partial evaluation to propagate static (i.e., ground) and dynamic (i.e., possibly non-ground) values through a program, in order to compute an argument filtering that fulfills the first two conditions of Lemma 11 (that, in turn, come from Definition 12 of safe argument filtering).
3. Finally, we construct a graph of functional dependencies (cf. Definition 15) and check the third condition of Lemma 11. If it holds, the the computed argument filtering is safe; otherwise, no safe argument filtering is produced and, thus, the termination of narrowing cannot be proved.[13]

Now, we introduce the simplified BTA that is used in step (2) above to propagate the static/dynamic values of the abstract term through all the functions in the TRS. Observe that we use g (ground) and v (possibly variable) as binding-times, rather than the more traditional S (static) and D (dynamic). Note also that the BTA only considers defined functions; in our approach, no argument of a constructor symbol is filtered.

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 $\mathsf{f}/n \in \mathscr{D}$, 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.,

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

[13] Of course, there is a loss of precision because we are using a sufficient condition for the safeness of argument filterings. Nevertheless, there are cases where no safe argument filtering exists. Consider, e.g., the TRS $\mathscr{R} = \{\mathsf{b}(x) \to \mathsf{f}(\mathsf{g}(\mathsf{a},\mathsf{c}(x))),\ \mathsf{f}(\mathsf{c}(x)) \to x,\ \mathsf{g}(x,y) \to y\}$. Given the abstract term $\mathsf{b}(v)$, there exists no safe argument filtering for $\mathsf{b}(v)$ in $\mathscr{R}$.
Intuitively speaking, the problem comes from the fact that the right-hand sides of the second and third rules are variables that cannot be filtered away. Removing this kind of extra variables is an interesting open problem for further research that is left out of the scope of this paper.

Following [44], our binding-time analysis includes two auxiliary functions, B_v and B_e. Function B_v takes a term t (usually the right-hand side of some rule), a defined function symbol f/n, a binding-time environment ρ for the variables of the considered rule and a division div, and returns a sequence of n binding-times for function f/n (roughly speaking, they denote the least upper bound of the binding-times of the calls to f/n that occur in t). Function B_e takes a term t, a binding-time environment ρ and a division div, and returns a binding-time for the given term (roughly speaking, it returns v if $x\rho = v$ for some variable x of t and g otherwise).

Functions B_v and B_e are defined in our context as follows:

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

$$
\begin{array}{lll}
B_e[[x]]\ \rho\ div & = x\rho & (\text{if } x \in \mathscr{V}) \\
B_e[[\mathsf{c}(t_1,\ldots,t_k)]]\ \rho\ div & = B_e[[t_1]]\ \rho\ div \sqcup \ldots \sqcup B_e[[t_k]]\ \rho\ div & (\text{if } \mathsf{c} \in \mathscr{C}) \\
B_e[[\mathsf{f}(t_1,\ldots,t_k)]]\ \rho\ div & = B_e[[t_{i_1}]]\ \rho\ div \sqcup \ldots \sqcup B_e[[t_{i_k}]]\ \rho\ div & (\text{if } \mathsf{f} \in \mathscr{D} \text{ and} \\
 & & \pi_{div}(f) = \{i_1,\ldots,i_k\})
\end{array}
$$

Here, we denote by π_{div} the argument filtering obtained by filtering away the positions of non-ground arguments in div, i.e., if the considered division is

$$div = \{\mathsf{f}_1 \mapsto (m^1_1,\ldots,m^1_{n_1}),\ldots,\mathsf{f}_k \mapsto (m^k_1,\ldots,m^k_{n_k})\}$$

then we have

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

Roughly speaking, an expression $(B_v[[t]]\ \mathsf{h}/n\ \rho\ div)$ returns a sequence of n binding-times that denote the (least upper bound of the) binding-times of the arguments (not filtered by π_{div}) of the calls to h/n that occur in t in the context of the binding-time environment ρ. An expression $(B_e[[t]]\ \rho\ div)$ then returns g if the filtered term $\pi_{div}(t)$ contains no variable which is bound to v in ρ, and v otherwise.

The only difference w.r.t. the original definitions of B_v and B_e is that we take into account the current division div so that those function arguments that would be filtered away by the argument filtering π_{div} associated to div are not considered in B_e.

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

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

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

$$e(div, \mathsf{f}(t_1, \ldots, t_n)) = \{x \mapsto m_1 \mid x \in \mathscr{V}\mathsf{ar}(t_1)\} \cup \ldots \cup \{x \mapsto m_n \mid x \in \mathscr{V}\mathsf{ar}(t_n)\}$$
$$\text{where } div(\mathsf{f}) = (m_1, \ldots, m_n)$$

Once we get a fixpoint,[14] i.e., $div_{i+1} = div_i$ for some $i \geq 0$, the corresponding argument filtering π_{div_i}, augmented with $\pi(c/k) = \{1, \ldots, k_c\}$ for all constructor symbol c/k, is returned as a result. This argument filtering fulfills the first two conditions of Lemma 11 since the computed division div_i is *congruent* [44] (i.e., a function argument is classified as g only when every call to this function has a ground term in this argument position). The following result states this result:

Lemma 12 *Let $\mathscr{R}$ be a left-linear constructor TRS and let $t^\alpha = \mathsf{f}(m_1, \ldots, m_n)$ be an abstract term. Let π_{div} be the argument filtering associated to the division div computed by the BTA shown above. Then, the following conditions hold:*

(1) *$m_i = g$ for all $i \in \pi_{div}(\mathsf{f})$;*
(2) *$\mathscr{V}\mathsf{ar}(\pi_{div}(t)) \subseteq \mathscr{V}\mathsf{ar}(\pi_{div}(s))$ for all dependency pairs $s \to t \in DP(\mathscr{R})$.*

Proof Condition (1) holds trivially by construction since the computation of a new division is monotone.

Now, we prove condition (2) by contradiction. Assume that there is some dependency pair $s \to t \in DP(\mathscr{R})$ and a variable $x \in \mathscr{V}\mathsf{ar}(\pi_{div}(t))$ such that $x \notin \mathscr{V}\mathsf{ar}(\pi_{div}(s))$. Let us consider $s = \mathsf{g}(s_1, \ldots, s_m)$ and $t = \mathsf{h}(t_1, \ldots, t_l)$. Since $x \in \mathscr{V}\mathsf{ar}(\pi_{div}(t))$, we have that $div(\mathsf{h}) = (b_1, \ldots, b_l)$ with $b_i = g$ for some i such that $x \in \mathscr{V}\mathsf{ar}(t_i)$. Also, since $x \notin \mathscr{V}\mathsf{ar}(\pi_{div}(s))$, we have that $div(\mathsf{g}) = (b'_1, \ldots, b'_m)$ with $b'_j = v$ for some j such that $x \in \mathscr{V}\mathsf{ar}(s_j)$. However, this means that the computed binding-time environment for this rule would contain the mapping $x \mapsto v$ which contradicts the fact that $b_i = g$ according to the definition of functions B_v and B_e. □

Example 20 Consider the following TRS $\mathscr{R}$ defining the addition and multiplication of natural numbers:

$$\begin{array}{rcl@{\qquad}rcl} \mathsf{mult}(\mathsf{z}, y) & \to & \mathsf{z} & \mathsf{add}(\mathsf{z}, y) & \to & y \\ \mathsf{mult}(\mathsf{s}(x), y) & \to & \mathsf{add}(\mathsf{mult}(x, y), y) & \mathsf{add}(\mathsf{s}(x), y) & \to & \mathsf{s}(\mathsf{add}(x, y)) \end{array}$$

Given the abstract term $\mathsf{mult}(g, v)$, the associated initial division is

$$div_0 = \{\mathsf{mult} \mapsto (g, v),\ \mathsf{add} \mapsto (g, g)\}$$

[14] It is well-known in partial evaluation that this fixpoint computation always terminates since the domain of possible divisions is finite and the computation of a new division is monotone (i.e., some 'v' arguments may be replaced by 'g' but not vice versa).

The next division, div_1, is obtained from the following expression:

$$\begin{array}{rl} div_1 = \{\ \mathsf{mult} \mapsto (g,v) & \sqcup\ B_v[[\mathsf{z}]]\ \mathsf{mult}/2\ \{y \mapsto v\}\ div_0 \\ & \sqcup\ B_v[[\mathsf{add}(\mathsf{mult}(x,y),y)]]\ \mathsf{mult}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0 \\ & \sqcup\ B_v[[y]]\ \mathsf{mult}/2\ \{y \mapsto g\}\ div_0 \\ & \sqcup\ B_v[[\mathsf{s}(\mathsf{add}(x,y))]]\ \mathsf{mult}/2\ \{x \mapsto g,\ y \mapsto g\}\ div_0, \\ \mathsf{add} \mapsto (g,g) & \sqcup\ B_v[[\mathsf{z}]]\ \mathsf{add}/2\ \{y \mapsto v\}\ div_0 \\ & \sqcup\ B_v[[\mathsf{add}(\mathsf{mult}(x,y),y)]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0 \\ & \sqcup\ B_v[[y]]\ \mathsf{add}/2\ \{y \mapsto g\}\ div_0 \\ & \sqcup\ B_v[[\mathsf{s}(\mathsf{add}(x,y))]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto g\}\ div_0 \qquad \} \end{array}$$

Therefore, by evaluating the calls to B_v, we get

$$div_1 = \{\mathsf{mult} \mapsto (g,v),\ \mathsf{add} \mapsto (g,v)\}$$

Note that the change in the binding-times of add comes from the evaluation of

$$B_v[[\mathsf{add}(\mathsf{mult}(x,y),y)]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0$$

which proceeds as follows (the reduced subexpression is underlined):

$$\begin{array}{ll} & \underline{B_v[[\mathsf{add}(\mathsf{mult}(x,y),y)]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0} \\ = & \underline{B_v[[\mathsf{mult}(x,y)]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0} \sqcup B_v[[y]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0 \\ & \sqcup\ (B_e[[\mathsf{mult}(x,y)]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0, \underline{B_e[[y]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0}) \\ = & \underline{B_v[[x]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0} \sqcup B_v[[y]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0 \\ & \sqcup\ B_v[[y]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0 \\ & \sqcup\ (B_e[[\mathsf{mult}(x,y)]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0, B_e[[y]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0) \\ = & (g,g) \sqcup \underline{B_v[[y]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0} \sqcup B_v[[y]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0 \\ & \sqcup\ (B_e[[\mathsf{mult}(x,y)]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0, \underline{B_e[[y]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0}) \\ = & (g,g) \sqcup (g,g) \sqcup \underline{B_v[[y]]\ \mathsf{add}/2\ \{x \mapsto g,\ y \mapsto v\}\ div_0} \\ & \sqcup\ (B_e[[\mathsf{mult}(x,y)]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0, \underline{B_e[[y]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0}) \\ = & (g,g) \sqcup (g,g) \sqcup (g,g) \\ & \sqcup\ (\underline{B_e[[\mathsf{mult}(x,y)]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0}, B_e[[y]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0) \\ = & (g,g) \sqcup (g,g) \sqcup (g,g) \qquad (\text{since } \pi_{div_0}(\mathsf{mult}) = \{1\}) \\ & \sqcup\ (\underline{B_e[[x]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0}, B_e[[y]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0) \\ = & (g,g) \sqcup (g,g) \sqcup (g,g) \sqcup (g, \underline{B_e[[y]]\ \{x \mapsto g,\ y \mapsto v\}\ div_0}) \\ = & \underline{(g,g) \sqcup (g,g) \sqcup (g,g) \sqcup (g,v)} \\ = & (g,v) \end{array}$$

If we compute div_2 we get the same result and, thus, div_1 is already a fixpoint. From this division, the associated argument filtering is

$$\pi_{div_1} = \{\ \mathsf{mult} \mapsto \{1\},\ \mathsf{add} \mapsto \{1\}, \mathsf{z} \mapsto \varnothing,\ \mathsf{s} \mapsto \{1\}\ \}$$

6 Practical Applicability and Extensions

In this section, we discuss the practical applicability of our termination analysis and propose a number of useful extensions that allow us to get more accurate results.

6.1 Applications

There are several interesting applications of our termination analysis. As mentioned in the introduction, it can be particularly useful as a basis to prove the termination of so-called functional logic programs. Other applications include partial evaluation, termination of functional programs, etc.

6.1.1 Termination of Functional Logic Programs

Functional logic languages combine the operational principles of the most important declarative programming paradigms, namely functional and logic programming [38]. The operational semantics of such languages is usually based on narrowing. The essential component of modern functional logic languages like Curry [27] or TOY [50] is an *inductively sequential* rewrite system [5], a subclass of the left-linear constructor systems that we consider in this work. These languages also include some additional features like higher-order functions, variable sharing, local declarations, (concurrent) constraints, etc., but most of these features are translated into a simpler core syntax called the *flat* representation of a program, which basically amounts to a first-order left-linear constructor system. Therefore, our results in this paper are straightforwardly applicable to analyze the termination of functional logic programs without constraints nor extra variables.

Essentially, the main difference between functional logic programming and a term rewriting is not the syntax of programs but the associated evaluation mechanism. While only ground terms are evaluated in a term rewriting setting, terms containing variables can also be non-deterministically evaluated in a functional logic setting, similarly to a logic programming language based on SLD resolution. Consider, for instance, the following functional logic program to perform pattern matching in strings:

```
data Letter = A | B

match :: [Letter] -> [Letter] -> Bool
match p s = loop p s p s

loop :: [Letter] -> [Letter] -> [Letter] -> [Letter] -> Bool
loop []     _      _  _  = True
loop (_:_)  []     _  _  = False
loop (p:ps) (s:ss) op os = if (eq p s) then loop ps ss op os
                                       else next op os
next :: [Letter] -> [Letter] -> Bool
next _  [] = False
next op (_:ss) = loop op ss op ss
```

```
(VAR p ps s ss op os x y)
(RULES

 match(p,s) -> loop(p,s,p,s)

 loop(nil,s,op,os) -> true
 loop(cons(p,ps),nil,op,os) -> false
 loop(cons(p,ps),cons(s,ss),op,os)
       -> ite(eq(p,s),loop(ps,ss,op,os),next(op,os))

 next(op,nil) -> false
 next(op,cons(s,ss)) -> loop(op,ss,op,ss)

 ite(true,x,y) -> x
 ite(false,x,y) -> y

 eq(a,a) -> true
 eq(b,b) -> true
 eq(a,b) -> false
 eq(b,a) -> false
)
```

Fig. 1 A TRS denoting the string pattern matcher

```
eq :: Letter -> Letter -> Bool
eq A A = True
eq B B = True
eq A B = False
eq B A = False
```

Here, every function definition is preceded by a type declaration (using Haskell's syntax), the underscore "_" denotes an anonymous variable, i.e., a variable that appears only once in the rule, `Bool` is the predefined type containing the Boolean constants `True` and `False`, and the conditional sentence `if_then_else` is implicitly defined by the rules

```
if :: Bool -> a -> a -> a
if  True then x else y = x
if False then x else y = y
```

where `a` is a *type variable*. For simplicity, we consider that strings can only contain letters `A` or `B`.

This program can easily be translated into a left-linear constructor TRS. It is shown in Fig. 1, where we consider the TRS format of the *Termination Problem Data Base* (TPDB), see `http://www.lri.fr/~marche/tpdb/`.

Now, we assume that the programmer is interested in proving that all calls to `match` with a known string and a partially unknown pattern (the case, e.g., when only a prefix of the pattern is known) terminate. We can specify these calls using the abstract term $\mathsf{match}(v,g)$. Then, by using the techniques of Section 5, we compute

```
(VAR p ps s ss op os x y)
(RULES

 match(s) -> loop(s,s)

 loop(s,os) -> true
 loop(nil,os) -> false
 loop(cons(s,ss),os) -> ite(eq(s),loop(ss,os))
 loop(cons(s,ss),os) -> next(os)

 next(nil) -> false
 next(cons(s,ss)) -> loop(ss,ss)

 ite(true,x) -> x
 ite(false,x) -> nullVar

 eq(a) -> true
 eq(b) -> true
 eq(b) -> false
 eq(a) -> false
)
```

Fig. 2 The output of the AFT transformation

the following safe argument filtering:

$$\pi = \{\ \mathsf{match} \mapsto \{2\},\ \mathsf{loop} \mapsto \{2,4\},\ \mathsf{next} \mapsto \{2\},\ \mathsf{ite} \mapsto \{1,2\},\ \mathsf{eq} \mapsto \{2\},\ \mathsf{nil} \mapsto \varnothing,\ \mathsf{cons} \mapsto \{1,2\}\ \}$$

The AFT transformation for the TRS of Fig. 1 using the argument filtering π above is shown in Fig. 2, where `nullVar` is a special constant that denotes $\bot$. The termination of this TRS can easily be proved using standard techniques for proving the termination of rewriting (e.g., it can be proved terminating using the AProVE tool [32]). Of course, its termination could also be proved using the direct approach of Section 4.2.

Clearly, there is some loss of precision in our analysis since it ignores the following language features:

- First, programs are evaluated using a *lazy* variant of narrowing called needed narrowing [8]. As a consequence, a program might terminate even if there exist non-terminating derivations that are not lazy.
- Almost all implementations consider some form of variable *sharing* to avoid recomputing the same expression.

Furthermore, there are some language features that we have not considered yet, like (concurrent) constraints or extra variables in the original programs. These are interesting extensions that are the subject of ongoing work.

6.1.2 Other Applications

Our results can also be used within the *narrowing-driven* partial evaluation scheme [1] for functional and functional logic programs in order to infer the right annotations for function calls. The most recent approach to narrowing-driven specialization

is introduced in [4], where *needed narrowing* [8] is used to perform symbolic computations at partial evaluation time.

In particular, within the so-called *offline* approach to partial evaluation, one should first annotate every function call of the source program (namely, a left-linear constructor TRS in [4]) with either unfold (i.e., all calls to this function terminate for the considered terms) and memo (i.e., the calls to this function might not terminate). In this context, our termination analysis is clearly useful to infer the right annotations for the partial evaluation process.

Another related application is concerned with the termination analysis of Haskell programs introduced in [33]. In this work, a restricted form of partial evaluation based on a narrowing-like mechanism is used to encode the lazy strategy of Haskell. Then, dependency pairs are extracted from the partial computations rather than from the rules of the original program. Here, our termination analysis could be used to drive the construction of the partial computations so that the search space is kept finite while avoiding too much generalization.

6.2 Extensions

In this section, we introduce several extensions that may improve the accuracy of our termination analysis.

6.2.1 Multiple Abstract Terms

Let us now consider that the user wants to check termination w.r.t. a *set* of abstract terms. In order to illustrate this issue, consider, e.g., the following TRS defining the equality on natural numbers:

$$\begin{array}{rcl} \mathsf{eq}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}(x,y) \end{array}$$

and the set of abstract terms $T^\alpha = \{\mathsf{eq}(g,v),\ \mathsf{eq}(v,g)\}$. In this case, we could straightforwardly extend the method in Section 5.2 so that the initial division maps eq to the least upper bound of the arguments of all abstract terms rooted by eq:

$$div_0 = \{\ \mathsf{eq} \mapsto (g,v) \sqcup (v,g)\ \} = \{\ \mathsf{eq} \mapsto (v,v)\ \}$$

Unfortunately, despite narrowing terminates for every term in $\gamma(T^\alpha)$,[15] our method would infer the safe argument filtering $\pi = \{\mathsf{eq} \mapsto \varnothing,\ \mathsf{z} \mapsto \varnothing,\ \mathsf{s} \mapsto \{1\}\}$ and, therefore, termination could not be proved.

This problem, though, can easily be solved. Basically, the idea is to consider the termination of every abstract term separately. The following trivial lemma justifies this approach:

Lemma 13 *Let $\mathscr{R}$ be a TRS and T^α be a finite set of abstract terms. $\gamma(T^\alpha)$ is $\leadsto_{\mathscr{R}}$-terminating iff $\gamma(t^\alpha)$ is $\leadsto_{\mathscr{R}}$-terminating for all $t^\alpha \in T^\alpha$.*

[15] We assume that the concretization function γ is extended over sets of terms in the natural way: $\gamma(T^\alpha) = \{\gamma(t^\alpha) \mid t^\alpha \in T^\alpha\}$.

In this way, we can prove termination of T^{α} above by proving the termination for $\mathsf{eq}(g,v)$ and $\mathsf{eq}(v,g)$ separately (using the inferred safe argument filterings $\pi_{gv} = \{\mathsf{eq} \mapsto \{1\},\ \mathsf{z} \mapsto \varnothing,\ \mathsf{s} \mapsto \{1\}\}$ and $\pi_{vg} = \{\mathsf{eq} \mapsto \{2\},\ \mathsf{z} \mapsto \varnothing,\ \mathsf{s} \mapsto \{1\}\}$, respectively.

6.2.2 Non Well-Moded TRSs

Even if we prove termination for every abstract term separately, we may still find some problems. Consider, e.g., the following variation of function eq above:

$$\begin{array}{rcl} \mathsf{eq}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}(y,x) \end{array}$$

Observe that the only difference is that the arguments of eq are switched in the right-hand side of the second rule. In this case, if we consider the abstract term $\mathsf{eq}(g,v)$, the only safe argument filtering has $\pi(\mathsf{eq}) = \varnothing$ (and, in fact, this is the case with the algorithm of Section 5.2). Now, the problem comes from the fact that the program is not "well-moded" (using terminology from logic programming; see, e.g., [22]).[16] In order to overcome this drawback, we proceed as follows.

First, we duplicate the *problematic* rules (i.e., the rules that define functions which are used with multiple modes) by introducing different labels denoting possible *g*/*v* combinations:

$$\begin{array}{rcl} \mathsf{eq}_{gg}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{gg}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}(y,x) \\ \mathsf{eq}_{vg}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{vg}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}(y,x) \end{array} \qquad \begin{array}{rcl} \mathsf{eq}_{gv}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{gv}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}(y,x) \\ \mathsf{eq}_{vv}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{vv}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}(y,x) \end{array}$$

Now, we replace every call in the right-hand sides of the rules by the appropriate call to the new labeled function (according to the instantiation degree of its arguments):

$$\begin{array}{rcl} \mathsf{eq}_{gg}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{gg}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}_{gg}(y,x) \\ \mathsf{eq}_{vg}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{vg}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}_{gv}(y,x) \end{array} \qquad \begin{array}{rcl} \mathsf{eq}_{gv}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{gv}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}_{vg}(y,x) \\ \mathsf{eq}_{vv}(\mathsf{z},\mathsf{z}) & \to & \mathsf{true} \\ \mathsf{eq}_{vv}(\mathsf{s}(x),\mathsf{s}(y)) & \to & \mathsf{eq}_{vv}(y,x) \end{array}$$

Note that this program returns the same values as the original one. The only difference is that every function is now well-moded.

Now, given the abstract term $\mathsf{eq}_{gv}(g,v)$, our algorithm infers the right safe argument filtering where $\pi(\mathsf{eq}_{gv}) = \{1\}$ and $\pi(\mathsf{eq}_{vg}) = \{2\}$. A similar approach has recently been introduced in [60] in order to improve the termination analysis of logic programs.

[16] Roughly speaking, a logic program is well-moded when the *input* positions of predicate calls are always filled by ground terms when considering the leftmost selection rule.

6.2.3 Removing Non-Reachable Functions

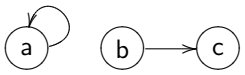
Another source of improvement for the transformation of Section 4.3 comes from the removal of those functions which are not *reachable* from the considered abstract term. Let us consider, for instance, the following simple TRS:

$$\begin{array}{lcl} \mathsf{a} & \to & \mathsf{a} \\ \mathsf{b} & \to & \mathsf{c} \\ \mathsf{c} & \to & \mathsf{d} \end{array}$$

Here, although narrowing clearly terminates for the abstract term b, we fail to prove termination because of the non-terminating function a.

In this case, we can improve the termination analysis by first removing the rules defining those function symbols that are not reachable from the function symbol of the abstract term. For this purpose, we can use the graph of functional dependencies introduced in Definition 15.

For instance, for the example above, we get the following graph:



Therefore, we can safely skip the rule defining the function a and, thus, the termination of narrowing for the abstract term b can easily be proved.

In general, we could consider more refined graphs of functional dependencies like those in [3] and [11].

6.3 The Termination Tool TNT

Now, we describe the implementation of the AFT transformation of Section 4.3, including the algorithm of Section 5.2 for the inference of safe argument filterings. The termination tool, called TNT, is publicly available from

`http://german.dsic.upv.es/filtering.html`

The tool is written in Prolog and SML/NJ and includes a parser for rewrite systems that accepts the TRS format of the Termination Problem Data Base. It also includes the pre-processing transformation of Sections 6.2.2 and 6.2.3. The tool is available through a web interface:

- The user can either write down a (left-linear constructor) TRS or choose it from a selection of TRSs (which can then be freely modified).
- The user also provides an abstract term t^α so that the termination of narrowing is analyzed w.r.t. the set of initial terms $\gamma(t^\alpha)$.
- Finally, the user can also select whether labeling of functions (cf. Section 6.2.2) should be applied and/or elimination of non-reachable functions (cf. Section 6.2.3).

The tool returns a transformed TRS $\mathscr{R}'$ whose termination w.r.t. standard rewriting implies the termination of narrowing for $\gamma(t^\alpha)$ in the original TRS $\mathscr{R}$. The termination of $\mathscr{R}'$ can then 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 [32].

We do not provide run times for the AFT transformation since it is rather simple (a few milliseconds for the considered TRSs). Actually, the most expensive part of the process lies generally in the application of the AProVE tool over the transformed TRSs. We note that the fact that AProVE demonstrates the infiniteness of the transformed TRS does not imply the non-termination of narrowing since the converse of Theorem 3 does not always hold.

7 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 [26] defined a narrowing procedure that incorporates pruning of some unsatisfiable goals. Similar approaches have been presented by Chabin and Réty [16], where narrowing is directed by a graph of terms, and by Alpuente *et al* [3], where the notion of *loop-check* is introduced to detect some unsatisfiable equations. Also, Antoy and Ariola [7] 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 [17] 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 Section 3, the termination of narrowing for arbitrary terms is quite a strong property that almost no TRS fulfills.

Recent approaches include [58, 10]. 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, Sakai and Sakabe [57] adapted the dependency pair method for proving the termination of narrowing for TRSs with extra variables. Nishida and Miura [56] then extended the method with the use of a dependency graph. These methods constitute a direct adaptation of the dependency pair approach to the case of narrowing. The main results in [57, 56] consider TRSs with the TRAT property. Although the class of TRSs with the TRAT property is slightly more general than the class of left-linear constructor systems, our notion is simpler, easy to check, and still widely applicable. The main differences w.r.t. [57, 56] are the following: we have considered the termination of narrowing via the termination of rewriting rather than defining a direct termination analysis for narrowing (which greatly simplifies the formal proofs), we have presented also a transformational approach based on the argument filtering transformation, we have presented several refinements of the basic technique (cf. Sec-

tion 6), and we have implemented a tool for the termination analysis of narrowing. Furthermore, regarding the dependency pair approach, our technique and that of [57, 56] are in principle not comparable since one can find examples where our technique is not applicable (e.g., when the TRS is not a left-linear constructor TRS) and the technique of [57,56] is, and examples where our technique is able to prove termination (e.g., thanks to the removal of some extra variables which are replaced by $\bot$) and the technique of [57,56] is not.

The closest approach is perhaps that of Schneider-Kamp *et al* [60], 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 Section 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, [60] 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.[17] Actually, we allow extra variables above the defined functions of the right-hand sides of the filtered rules in some cases. 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 transformational approach to proving termination, while [60] only introduces a direct approach based on the dependency pair framework.

Alpuente, Escobar and Iborra extended the dependency pair framework to proving the termination of narrowing for arbitrary TRSs [2], adding pairs for *echoing* of narrowing into the dependency pairs. In principle, their framework is similar to ours when left-linear constructor TRSs are considered. In contrast to our approach, however, they do not specify a set of initial terms. As a consequence, their argument filterings are required to satisfy the variable condition for filtered rewrite systems, which is stronger than our notion of safe argument filtering that takes into account an abstract term specifying the possible initial terms and some extra variables are safely replaced by $\bot$.

Finally, regarding the preliminary version of this work [65], we have made a number of extensions and improvements. In particular, we now consider a more relaxed notion of safe argument filtering (the one in [65] was basically equivalent to the variable condition of [60]). Furthermore, we have introduced a number of additional results (e.g., Theorem 5, Lemma 11) that allow us to obtain more accurate results.

8 Conclusions

In this paper, we have presented novel techniques for proving the termination of narrowing in left-linear constructor systems, a widely accepted class of systems that

[17] In fact, this is only a sufficient condition in our case, as shown in Lemmas 9 and 10.

forms the (first-order) basis of many functional and functional logic programming languages. 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 required extensions to cope with realistic functional logic languages like Curry [27] and TOY [50], e.g., to consider functional logic programs including (concurrent) constraints, guarded rules, extra variables, etc. Also, it would be useful to improve the accuracy of our current technique by taking into account the particular operational semantics of these languages, i.e., needed narrowing with sharing of variables. For this purpose, we could follow the approach of [33] for analyzing the termination of Haskell programs. Finally, we have opened an alternative line of research (see [42]) based on extending the dependency pair framework to only consider derivations from a given initial set of terms, and then using the extended framework to solve relative termination problems (and, hence, narrowing termination problems).

Acknowledgements We thank the developers of the AProVE tool for allowing us to interface the TNT tool with the web interface of AProVE. We also thank Michael Codish, Manuel Hermenegildo, Nao Hirokawa, José Iborra, Francisco López-Fraguas, Juan Rodríguez-Hortalá, and Peter Schneider-Kamp for useful suggestions that helped us to improve the paper. Finally, we acknowledge the anonymous referees for their detailed comments and helpful suggestions, which have allowed us to substantially improve this article.

Part of this research was done while the second author was visiting the Sakabe/Sakai Lab at Nagoya University. Germán Vidal gratefully acknowledges their hospitality.

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, S. Escobar, and J. Iborra. Termination of Narrowing Using Dependency Pairs. In Maria Garcia de la Banda and Enrico Pontelli, editors, *Proc. of the 24th International Conference on Logic Programming (ICLP 2008)*, pages 317–331. Springer LNCS 5366, 2008.
3. M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Narrowing Approximations as an Optimization for Equational Logic Programs. In *Proc. of the 5th International Symposium on Programming Language Implementation and Logic Programming (PLILP'93)*, pages 391–409. Springer LNCS 714, 1993.
4. M. Alpuente, M. Hanus, S. Lucas, and G. Vidal. Specialization of Functional Logic Programs Based on Needed Narrowing. *Theory and Practice of Logic Programming*, 5(3):273–303, 2005.
5. S. Antoy. Definitional trees. In *Proc. of the 3rd International Conference on Algebraic and Logic Programming (ALP'92)*, pages 143–157. Springer LNCS 632, 1992.
6. S. Antoy. Optimal non-deterministic functional logic computations. In *Proc. of the International Conference on Algebraic and Logic Programming (ALP'97)*, pages 16–30. Springer LNCS 1298, 1997.
7. S. Antoy and Z. Ariola. Narrowing the Narrowing Space. In *Proc. of the 9th International Symposium on Programming Languages: Implementations, Logics, and Programs (PLILP'97)*, pages 1–15. Springer LNCS 1292, 1997.
8. S. Antoy, R. Echahed, and M. Hanus. A Needed Narrowing Strategy. *Journal of the ACM*, 47(4):776–822, 2000.
9. S. Antoy and M. Hanus. Overlapping Rules and Logic Variables in Functional Logic Programs. In *Proc. of the 22nd International Conference on Logic Programming (ICLP'06)*, pages 87–101. Springer LNCS 4079, 2006.

10. G. Arroyo, J.G. Ramos, J. Silva, and G. Vidal. Improving Offline Narrowing-Driven Partial Evaluation using Size-Change Graphs. In *Logic-Based Program Synthesis and Transformation. Revised and Selected Papers from LOPSTR'06*, pages 60–76. Springer LNCS 4407, 2007.
11. T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs. *Theoretical Computer Science*, 236(1-2):133–178, 2000.
12. T. Arts and H. Zantema. Termination of Logic Programs Using Semantic Unification. In *Proc. of the 5th International Workshop on Logic Programming Synthesis and Transformation (LOPSTR'95)*, pages 219–233. Springer LNCS 1048, 1996.
13. F. Baader and T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.
14. F. Baader and W. Snyder. Unification Theory. In *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.
15. D. Bert and R. Echahed. Design and Implementation of a Generic, Logic and Functional Programming Language. In *Proc. of the 1st European Symposium on Programming (ESOP'86)*, pages 119–132. Springer LNCS 213, 1986.
16. J. Chabin and P. Réty. Narrowing Directed by a Graph of Terms. In *Proc. of the 4th International Conference on Rewriting Techniques and Applications (RTA'91)*, pages 112–123. Springer LNCS 488, 1991.
17. J. Christian. Some Termination Criteria for Narrowing and E-narrowing. In *Proc. of the 11th International Conference on Automated Deduction (CADE-11)*, pages 582–588. Springer LNCS 607, 1992.
18. M. Codish and C. Taboch. A Semantic Basis for the Termination Analysis of Logic Programs. *Journal of Logic Programming*, 41(1):103–123, 1999.
19. H. Comon-Lundh and S. Delaune. The Finite Variant Property: How to Get Rid of Some Algebraic Properties. In *Proc. of the 16th International Conference on Term Rewriting and Applications (RTA'05)*, pages 294–307. Springer LNCS 3467, 2005.
20. J. Darlington and Y. Guo. Narrowing and Unification in Functional Programming: an Evaluation Mechanism for Absolute Set Abstraction. In *Proc. of the 3rd International Conference on Rewriting Techniques and Applications (RTA'89)*, pages 92–108. Springer LNCS 355, 1989.
21. J. de Dios-Castro and F. López-Fraguas. Extra Variables Can Be Eliminated from Functional Logic Programs. In *Proc. of the 6th Spanish Conference on Programming and Languages (PROLE'06)*, pages 3–19. ENTCS 188, 2007.
22. D. De Schreye and S. Decorte. Termination of Logic Programs: The Never-Ending Story. *Journal of Logic Programming*, 19/20:199–260, 1994.
23. S. Debray and D.S. Warren. Automatic Mode Inference for Logic Programs. *Journal of Logic Programming*, 5:207–230, 1988.
24. N. Dershowitz. Termination of Rewriting. *Journal of Symbolic Computation*, 3(1&2):69–115, 1987.
25. N. Dershowitz. Goal Solving as Operational Semantics. In *Proc. of ILPS'95*, pages 3–17. The MIT Press, Cambridge, MA, 1995.
26. N. Dershowitz and G. Sivakumar. Goal-Directed Equation Solving. In *Proc. of 7th National Conference on Artificial Intelligence*, pages 166–170. Morgan Kaufmann, 1988.
27. M. Hanus (ed.). Curry: An Integrated Functional Logic Language. Available at: `http://www.informatik.uni-kiel.de/~mh/curry/`.
28. J. Endrullis, J. Waldmann, and H. Zantema. Matrix Interpretations for Proving Termination of Term Rewriting. *J. Autom. Reasoning*, 40(2-3):195–220, 2008.
29. 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.
30. S. Escobar and J. Meseguer. Symbolic Model Checking of Infinite-State Systems Using Narrowing. In *Proc. of 18th International Conference on Rewriting Techniques and Applications (RTA'07)*, pages 153–168. Springer LNCS 4533, 2007.
31. 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.
32. J. Giesl, P. Schneider-Kamp, and R. Thiemann. AProVE 1.2: Automatic Termination Proofs in the Dependency Pair Framework. In *Proc. of the International Joint Conference on Automated Reasoning (IJCAR'06)*, pages 281–286. Springer LNCS 4130, 2006.
33. J. Giesl, S. Swiderski, P. Schneider-Kamp, and R. Thiemann. Automated Termination Analysis for Haskell: From Term Rewriting to Programming Languages. In Frank Pfenning, editor, *Proc. of the 17th International Conference on Term Rewriting and Applications (RTA 2006)*, pages 297–312. Springer LNCS 4098, 2006.

34. J. Giesl, R. Thiemann, and P. Schneider-Kamp. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In *Proc. of the 11th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'04)*, pages 301–331. Springer LNCS 3452, 2005.
35. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Mechanizing and Improving Dependency Pairs. *Journal of Automated Reasoning*, 37(3):155–203, 2006.
36. M. Hanus. Compiling Logic Programs with Equality. In *Proc. of the 2nd International Workshop on Programming Language Implementation and Logic Programming (PLILP'90)*, pages 387–401. Springer LNCS 456, 1990.
37. M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 19&20:583–628, 1994.
38. M. Hanus. Multi-paradigm Declarative Languages. In *Proc. of the International Conference on Logic Programming (ICLP 2007)*, pages 45–75. Springer LNCS 4670, 2007.
39. S. Hölldobler. *Foundations of Equational Logic Programming*. Springer LNAI 353, 1989.
40. 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.
41. J.M. Hullot. Canonical Forms and Unification. In *Proc of the 5th International Conference on Automated Deduction*, pages 318–334. Springer LNCS 87, 1980.
42. J. Iborra, N. Nishida and G. Vidal. Goal-directed and Relative Dependency Pairs for Proving the Termination of Narrowing. In Danny De Schreye, editor, *Proc. of the 19th International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR 2009)*. Springer LNCS, to appear, 2010.
43. F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In *Proc. of the 7th International Conference on Logic for Programming and Automated Reasoning (LPAR'00)*, pages 131–160. Springer LNAI 1955, 2000.
44. N.D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Englewood Cliffs, NJ, 1993.
45. J.W. Klop. Term Rewriting Systems: A Tutorial. *Bulletin of the European Association for Theoretical Computer Science*, 32:143–183, 1987.
46. A. Koprowski. TPA: Termination Proved Automatically. In Frank Pfenning, editor, *Proc. of the 17th International Conference on Term Rewriting and Applications (RTA 2006)*, pages 257–266. Springer LNCS 4098, 2006.
47. K. Kusakari, M. Nakamura, and Y. Toyama. Argument Filtering Transformation. In *Proc. of the International Conference on Principles and Practice of Declarative Programming (PPDP'99)*, pages 48–62. Springer LNCS 1702, 1999.
48. N. Lindenstrauss, Y. Sagiv, and A. Serebrenik. Proving Termination for Logic Programs by the Query-Mapping Pairs Approach. In M. Bruynooghe and K.-K. Lau, editors, *Program Development in Computational Logic*, pages 453–498. Springer LNCS 3049, 2004.
49. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987. Second edition.
50. F. López-Fraguas and J. Sánchez-Hernández. TOY: A Multiparadigm Declarative System. In *Proc. of the 10th International Conference on Rewriting Techniques and Applications (RTA'99)*, pages 244–247. Springer LNCS 1631, 1999.
51. A. Martelli and U. Montanari. An Efficient Unification Algorithm. *ACM Transactions on Programming Languages and Systems*, 4:258–282, 1982.
52. 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.
53. A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.
54. J.J. Moreno-Navarro, H. Kuchen, R. Loogen, and M. Rodriguez-Artalejo. Lazy Narrowing in a Graph Machine. In *Proc. of the 2nd International Conference on Algebraic and Logic Programming (ALP'90)*, pages 298–317. Springer LNCS 463, 1990.
55. 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.
56. N. Nishida and K. Miura. Dependency Graph Method for Proving Termination of Narrowing. In *Proc. of the 8th International Workshop on Termination (WST'06)*, pages 12–16, 2006.
57. N. Nishida, M. Sakai, and T. Sakabe. Narrowing-Based Simulation of Term Rewriting Systems with Extra Variables. *ENTCS*, 86(3), 2003.

58. J.G. Ramos, J. Silva, and G. Vidal. Fast Narrowing-Driven Partial Evaluation for Inductively Sequential Systems. In *Proc. of the 10th ACM SIGPLAN International Conference on Functional Programming (ICFP'05)*, pages 228–239. ACM Press, 2005.
59. 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.
60. P. Schneider-Kamp, J. Giesl, A. Serebrenik, and R. Thiemann. Automated Termination Proofs for Logic Programs by Term Rewriting. *ACM Transactions on Programming Languages*, 11(1):1–52, 2009.
61. 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.
62. J.R. Slagle. Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity. *Journal of the ACM*, 21(4):622–642, 1974.
63. J. Steinbach. Simplification Orderings: Histrory of Results. *Fundamenta Informaticae*, 24(1/2):47–87, 1995.
64. S. Verbaeten, K. Sagonas, and D. De Schreye. Termination Proofs for Logic Programs with Tabling. *ACM Transactions on Computational Logic*, 2(1):57–92, 2001.
65. G. Vidal. Termination of Narrowing in Left-Linear Constructor Systems. In J. Garrigue and M. Hermenegildo, editors, *Proc. of the 9th International Symposium on Functional and Logic Languages (FLOPS 2008)*, pages 113–129. Springer LNCS 4989, 2008.

A Proofs of Technical Results

In this section, we provide the proofs of those technical results that do not appear in the body of the paper.

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

Proof In order to prove this claim, we consider a rule-based unification algorithm [14] (basically, a variant of the Martelli & Montanari unification algorithm [51]). 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 Figure 3. 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.

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

(rule binding)
$$\{x = s\} \cup P;S \Longrightarrow P\{x \mapsto s\};S\{x \mapsto s\} \quad \text{if } x \notin \mathscr{V}\mathsf{ar}(s)$$

Fig. 3 Simplified unification rules

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 $P; S$. □

Lemma 3 *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F}$ and let $t \in \mathscr{T}(\mathscr{F}, \mathscr{V})$ be a term. If $\widehat{t} \to^+_{GEN(\mathscr{R})} s$ is an admissible derivation, 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 $\widehat{t} \to^{+}_{GEN(\mathscr{R})} s$.

Base case ($k = 1$). Then, $\widehat{t} \to_{GEN(\mathscr{R})} s$ and we have $t|_p = x \in \mathscr{V}$ for some position p, $(\mathsf{gen} \to \mathsf{c}(\mathsf{gen},\ldots,\mathsf{gen})) \in \mathscr{R}_{\mathsf{gen}}$, 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 $\widehat{t} \to^{+}_{GEN(\mathscr{R})} t'$ of the derivation $\widehat{t} \to^{+}_{GEN(\mathscr{R})} s$ such that in $\widehat{t} \to^{+}_{GEN(\mathscr{R})} t'$ all occurrences of gen associated to the same variable are reduced one step (thus, both $\widehat{t} \to^{+}_{GEN(\mathscr{R})} t'$ and $t' \to^{+}_{GEN(\mathscr{R})} s$ are trivially admissible). Therefore, there exists positions $p_1,\ldots,p_n$, $n > 0$, such that $t|_{p_i} = x \in \mathscr{V}$, $i = 1,\ldots,n$, $(\mathsf{gen} \to \mathsf{c}(\mathsf{gen},\ldots,\mathsf{gen})) \in \mathscr{R}_{\mathsf{gen}}$, 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 4 *Let $\mathscr{R}$ be a left-linear constructor TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and $t \in \mathscr{T}(\mathscr{F},\mathscr{V})$ a term. If $\widehat{t} \to_{p,R} s'$ with $R \in \mathscr{R}$, then $t \to_{p,R} s$ such that $\widehat{s} = s'$.*

Proof Since $\widehat{t} \to_{p,R} s'$, by the definition of a rewrite step, we have $R = (l \to r) \in \mathscr{R}$, $t|_p = l\sigma$ 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. □

Lemma 5 *Let $\mathscr{R}$ be a TRS over a signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$. Let π be an argument filtering over $\mathscr{F}$ that is extended over tuple symbols so that $\pi(f^{\sharp}) = \pi(f)$ for all $f \in \mathscr{D}$, and let t^{α} be an abstract term. If π is safe for t^{α} in $\mathscr{R}$, then $\pi(s) \in \mathscr{T}(\mathscr{F})$ for all terms $s \in calls_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$ with $t \in \gamma(t^{\alpha})$.*

Proof We prove the claim by induction on the length n of the rewrite derivation from $\widehat{t}$ to s.

Base case ($n = 0$). In this case, we have that $s = \widehat{t}$ (since t only contains a defined symbol at the root) and, thus, the claim follows straightforwardly by condition (1) of safe argument filtering.

Inductive case ($n > 0$). Let us consider some term $s \in calls_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$ such that the rewrite sequence $\widehat{t} \to^{*}_{\mathscr{R}_{\mathsf{gen}}} t'$, with s a subterm of t' rooted by a defined symbol, performs exactly n steps. W.l.o.g., we assume that the rewrite sequence has this form: $\widehat{t} \to^{*}_{\mathscr{R}_{\mathsf{gen}}} t'[l\sigma]_p \to_{p,l\to r} t'[r\sigma]_p$ for some rule $l \to r \in \mathscr{R}_{\mathsf{gen}}$, position p, and substitution σ. Let us consider that s occurs in $t'[r\sigma]_p$ at position q, i.e., $(t'[r\sigma]_p)|_q = s$. Now, we distinguish the following cases:

– Consider first that $p \leq q$, i.e., that s is a subterm of $r\sigma$. By the inductive hypothesis, we have that $\pi(l\sigma) \in \mathscr{T}(\mathscr{F})$. By condition (2) in the definition of safe argument filtering, we have that $\mathscr{V}\mathsf{ar}(\pi(s)) \subseteq \mathscr{V}\mathsf{ar}(\pi(l))$. Therefore, all the bindings of σ that can affect the variables of $\pi(s)$ come from $\pi(s')$ and, hence, $\pi(s) \in \mathscr{T}(\mathscr{F})$ and the claim follows.
– Otherwise, we have $p > q$ and thus s occurs above $r\sigma$. We prove this case by contradiction. Assume that $\pi(s)$ contains some occurrence of gen. Let $(t'[l\sigma]_p)|_q = s'$ and $p.u = q$. Since we have $\widehat{t} \to^*_{\mathscr{R}_{\mathsf{gen}}} t'[l\sigma]_p$ and s' is rooted by a defined symbol by assumptions, then $s' \in \mathit{calls}_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$. Therefore, we have a derivation of the form $s' \to_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} s'[r\sigma]_u = s$ and $\pi(s'[r\sigma]_u)$ contains some occurrence of gen. However, by condition (3) in the definition of safe argument filtering, there must be a derivation $\pi(s') \to_{>\varepsilon,\pi(\mathscr{R})} \pi(s'[r\sigma]_u)$, which is not possible since neither $\pi(s')$ nor $\pi(\mathscr{R})$ contain occurrences of gen. Hence we get a contradiction and the claim follows.

□

Lemma 6 *Let $\mathscr{R}$ be a TRS over the signature $\mathscr{F} = \mathscr{D} \uplus \mathscr{C}$ and let $\mathscr{P} \subseteq DP(\mathscr{R})$ be a set of dependency pairs. Let π be an argument filtering over $\mathscr{F}$ that is extended over tuple symbols so that $\pi(f^\sharp) = \pi(f)$ for all $f \in \mathscr{D}$ and let t^α be an abstract term. For every $(t^\alpha, \mathscr{P}, \mathscr{R}, \pi)$-chain of the form $s^\sharp_1 \to t^\sharp_1, s^\sharp_2 \to t^\sharp_2, \ldots$, there exists a rewrite derivation of the form*

$$\begin{array}{lll} \widehat{t} & \to^*_{\mathscr{R}_{\mathsf{gen}}} & t'[\widehat{s_1\sigma}]_p \to_{p,\mathscr{R}} t'[r_1\widehat{\sigma[t_1\sigma]}_{p_1}]_p \\ & \to^*_{>p.p_1,\mathscr{R}_{\mathsf{gen}}} & t'[r_1\widehat{\sigma[s_2\sigma]}_{p_1}]_p \to_{p.p_1,\mathscr{R}} t'[r_1\sigma[r_2\widehat{\sigma[t_2\sigma]}_{p_2}]_{p_1}]_p \\ & \to^*_{>p.p_1.p_2,\mathscr{R}_{\mathsf{gen}}} & \cdots \end{array}$$

for some substitution σ.

Proof By the definition of chain, there exists a substitution $\sigma : \mathscr{V} \mapsto \mathscr{T}(\mathscr{F}, \mathscr{V})$ such that the following conditions hold:

1. there exists a term $s \in \mathit{calls}_{\mathscr{R}_{\mathsf{gen}}}(\widehat{t})$ for some $t \in \gamma(t^\alpha)$ such that $s = \widehat{s_1\sigma}$ and
2. $\widehat{t_i\sigma} \to^*_{>\varepsilon,\mathscr{R}_{\mathsf{gen}}} \widehat{s_{i+1}\sigma}$ for every two consecutive pairs in the sequence for all $i > 0$.

By definition of dependency pair we have that, for all $s^\sharp_i \to t^\sharp_i \in DP(\mathscr{R})$, there exists a rule $R_i = s_i \to r_i \in \mathscr{R}$ with $r_i|_{p_i} = t_i$ a subterm of r_i rooted by a defined function symbol. Thus, by the stability of rewriting, we have $s_i\sigma \to_{\mathscr{R}} r_i\sigma[t_i\sigma]_{p_i}$ for all $i > 0$. Again by the stability of rewriting, we have $\widehat{s_i\sigma} \to_{\mathscr{R}} r_i\widehat{\sigma[t_i\sigma]}_{p_i}$ for all $i > 0$. Now, by using condition (2) above, we have:

$$\begin{array}{lll} \widehat{s_1\sigma} \to_{\varepsilon,\mathscr{R}} & r_1\widehat{\sigma[t_1\sigma]}_{p_1} & \to^*_{>p_1,\mathscr{R}_{\mathsf{gen}}} \quad r_1\widehat{\sigma[s_2\sigma]}_{p_1} \\ \to_{p_1,\mathscr{R}} & r_1\sigma[r_2\widehat{\sigma[t_2\sigma]}_{p_2}]_{p_1} & \to^*_{>p_1.p_2,\mathscr{R}_{\mathsf{gen}}} \cdots \end{array}$$

Finally, by using condition (1) above, we have that $\widehat{t} \to^*_{\mathscr{R}_{\mathsf{gen}}} t'$, with $t' = t'[s]_p$ for some position p, and that $s = \widehat{s_1\sigma}$, which completes the proof. □

Lemma 7 *Let π be an argument filtering and let r be a term. Then, for every subterm $r|_p$ of r rooted by a defined symbol, either $\pi(r|_p)$ is a subterm of $\pi(r)$ or there exists a term $r' \in \mathsf{dec}_\pi(r)$ such that $\pi(r|_p)$ is a subterm of $\pi(r')$.*

Proof We prove this claim by induction on the structure of r. Since the base case (i.e., $r|_p = r$) is trivial, we only consider the inductive case. Then, we consider that $r|_p$ is a proper subterm of r. Let us assume that $r = \mathsf{f}(t_1,\ldots,t_n)$ and that $r|_p$ is a subterm of t_i, $1 \leqslant i \leqslant n$. By the induction hypothesis, we have that either $\pi(r|_p)$ is a subterm of $\pi(t_i)$ or there exists a term $r' \in \mathsf{dec}_\pi(t_i)$ such that $\pi(r|_p)$ is a subterm of $\pi(r')$. Now, we distinguish the following cases:

- Let $i \in \pi(f)$. Here, we have that $\pi(r) = \mathsf{f}(\ldots,\pi(t_i),\ldots)$ and $\mathsf{dec}_\pi(t_i) \subseteq \mathsf{dec}_\pi(r)$ and, thus, the claim follows by the inductive hypothesis.
- Let $i \notin \pi(f)$. Here, we have that $t_i \in \mathsf{dec}_\pi(r)$. By the inductive hypothesis, either $\pi(r|_p)$ is a subterm of $\pi(t_i)$ or there exists a term $r' \in \mathsf{dec}_\pi(t_i)$ such that $\pi(r|_p)$ is a subterm of $\pi(r')$. Assume first that $\pi(r|_p)$ is a subterm of $\pi(t_i)$. Therefore, we have that $\pi(r|_p)$ is a subterm of $\pi(t_i)$ and $t_i \in \mathsf{dec}_\pi(r)$. Assume now that there exists a term $r' \in \mathsf{dec}_\pi(t_i)$ such that $\pi(r|_p)$ is a subterm of $\pi(r')$. Since r' is a subterm of t_i and $t_i \in \mathsf{dec}_\pi(r)$, the claim follows. □

Lemma 8 *Let $\mathscr{R}$ be a left-linear constructor TRS, t^α be an abstract term, and π be a safe argument filtering for t^α in $\mathscr{R}$. Then, $\pi(DP(\mathscr{R})) \subseteq DP(\mathsf{AFT}_\pi(\mathscr{R}))$.*

Proof By the definition of safe argument filtering (second condition), there are no occurrences of extra variables in the filtered dependency pairs, i.e., $\mathscr{V}\mathsf{ar}(\pi(s)) \supseteq \mathscr{V}\mathsf{ar}(\pi(t))$ for all $s \to t \in DP(\mathscr{R})$. Therefore, we have $\pi(DP(\mathscr{R})) = \{\pi(s) \to \pi(t) \mid s \to t \in DP(\mathscr{R})\}$, i.e., $[\pi(s) \to \pi(t)]_\bot = \pi(s) \to \pi(t)$ for all $s \to t \in DP(\mathscr{R})$.

Consider an arbitrary filtered dependency pair $\pi(l^\sharp) \to \pi(u^\sharp) \in \pi(DP(\mathscr{R}))$. By definition of dependency pair, there must be a rule $l \to r \in \mathscr{R}$ such that u is a subterm of r rooted by a defined symbol. By Lemma 7, we have that either

- $\pi(u)$ is a subterm of $\pi(r)$ or
- there is a term $r' \in \mathsf{dec}_\pi(r)$ such that $\pi(u)$ is a subterm of $\pi(r')$.

Now we show that, in both cases, we have $[\pi(l^\sharp) \to \pi(u^\sharp)]_\bot \in DP(\mathsf{AFT}_\pi(\mathscr{R}))$ for every filtered dependency pair $\pi(l^\sharp) \to \pi(u^\sharp) \in \pi(DP(\mathscr{R}))$:

- Consider first that $\pi(u)$ is a subterm of $\pi(r)$. Then, $\pi(l)^\sharp \to \pi(u)^\sharp$ is a dependency pair obtained from some rule $[\pi(l) \to \pi(r)]_\bot \in \pi(\mathscr{R})$. Therefore, we have $[\pi(l^\sharp) \to \pi(r'^\sharp)]_\bot \in DP(\mathsf{AFT}_\pi(\mathscr{R}))$ and the claim follows.
- Consider now that there exists a term $r' \in \mathsf{dec}_\pi(r)$ such that $\pi(u)$ is a subterm of $\pi(r')$. Then, $[\pi(l) \to \pi(r')]_\bot \in \mathsf{AFT}_\pi(\mathscr{R})$ and, thus, $\pi(l)^\sharp \to \pi(u)^\sharp$ is a dependency pair obtained from $[\pi(l) \to \pi(r')]_\bot \in \mathsf{AFT}_\pi(\mathscr{R}) \setminus \pi(\mathscr{R})$. Therefore, $[\pi(l^\sharp) \to \pi(u^\sharp)]_\bot \in DP(\mathsf{AFT}_\pi(\mathscr{R}))$. □