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 data structure that still represents all the narrowing derivations. Furthermore, we extract from this data structure a novel, compact equational representation of the (possibly infinite) answers computed by narrowing for a given initial term. Both the finite data structure 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 [28], 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 by 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 where several branches are created every time a function call matches with the left-hand side of more than one program rule. Currently, narrowing is regaining popularity in a number of areas other than functional logic programming, like protocol verification [10, 18], model checking [8, 11], partial evaluation [1, 27], 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, so called *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.

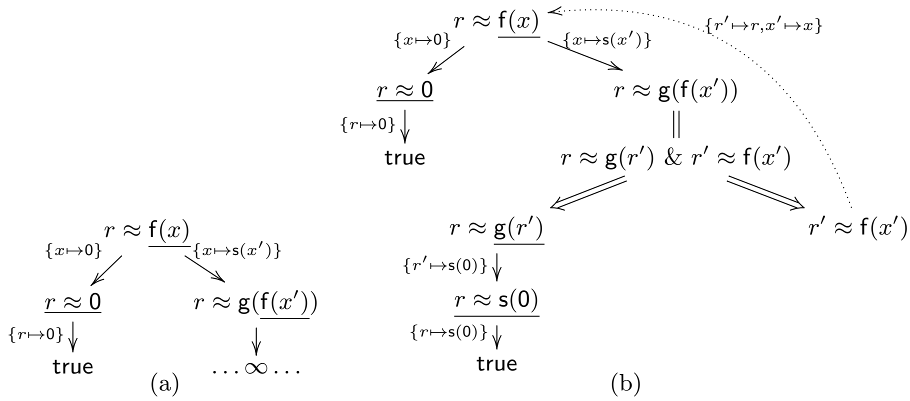

Fig. 1. Building a finite representation of the narrowing space for $\mathsf{f}(x)$.

use of these operators can be used to construct a finite tree that still represents all possible derivations. In this work, we present a new approach to produce a finite data structure that still represents all the narrowing derivations for any given 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 $x \approx e|_p \;\&\; e[x]_p$, where the subterm $e|_p$ of e is replaced by a fresh variable x not 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):

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

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 tree 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. We consider these dotted arrows *implicit* to keep the data structure a tree (i.e., they are only used to identify the occurrence of a variant of the given node, and could be replaced by just adding the information locally to this node).

Designing a technique for producing finite 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 never produce a

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

computed answer (which is useful, e.g., in the context of the *more specific transformation* recently introduced in [23]), 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 compact representations of the computed answers of a term from its finite tree. E.g., the set of computed answers $\Gamma_{\mathsf{f}(x)}$ associated to the narrowing tree depicted in Figure 1 (a) can be succinctly represented by

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

which is extracted from the finite tree in Figure 1 (b). 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 original narrowing tree is infinite. In this case, 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 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. Proofs of technical results can be found in the appendix.

2 Preliminaries

We assume familiarity with basic concepts of term rewriting and narrowing. We refer the reader to, e.g., [7], [25], 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. 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 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 (lhs) and the right-hand side (rhs) 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 lhs's 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 lhs's 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. The instantiated lhs $l\sigma$ 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 [28] mainly extends term rewriting by replacing pattern matching with unification, so that terms containing logic variables can also be reduced by non-deterministically instantiating these variables. Formally, given a TRS $\mathcal{R}$ and two terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have that $s \leadsto_{\mathcal{R}} t$ is a *narrowing step* iff there exist[4]

- a non-variable 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.

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.

Innermost Narrowing. In this paper, we consider a particular narrowing strategy called *innermost narrowing* (see, e.g., [12]). Innermost narrowing only reduces subterms of the form $f(t_1, \ldots, t_n)$, with f a defined function symbol and $t_1, \ldots, t_n$ constructor terms; if there are several such subterms, we consider in this paper that the leftmost one is selected. Innermost narrowing steps are denoted using arrows of the form "$\stackrel{i}{\leadsto}$". A well-known result for innermost narrowing states its completeness for (confluent and terminating) constructor TRSs that are *completely defined* (CD) (or *sufficiently complete*): TRSs in which no function symbol occurs in any ground term in normal form (i.e., functions are always reducible on all ground terms). The CD condition is common when using types and each function is defined for all constructors of its argument types. It is easy to extend innermost narrowing to incompletely defined functions, by just adding a so called innermost *reflection* rule which skips an innermost function call that cannot be reduced [17], given rise to so called innermost *basic* narrowing. For the sake of simplicity, here we assume that the CD condition holds for all functions so that innermost narrowing suffices to compute all answers.

Example 2. Consider the TRS

$$\mathcal{R} = \{\ \mathsf{add}(\mathsf{0}, y) \to y \ \ (R_1), \quad \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y)) \ \ (R_2)\ \}$$

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 innermost narrowing derivations starting from $\mathsf{add}(x, \mathsf{s}(\mathsf{0}))$, e.g.,

$$\begin{array}{l}
\mathsf{add}(x, \mathsf{s}(\mathsf{0})) \stackrel{i}{\leadsto}_{\epsilon, R_1, \{x \mapsto \mathsf{0}\}} \quad \mathsf{s}(\mathsf{0}) \\
\mathsf{add}(x, \mathsf{s}(\mathsf{0})) \stackrel{i}{\leadsto}_{\epsilon, R_2, \{x \mapsto \mathsf{s}(y_1)\}} \mathsf{s}(\mathsf{add}(y_1, \mathsf{s}(\mathsf{0}))) \stackrel{i}{\leadsto}_{1, R_1, \{y_1 \mapsto \mathsf{0}\}} \mathsf{s}(\mathsf{s}(\mathsf{0})) \\
\ldots
\end{array}$$

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 lhs 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., we say that 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 w.r.t. the position 2.1 (i.e., w.r.t. $\mathsf{g}(y)$ since $x \approx \mathsf{f}(\mathsf{g}(y))|_{2.1} = \mathsf{g}(y)$) returns $x' \approx \mathsf{g}(y)$ & $x \approx \mathsf{f}(x')$, where x' is a fresh variable. Therefore, flattening can be used to *distribute* the narrowing tasks among different equations.

Intuitively speaking, 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 whether 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, we will focus on (unconditional) innermost narrowing (though other narrowing strategies would also be equally appropriate, e.g., basic narrowing or innermost 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 and conjunction:

$$\mathcal{R}_{eq} = \{x \approx x \to \mathsf{true}\} \qquad \mathcal{R}_{\&} = \{\mathsf{true}\ \&\ x \to x,\ \mathsf{false}\ \&\ x \to \mathsf{false}\}$$

Hence, we have that $s \approx t$ holds if s and t are syntactically equal. Also, when using innermost narrowing, we can only reduce $s \approx t$ using the rule $x \approx x \to \mathsf{true}$ if both s and t are constructor terms. Narrowing deals with equations and conjunctions as ordinary terms. We often call such terms *equational* terms to make it explicit that they contain occurrences of "≈" 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 TRS and let t be a term. We define the* success set $\mathcal{S}_{\mathcal{R}}(t)$ *of t in $\mathcal{R}$ as follows:*

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

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

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, 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, 26]. 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 [26]). *Let θ_1 and θ_2 be two idempotent substitutions. Then, we define $\Uparrow$ as follows:*

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

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 CD TRS. Let e_1 & e_2 be an equational term. Then, we have $\mathcal{S}_{\mathcal{R}}(e_1 \mathbin{\&} e_2) = \mathcal{S}_{\mathcal{R}}(e_1) \Uparrow \mathcal{S}_{\mathcal{R}}(e_2)$ 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 CD TRS. Let e_1 & e_2 be an equational term. Then, we have $\mathcal{S}_{\mathcal{R}}(e_1 \mathbin{\&} e_2) = \mathcal{S}_{\mathcal{R}}(e_2 \mathbin{\&} e_1)$ 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 [24]) that will become useful in the next section, and prove its correctness.

Definition 7 (flattening). *Let e be an equational term and $p \in \mathcal{P}os(e)$ be a position of e such that $e|_p$ is not a variable, and the root of $e|_p$ is neither $\approx$ nor &. Then, the* flattening *of e w.r.t. p is given by $x \approx e|_p \mathbin{\&} e[x]_p$.*

We say that a flattening is trivial *when e has an equation $y \approx t$ and flattening just replaces it with $x \approx t \mathbin{\&} y \approx x$ (so that just another level of indirection is created). In the following, we assume that all flattenings are non-trivial.*

The following property states the correctness of the flattening operation:

Theorem 8. *Let $\mathcal{R}$ be a constructor CD TRS. Let e be an equational term and e' be a non-trivial flattening of e w.r.t. some position p. 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

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

4.1 Constructing Finite Narrowing Trees

We produce finite trees representing all the (possibly infinite) narrowing derivations of a term as follows. Basically, 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 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 occurring in t;*
- *a leaf is either a node labeled with* true *(a success node) or a node containing defined functions that cannot be further narrowed, which is labeled with* fail *to make it explicit that it represents a failing derivation;*
- subsumption: *if a node is labeled with a non-constructor 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]
- *otherwise, given a node labeled with e, we expand it (do not care non-deterministically) using one of the following rules:*
  - narrowing: *if e is narrowable, we have an output edge labeled with σ from node e to node e' for each innermost narrowing step $e \overset{i}{\leadsto}_\sigma e'$;*
  - constructor decomposition: *if $e \equiv (y \approx \mathsf{c}(t_1, \ldots, t_n)\ \&\ e')$ ($\mathsf{c} \in \mathcal{C}$), 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 $\{y \mapsto \mathsf{c}(y_1, \ldots, y_n)\}$;*
  - splitting: *if $e \equiv (e_1\ \&\ \cdots\ \&\ e_{n-1}\ \&\ e_n)$, we add output edges from e to new nodes labeled with $e_1, \ldots, e_{n-1}$, and e_n;*
  - flattening: *we add an output edge from node e to a node $y \approx e|_p\ \&\ e[y]_p$, where y is a fresh variable not occurring anywhere in the tree.*

The operations considered in the previous definition can also be found in the literature (perhaps with slightly different definitions). For instance, flattening is introduced in [24] (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., [19]); finally, splitting is considered when proving compositionality results (e.g., [3]) and in the partial evaluation of logic programs [9].

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

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

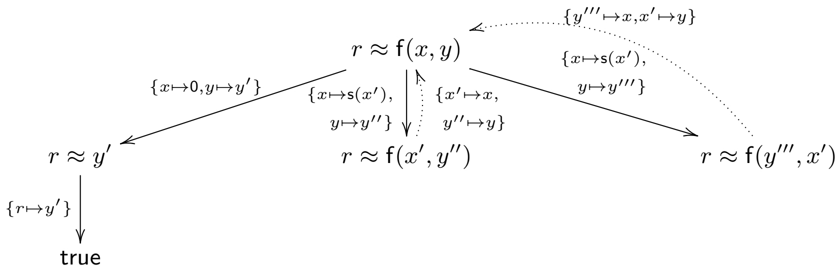

Fig. 2. Finite narrowing tree for $\mathsf{f}(x, y)$.

- narrowing and constructor decomposition: (labeled) solid arrow ($\longrightarrow$);
- subsumption: (labeled) dotted arrow ($\cdots\!>$);
- 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 (where no fixed strategy is considered). 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 $\mathcal{R} = \{\ \mathsf{f}(0, y) \to y,\ \mathsf{f}(\mathsf{s}(x), y) \to \mathsf{f}(x, y),\ \mathsf{f}(\mathsf{s}(x), y) \to \mathsf{f}(y, x)\ \}$. 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 following TRS $\mathcal{R} = \{\mathsf{f}(0, y) \to y,\ \mathsf{f}(\mathsf{s}(x), y) \to \mathsf{c}(\mathsf{f}(x, y), \mathsf{f}(y, x))\}$ 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.

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:

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

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

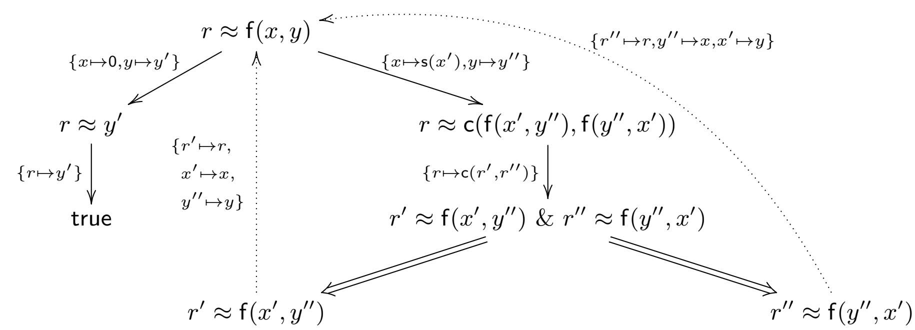

Fig. 3. Finite narrowing tree for $\mathsf{f}(x, y)$.

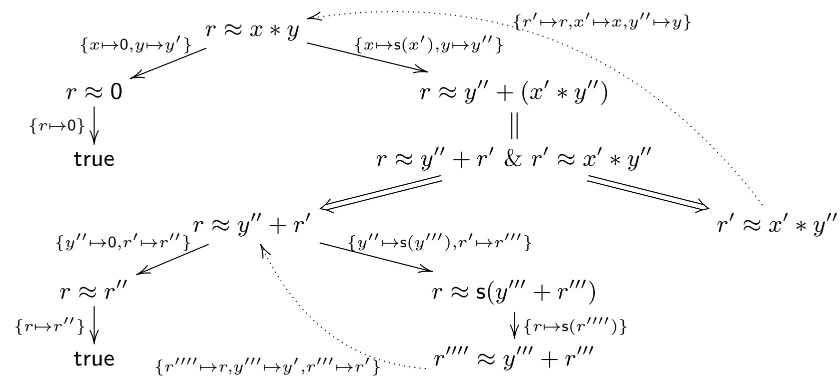

Fig. 4. Finite narrowing tree for $x * y$.

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 flattening and splitting are necessary to produce a finite extended narrowing tree, as shown in Figure 4.

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 $\mathsf{out}(\tau) = \{r \approx x * y \to_{\{x \mapsto \mathsf{0}, y \mapsto y'\}} \tau_1,\ r \approx x * y \to_{\{x \mapsto \mathsf{s}(x'), y \mapsto y''\}} \tau_2\}$, where τ_1 and τ_2 are the subtrees rooted by $r \approx \mathsf{0}$ 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 $r \approx y'' + r'$.

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 replacing a subterm by a fresh variable—and splitting one can keep the set of non-variant terms finite.

Extended narrowing trees represent all possible computed answer substitutions in the following sense:

Definition 13 (success set of an extended narrowing tree). *Let τ_0 be a extended narrowing tree for a term t. Then, the success set of a subtree τ for τ_0, $\mathcal{SS}(\tau)$, is defined as follows:*[8]

$$\mathcal{SS}(\tau) = \begin{cases} \{id\} & \textit{if } \tau \equiv \mathsf{true} \\ \{\,\} & \textit{if } \tau \equiv \mathsf{fail} \textit{ (a failing derivation)} \\ \sigma \cdot \mathcal{SS}(\tau') & \textit{if } \tau \equiv (\ t \underset{\sigma}{\dashrightarrow} \tau'\ ) \\ \mathcal{SS}(\tau') & \textit{if } \tau \equiv (e \Longrightarrow \tau') \\ \mathcal{SS}(\tau_1) \Uparrow \cdots \Uparrow \mathcal{SS}(\tau_n) & \textit{if } \mathsf{out}(\tau) = \{e \Rightarrow \tau_i \mid i = 1, \ldots, n\} \\ \sigma_1 \cdot \mathcal{SS}(\tau_1) \cup \cdots \cup \sigma_n \cdot \mathcal{SS}(\tau_n) & \textit{if } \mathsf{out}(\tau) = \{e \rightarrow_\sigma \tau_i \mid i = 1, \ldots, n\} \end{cases}$$

The correctness of the extended narrowing trees is then stated as follows:

Theorem 14. *Given a finite narrowing tree τ for a term t, $\mathcal{S}_\mathcal{R}(t) = \mathcal{SS}(\tau)$.*

Observe that the four operations—narrowing, constructor decomposition, splitting and flattening—might be applicable to the same node. A *strategy* is needed in order to decide which step should be applied and when. 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.

For instance, a simple strategy that always guarantees the construction of a finite extended narrowing tree may proceed as follows. Basically, every time a node e is narrowed at some position p with $e|_p$ rooted by a defined function symbol: $e \overset{i}{\leadsto}_\sigma e[r]_p\sigma'$ with $\sigma' = \sigma\!\restriction_{\mathcal{V}\mathsf{ar}(e)}$, we apply a flattening step:

$$e[r]_p\sigma' \Longrightarrow x \approx r \ \&\ e[x]_p\sigma'$$

followed by these splitting steps:

$$\begin{tikzcd} & x \approx r \\ x \approx r \ \&\ e[x]_p\sigma' \arrow[ur, Rightarrow] \arrow[r, Rightarrow] \arrow[dr, Rightarrow] & e[x]_p \\ & \widehat{\sigma'} \end{tikzcd}$$

By abuse of notation, for $\sigma' = \{x_1 \mapsto t_n, \ldots, x_n \mapsto t_n\}$, we use $\widehat{\sigma'}$ to denote the

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

[8] Observe that a failing derivation returns an empty set. Here, we assume that both $\sigma \cdot \{\,\} = \{\}$ and $\{\,\} \Uparrow \Theta = \Theta \Uparrow \{\,\} = \{\}$.

equational term $x_1 \approx t_1 \,\&\, \cdots \,\&\, x_n \approx t_n$. Roughly speaking, the construction of the extended narrowing tree will be finite since i) the number of nodes of the form $x \approx r$, with r an rhs of the TRS, is finite modulo variable renaming; ii) the new node $e[x]_p$ contains strictly less defined function symbols than e; and iii) $\widehat{\sigma'}$ only contains constructor symbols, $\approx$, and &.

More refined strategies involve the use of appropriate orders on terms so that flattening and/or splitting steps are only applied when there is a risk of non-termination. We refer the interested reader to [2, 1], where terminating strategies for narrowing-driven partial evaluation are introduced. Similar strategies could be defined using the operations of Definition 9.

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; the success sets of the different branches issuing from a term are put together using the alternative operator; 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.

Definition 15 (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:*

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

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

Example 16. 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:[9]

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

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 $\Gamma_{t_1} = r_1, \ldots, \Gamma_{t_n} = r_n$ 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:

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

   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 $j = 1, \ldots, n$ and $t \in \{t_1, \ldots, t_n\}$.

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.

Example 17. Given the extended narrowing tree shown in Figure 3, we produce the following success set equation:

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

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

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

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

...

Example 18. 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 19. 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 20. *Let $\mathcal{R}$ be a constructor CD TRS and let t be a term. Let τ be a finite extended narrowing tree for t in $\mathcal{R}$ rooted with $x \approx t$, and let $\Gamma_{x\approx t}$ be its associated success set equation. Then, we have $\mathcal{S}_{\mathcal{R}}(x \approx t) = \mathsf{sols}(\Gamma_{x\approx t})$ 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 like Prolog. For instance, the splitting operation is well-known in this context and allows one to partition a query Q into a number of queries $Q_1, \ldots, Q_n$ such that $Q = Q_1, \ldots, Q_n$ (see, e.g., [9] for a precise definition, where the reordering of atoms in a query is also allowed). As for flattening, it can be seen as a simplified version of our notion since predicate symbols cannot be nested. For instance, the flattening of a query $p(X), q(f(Y), Z)$ w.r.t. the position of $f(Y)$ would be $p(X), W = f(Y), q(W, Z)$, where Z is a fresh variable and "=" is the syntactic equality defined by the clause $X = X \leftarrow$. Thus,

$$
\begin{tikzcd}
 & r \approx \mathsf{s}(\mathsf{log}(\mathsf{div}(x, \mathsf{s}(\mathsf{s}(y))), \mathsf{s}(\mathsf{s}(y)))) \arrow[d, "\{r \mapsto \mathsf{s}(r_1)\}"'] \\
 & r_1 \approx \mathsf{log}(\mathsf{div}(x, \mathsf{s}(\mathsf{s}(y))), \mathsf{s}(\mathsf{s}(y))) \arrow[dl, "\{x \mapsto \mathsf{0}\}"'] \arrow[d, "\{x \mapsto \mathsf{s}(x_1)\}"] \\
r_1 \approx \mathsf{log}(\mathsf{0}, \mathsf{s}(\mathsf{s}(y))) \arrow[d, "\{r_1 \mapsto \mathsf{s}(r_2)\}"] & r_1 \approx \mathsf{log}(\mathsf{div}(\mathsf{s}((\mathsf{minus}(x_1, \mathsf{s}(y))), \mathsf{s}(\mathsf{s}(y)))), \mathsf{s}(\mathsf{s}(y))) \arrow[d, "\{x_1 \mapsto \mathsf{s}(x_2)\}"] \\
r_2 \approx \mathsf{log}(\mathsf{0}, \mathsf{s}(\mathsf{s}(y))) \arrow[u, dotted, bend left, "\{r_2 \mapsto r_1\}"] & \tau'
\end{tikzcd}
$$

Fig. 5. Finite narrowing tree for $\mathsf{s}(\mathsf{log}(\mathsf{div}(x, \mathsf{s}(\mathsf{s}(y))), \mathsf{s}(\mathsf{s}(y))))$.

it should not be difficult to adapt the notions of extended narrowing tree and success set equations to logic programming.

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* (MSV) recently introduced in [23]), and so forth. For instance, let us consider the following TRS:

$$
\begin{aligned}
\mathsf{log}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(y))) &\to \mathsf{0} \\
\mathsf{log}(x, \mathsf{s}(\mathsf{s}(y))) &\to \mathsf{s}(\mathsf{log}(\mathsf{div}(x, \mathsf{s}(\mathsf{s}(y))), \mathsf{s}(\mathsf{s}(y)))) \\
\mathsf{div}(\mathsf{0}, \mathsf{s}(y)) &\to \mathsf{0} \\
\mathsf{div}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{s}(\mathsf{div}(\mathsf{minus}(x, y), \mathsf{s}(y))) \\
\mathsf{minus}(x, \mathsf{0}) &\to x \\
\mathsf{minus}(\mathsf{s}(x), \mathsf{s}(y)) &\to \mathsf{minus}(x, y)
\end{aligned}
$$

which is obtained by applying the inverse transformation of [21]. Now, we aim at producing a non-overlapping definition of function log. Unfortunately, by applying the original MSV transformation [23] to the body of the second rule, we construct an incomplete narrowing tree for $r \approx \mathsf{s}(\mathsf{log}(\mathsf{div}(x, \mathsf{s}(\mathsf{s}(y))), \mathsf{s}(\mathsf{s}(y))))$ that still produces overlapping (partial) computed answers. In this context, we can construct the finite extended narrowing tree shown in Figure 5 instead.

In the extended narrowing tree, one can easily see that the leftmost subtree rooted by $r_1 \approx \mathsf{log}(\mathsf{0}, \mathsf{s}(\mathsf{s}(y)))$ cannot produce any computed answer since there is no leaf. Therefore, it is still safe if the MSV transformation ignores the substitution $\{x \mapsto \mathsf{0}\}$ of the leftmost subtree. Thus we know that the variable x of the rule $\mathsf{log}(x, \mathsf{s}(\mathsf{s}(y))) \to \mathsf{s}(\mathsf{log}(\mathsf{div}(x, \mathsf{s}(\mathsf{s}(y))), \mathsf{s}(\mathsf{s}(y))))$ needs to be bound only to $\mathsf{s}(\mathsf{s}(x_2))$, so that the following non-overlapping definition of log is obtained:[10]

$$
\begin{aligned}
\mathsf{log}(\mathsf{s}(\mathsf{0}), \mathsf{s}(\mathsf{s}(y))) &\to \mathsf{0} \\
\mathsf{log}(\mathsf{s}(\mathsf{s}(x_2)), \mathsf{s}(\mathsf{s}(y))) &\to \mathsf{s}(\mathsf{log}(\mathsf{div}(\mathsf{s}(\mathsf{s}(x_2)), \mathsf{s}(\mathsf{s}(y))), \mathsf{s}(\mathsf{s}(y))))
\end{aligned}
$$

[10] Actually, since the initial TRS is not completely-defined, a reflection rule for innermost narrowing is required, as discussed in Section 2. However, since the result would be the same, we prefer to ignore this rule here and keep the example simpler.

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 [21, 20, 22]).

Acknowledgements

We thank the anonymous reviewers and the participants of LOPSTR 2013 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 Transactions on Programming Languages and Systems 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.) Proceedings of the 4th International Conference on 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: Proceedings of the 9th International Symposium on Programming Languages: Implementations, Logics, and Programs. Lecture Notes in Computer Science, vol. 1292, pp. 1–15. Springer (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: Proietti, M. (ed.) Proceedings of the 5th International Workshop on Logic Programming Synthesis and Transformation. Lecture Notes in Computer Science, vol. 1048, pp. 219–233. Springer (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: Proceedings of the 24th International Conference on Rewriting Techniques and Applications. LIPIcs, vol. 21, pp. 81–96. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2013)
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: Proceedings of the 18th International Conference on Rewriting Techniques and Applications. Lecture Notes in Computer Science, vol. 4533, pp. 153–168. Springer (2007)
12. Fribourg, L.: SLOG: A Logic Programming Language Interpreter Based on Clausal Superposition and Rewriting. In: Proceedings of the Symposium on Logic Programming. pp. 172–185. IEEE Press (1985)
13. Gnaedig, I., Kirchner, H.: Proving weak properties of rewriting. Theoretical Computer Science 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.) Proceedings of the 1989 North American Conf. on Logic Programming. pp. 369–389. The MIT Press, Cambridge, MA (1989)
17. Hölldobler, S.: Foundations of Equational Logic Programming, Lecture Notes in Artificial Intelligence, vol. 353. Springer (1989)
18. 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)
19. Middeldorp, A., Okui, S.: A Deterministic Lazy Narrowing Calculus. Journal of Symbolic Computation 25(6), 733–757 (1998)
20. 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 (2005), in Japanese
21. 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. Lecture Notes in Computer Science, vol. 3467, pp. 264–278. Springer (2005)
22. 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. LIPIcs, vol. 10, pp. 283–298. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2011)
23. Nishida, N., Vidal, G.: Computing More Specific Versions of Conditional Rewriting Systems. In: Albert, E. (ed.) Proceedings of the 20th International Symposium on Logic-Based Program Synthesis and Transformation. Lecture Notes in Computer Science, vol. 7844, pp. 137–154. Springer (2013)
24. Nutt, W., Réty, P., Smolka, G.: Basic Narrowing Revisited. Journal of Symbolic Computation 7(3/4), 295–317 (1989)
25. Ohlebusch, E.: Advanced Topics in Term Rewriting. Springer-Verlag, London, UK (2002)
26. Palamidessi, C.: Algebraic Properties of Idempotent Substitutions. In: Paterson, M. (ed.) Proceedings of 17th Int'l Colloquium on Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 443, pp. 386–399. Springer (1990)
27. 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. pp. 228–239. ACM Press (2005)
28. Slagle, J.R.: Automated theorem-proving for theories with simplifiers, commutativity and associativity. Journal of the ACM 21(4), 622–642 (1974)

A Proofs of Technical Results

First, we consider the proof of compositionality for innermost narrowing (Section 3). In order to prove it, we first need the following property for the parallel composition operator:

Lemma 21 ([?,26]). *Let θ_1 and θ_2 be idempotent substitutions. Then*

$$\theta_1 \Uparrow \theta_2 = \theta_1 \mathsf{mgu}(\widehat{\theta_2}\theta_1) = \theta_2 \mathsf{mgu}(\widehat{\theta_1}\theta_2)$$

Let us now proceed with the compositionality result:

Theorem 5. *Let $\mathcal{R}$ be a constructor CD 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.*

Proof. To simplify the proof, we assume that both derivations consider the same renamings for program rules and thus the computed substitutions are just equal (instead of equal up to variable renaming).

(Soundness) Consider a successful derivation of the form $e_1 \;\&\; e_2 \stackrel{i}{\leadsto}{}^*_\theta \mathsf{true}$. We will prove that there exist narrowing derivations $e_1 \stackrel{i}{\leadsto}{}^*_{\sigma_1} \mathsf{true}$ and $e_2 \stackrel{i}{\leadsto}{}^*_{\sigma_2} \mathsf{true}$ such that $\theta = \sigma_1 \Uparrow \sigma_2 \neq \mathsf{fail}$. Following the innermost strategy, the former derivation can be written as follows:

$$e_1 \;\&\; e_2 \stackrel{i}{\leadsto}{}^*_{\theta_1} \mathsf{true} \;\&\; e_2\theta_1 \stackrel{i}{\leadsto}{}^*_{\theta_2} \mathsf{true} \;\&\; \mathsf{true} \stackrel{i}{\leadsto} \mathsf{true}$$

with $\theta = \theta_1\theta_2$. Hence, we have $e_1 \stackrel{i}{\leadsto}{}^*_{\sigma_1} \mathsf{true}$ such that $\sigma_1 = \theta_1$. Therefore, we have to prove that $e_2\theta_1 \stackrel{i}{\leadsto}{}^*_{\theta_2} \mathsf{true}$ implies that $e_2 \stackrel{i}{\leadsto}{}^*_{\sigma_2} \mathsf{true}$ with $\theta_1\theta_2 = \theta_1 \Uparrow \sigma_2 \neq \mathsf{fail}$. We prove this claim by induction on the length n of the former derivation.

Base case ($n = 0$). This case follows trivially.

Inductive case ($n > 0$). Here, we consider that the first derivation has the form $e_2\theta_1 \stackrel{i}{\leadsto}_{p,l\to r,\theta_{21}} (e_2\theta_1[r]_p)\theta_{21} \stackrel{i}{\leadsto}{}^*_{\theta_{22}} \mathsf{true}$ with $\theta_{21} = \mathsf{mgu}(\{e_2|_p\theta_1 = l\})$ and $\theta_2 = \theta_{21}\theta_{22}$. Therefore, since θ_1 is a constructor substitution, we have that p is also an innermost narrowing position of e_2 and $e_2 \stackrel{i}{\leadsto}_{p,l\to r,\sigma_{21}} (e_2[r]_p)\sigma_{21}$, where $\sigma_{21} = \mathsf{mgu}(\{e_2|_p = l\})$. Now, we have that $(e_2\theta_1[r]_p)\theta_{21} = (e_2[r]_p)\theta_1\theta_{21}$ since $l \to r$ has fresh variables. Moreover, the following sequence of equivalences hold:

$$\begin{aligned}
\theta_1\theta_{21} &= \theta_1\mathsf{mgu}(\{(e_2\theta_1)|_p = l\}) && \text{(since } \mathcal{D}\mathsf{om}(\theta_1) \cap \mathcal{V}\mathsf{ar}(l) = \varnothing\text{)}\\
&= \theta_1\mathsf{mgu}(\widehat{\mathsf{mgu}}\{e_2|_p = l\}\theta_1) &&\\
&= \theta_1\mathsf{mgu}(\widehat{\sigma_{21}}\theta_1) && \text{(by Lemma 21)}\\
&= \sigma_{21}\mathsf{mgu}(\widehat{\theta_1}\sigma_{21}) &&
\end{aligned}$$

Hence, we have $(e_2\theta_1[r]_p)\theta_{21} = (e_2[r]_p)\sigma_{21}\mathsf{mgu}(\widehat{\theta_1}\sigma_{21}) \stackrel{i}{\leadsto}{}^*_{\theta_{22}} \mathsf{true}$. By the induction hypothesis, we have that $(e_2[r]_p)\sigma_{21} \stackrel{i}{\leadsto}{}^*_{\sigma_{22}} \mathsf{true}$ with $\mathsf{mgu}(\widehat{\theta_1}\sigma_{21})\theta_{22} = \mathsf{mgu}(\widehat{\theta_1}\sigma_{21}) \Uparrow \sigma_{22} \neq \mathsf{fail}$. Putting all pieces together, we have

$$e_2 \stackrel{i}{\leadsto}_{p,l\to r,\sigma_{21}} (e_2[r]_p)\sigma_{21} \stackrel{i}{\leadsto}{}^*_{\sigma_{22}} \mathsf{true}$$

with

$$\begin{aligned}
\theta_1\theta_2 &= \theta_1\theta_{21}\theta_{22} \\
&= \theta_1\sigma_{21}\mathsf{mgu}(\widehat{\theta_1}\sigma_{21})\theta_{22} \\
&= \theta_1\sigma_{21}(\mathsf{mgu}(\widehat{\theta_1}\sigma_{21}) \Uparrow \sigma_{22}) && \text{(by Lemma 21)} \\
&= \theta_1\sigma_{21}(\sigma_{22}\mathsf{mgu}(\widehat{\mathsf{mgu}}(\widehat{\theta_1}\sigma_{21})\sigma_{22})) \\
&= \theta_1(\sigma_{21}\sigma_{22})\mathsf{mgu}(\widehat{\mathsf{mgu}}(\widehat{\theta_1})(\sigma_{21}\sigma_{22}))) \\
&= \theta_1(\sigma_{21}\sigma_{22})\mathsf{mgu}(\widehat{\theta_1}(\sigma_{21}\sigma_{22}))) && \text{(by Lemma 21)} \\
&= \theta_1 \Uparrow (\sigma_{21}\sigma_{22}) \\
&= \theta_1 \Uparrow \sigma_2
\end{aligned}$$

and the claim follows.

(Completeness) Now, we consider successful derivations of the form $e_1 \overset{i}{\leadsto}{}^*_{\sigma_1}\mathsf{true}$ and $e_2 \overset{i}{\leadsto}{}^*_{\sigma_2}\mathsf{true}$ such that $\sigma_1 \Uparrow \sigma_2 \neq \mathsf{fail}$. We will prove that there exists a narrowing derivation $e_1\ \&\ e_2 \overset{i}{\leadsto}{}^*_{\theta}\ \mathsf{true}$ such that $\theta = \sigma_1 \Uparrow \sigma_2$. Trivially, we have $e_1\ \&\ e_2 \overset{i}{\leadsto}{}^*_{\theta_1}\mathsf{true}\ \&\ e_2\theta_1$ such that $\sigma_1 = \theta_1$. Therefore, we have to prove that $e_2 \overset{i}{\leadsto}{}^*_{\sigma_2}\mathsf{true}$ with $\sigma_1 \Uparrow \sigma_2 = \theta_1 \Uparrow \sigma_2 \neq \mathsf{fail}$ implies $e_2\theta_1 \overset{i}{\leadsto}{}^*_{\theta_2}\mathsf{true}$ with $\theta = \theta_1\theta_2 = \theta_1 \Uparrow \sigma_2$. We prove this claim by induction on the length n of the former derivation $e_2 \overset{i}{\leadsto}{}^*_{\sigma_2}\mathsf{true}$.

Base case ($n = 0$). This case follows trivially.

Inductive case ($n > 0$). Here, we consider that the first derivation has the form $e_2 \overset{i}{\leadsto}_{p,l\to r,\sigma_{21}} (e_2[r]_p)\sigma_{21} \overset{i}{\leadsto}{}^*_{\sigma_{22}}\mathsf{true}$ with $\sigma_{21} = \mathsf{mgu}(\{e_2|_p = l\})$ and $\sigma_2 = \sigma_{21}\sigma_{22}$. Now, since $\theta_1 \Uparrow \sigma_2 \neq \mathsf{fail}$ and $\sigma_{21} \leq \sigma_2$, we have that $\theta_1 \Uparrow \sigma_{21} \neq \mathsf{fail}$. Therefore, by Lemma 21 and the fact that $\mathcal{D}\mathsf{om}(\theta_1) \cap \mathcal{V}\mathsf{ar}(l) = \varnothing$, we have $\theta_1 \Uparrow \sigma_{21} = \theta_1\mathsf{mgu}(\widehat{\sigma_{21}}\theta_1) = \theta_1\mathsf{mgu}(\widehat{\mathsf{mgu}}(\{e_2|_p = l\})\theta_1) = \theta_1\mathsf{mgu}(\widehat{\mathsf{mgu}}(\{(e_2\theta_1)|_p = l\})) = \theta_1\mathsf{mgu}(\{(e_2\theta_1)|_p = l\}) \neq \mathsf{fail}$ and, thus, $\mathsf{mgu}(\{(e_2\theta_1)|_p = l\}) \neq \mathsf{fail}$. Let us recall $\theta_{21} = \mathsf{mgu}(\{(e_2\theta_1)|_p = l\})$. Then, since θ_1 is a constructor substitution, p is also an innermost narrowing position of $e_2\theta_1$ and we have $e_2\theta_1 \overset{i}{\leadsto}_{p,l\to r,\theta_{21}} (e_2\theta_1[r]_p)\theta_{21}$.

Now, we have that $(e_2\theta_1[r]_p)\theta_{21} = (e_2[r]_p)\theta_1\theta_{21}$ since $l \to r$ has fresh variables. As in the previous case, the following sequence of equivalences hold:

$$\begin{aligned}
\theta_1\theta_{21} &= \theta_1\mathsf{mgu}(\{(e_2\theta_1)|_p = l\}) && \text{(since } \mathcal{D}\mathsf{om}(\theta_1) \cap \mathcal{V}\mathsf{ar}(l) = \varnothing\text{)} \\
&= \theta_1\mathsf{mgu}(\widehat{\mathsf{mgu}}\{e_2|_p = l\}\theta_1) \\
&= \theta_1\mathsf{mgu}(\widehat{\sigma_{21}}\theta_1) && \text{(by Lemma 21)} \\
&= \sigma_{21}\mathsf{mgu}(\widehat{\theta_1}\sigma_{21})
\end{aligned}$$

Hence, we have $(e_2\theta_1[r]_p)\theta_{21} = (e_2[r]_p)\sigma_{21}\mathsf{mgu}(\widehat{\theta_1}\sigma_{21})$. Since $(e_2[r]_p)\sigma_{21} \overset{i}{\leadsto}{}^*_{\sigma_{22}}\mathsf{true}$ with $\mathsf{mgu}(\widehat{\theta_1}\sigma_{21}) \Uparrow \sigma_{22} \neq \mathsf{fail}$, by the induction hypothesis, we have $(e_2\theta_1[r]_p)\theta_{21} = (e_2[r]_p)\sigma_{21}\mathsf{mgu}(\widehat{\theta_1}\sigma_{21}) \overset{i}{\leadsto}{}^*_{\theta_{22}}\mathsf{true}$ with $\mathsf{mgu}(\widehat{\theta_1}\sigma_{21}) \Uparrow \sigma_{22} = \mathsf{mgu}(\widehat{\theta_1}\sigma_{21})\theta_{22} \neq \mathsf{fail}$. Putting all pieces together, we have

$$e_2\theta_1 \overset{i}{\leadsto}_{p,l\to r,\theta_{21}} (e_2\theta_1[r]_p)\theta_{21} \overset{i}{\leadsto}{}^*_{\theta_{22}}\mathsf{true}$$

with

$$\begin{aligned}
\theta_1\theta_2 &= \theta_1\theta_{21}\theta_{22} \\
&= \theta_1\sigma_{21}\mathsf{mgu}(\widehat{\theta_1}\sigma_{21})\theta_{22} \\
&= \theta_1\sigma_{21}(\mathsf{mgu}(\widehat{\theta_1}\sigma_{21}) \Uparrow \sigma_{22}) && \text{(by Lemma 21)} \\
&= \theta_1\sigma_{21}(\sigma_{22}\mathsf{mgu}(\widehat{\mathsf{mgu}}(\widehat{\theta_1}\sigma_{21})\sigma_{22})) \\
&= \theta_1(\sigma_{21}\sigma_{22})\mathsf{mgu}(\widehat{\mathsf{mgu}}(\widehat{\theta_1})(\sigma_{21}\sigma_{22}))) \\
&= \theta_1(\sigma_{21}\sigma_{22})\mathsf{mgu}(\widehat{\theta_1}(\sigma_{21}\sigma_{22}))) && \text{(by Lemma 21)} \\
&= \theta_1 \Uparrow (\sigma_{21}\sigma_{22}) \\
&= \theta_1 \Uparrow \sigma_2
\end{aligned}$$

and the claim follows. □

The following corollary is a consequence of the theorem above:

Corollary 6. *Let $\mathcal{R}$ be a constructor CD 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.*

Proof. By Theorem 5, we have $\mathcal{S}_\mathcal{R}(e_1 \;\&\; e_2) = \mathcal{S}_\mathcal{R}(e_1) \Uparrow \mathcal{S}_\mathcal{R}(e_2) = \mathcal{S}_\mathcal{R}(e_2) \Uparrow \mathcal{S}_\mathcal{R}(e_1) = \mathcal{S}_\mathcal{R}(e_2 \;\&\; e_1)$, since $\Uparrow$ is trivially commutative.

Now, we consider the proof of correctness for the flattening transformation:

Theorem 8. *Let $\mathcal{R}$ be a constructor CD TRS. Let e be an equational term and e' be a non-trivial flattening of e w.r.t. some position p. Then, we have $\mathcal{S}_\mathcal{R}(e) = \mathcal{S}_\mathcal{R}(e')$ $[\mathcal{V}\mathsf{ar}(e)]$ up to variable renaming.*

Proof. We consider an equational term e and its flattening $x \approx e|_p \;\&\; e[x]_p$, where x is a fresh variable not occurring in e and $p \in \mathcal{P}os(e)$ with $p \neq \epsilon$.

(Soundness) Since we consider innermost narrowing, we can write any successful derivation for e as follows:

$$e[e|_p]_p \overset{i}{\leadsto}{}^*_{\sigma_1} (e'[e|_p]_p)\sigma_1 \overset{i}{\leadsto}{}^*_{\sigma_2} (e'[c]_p)\sigma_1\sigma_2 \overset{i}{\leadsto}{}^*_{\sigma_3} \mathsf{true}$$

where c cannot be further narrowed, i.e., after some initial subderivation, once we start narrowing $e|_p$ it should continue until we reach a constructor term. By Corollary 6, we have that the equations in a conjunction can be narrowed in any order without affecting to the computed answers. Hence, we can consider the following derivation $x \approx e|_p \;\&\; e[x]_p \overset{i}{\leadsto}{}^*_{\sigma'_1} (x \approx e|_p \;\&\; e'[x]_p)\sigma'_1 \overset{i}{\leadsto}{}^*_{\sigma'_2} (x \approx c \;\&\; e[x]_p)\sigma'_1\sigma'_2$, where σ_i and σ'_i are equal up to variable renaming, $i = 1, 2$. Now, by narrowing the equality (observe that x cannot be bound by $\sigma'_1\sigma'_2$ since it was a fresh variable), we get the derivation $(x \approx c \;\&\; e[x]_p)\sigma'_1\sigma'_2 \overset{i}{\leadsto}_{\{x \mapsto c\sigma'_1\sigma'_2\}}$ $(\mathsf{true} \;\&\; e[c]_p)\sigma'_1\sigma'_2 \overset{i}{\leadsto} (e[c]_p)\sigma'_1\sigma'_2$, and the claim follows.

(Completeness) This proof is analogous to the previous case. □

Let us now consider the correctness of extended narrowing trees. For simplicity, in the following we often use the term e labeling a node to refer to this node.

Lemma 22. *Let τ_0 be a finite extended narrowing tree for a term t. Then, for any subtree τ of τ_0, $\mathcal{SS}(\tau)$ is a set of idempotent substitutions.*

Proof. This lemma follows from the construction of τ since in unfolding nodes, fresh variables are introduced. $\square$

Lemma 23. *Let τ_0 be a finite extended narrowing tree for a term t, τ be a subtree of τ, and e be a term which is a label of the root node for τ. Then,*

- *(Completeness) for any constructor term u, if τ is not a leaf itself subsuming another node and $e \overset{i}{\rightsquigarrow}{}^{*}_{\theta}\, u$, then $\theta \in \mathcal{SS}(\tau)$, and*
- *(Soundness) for any $k > 0$ and any substitution $\theta \in \mathcal{SS}(\tau)$ such that θ is obtained from the k iteration of $\mathcal{SS}$, there exists a constructor term u such that $e \overset{i}{\rightsquigarrow}{}^{*}_{\theta}\, u$.*

Proof. In the proof below, we do not consider any node unfolded by constructor decomposition because such a node can be unfolded by flattening with providing the same substitutions; a subtree $y \approx \mathsf{c}(t_1, \ldots, t_n) \mathbin{\&} e' \xrightarrow[\{y \mapsto \mathsf{c}(y_1,\ldots,y_n)\}]{} \tau'$ such that the root of τ' is $y_1 \approx t_1 \mathbin{\&} \cdots \mathbin{\&} y_n \approx t_n \mathbin{\&} e'$ is equivalent to the following subtree:

$$
\begin{array}{ccc}
 & y \approx \mathsf{c}(t_1, \ldots, t_n) \mathbin{\&} e' & \\
 & \| & \\
 & y \approx \mathsf{c}(y_1, \ldots, y_n) \mathbin{\&} y_1 \approx t_1 \mathbin{\&} \cdots \mathbin{\&} y_n \approx t_n \mathbin{\&} e' & \\
 \swarrow\!\!\swarrow & & \searrow\!\!\searrow \\
 y \approx \mathsf{c}(y_1, \ldots, y_n) & & \tau' \\
 \big\downarrow{\scriptstyle \{y \mapsto \mathsf{c}(y_1,\ldots,y_n)\}} & & \\
 \mathsf{true} & &
\end{array}
$$

It follows from Lemma 21 that the sets of substitutions provided by these trees are equivalent.

(Completeness) Before proving this case, we prepare a weight function w for expressions as follows:

- $w(x \approx t) = \sum_{p \in \mathcal{P}os(t), t|_p \in \mathcal{F}} \mathsf{length}(p)$, and
- $w(e_1 \mathbin{\&} \cdots \mathbin{\&} e_n) = (n-1) + \sum_{i=1}^{n} w(e_i)$ where $n > 1$.

Roughly speaking, $w(e)$ is the summation of the depth of occurrences of function symbols in e with the number of occurrences of &. For an expression e and its flattening e', $w(e) \geq w(e')$ because e' is of the form $x \approx e|_p \mathbin{\&} e[x]_p$, $p \neq \epsilon$, and $e|_p$ is not a variable. For expressions e_1 and e_2, $w(e_1 \mathbin{\&} e_2) > w(e_i)$ for $i = 1, 2$.

We prove the first claim by induction on the lexicographic product $(j, w(e), m)$ of the steps j of $e \overset{i}{\rightsquigarrow}{}^{*}_{\theta}\, u$, the weight of e, and the size m of τ. We make a case distinction depending on how the node e is unfolded.

- Case that τ is true (i.e., e is true). In this case, θ is the identity substitution, and $e = u$. Thus, $\theta = id \in \mathcal{SS}(\tau)$.

– Case that $\tau \equiv (e == \tau')$. In this case, we can assume that the root e' of τ' is of the form $x \approx e|_p \,\&\, e[x]_p$, and hence $w(e) \geq w(e')$. It follows from Theorem 8 that $x \approx e|_p \,\&\, e[x]_p \overset{i}{\rightsquigarrow}{}^*_\theta\, u$ with j' steps where $j' \leq j$.[11]
  • Consider the case that e' is not a leaf subsuming another node. In this case, by the induction hypothesis, $\theta \in \mathcal{SS}(\tau')$, and thus, $\theta \in \mathcal{SS}(\tau)$.
  • Consider the other case. In this case, τ' is e' itself, and there exists another subtree τ'' whose root e'' is a variant of e'. Let σ be a renaming such that $e'\sigma = e''$. Let σ^{-1} be the inverse mapping of σ which is also a variable renaming. Then, by the induction hypothesis, $\sigma^{-1} \cdot \theta \in \mathcal{SS}(\tau'')$, and hence $\theta = \sigma \cdot \sigma^{-1} \cdot \theta \in \mathcal{SS}(\tau)$.

– Case that $\mathsf{out}(\tau) = \{e \Rightarrow \tau_i \mid i = 1, \ldots, n\}$ where $n > 1$. We can assume that e is of the form $e_1 \,\&\, \cdots \,\&\, e_n$, and τ_i is rooted by e_i. It follows from Theorem 5 that $\theta \in \mathcal{S}_\mathcal{R}(e_1) \Uparrow \cdots \Uparrow \mathcal{S}_\mathcal{R}(e_n)$, and hence, there exist substitutions $\theta_1, \ldots, \theta_n$ such that $\theta = \theta_1 \Uparrow \cdots \Uparrow \theta_n$, and $\theta_i \in \mathcal{S}_\mathcal{R}(e_i)$ for each i. Thus, for each i, $e_i \overset{i}{\rightsquigarrow}{}^*_{\theta_i}\, u_i$ for some constructor term u_i with j_i steps, where $j_i \leq j$. If τ_i is e_i itself subsuming another node which is the root of a subtree τ_i', then as in the previous case, we can replace τ_i by τ_i'. For the sake of readability, we assume that none of the roots of $\tau_1, \ldots, \tau_n$ subsumes any other node. By the induction hypothesis, $\theta_i \in \mathcal{SS}(\tau_i)$ for all i. Therefore, $\theta = \theta_1 \Uparrow \cdots \Uparrow \theta_n \in \mathcal{SS}(\tau)$.

– Case that $\mathsf{out}(\tau) = \{e \to_\sigma \tau_i \mid i = 1, \ldots, n\}$. By definition, $e \overset{i}{\rightsquigarrow}_{\theta_i} e_i \overset{i}{\rightsquigarrow}{}^*_{\theta'}\, u$ and $\theta = \theta_i \cdot \theta'$ for some i, where e_i is the root of τ_i. As in the previous case, we assume that the root of τ_i does not subsume any other node. By the induction hypothesis, $\theta' \in \mathcal{SS}(\tau_i)$. Then, by definition, $\theta_i \cdot \theta_i \in \theta_i \cdot \mathcal{SS}(\tau_i) \subseteq \mathcal{SS}(\tau)$. Therefore, $\theta \in \mathcal{SS}(\tau)$.

(Soundness) We prove the second claim by induction on the lexicographic product (k, m) of k and the size m of the subtree rooted by e. Let $\theta \in \mathcal{SS}(\tau)$ such that θ is computed by k applications of $\mathcal{SS}$. We make a case distinction depending on k and how the node e is unfolded. Note that $\mathcal{SS}(\tau) \neq \varnothing$ due to the existence of θ.

– Case that $e \equiv \mathsf{true}$. In this case, θ is the identity substitution, and we can choose e as u in the claim.
– Case that $\tau \equiv (e == \tau')$. In this case, we can assume that the root of τ' is of the form $x \approx e|_p \,\&\, e[x]_p$. By definition, $\theta \in \mathcal{SS}(\tau')$. By the induction hypothesis, $x \approx e|_p \,\&\, e[x]_p \overset{i}{\rightsquigarrow}{}^*_\theta\, u$ for some constructor term u. Thus, it follows from Theorem 8 that $e \overset{i}{\rightsquigarrow}{}^*_\theta\, u$.
– Case that $\mathsf{out}(\tau) = \{e \Rightarrow \tau_i \mid i = 1, \ldots, n\}$ where $n > 1$. We can assume that e is of the form $e_1 \,\&\, \cdots \,\&\, e_n$, and T_i is rooted by e_i, resp. By definition, there exist substitutions $\theta_1, \ldots, \theta_n$ such that $\theta = \theta_1 \Uparrow \cdots \Uparrow \theta_n$, and $\theta_i \in \mathcal{SS}(\tau_i)$ with $k_i \leq k$. By the induction hypothesis, for each i, $e_i \overset{i}{\rightsquigarrow}{}^*_{\theta_i}\, u_i \in T(\mathcal{C}, \mathcal{V})$, and hence $\theta_i \in \mathcal{S}_\mathcal{R}(e_i)$. It follows from Theorem 5 that $\theta \in \mathcal{S}_\mathcal{R}(e_1 \,\&\, \cdots \,\&\, e_n)$, and hence $e \overset{i}{\rightsquigarrow}{}^*_\theta\, u$ for some constructor term u.

[11] From the proof of Theorem 8, it can be seen that $x \approx e|_p \,\&\, e[x]_p \overset{i}{\rightsquigarrow}{}^*_\theta\, u$ with j' ($< j$) steps.

– Case that $\mathsf{out}(\tau) = \{e \to_\sigma \tau_i \mid i = 1, \ldots, n\}$. By definition, there exists a substitution θ' such that $\theta = \theta_i \cdot \theta'$ and $\theta' \in \mathcal{SS}(\tau_i)$ for some i, where e_i is the root of τ_i. By the induction hypothesis, $e_i \overset{i}{\leadsto}{}^*_{\theta'} u$ for some constructor term u. Therefore, $e \overset{i}{\leadsto}_{\theta_i} e_i \overset{i}{\leadsto}{}^*_{\theta'} u$.
– Case that $k > 1$ and $\tau \equiv (\, t \underset{\sigma}{\dashrightarrow} \tau' \,)$. By definition, there exists a substitution θ' such that $\theta = \sigma \cdot \theta'$ and $\theta' \in \mathcal{SS}(\tau')$, where e' is the root of τ' and θ' is obtained by $k-1$ applications of $\mathcal{SS}$. By the induction hypothesis, e' $\overset{i}{\leadsto}{}^*_{\theta'} u$ for some constructor term u. Since σ is a variable renaming, we have a variable renaming which is the inverse mapping of σ. We denote it by σ^{-1}. Then, $e = e'\sigma^{-1} \overset{i}{\leadsto}{}^*_{\sigma^{-1}\cdot\theta'} u$. □

Theorem 14. *Given a finite narrowing tree τ for a term t, $\mathcal{S}_\mathcal{R}(t) = \mathcal{SS}(\tau)$.*

Proof. Trivial by Lemma 23. □

Finally, the proof of correctness for the success set equations may proceed as follows:

Theorem 20. *Let $\mathcal{R}$ be a constructor CD TRS and let t be a term. Let τ be a finite extended narrowing tree for t in $\mathcal{R}$ rooted with $x \approx t$, and let $\Gamma_{x\approx t}$ be its associated success set equation. Then, we have $\mathcal{S}_\mathcal{R}(x \approx t) = \mathsf{sols}(\Gamma_{x\approx t})$ up to variable renaming.*

Proof (Sketch). We should prove that, for every successful derivation, there is a solution of Γ and vice versa.

($\Rightarrow$) We proceed by induction on the length of the successful derivation. Let $x \approx t \overset{i}{\leadsto}_\sigma e$.

– If the extended narrowing tree τ applies a narrowing step, i.e., $\tau \equiv (x \approx t \to_\sigma \tau')$ then we have $\Gamma_{x\approx t} = \sigma \cdot \mathcal{SF}(\tau')$ and the proof follows by the induction hypothesis.
– If the extended narrowing tree τ applies a flattening step, correctness follows by Theorem 8 and the induction hypothesis.
– If the extended narrowing tree τ applies a splitting step, correctness follows by Theorem 5 and the induction hypothesis.
– If the extended narrowing tree τ applies a constructor decomposition step, correctness follows by Theorem 8 (since constructor decomposition can be achieved as a sequence of flattening steps) and the induction hypothesis.
– If the extended narrowing tree τ applies a subsumption step, correctness follows by the fact that $\mathcal{S}(e) = \mathcal{S}(e')$ for all equations e, e' that are variants, and the induction hypothesis.

($\Leftarrow$) The proof is analogous to the previous case. □