A Finite Representation of the Narrowing Space*

Naoki Nishida[1] and Germán Vidal[2]

[1] Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, 4648603 Nagoya, Japan,
nishida@is.nagoya-u.ac.jp
[2] MiST, DSIC, Universitat Politècnica de València
Camino de Vera, s/n, 46022 Valencia, Spain
gvidal@dsic.upv.es

Abstract. Narrowing basically extends rewriting by allowing free variables in terms and by replacing matching with unification. As a consequence, the search space of narrowing becomes usually infinite, as in logic programming. In this paper, we introduce the use of some operators that allow one to always produce a finite graph that still represents all the narrowing derivations. Furthermore, we introduce a novel, compact equational representation of the (possibly infinite) answers computed by narrowing for a given initial term. Both the finite graphs and the equational representation of the computed answers might be useful in a number of areas, like program comprehension, static analysis, program transformation, etc.

1 Introduction

The narrowing relation [27], originally introduced in the context of theorem proving, was later adopted as the operational semantics of so called functional logic programming languages (like Curry [15]). Basically, narrowing extends term rewriting by allowing terms with variables and replacing matching with unification. Therefore, narrowing has many similarities with the SLD resolution principle of logic programming. Indeed, both narrowing and SLD resolution usually produce an infinite search space, i.e., an infinite tree-like structure. Currently, narrowing is regaining popularity in a number of areas other than functional logic programming, like protocol verification [10, 17], model checking [8, 11], partial evaluation [1, 26], refining methods for proving the termination of rewriting [5, 6], etc. In many—if not all—of these applications, producing a finite representation—usually in the form of a finite graph—of the narrowing space is essential.

The generation of a finite representation of the narrowing space has been tackled, e.g., by partial evaluation techniques (see, e.g., [1]). Here, some *subsumption* and *abstraction* operators are introduced in order to stop potentially infinite derivations. However, no previous work has formally considered how the

* This work has been partially supported by the *Generalitat Valenciana* under grant PROMETEO/2011/052.

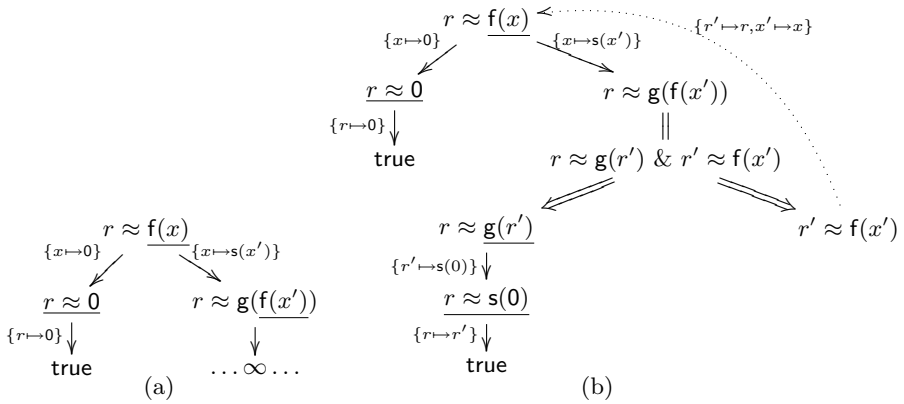

Fig. 1. Building a finite representation of the narrowing space for $\mathsf{f}(x)$.

use of subsumption and abstraction operators can be used to construct finite narrowing trees that still represent all possible derivations. In this work, we present a new approach to produce a finite narrowing tree for any term. For this purpose, we introduce two basic operators: splitting and flattening. *Splitting* a conjunction like e_1 & e_2 implies the parallel evaluation of the conjuncts e_1 and e_2. On the other hand, *flattening* an equation e returns a conjunction of the form $e|_p \approx x$ & $e[x]_p$, where the subterm $e|_p$ of e is replaced by a fresh variable x in e and a new equation is added. These two operations suffice to always produce a finite representation of the narrowing space.

Example 1. Consider the following simple program (a term rewriting system):

$$\mathcal{R} = \{\mathsf{f}(\mathsf{0}) \to \mathsf{0}, \quad \mathsf{f}(\mathsf{s}(x)) \to \mathsf{g}(\mathsf{f}(x)), \quad \mathsf{g}(\mathsf{s}(\mathsf{0})) \to \mathsf{s}(\mathsf{0})\}$$

where natural numbers are built using the constructors $\mathsf{0}$ and $\mathsf{s}(\ )$. Given the initial equation $r \approx \mathsf{f}(x)$, the narrowing space using an *innermost* strategy is infinite, as shown in Figure 1 (a), where the terms selected to be narrowed are underlined. Even by using some sort of memoization (as in [4]), where variants of a previously narrowed term are not unfolded, we still get an infinite narrowing space. In contrast, by using flattening (depicted with a double line) and splitting (depicted with a double arrow), we can obtain a finite representation of the narrowing space that still represents all the possible narrowing derivations, as shown in Figure 1 (b), where dotted arrows are used to point to a previous variant of a term or equation.

Designing a technique for producing finite narrowing trees can be useful in many different areas. For instance, one can use them to better understand the program's control flow, to analyze *weak* termination,[3] to detect subtrees that will

[3] A TRS is weakly terminating if, for any term, there is at least one terminating derivation [13].

never produce a computed answer (which is useful, e.g., in the context of the *more specific transformation* recently introduced in [22]), and so forth. In this paper, we present the building blocks for designing such techniques.

Furthermore, we also introduce a novel, compact equational representation of the (possibly infinite) answers computed by narrowing for a given initial term. In particular, we only need three operators:

- standard composition ($\cdot$),
- alternative ($+$), that represents the union of sets of substitutions, and
- parallel composition ($\Uparrow$), that denotes the *unification* on sets of substitutions.

The precise definitions will be introduced in Section 4.2. Using these operators, we are able to produce finite and compact representations for the computed answers of a term. For instance, the set of computed answers $\Gamma_{\mathsf{f}(x)}$ associated to the narrowing tree depicted in Figure 1 can be succinctly represented by

$$\begin{aligned}\Gamma_{\mathsf{f}(x)} = {} & \{x \mapsto \mathsf{0}, r \mapsto \mathsf{0}\} \\ & + \{x \mapsto \mathsf{s}(x')\} \cdot (\{r \mapsto \mathsf{s}(\mathsf{0}), \mathsf{r}' \mapsto \mathsf{s}(\mathsf{0})\} \Uparrow \{r' \mapsto r, x' \mapsto x\} \cdot \Gamma_{\mathsf{f}(x)})\end{aligned}$$

Interestingly, one can easily see that there is no solution to

$$\{r \mapsto \mathsf{s}(\mathsf{0}), \mathsf{r}' \mapsto \mathsf{s}(\mathsf{0})\} \Uparrow \{r' \mapsto r, x' \mapsto x\} \cdot \Gamma_{\mathsf{f}(x)}$$

since $\{r \mapsto \mathsf{s}(\mathsf{0}), \mathsf{r}' \mapsto \mathsf{s}(\mathsf{0})\}$ maps r' to $\mathsf{s}(\mathsf{0})$ while $\{r' \mapsto r, x' \mapsto x\} \cdot \Gamma_{\mathsf{f}(x)}$ can only bind r' to $\mathsf{0}$ (because the only non-recursive solution of $\Gamma_{\mathsf{f}(x)}$ binds r to $\mathsf{0}$), and $\mathsf{s}(\mathsf{0})$ and $\mathsf{0}$ clearly do not unify. Therefore, one can conclude that the only solution is $\{x \mapsto \mathsf{0}, r \mapsto \mathsf{0}\}$ despite the fact that the narrowing tree is infinite. Here, this was already obvious from the inspection of the narrowing tree. In general, however, our equational representation may be useful to analyze the computed answers of more complex programs.

This paper is organized as follows. In Section 2, we briefly review some notions and notations of term rewriting and narrowing. Section 3 presents some results on the compositionality of narrowing, introduces the flattening operator and proves its correctness. Section 4 then presents our method to produce finite narrowing trees by using subsumption, constructor decomposition, flattening, and splitting. We also introduce an equational representation for the computed answers in this section. Finally, Section 6 concludes and points out some directions for future research.

2 Preliminaries

We assume familiarity with basic concepts of term rewriting and narrowing. We refer the reader to, e.g., [7], [24], and [14] for further details.

Terms and Substitutions. A *signature* $\mathcal{F}$ is a set of function symbols. Given a set of variables $\mathcal{V}$ with $\mathcal{F} \cap \mathcal{V} = \varnothing$, we denote the domain of *terms* by $\mathcal{T}(\mathcal{F}, \mathcal{V})$. We assume that $\mathcal{F}$ always contains at least one constant $\mathsf{f}/0$. We use $\mathsf{f}, \mathsf{g}, \ldots$ to

denote functions and $x, y, \ldots$ to denote variables. Positions are used to address the nodes of a term viewed as a tree. A *position* p in a term t is represented by a finite sequence of natural numbers, where ϵ denotes the root position. The set of positions of a term t is denoted by $\mathcal{P}os(t)$. We let $t|_p$ denote the *subterm* of t at position p and $t[s]_p$ the result of *replacing the subterm* $t|_p$ by the term s. $\mathcal{V}ar(t)$ denotes the set of variables appearing in t. A term t is *ground* if $\mathcal{V}ar(t) = \varnothing$.

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

A term t_2 is an *instance* of a term t_1 (or, equivalently, t_1 is *more general* than t_2), in symbols $t_1 \leqslant t_2$, if there is a substitution σ with $t_2 = t_1\sigma$. Two terms t_1 and t_2 are *variants* (or equal up to variable renaming) if $t_1 = t_2\rho$ for some variable renaming ρ. A *unifier* of two terms t_1 and t_2 is a substitution σ with $t_1\sigma = t_2\sigma$. This notion is naturally extended to a set of equations: σ is a unifier of a set of equations $\{s_1 = t_1, \ldots, s_n = t_n\}$ if $s_i\sigma = t_i\sigma$ for $i = 1, \ldots, n$; furthermore, σ is the *most general unifier* of $\{s_1 = t_1, \ldots, s_n = t_n\}$, denoted by $\mathsf{mgu}(\{s_1 = t_1, \ldots, s_n = t_n\})$ if, for every other unifier θ of $\{s_1 = t_1, \ldots, s_n = t_n\}$, we have that $\sigma \leqslant \theta$.

TRSs and Rewriting. A set of rewrite rules $l \to r$ such that l is a nonvariable term and r is a term whose variables appear in l is called a *term rewriting system* (TRS for short); terms l and r are called the left-hand side and the right-hand side of the rule, respectively. We restrict ourselves to finite signatures and TRSs. Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined* symbols $\mathcal{D}_\mathcal{R}$ are the root symbols of the left-hand sides of the rules and the *constructors* are $\mathcal{C}_\mathcal{R} = \mathcal{F} \setminus \mathcal{D}_\mathcal{R}$. *Constructor terms* of $\mathcal{R}$ are terms over $\mathcal{C}_\mathcal{R}$ and $\mathcal{V}$, i.e., $\mathcal{T}(\mathcal{C}_\mathcal{R}, \mathcal{V})$. We omit $\mathcal{R}$ from $\mathcal{D}_\mathcal{R}$ and $\mathcal{C}_\mathcal{R}$ if it is clear from the context. A substitution σ is a *constructor substitution* (of $\mathcal{R}$) if $x\sigma \in \mathcal{T}(\mathcal{C}_\mathcal{R}, \mathcal{V})$ for all variables x. A TRS $\mathcal{R}$ is a *constructor system* if the left-hand sides of its rules have the form $\mathsf{f}(s_1, \ldots, s_n)$ where s_i are constructor terms, i.e., $s_i \in \mathcal{T}(\mathcal{C}, \mathcal{V})$, for all $i = 1, \ldots, n$.

For a TRS $\mathcal{R}$, we define the associated rewrite relation $\to_\mathcal{R}$ as the smallest binary relation satisfying the following: given terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have $s \to_\mathcal{R} t$ iff there exist a position p in s, a rewrite rule $l \to r \in \mathcal{R}$ and a substitution σ with $s|_p = l\sigma$ and $t = s[r\sigma]_p$; the rewrite step is usually denoted by $s \to_{p,l\to r} t$ to make explicit the position and rule used in this step. Moreover, if no proper subterms of $s|_p$ are reducible, then we speak of an *innermost* reduction step, denoted by $s \xrightarrow{i}_\mathcal{R} t$. 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 $\mathcal{R}$ if there is no term s with $t \to_{\mathcal{R}} s$. A *derivation* is a (possibly empty) sequence of rewrite steps. Given a binary relation $\to$, we denote by $\to^*$ its reflexive and transitive closure. Thus $t \to^*_{\mathcal{R}} s$ means that t can be reduced to s in $\mathcal{R}$ in zero or more steps.

Narrowing. The *narrowing* relation [27] mainly extends term rewriting by replacing pattern matching with unification, so that terms containing logic (i.e., *free*) variables can also be reduced by non-deterministically instantiating these variables. Formally, given a TRS $\mathcal{R}$ and two terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have that $s \leadsto_{\mathcal{R}} t$ is a *narrowing step* iff there exist[4]

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

and $t = (s[r]_p)\sigma$. We usually write $s \leadsto_{p,l\to r,\theta} t$ (or simply $s \leadsto_\theta t$) to make explicit the position, rule, and substitution of the narrowing step. Moreover, if $s|_p$ contains no proper narrowable subterms, then we speak of an *innermost narrowing* step (see, e.g., [12]), denoted by $s \overset{i}{\leadsto}_{p,l\to r,\theta} t$. Also, when there are several possible innermost narrowing steps, we consider the *leftmost* one, i.e., we consider a leftmost innermost narrowing strategy in this paper.

A *narrowing derivation* $t_0 \leadsto^*_\sigma t_n$ denotes a sequence of narrowing steps $t_0 \leadsto_{\sigma_1} \cdots \leadsto_{\sigma_n} t_n$ with $\sigma = \sigma_n \cdot \cdots \cdot \sigma_1$ (if $n = 0$ then $\sigma = id$). Given a narrowing derivation $s \leadsto^*_\sigma t$ with t a constructor term, we say that σ is a *computed answer* for s. We say that a substitution is *normalized* w.r.t. narrowing (and $\mathcal{R}$) if every variable in the domain is replaced by a term that is not narrowable in $\mathcal{R}$. A well-known result for innermost narrowing in constructor systems states that only substitutions normalized w.r.t. narrowing are computed.

Example 2. Consider the TRS

$$\mathcal{R} = \left\{ \begin{array}{ll} \mathsf{add}(\mathsf{0}, y) \to y & (R_1) \\ \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y)) & (R_2) \end{array} \right\}$$

defining the addition $\mathsf{add}/2$ on natural numbers built from $\mathsf{0}/0$ and $\mathsf{s}/1$. Given the term $\mathsf{add}(x, \mathsf{s}(\mathsf{0}))$, we have infinitely many narrowing derivations starting from $\mathsf{add}(x, \mathsf{s}(\mathsf{0}))$, e.g.,

$$\begin{array}{ll} \mathsf{add}(x, \mathsf{s}(\mathsf{0})) \leadsto_{\epsilon, R_1, \{x \mapsto \mathsf{0}\}} & \mathsf{s}(\mathsf{0}) \\ \mathsf{add}(x, \mathsf{s}(\mathsf{0})) \leadsto_{\epsilon, R_2, \{x \mapsto \mathsf{s}(y_1)\}} & \mathsf{s}(\mathsf{add}(y_1, \mathsf{s}(\mathsf{0}))) \leadsto_{1, R_1, \{y_1 \mapsto \mathsf{0}\}} \mathsf{s}(\mathsf{s}(\mathsf{0})) \\ \ldots & \end{array}$$

with computed answers $\{x \mapsto \mathsf{0}\}$, $\{x \mapsto \mathsf{s}(\mathsf{0})\}$, etc.

[4] We consider the so called *most general* narrowing, i.e., the mgu of the selected subterm and the left-hand side of a rule—rather than an arbitrary unifier—is computed at each narrowing step.

3 Compositionality and Flattening

The compositionality property can be simply formalized at the level of equations, i.e., narrowing is compositional when the computed answers of e_1 & e_2 can be obtained from the computed answers of e_1 and e_2, where "&" denotes the Boolean conjunction operator. As for the flattening operation, given an equation $x \approx \mathsf{f}(\mathsf{g}(y))$,[5] its flattening returns, e.g., $x' \approx \mathsf{g}(y)$ & $x \approx \mathsf{f}(x')$, where x' is a fresh variable. Therefore, flattening can be used to *distribute* the narrowing tasks among different equations.

In principle, compositionality holds for any narrowing strategy that fulfills the following conditions:

- Independence of the context. This is the case, for instance, of unrestricted narrowing, basic narrowing, innermost narrowing, etc. Lazy or needed narrowing, in contrast, are not independent of the context because, given an expression $s[t]_p$, we cannot determine wheter t should be narrowed (and to what extent) without looking at the context $s[\ ]_p$.
- Terms introduced by instantiation should not be narrowable. This is the case, for instance, of basic narrowing, innermost narrowing, lazy and needed narrowing (for left-linear constructor systems), etc. This is not the case of unrestricted narrowing though.

In the following, for simplicity, we will focus on (unconditional) innermost narrowing (though other narrowing strategies would also be equally appropriate, e.g., basic narrowing). Furthermore, some strategies not fulfilling the above conditions, like lazy and needed narrowing, can also be proved compositional by restricting the narrowing derivations to head normal form (so that they become essentially independent of the context).

In this paper, we consider the usual definitions for syntactic equality $\mathcal{R}_{eq} = \{x \approx x \to \mathsf{true}\}$ and conjunction $\mathcal{R}_{\&} = \{\mathsf{true}\ \&\ x \to x,\ \mathsf{false}\ \&\ x \to \mathsf{false}\}$. Therefore, narrowing deals with equations and conjunctions as ordinary terms. Sometimes we call such terms *equational* terms to make it explicit that they contain occurrences of "$\approx$" and/or "&". In the following, we assume that every TRS implicitly includes the rules of $\mathcal{R}_{eq}$ and $\mathcal{R}_{\&}$.

Here, we only aim at preserving the answers computed in *successful* derivations, i.e., derivations ending with a constructor term (true, when the initial term is an equation or a conjunction of equations).

Definition 3 (success set). *Let $\mathcal{R}$ be a constructor TRS and let t be a term. We define the* success set $\mathcal{S}_{\mathcal{R}}(t)$ *of t in $\mathcal{R}$ as follows:*

$$\mathcal{S}_{\mathcal{R}}(t) = \{\sigma\restriction_{\mathcal{V}\mathsf{ar}(t)} |\ t \overset{i}{\leadsto}{}^*_{\sigma}\ c \text{ in } \mathcal{R} \text{ and } c \in \mathcal{T}(\mathcal{C},\mathcal{V}) \text{ is a constructor term}\}$$

Observe that function $\mathcal{S}$ does not return the computed normal forms. Nevertheless, we can still get the computed normal form as follows: given a term t,

[5] Here, "$\approx$" is a binary symbol to denote syntactic equality on terms, see below.

we consider an initial equation of the form $x \approx t$, where x is a fresh variable not occurring in t; therefore, x will be bound to the normal form of t in any successful derivation (i.e., any derivation that ends with true).

Let us now recall the definition of *parallel composition* of substitutions, denoted by $\Uparrow$ in [16, 25]. Informally speaking, this operation corresponds to the notion of unification generalized to substitutions. Here, $\widehat{\theta}$ denotes the *equational representation* of a substitution θ, i.e., if $\theta = \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ then $\widehat{\theta} = \{x_1 = t_1, \ldots, x_n = t_n\}$.

Definition 4 (parallel composition [25]). *Let θ_1 and θ_2 be two idempotent substitutions. Then, we define $\Uparrow$ as follows:*

$$\theta_1 \Uparrow \theta_2 = \begin{cases} \mathsf{mgu}(\widehat{\theta_1} \cup \widehat{\theta_2}) & \text{if } \widehat{\theta_1} \cup \widehat{\theta_2} \text{ has a solution (a unifier)} \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

Parallel composition is extended to sets of substitutions in the natural way:

$$\Theta_1 \Uparrow \Theta_2 = \{\theta_1 \Uparrow \theta_2 \mid \theta_1 \in \Theta_1,\ \theta_2 \in \Theta_2,\ \theta_1 \Uparrow \theta_2 \neq \mathsf{fail}\}$$

Now, we state the main compositional result for innermost narrowing:

Theorem 5. *Let $\mathcal{R}$ be a constructor TRS. Let e_1 & e_2 be an equational term. Then, we have $\mathcal{S}_{\mathcal{R}}(e_1 \,\&\, e_2) = \mathcal{S}_{\mathcal{R}}(e_1) \Uparrow \mathcal{S}_{\mathcal{R}}(e_2)$ up to variable renaming.*

As a useful consequence of the above compositionality result, we can state the following corollary:

Corollary 6. *Let $\mathcal{R}$ be a constructor TRS. Let e_1 & e_2 be an equational term. Then, we have $\mathcal{S}_{\mathcal{R}}(e_1 \,\&\, e_2) = \mathcal{S}_{\mathcal{R}}(e_2 \,\&\, e_1)$ up to variable renaming.*

In practice, this result implies that innermost narrowing can select the equations to be narrowed in any order (and not necessarily in a left-to-right order) while preserving the computed answers. This is equivalent to the *independence of the selection rule* of logic programming.

Now, we recall the flattening transformation (called *unfolding* in [23]) that will become useful in the next section, and prove its correctness.

Definition 7 (flattening). *Let e be an equational term. Then, we say that $x \approx e|_p \,\&\, e[x]_p$ is a* flattening *of e, with $p \in \mathcal{P}os(e)$ and $p \neq \epsilon$.*

The following property states the correctness of the flattening operation:

Theorem 8. *Let $\mathcal{R}$ be a constructor TRS. Let e be an equational term and e' be a flattening of e. Then, we have $\mathcal{S}_{\mathcal{R}}(e) = \mathcal{S}_{\mathcal{R}}(e')\ [\mathcal{V}\mathsf{ar}(e)]$ up to variable renaming.*

4 A Finite Representation of the Narrowing Space

Now, we introduce a framework to obtain a finite representation of a (possibly infinite) narrowing space. Then, given a finite narrowing tree, we also present a method to extract an equational representation of the associated success set.

4.1 Construction of Finite Narrowing Trees

We proceed as in the construction of a standard narrowing tree, but we also introduce some new operators in order to ensure that the tree can be kept finite.

Definition 9 (extended narrowing tree). *Let $\mathcal{R}$ be a constructor TRS and t be a term. An* extended narrowing tree *for t in $\mathcal{R}$ is a directed rooted node- and edge-labeled graph τ built as follows:*

- *the root node of τ is labeled with $x \approx t$, where x is a fresh variable not occuring in t;*
- *if a node is labeled with a term that cannot be further narrowed, it is considered a leaf; moreover, we label this node with* fail *when it is not* true *(to make it explicit that this is a failing derivation);*
- subsumption: *if a node is labeled with a term e' that is a variant of a previous node e in the same root-to-leaf derivation, i.e., $e'\vartheta = e$, it is also considered a leaf, and we add an implicit edge between these nodes labeled with ϑ;*[6]
- constructor decomposition: *if a node is labeled with $y \approx \mathsf{c}(t_1, \ldots, t_n)$ & e ($\mathsf{c} \in \mathcal{C}$), we add an edge to a node $y_1 \approx t_1$ & $\ldots$ & $y_n \approx t_n$ & e, with $y_1, \ldots, y_n$ fresh variables, and the edge is labeled with $\{y \mapsto \mathsf{c}(y_1, \ldots, y_n)\}$;*
- *otherwise, we expand (don't care nondeterministically) the node using one of the following rules:*
  - narrowing: *we have an output edge labeled with σ from a node e to a node e' for all innermost narrowing steps $e \overset{i}{\leadsto}_\sigma e'$;*
  - flattening: *there is an output edge from a node e to a node $y \approx e|_p$ & $e[y]_p$, where y is a fresh variable not occurring anywhere in the tree;*
  - splitting: *we have output edges from a node labeled with a term of the form e_1 & $\cdots$ & e_{n-1} & e_n to the nodes labeled with e_1, $\ldots$, e_{n-1}, and e_n.*

The operations considered in the previous definition can also be found in the literature (perhaps with some slightly different definitions). For instance, flattening is introduced in [23] (where it is called *unfolding*); subsumption is used in many different contexts (e.g., [4, 1]); (constructor) decomposition rules are used in different narrowing calculi (see, e.g., [18]); finally, splitting is considered when proving compositionality results (e.g., [3]) and in the partial evaluation of logic programs [9].

The relevance of the notion of extended narrowing tree is that, thanks to the use of the rules of flattening, constructor decomposition,[7] and splitting, one can always produce a tree with finitely many non-variant nodes. We do not provide a formal proof of this claim, but it is an easy consequence of the fact that using flattening—which involves generalizing a subterm—and splitting one can keep the set of non-variant terms finite. The correctness of the extended narrowing trees, i.e., the fact that they still represent all possible narrowing derivations, is an easy consequence of the results in Section 3.

[6] We consider these edges *implicit* to keep the data structure a tree.

[7] The rule of constructor decomposition is mainly introduced for simplicity, but could be replaced by a sequence of flattening steps.

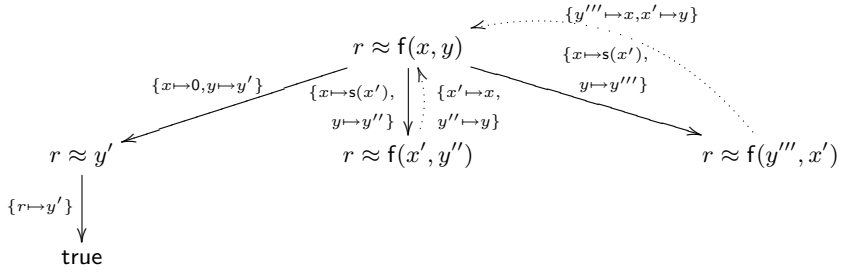

Fig. 2. Finite narrowing tree for $\mathsf{f}(x, y)$.

In this paper, we do not introduce a particular strategy for automating the construction of finite extended narrowing trees. Some strategies can produce very compact representations by applying constructor decomposition/flattening and splitting as much as possible. However, in this case, we also get less accurate results in general. Other strategies may try to avoid breaking down a term as long as possible. Here, one should be very careful to avoid entering an infinite loop. The interested reader can find suitable strategies, e.g., in the literature of narrowing-driven partial evaluation [2, 1]. Similar strategies could be defined using the above operations.

In the following, we will use these graphical conventions when depicting the steps of an extended narrowing tree:

- narrowing and constructor decomposition: (labeled) solid arrow ($\longrightarrow$);
- subsumption: (labeled) dotted arrow ($\cdots\!\!\succ$);
- flattening: double line ($=\!=$);
- splitting: double arrow ($\Longrightarrow$).

By abuse of notation, we often use in the text $e \longrightarrow^*_\sigma e'$ to denote a path in the tree, no matter the type of rules applied from node e to node e' (except subsumption), where σ is the composition of the substitutions in the labeled edges along this path (if any, and id otherwise).

Let us now illustrate the construction of finite extended narrowing trees with some examples. Let us note that rule variables are always renamed with fresh names; this is mandatory to produce correct equations in the next section.

Example 10. Let us consider the following (non confluent) TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{0}, y) \to y,\ \mathsf{f}(\mathsf{s}(x), y) \to \mathsf{f}(x, y),\ \mathsf{f}(\mathsf{s}(x), y) \to \mathsf{f}(y, x)\}$. Given the initial term $\mathsf{f}(x, y)$, the narrowing space is clearly infinite because of the recursive calls to f. Here, a couple of subsumption steps suffice to get a finite extended narrowing tree, as shown in Figure 2.

Example 11. Consider the TRS $\mathcal{R} = \{\mathsf{f}(\mathsf{0}, y) \to y,\ \mathsf{f}(\mathsf{s}(x), y) \to \mathsf{c}(\mathsf{f}(x, y), \mathsf{f}(y, x))\}$ and the initial term $\mathsf{f}(x, y)$. In this case, subsumption does not suffice and constructor decomposition and splitting becomes necessary, as shown in Figure 3. This is a simple pattern that could be routinely applied to all constructor-rooted terms in order to get a finite representation of the narrowing space.

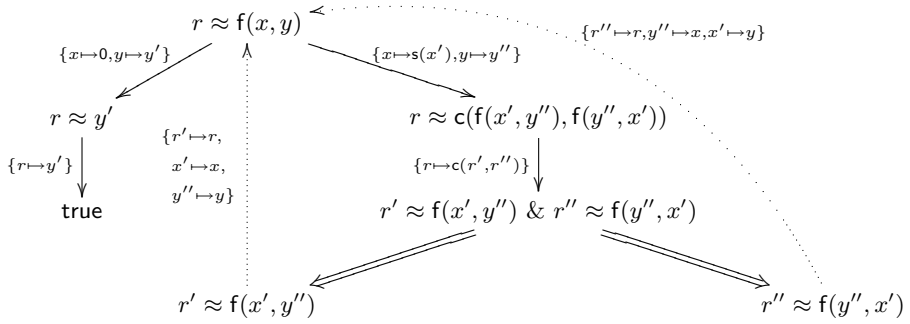

Fig. 3. Finite narrowing tree for $\mathsf{f}(x,y)$.

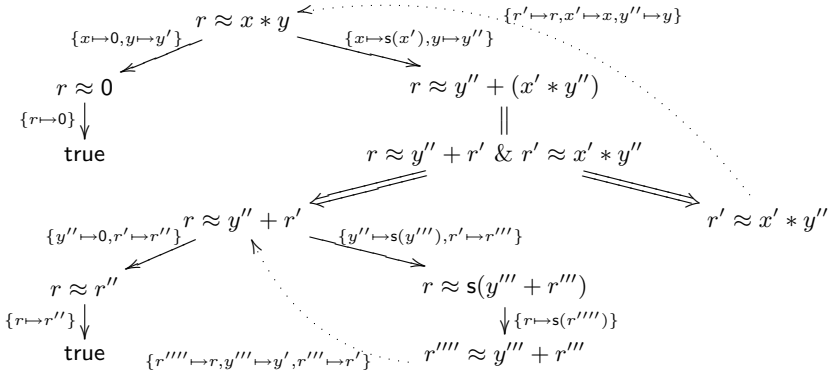

Fig. 4. Finite narrowing tree for $x * y$.

Observe that the constructor decomposition step is not really needed and could be mimicked by performing two flattening steps and, then, reducing the last equation as follows:

$$
\begin{array}{ll}
r \approx \mathsf{c}(\mathsf{f}(x',y''),\mathsf{f}(y'',x')) & \\
\quad\Longrightarrow & r' \approx \mathsf{f}(x',y'') \;\&\; r \approx \mathsf{c}(r',\mathsf{f}(y'',x')) \\
\quad\Longrightarrow & r' \approx \mathsf{f}(x',y'') \;\&\; r'' \approx \mathsf{f}(y'',x') \;\&\; r \approx \mathsf{c}(r',r'') \\
\quad\longrightarrow_{\{r\mapsto \mathsf{c}(r',r'')\}} & r' \approx \mathsf{f}(x',y'') \;\&\; r'' \approx \mathsf{f}(y'',x') \;\&\; \mathsf{true}
\end{array}
$$

However, we prefer to keep the constructor decomposition steps for simplicity.

Example 12. Finally, consider the following TRS $\mathcal{R} = \{\mathsf{0} + y \to y,\ \mathsf{s}(x) + y \to \mathsf{s}(x + y),\ \mathsf{0} * y \to \mathsf{0},\ \mathsf{s}(x) * y \to y + (x * y)\}$. Given the initial term $x * y$, both flattenning and splitting are necessary to produce a finite extended narrowing tree, as shown in Figure 4.

4.2 Success Set Equations

In this section, we introduce an equational notation for representing the success set of a term, that we call its *success set equations*. Here, we consider the following three operators:

- Composition ($\cdot$). For simplicity, besides the standard composition of substitutions, we also consider its extension to sets of substitutions as follows. Given a set of substitutions Θ and a substitution σ, we let $\sigma \cdot \Theta = \{\sigma \cdot \theta \mid \theta \in \Theta\}$ and $\Theta \cdot \sigma = \{\theta \cdot \sigma \mid \theta \in \Theta\}$.
- Alternative ($+$). In our context, an expression like $ss_1 + ss_2$ denotes the union of the success sets denoted by ss_1 and ss_2. Again, for simplicity, we let a substitution denote a singleton set with this substitution.
- Parallel composition ($\Uparrow$). This is the standard parallel composition operator introduced in Definition 4.

As for the operator precedence, we assume that composition has a higher priority than parallel composition, which has a higher priority than alternative.

Now, we introduce a technique to extract the success set equations of a term from a given (finite) extended narrowing tree. Loosely speaking, substitutions along derivations with narrowing steps are just composed. Flattening and constructor decomposition steps are ignored. Splitting steps involve computing the parallel composition of the success sets of the different branches. Finally, for subsumption steps, we compose the current set with the substitution labeling the step and, then, with the success set of the previous variant term.

In the following, we use the following notation. Given an extended narrowing tree τ, we let $\mathsf{root}(\tau)$ denote the root of τ. We also let $\tau \equiv (t \to_\sigma \tau')$ denote the fact that τ is rooted by term t and has a (possibly labeled) output edge to a subtree τ'. Moreover, we use the auxiliary function $\mathsf{out}(\tau)$ that returns the output edges from $\mathsf{root}(\tau)$ (if any). E.g., let τ be the extended narrowing tree of Figure 4; here, we have $\mathsf{out}(\tau) = \{r \approx x * y \to_{\{x \mapsto 0, y \mapsto y'\}} \tau_1,\ r \approx x * y \to_{\{x \mapsto \mathsf{s}(x'), y \mapsto y''\}} \tau_2\}$, where τ_1 and τ_2 are the subtrees rooted by $r \approx 0$ and $r \approx y'' + (x' * y'')$, respectively. Finally, we let $\mathsf{subtrees}(\tau)$ denote the set of subtrees of a tree τ that are obtained by partitioning τ into those subtrees that are rooted by a term with an incoming subsumption edge. E.g., for the tree τ of Figure 4, $\mathsf{subtrees}(\tau)$ returns two subtrees, one rooted by $r \approx x * y$ and another one rooted by $r \approx y'' + r'$.

Definition 13 (success set equations). *Let τ be a finite extended narrowing tree for a term t. Let $\mathcal{T} = \mathsf{subtrees}(\tau)$. Then, we produce a success set equation $\Gamma_t = \mathcal{SF}(\tau')$ for each tree in $\tau' \in \mathcal{T}$ with $\mathsf{root}(\tau') = t$, where the auxiliary function $\mathcal{SF}$ is defined as follows:*

$$\mathcal{SF}(\tau) = \begin{cases} id & \text{if } \tau \equiv \mathsf{true} \\ \mathsf{fail} & \text{if } \tau \equiv \mathsf{fail} \text{ (a failing derivation)} \\ \sigma \cdot \Gamma_{t'} & \text{if } \tau \equiv (\ t \overset{\cdots}{\gg}_\sigma \tau'\ ),\ t' = \mathsf{root}(\tau') \\ \mathcal{SF}(\tau') & \text{if } \tau \equiv (e == \tau') \\ \mathcal{SF}(\tau_1) \Uparrow \cdots \Uparrow \mathcal{SF}(\tau_n) & \text{if } \mathsf{out}(\tau) = \{e \Rightarrow \tau_i \mid i = 1, \ldots, n\} \\ \sigma_1 \cdot \mathcal{SF}(\tau_1) + \cdots + \sigma_n \cdot \mathcal{SF}(\tau_n) & \text{if } \mathsf{out}(\tau) = \{e \to_\sigma \tau_i \mid i = 1, \ldots, n\} \end{cases}$$

For clarity, when no confusion can arise, we often label function Γ with term t rather than with the equation $r \approx t$.

Example 14. Given the extended narrowing tree of Figure 2, we produce the following success set equation:

$$\begin{aligned}\Gamma_{\mathsf{f}(x,y)} &= \{x \mapsto \mathsf{0}, y \mapsto y', r \mapsto y'\} \\ &+ \{x \mapsto \mathsf{s}(x'), y \mapsto y''\} \cdot (\{x' \mapsto x, y'' \mapsto y\} \cdot \Gamma_{\mathsf{f}(x,y)}) \\ &+ \{x \mapsto \mathsf{s}(x'), y \mapsto y'''\} \cdot (\{y''' \mapsto x, x' \mapsto y\} \cdot \Gamma_{\mathsf{f}(x,y)})\end{aligned}$$

Informally speaking, the (infinite) solutions of this equation can be enumerated iteratively as follows. One starts with $\Gamma^0_{\mathsf{f}(x,y)} = \{\}$. Then, we compute the next iteration $i > 0$ as follows:

$$\begin{aligned}\Gamma^i_{\mathsf{f}(x,y)} &= \{x \mapsto \mathsf{0}, y \mapsto y', r \mapsto y'\} \\ &+ \{x \mapsto \mathsf{s}(x'), y \mapsto y''\} \cdot (\{x' \mapsto x, y'' \mapsto y\} \cdot \Gamma^{i-1}_{\mathsf{f}(x,y)}) \\ &+ \{x \mapsto \mathsf{s}(x'), y \mapsto y'''\} \cdot (\{y''' \mapsto x, x' \mapsto y\} \cdot \Gamma^{i-1}_{\mathsf{f}(x,y)})\end{aligned}$$

Therefore, we have the following infinite sequence[8]

$$\begin{aligned}\Gamma^1_{\mathsf{f}(x,y)} &= \{\{x \mapsto \mathsf{0}, y \mapsto y'\}\} \\ \Gamma^2_{\mathsf{f}(x,y)} &= \Gamma^1_{\mathsf{f}(x,y)} \cup \{\{x \mapsto \mathsf{s}(\mathsf{0}), y \mapsto y'\},\ \{x \mapsto \mathsf{s}(y'), y \mapsto \mathsf{0}\}\} \\ \Gamma^3_{\mathsf{f}(x,y)} &= \Gamma^2_{\mathsf{f}(x,y)} \cup \{\{x \mapsto \mathsf{s}(\mathsf{s}(\mathsf{0})), y \mapsto y'\},\ \{x \mapsto \mathsf{s}(\mathsf{s}(y')), y \mapsto \mathsf{0}\}, \\ &\qquad\qquad \{x \mapsto \mathsf{s}(y'), y \mapsto \mathsf{s}(\mathsf{0})\},\ \{x \mapsto \mathsf{s}(\mathsf{0}), y \mapsto \mathsf{s}(y')\}\} \\ &\dots\end{aligned}$$

In the following, we denote by $\mathsf{sols}(\Gamma_t)$ the (possibly infinite) set of solutions of the success set equation Γ_t for some term t. Let us consider a set of success set equations $\Gamma_{t_1} = r_1, \dots, \Gamma_{t_n} = r_n$ associated to the narrowing derivations starting from term t_1. A procedure to enumerate the substitutions in $\mathsf{sols}(\Gamma_{t_1})$ can proceed as follows:

1. Initialization. $\Gamma^0_{t_1} = \cdots = \Gamma^0_{t_n} = \{\ \}$.
2. Iterative process. for all $i > 0$, we compute the following sets:

$$\Gamma^i_{t_1} = r_1[\Gamma_t \mapsto \Gamma^{i-1}_t] \qquad \dots \qquad \Gamma^i_{t_n} = r_n[\Gamma_t \mapsto \Gamma^{i-1}_t]$$

   where $r_j[\Gamma_t \mapsto \Gamma^{i-1}_t]$ denotes the expression that results from r_j by replacing every occurrence of Γ_t by Γ^{i-1}_t, with $j = 1, \dots, n$ and $t \in \{t_1, \dots, t_n\}$.

Then, we have $\mathsf{sols}(\Gamma_{t_1}) = \bigcup_{i>0} \Gamma^i_{t_1}$, where the $\Gamma^i_{t_1}$ are computed as above.

We do not formally prove the correctness of the above procedure for computing $\mathsf{sols}(\Gamma_t)$, but it is rather straightforward.

[8] We restrict substitutions to $\mathcal{V}\mathsf{ar}(\mathsf{f}(x,y))$ for conciseness.

Example 15. Given the extended narrowing tree shown in Figure 3, we produce the following success set equation:

$$\begin{aligned}\Gamma_{\mathsf{f}(x,y)} &= \{x \mapsto \mathsf{0}, y \mapsto y', r \mapsto y'\} \\ &\quad + \{x \mapsto \mathsf{s}(x'), y \mapsto y'', r \mapsto \mathsf{c}(r', r'')\} \cdot (\{r' \mapsto r, x' \mapsto x, y'' \mapsto y\} \cdot \Gamma_{\mathsf{f}(x,y)} \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \Uparrow \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \{r'' \mapsto r, y'' \mapsto x, x' \mapsto y\} \cdot \Gamma_{\mathsf{f}(x,y)})\end{aligned}$$

Computing the success set is slightly more difficult now since it involves parallel compositions. The sequence of success sets is as follows:

$$\begin{aligned}\Gamma^0_{\mathsf{f}(x,y)} &= \{\ \} \\ \Gamma^1_{\mathsf{f}(x,y)} &= \{\{x \mapsto \mathsf{0}, y \mapsto y', r \mapsto y'\}\} \\ \Gamma^2_{\mathsf{f}(x,y)} &= \Gamma^1_{\mathsf{f}(x,y)} \cup \{\{x \mapsto \mathsf{s}(x'), y \mapsto y'', r \mapsto \mathsf{c}(r', r'')\} \\ &\qquad\qquad \cdot(\{r' \mapsto y', x' \mapsto \mathsf{0}, y'' \mapsto y', x \mapsto \mathsf{0}, y \mapsto y', r \mapsto y'\} \\ &\qquad\qquad\quad \Uparrow \{r'' \mapsto y', y'' \mapsto \mathsf{0}, x' \mapsto y', x \mapsto \mathsf{0}, y \mapsto y', r \mapsto y'\})\} \\ &= \Gamma^1_{\mathsf{f}(x,y)} \cup \{\{x \mapsto \mathsf{s}(x'), y \mapsto y'', r \mapsto \mathsf{c}(r', r'')\} \\ &\qquad\qquad \cdot\{r' \mapsto \mathsf{0}, r'' \mapsto \mathsf{0}, x' \mapsto \mathsf{0}, y'' \mapsto \mathsf{0}, x \mapsto \mathsf{0}, y \mapsto \mathsf{0}, r \mapsto \mathsf{0}\}\} \\ &= \Gamma^1_{\mathsf{f}(x,y)} \cup \{\{x \mapsto \mathsf{s}(\mathsf{0}), y \mapsto \mathsf{0}, r \mapsto \mathsf{c}(\mathsf{0}, \mathsf{0})\}\} \\ &\ldots\end{aligned}$$

Example 16. Given the extended narrowing tree shown in Figure 1, we produce the following success set equation:

$$\begin{aligned}\Gamma_{\mathsf{f}(x)} &= \{x \mapsto \mathsf{0}, r \mapsto \mathsf{0}\} \\ &\quad + \{x \mapsto \mathsf{s}(x')\} \cdot (\{r' \mapsto \mathsf{s}(\mathsf{0}), r \mapsto r'\} \Uparrow \{r' \mapsto r, x' \mapsto x\} \cdot \Gamma_{\mathsf{f}(x)})\end{aligned}$$

The sequence of success sets is as follows:

$$\begin{aligned}\Gamma^0_{\mathsf{f}(x)} &= \{\ \} \\ \Gamma^1_{\mathsf{f}(x)} &= \{\{x \mapsto \mathsf{0}, r \mapsto \mathsf{0}\}\} \\ \Gamma^2_{\mathsf{f}(x)} &= \Gamma^1_{\mathsf{f}(x)} \\ &\quad \cup\{\{x \mapsto \mathsf{s}(x')\} \cdot (\{r' \mapsto \mathsf{s}(\mathsf{0}), r \mapsto r'\} \Uparrow \{r' \mapsto \mathsf{0}, x' \mapsto \mathsf{0}, x \mapsto \mathsf{0}, r \mapsto \mathsf{0}\})\} \\ &= \Gamma^1_{\mathsf{f}(x)}\end{aligned}$$

Thus, the success set equation denote the singleton set $\{\{x \mapsto \mathsf{0}, r \mapsto \mathsf{0}\}\}$.

Example 17. Given the extended narrowing tree shown in Figure 4, we produce the following success set equations:

$$\begin{aligned}\Gamma_{x*y} &= \{x \mapsto \mathsf{0}, y \mapsto y', r \mapsto \mathsf{0}\} \\ &\quad + \{x \mapsto \mathsf{s}(x'), y \mapsto y''\} \cdot (\Gamma_{y''+r'} \Uparrow \{r' \mapsto r, x' \mapsto x, y'' \mapsto y\} \cdot \Gamma_{x*y}) \\ \Gamma_{y''+r'} &= \{y'' \mapsto \mathsf{0}, r' \mapsto r'', r \mapsto r''\} \\ &\quad + \{y'' \mapsto \mathsf{s}(y'), r' \mapsto r, r \mapsto \mathsf{s}(r), r'''' \mapsto r, y''' \mapsto y', r''' \mapsto r'\} \cdot \Gamma_{y''+r'}\end{aligned}$$

The success set is the obvious one for addition and multiplication.

The correctness of success set equations can be stated as follows:

Theorem 18. *Let $\mathcal{R}$ be a constructor TRS and let t be a term. Let τ be a finite extended narrowing tree for t in $\mathcal{R}$ rooted with $r \approx t$, and let $\Gamma_{r \approx t}$ be its associated success set equation. Then, we have $\mathcal{S}_{\mathcal{R}}(r \approx t) = \mathsf{sols}(\Gamma_{r \approx t})$ up to variable renaming.*

5 Related Work

There are basically two closely related lines of research. On the one hand, we have a work by Antoy and Ariola [4] that aims at finding a finite representation of the (possibly infinite) narrowing space. In contrast to our approach, however, they only consider subsumption. Therefore, there is no guarantee that the representation of the narrowing space is going to be finite. They also propose a finite representation inspired by regular expressions to denote a (possibly infinite) enumeration of computed answers. This is somehow similar to our success set equations; nevertheless, our equations are more complex since they may also include parallel compositions.

On the other hand, there are a number of papers on the so-called *narrowing-driven* partial evaluation (see [1] and references herein) that also require the construction of a finite representation of the narrowing space. In contrast to [4], other operators like generalization (i.e., replacing some subterms by fresh variables) and splitting are used to ensure that the representation of the narrowing space is finite. However, no single narrowing tree is constructed, but a sequence of (possibly incomplete) narrowing trees, which are then used to extract the residual program (a sequence of *resultants* associated to each root-to-leaf narrowing derivation). The correctness of the transformation is proved for some narrowing strategies (under the *closedness* condition of the narrowing trees). However, no general properties are proved for the different operators.

Our approach can be seen as a combination of the above lines of research. We aim at constructing finite representations of the narrowing space, as in [4], but we also allow the use of powerful operators like flattening and splitting, similarly to the works on narrowing-driven partial evaluation.

6 Conclusion and Future Work

In this work, we have introduced a framework that provides the building blocks that are required to produce a finite representation of the (possibly infinite) narrowing space. For this purpose, we have considered three simple operations: constructor decomposition, flattening and splitting, and have proved its correctness. Then, we have introduced the notion of *extended* narrowing tree, where the above operations can be applied to make the tree finite. Finally, we have introduced a compact equational representation of the success set that follows the structure of a finite extended narrowing tree. Let us note that our approach could easily be transferred to other logic-based programming languages; in particular, it should be straightforward to adapt it to definite logic programs.

Among the possible applications, one can consider the use of extended narrowing trees and success set equations to better understand the program's control flow, to analyze *weak* termination [13], to detect subtrees that will never produce a computed answer as in Example 1 (which could be useful, e.g., in the context of the *more specific transformation* recently introduced in [22]), and so forth. This work opens many possibilities for future work. In particular, we would like to design fully automatic strategies for producing finite extended narrowing trees (e.g., following the methods used in the context of narrowing-driven partial evaluation [1]). We find also interesting the definition of methods to automatically analyze success set equations and infer useful properties that can be used in other contexts (like the *more specific transformation* mentioned above, that is currently being used for improving program inversion [20, 19, 21]).

Acknowledgements

We thank the anonymous reviewers for their useful comments to improve this paper. 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 and support.

References

1. Albert, E., Vidal, G.: The narrowing-driven approach to functional logic program specialization. New Generation Computing 20(1), 3–26 (2002)
2. Alpuente, M., Falaschi, M., Vidal, G.: Partial Evaluation of Functional Logic Programs. ACM TOPLAS 20(4), 768–844 (1998)
3. Alpuente, M., Falaschi, M., Vidal, G.: Compositional analysis for equational Horn programs. In: Levi, G., Rodríguez-Artalejo, M. (eds.) Algebraic and Logic Programming, Lecture Notes in Computer Science, vol. 850, pp. 77–94. Springer Berlin Heidelberg (1994)
4. Antoy, S., Ariola, Z.: Narrowing the Narrowing Space. In: Proc. of the 9th Int'l Symposium on Programming Languages: Implementations, Logics, and Programs (PLILP'97). pp. 1–15. Springer LNCS 1292 (1997)
5. Arts, T., Giesl, J.: Termination of Term Rewriting Using Dependency Pairs. Theoretical Computer Science 236(1-2), 133–178 (2000)
6. Arts, T., Zantema, H.: Termination of Logic Programs Using Semantic Unification. In: Proc. of the 5th Int'l Workshop on Logic Programming Synthesis and Transformation (LOPSTR'95). pp. 219–233. Springer LNCS 1048 (1996)
7. Baader, F., Nipkow, T.: Term Rewriting and All That. Cambridge University Press (1998)
8. Bae, K., Escobar, S., Meseguer, J.: Abstract Logical Model Checking of Infinite-State Systems Using Narrowing. In: Proc. of the 24th International Conference on Rewriting Techniques and Applications (RTA 2013). Lecture Notes in Computer Science, Springer (2013), to appear
9. De Schreye, D., Glück, R., Jørgensen, J., Leuschel, M., Martens, B., Sørensen, M.: Conjunctive Partial Deduction: Foundations, Control, Algorihtms, and Experiments. Journal of Logic Programming 41(2&3), 231–277 (1999)

10. Escobar, S., Meadows, C., Meseguer, J.: A Rewriting-Based Inference System for the NRL Protocol Analyzer and its Meta-Logical Properties. Theoretical Computer Science 367(1-2), 162–202 (2006)
11. Escobar, S., Meseguer, J.: Symbolic Model Checking of Infinite-State Systems Using Narrowing. In: Proc. of RTA'07. pp. 153–168. Springer LNCS 4533 (2007)
12. Fribourg, L.: SLOG: a logic programming language interpreter based on clausal superposition and rewriting. In: Proceedings of the Symposium on Logic Programming (SLP'85). pp. 172–185. IEEE Press (1985)
13. Gnaedig, I., Kirchner, H.: Proving weak properties of rewriting. Theor. Comput. Sci. 412(34), 4405–4438 (2011)
14. Hanus, M.: The integration of functions into logic programming: From theory to practice. Journal of Logic Programming 19&20, 583–628 (1994)
15. Hanus (ed.), M.: Curry: An integrated functional logic language (vers. 0.8.3). Available at http://www.curry-language.org (2012)
16. Hermenegildo, M., Rossi, F.: On the Correctness and Efficiency of Independent And-Parallelism in Logic Programs. In: Lusk, E., Overbeck, R. (eds.) Proc. of the 1989 North American Conf. on Logic Programming. pp. 369–389. The MIT Press, Cambridge, MA (1989)
17. Meseguer, J., Thati, P.: Symbolic Reachability Analysis Using Narrowing and its Application to Verification of Cryptographic Protocols. Electronic Notes in Theoretical Computer Science 117, 153–182 (2005)
18. Middeldorp, A., Okui, S.: A deterministic lazy narrowing calculus. Journal of Symbolic Computation 25(6), 733–757 (1998)
19. Nishida, N., Sakai, M., Sakabe, T.: Generation of inverse computation programs of constructor term rewriting systems. IEICE Transactions on Information and Systems J88-D-I(8), 1171–1183 (Aug 2005), in Japanese
20. Nishida, N., Sakai, M., Sakabe, T.: Partial inversion of constructor term rewriting systems. In: Giesl, J. (ed.) Proceedings of the 16th International Conference on Rewriting Techniques and Applications (RTA 2005). Lecture Notes in Computer Science, vol. 3467, pp. 264–278. Springer (2005)
21. Nishida, N., Vidal, G.: Program inversion for tail recursive functions. In: Schmidt-Schauß, M. (ed.) Proceedings of the 22nd International Conference on Rewriting Techniques and Applications (RTA 2011). LIPIcs, vol. 10, pp. 283–298. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2011)
22. Nishida, N., Vidal, G.: Computing More Specific Versions of Conditional Rewriting Systems. In: Albert, E. (ed.) Proc. of the 20th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2012). Lecture Notes in Computer Science, vol. 7844, pp. 137–154. Springer (2013)
23. Nutt, W., Réty, P., Smolka, G.: Basic narrowing revisited. Journal of Symbolic Computation 7, 295–317 (1989)
24. Ohlebusch, E.: Advanced topics in term rewriting. Springer-Verlag, London, UK (2002)
25. Palamidessi, C.: Algebraic Properties of Idempotent Substitutions. In: Paterson, M. (ed.) Proc. of 17th Int'l Colloquium on Automata, Languages and Programming. pp. 386–399. Springer LNCS 443 (1990)
26. Ramos, J.G., Silva, J., Vidal, G.: Fast Narrowing-Driven Partial Evaluation for Inductively Sequential Systems. In: Danvy, O., Pierce, B.C. (eds.) Proceedings of the 10th ACM SIGPLAN International Conference on Functional Programming (ICFP'05). pp. 228–239. ACM Press (2005)
27. Slagle, J.R.: Automated theorem-proving for theories with simplifiers, commutativity and associativity. Journal of the ACM 21(4), 622–642 (1974)