More Specific Term Rewriting Systems*

Naoki Nishida
Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, 4648603 Nagoya, Japan
nishida@is.nagoya-u.ac.jp

Germán Vidal
MiST, DSIC, Universitat Politècnica de València
Camino de Vera, s/n, 46022 Valencia, Spain
gvidal@dsic.upv.es

Abstract

There are properties of rewriting systems that are characterized by means of some syntactic conditions (e.g., requiring left-linear and non-overlapping rules for ensuring confluence). Sometimes, though, a given property might hold but the syntactic conditions are not met. This is particularly true when the systems are obtained by some automated transformation. In this paper, we introduce a technique that allows us to replace a rule of a rewriting system by a more specific version (an instance) of this rule so that a particular class of reductions can still be performed in the more specific system. This transformation might help to make some properties explicit (e.g., transforming an overlapping system having the unique normal form property into a non-overlapping one). We provide an algorithm to compute more specific versions of rewriting systems based on narrowing.

1 Introduction

Rewriting systems that are automatically generated (e.g., by program inversion [2, 11, 12, 14, 17, 21, 20, 22] or partial evaluation [1, 5, 6, 24]) have often a poor syntactic structure that might hide some properties. For instance, the rewriting systems generated by program inversion sometimes have overlapping left-hand sides despite the fact that they actually have the *unique normal form* property w.r.t. constructor terms or are even confluent. Consider, e.g., the following TRS from [22] (where we use cons and nil as list constructors):

$$\left\{ \begin{array}{r} \mathsf{inc}(\mathsf{nil}) \to \mathsf{cons}(0, \mathsf{nil}) \\ \mathsf{inc}(\mathsf{cons}(0, xs)) \to \mathsf{cons}(1, xs) \\ \mathsf{inc}(\mathsf{cons}(1, xs)) \to \mathsf{cons}(0, \mathsf{inc}(xs)) \end{array} \right\}$$

and its inversion:

$$\left\{ \begin{array}{l} \mathsf{inc}^{-1}(\mathsf{cons}(0, \mathsf{nil})) \to \mathsf{nil} \\ \mathsf{inc}^{-1}(\mathsf{cons}(1, xs)) \to \mathsf{cons}(0, xs) \\ \mathsf{inc}^{-1}(\mathsf{cons}(0, ys)) \to \mathsf{cons}(1, \mathsf{inc}^{-1}(ys)) \end{array} \right\}$$

Now, observe that every instance of $\mathsf{inc}^{-1}(x)$ using constructor terms has a unique constructor normal form. However, this system is not confluent—consider, e.g., the reductions starting from $\mathsf{inc}^{-1}(\mathsf{cons}(0, \mathsf{nil}))$—and, moreover some of the left-hand sides overlap, thus preventing us from obtaining a typical (deterministic) functional program. In this case, one can observe that

*This work has been partially supported by the Spanish *Ministerio de Economía y Competitividad (Secretaría de Estado de Investigación, Desarrollo e Innovación)* under grant TIN2008-06622-C03-02, by the *Generalitat Valenciana* under grant PROMETEO/2011/052, and by *MEXT KAKENHI* #21700011.

the recursive call to inc^{-1} in the third rule can only bind variable ys to a non-empty list, say $\mathsf{cons}(z, zs)$. Therefore, the system above could be transformed as follows:

$$\left\{ \begin{array}{c} \mathsf{inc}^{-1}(\mathsf{cons}(0, \mathsf{nil})) \to \mathsf{nil} \\ \mathsf{inc}^{-1}(\mathsf{cons}(1, xs)) \to \mathsf{cons}(0, xs) \\ \mathsf{inc}^{-1}(\mathsf{cons}(0, \mathsf{cons}(z, zs))) \to \mathsf{cons}(1, \mathsf{inc}^{-1}(\mathsf{cons}(z, zs))) \end{array} \right\}$$

Now, the transformed system is non-overlapping (and confluent) and, under some conditions, it is equivalent to the original system (roughly speaking, the derivations to constructor terms are the same).

Observe that the computed inverse inc^{-1} of function inc is not trivial. Computing inverse functions automatically is a challenging task in the context of functional programming. In (functional) logic programming, the standard operational semantics is already able to perform some sort of inverse computation automatically. However, it proceeds by a "generate and test" strategy that is not driven by the given output and it is often very inefficient. Therefore, even in this context, the definition of an automatic inversion technique is a useful contribution.

A "more specific" transformation was originally introduced by Marriott et al. [18] in the context of logic programming. In this context, a *more specific version* of a logic program is a version of this program where each clause is further instantiated or removed while preserving the successful derivations of the original program. According to [18], the transformation increases the number of finitely failed goals (i.e., some infinite derivations are transformed into finitely failed ones), detects failure more quickly, etc. In general, the information about the allowed variable bindings which is hidden in the original program may be made explicit in a more specific version, thus improving the static analysis of the program's properties.

In this paper, we adapt the notion of a more specific program to the context of term rewriting systems. In principle, the transformation may achieve similar benefits as in logic programming by making some variable bindings explicit (as illustrated in the transformation of function inc^{-1} above). Adapting this notion to rewriting systems, however, is far from trivial. In contrast to logic programming, there is no notion of "successful" derivation. Therefore, one could in principle aim at preserving all possible reductions in more specific versions. Unfortunately, this is not useful in practice since only a trivial instance would be correct under these assumptions.

Example 1. Consider the following rewriting systems:

$$\mathcal{R} = \left\{ \begin{array}{l} \mathsf{f}(x) \to \mathsf{g}(x) \\ \mathsf{g}(\mathsf{a}) \to \mathsf{b} \end{array} \right\} \qquad \mathcal{R}' = \left\{ \begin{array}{l} \mathsf{f}(\mathsf{a}) \to \mathsf{g}(\mathsf{a}) \\ \mathsf{g}(\mathsf{a}) \to \mathsf{b} \end{array} \right\}$$

Here, one would expect $\mathcal{R}'$ to be a correct more specific version of $\mathcal{R}$ (in the sense of $\to_{\mathcal{R}} = \to_{\mathcal{R}'}$). However, the reduction $\mathsf{f}(\mathsf{b}) \to_{\mathcal{R}} \mathsf{g}(\mathsf{b})$ is not possible in $\mathcal{R}'$.

Hence we should restrict the class of reductions that must be preserved by the more specific version (MSV) transformation in order to make it practical. Essentially, we will only preserve reductions from a constructor instance of the left-hand side of a rewrite rule to a constructor term.

Furthermore, we provide an algorithm for computing more specific versions of a rewriting system. The algorithm is based on constructing finite (possibly incomplete) *narrowing* trees for the right-hand sides of the original rewrite rules. Here, narrowing [25, 16], an extension of term rewriting by replacing pattern matching with unification, is used to *guess* the allowed variable bindings for a given rewrite rule. We prove the correctness of the algorithm (i.e., that it actually outputs a more specific version of the input rewriting system) for both *constructor-based reduction* and non-erasing constructor systems as well as some basic properties like

- is the unique constructor normal form property of the rewrite system preserved?
- is the more specific version the *most* specific one?

This paper is organized as follows. In Section 2, we briefly review some notions and notations of term rewriting and narrowing. Section 3 introduces the notions of more and most specific version of a rewriting system. Then, we introduce an algorithm for computing more specific versions in Section 4 and prove some basic results. Finally, Section 5 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., [4] and [13] 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. 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 $\delta \circ \sigma = \theta$, where "$\circ$" denotes the composition of substitutions (i.e., $\sigma \circ \theta(x) = (x\theta)\sigma$). The *restriction* $\theta\restriction_V$ of a substitution θ to a set of variables V is defined as follows: $x\theta\restriction_V = x\theta$ if $x \in V$ and $x\theta\restriction_V = x$ otherwise. We say that $\theta = \sigma\ [V]$ if $\theta\restriction_V = \sigma\restriction_V$.

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

TRSs and Rewriting. A set of rewrite rules $l \to r$ such that l is a nonvariable term and r is a term whose variables appear in l is called a *term rewriting system* (TRS for short); terms l and r are called the left-hand side and the right-hand side of the rule, respectively. We restrict ourselves to finite signatures and TRSs. Given a TRS $\mathcal{R}$ over a signature $\mathcal{F}$, the *defined* symbols $\mathcal{D}_\mathcal{R}$ 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 $\mathcal{V}$. We sometimes omit $\mathcal{R}$ from $\mathcal{D}_\mathcal{R}$ and $\mathcal{C}_\mathcal{R}$ if it is clear in context. A substitution σ is a *constructor substitution* if $x\sigma \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ 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 $i = 1, \ldots, n$. A TRS $\mathcal{R}$ is *non-erasing* if every rule $l \to r$ in $\mathcal{R}$ satisfies $\mathcal{V}ar(l) = \mathcal{V}ar(r)$.

For a TRS $\mathcal{R}$, we define the associated rewrite relation $\to_{\mathcal{R}}$ as follows: given terms $s, t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, we have $s \to_{\mathcal{R}} t$ iff there exists a position p in s, a rewrite rule $l \to r \in \mathcal{R}$ and a substitution σ with $s|_p = l\sigma$ and $t = s[r\sigma]_p$; the rewrite step is often denoted by $s \to_{p,l\to r} t$ to make explicit the position and rule used in this step. 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 substitution is called *normalized* w.r.t. $\mathcal{R}$ if every variable in the domain is replaced by a normal form w.r.t. $\mathcal{R}$. We sometimes omit "w.r.t. $\mathcal{R}$" if it is clear in context. We denote the set of normal forms by $NF_{\mathcal{R}}$. 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; we also use $t \to^n_{\mathcal{R}} s$ to denote that t can be reduced to s in exactly n rewrite steps.

Narrowing. The *narrowing* principle [25] 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. Conceptually, this is not significantly different from ordinary rewriting when TRSs contain *extra-variables* (i.e., variables that appear in the right-hand side of a rule but not in its left-hand side), as noted in [3]. 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[1]

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

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

Example 2. Consider the TRS

$$\mathcal{R} = \left\{ \begin{array}{ll} (1) & \mathsf{add}(\mathsf{0}, y) \to y \\ (2) & \mathsf{add}(\mathsf{s}(x), y) \to \mathsf{s}(\mathsf{add}(x, y)) \end{array} \right\}$$

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.,

$$\begin{array}{l} \mathsf{add}(x, \mathsf{s}(\mathsf{0})) \leadsto_{\epsilon,(1),\{x\mapsto \mathsf{0}\}} \mathsf{s}(\mathsf{0}) \\ \mathsf{add}(x, \mathsf{s}(\mathsf{0})) \leadsto_{\epsilon,(2),\{x\mapsto \mathsf{s}(y_1)\}} \mathsf{s}(\mathsf{add}(y_1, \mathsf{s}(\mathsf{0}))) \leadsto_{1,(1),\{y_1\mapsto \mathsf{0}\}} \mathsf{s}(\mathsf{s}(\mathsf{0})) \\ \ldots \end{array}$$

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

[1] 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 ordinary unifier—is computed at each narrowing step.

3 More Specific Rewriting Systems

In this section, we introduce the notion of a *more specific* rewriting system. Intuitively speaking, we produce a more specific rewriting system $\mathcal{R}'$ from a rewriting system $\mathcal{R}$ by replacing a rewrite rule $(l \to r) \in \mathcal{R}$ with an *instance* of this rule, i.e., $\mathcal{R}' = (\mathcal{R} \setminus \{l \to r\}) \cup \{l\sigma \to r\sigma\}$, such that $\mathcal{R}'$ is *semantically* equivalent to $\mathcal{R}$ under some conditions.

The key idea is that more specific versions should still allow the same reductions of the original system. However, as mentioned in Section 1, if we aimed at preserving *all* possible rewrite reductions, the resulting notion would be useless since it would never happen in practice. Therefore, we will only focus on preserving *some* reductions. In particular, in order to have a practically applicable technique, we only aim at preserving what we call *successful* reductions. In the following, we denote by $\xrightarrow{s}$ a generic rewrite relation based on some strategy (e.g., innermost reduction $\xrightarrow{i}$).

Definition 3 (successful reduction w.r.t. $\xrightarrow{s}$)**.** *Let $\mathcal{R}$ be a TRS and let $\xrightarrow{s}$ be a rewrite relation. Any suffix of a rewrite reduction $l\sigma \xrightarrow{s}{}^*_{\mathcal{R}} t$ where $l \to r \in \mathcal{R}$, σ is a constructor substitution and $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ is called a* successful *reduction w.r.t. $\xrightarrow{s}$. We say that the successful reduction* starts *from rule $l \to r$.*

The restriction to successful reductions is mainly imposed in order to guarantee the existence of a constructive method to produce more specific versions. As mentioned in Section 1, we will compute the allowed variable bindings by narrowing. However, the use of narrowing has some limitations, as witnessed by the following example:

Example 4. Consider the following TRS:

$$\mathcal{R} = \left\{ \begin{array}{c} \mathsf{f}(x) \to \mathsf{g}(x,x) \\ \mathsf{g}(\mathsf{a},\mathsf{b}) \to \mathsf{b} \\ \mathsf{a} \to \mathsf{b} \end{array} \right\}$$

where b is a constructor symbol. Here, there is no narrowing derivation starting from $\mathsf{g}(x,x)$ since it does not unify with any left-hand side. Consequently, one would conclude that the first rule is useless (and can be removed from $\mathcal{R}$), which is wrong regarding arbitrary reductions (i.e., regarding rewrite derivations where any possible subterm can be selected for reduction) since the following reduction

$$\mathsf{f}(\mathsf{a}) \to_{\mathcal{R}} \mathsf{g}(\mathsf{a},\mathsf{a}) \to_{\mathcal{R}} \mathsf{g}(\mathsf{a},\mathsf{b}) \to_{\mathcal{R}} \mathsf{b}$$

would not be possible in $\mathcal{R}\setminus\{\mathsf{f}(x) \to \mathsf{g}(x,x)\}$ anymore. In contrast, if we only consider innermost reductions, then the use of narrowing would be fine since there is no innermost reduction starting from $\mathsf{f}(s)$ to a constructor value for any constructor term s and, thus, removing the first rule would be correct in this case.

Now, we introduce our notion of *more specific version* of a rewrite rule:

Definition 5 (more specific version of a rule)**.** *Let $\mathcal{R}$ be a TRS and $\xrightarrow{s}$ be a rewrite relation. Let $l \to r \in \mathcal{R}$ be a rewrite rule. We say that a rewrite rule $l' \to r'$ is a* more specific version of $l \to r$ in $\mathcal{R}$ w.r.t. $\xrightarrow{s}$ *if*

- *there exists a substitution σ such that $(l \to r)\sigma = l' \to r'$ and*
- *for all terms s, t, $s \xrightarrow{s}{}^*_{\mathcal{R}} t$ is successful in $\mathcal{R}$ w.r.t. $\xrightarrow{s}$ iff $s \xrightarrow{s}{}^*_{\mathcal{R}'} t$ is successful in $\mathcal{R}'$ w.r.t. $\xrightarrow{s}$, with $\mathcal{R}' = (\mathcal{R} \setminus \{l \to r\}) \cup \{l' \to r'\}$.*

Note that a rewrite rule is always a more specific version of itself. The notion of *most* specific version is then very natural:

Definition 6 (most specific version of a rule)**.** *Let $\mathcal{R}$ be a TRS and let $\overset{s}{\to}$ be a rewrite relation. Let $l \to r \in \mathcal{R}$ be a rewrite rule. We say that a rewrite rule $l' \to r'$ is a* most specific version *of $l \to r$ in $\mathcal{R}$ w.r.t. $\overset{s}{\to}$ if $l' \to r'$ is a more specific version of $l \to r$ in $\mathcal{R}$ w.r.t. $\overset{s}{\to}$ and, for all other more specific versions $l'' \to r''$ of $l \to r$ in $\mathcal{R}$ w.r.t. $\overset{s}{\to}$, we have $(l'' \to r'') \leq (l' \to r')$.*

The notions of more and most specific versions of a rule are extended to TRSs in a stepwise manner: given a TRS $\mathcal{R}$, we first replace a rule of $\mathcal{R}$ by its more (resp. most) specific version thus producing $\mathcal{R}'$, then we replace another rule of $\mathcal{R}'$ by its more (resp. most) specific version, and so forth. We denote each of this steps by $\mathcal{R} \mapsto_{\mathsf{more}} \mathcal{R}'$ (resp. $\mathcal{R} \mapsto_{\mathsf{most}} \mathcal{R}'$). We say that $\mathcal{R}'$ is a *more specific version* of $\mathcal{R}$ if there is a sequence of (zero or more) $\mapsto_{\mathsf{more}}$ steps leading from $\mathcal{R}$ to $\mathcal{R}'$. Also, we say that $\mathcal{R}'$ is a *most specific version* of $\mathcal{R}$ if there is a sequence of (zero or more) $\mapsto_{\mathsf{most}}$ steps and, moreover, no further $\mapsto_{\mathsf{most}}$ step is possible. Note that, given a TRS $\mathcal{R}$ and one of its more specific versions $\mathcal{R}'$, we have that $\to_{\mathcal{R}'} \subseteq \to_{\mathcal{R}}$ (i.e., $NF_{\mathcal{R}} \subseteq NF_{\mathcal{R}'}$) and $\mathcal{D}_{\mathcal{R}} = \mathcal{D}_{\mathcal{R}'}$ (i.e., $\mathcal{C}_{\mathcal{R}} = \mathcal{C}_{\mathcal{R}'}$). For instance, in Example 1, the TRS $\mathcal{R}'$ is clearly a *most* specific version of $\mathcal{R}$.

Observe that a TRS is always a more specific version of itself. Also, we do not need to compute more specific versions of *all* rules since a rule is always a more specific version of itself. Thus one can only replace some rules by their more specific versions and still produce a TRS which is a more specific version. This is in contrast to the case of a *most* specific version of a TRS, which requires computing a most specific version of *each* rule.

Now, we show a basic property of more specific versions of a TRS. In the following, we say that a TRS *has the unique constructor normal form property w.r.t.* $\overset{s}{\to}$ if any instance of the left-hand sides with constructor substitutions have unique constructor normal forms w.r.t. $\overset{s}{\to}$-reductions, i.e., for any rule $l \to r$ and any constructor substitution σ, derivations $l\sigma \overset{s}{\to}^* t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ and $l\sigma \overset{s}{\to}^* t' \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ implies $t = t'$.

Theorem 7. *Let $\mathcal{R}$ be a TRS and let $\mathcal{R}'$ be a more specific version of $\mathcal{R}$ w.r.t. $\overset{s}{\to}$. Then, $\mathcal{R}$ has the unique constructor normal form property w.r.t. $\overset{s}{\to}$ iff so does $\mathcal{R}'$.*

Proof. First, we note that reductions of the form $l\sigma \overset{s}{\to}^* t$ with $l \to r \in \mathcal{R} \cup \mathcal{R}'$, σ a constructor substitution and $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ are successful reductions w.r.t. $\overset{s}{\to}$ by definition (cf. Def. 3). Therefore, the claim follows straightforwardly since successful reductions are preserved by the MSV transformation. □

4 Computing More Specific Versions

In this section, we tackle the definition of a constructive method for computing more specific versions of a TRS w.r.t. *innermost* reduction, which is denoted by $\overset{i}{\to}$. Ideally, to compute a more specific version of a rule $l \to r$ one could proceed as follows:

- first, one determines all substitutions $\sigma_1, \sigma_2, \ldots$ such that $r\sigma_1 \overset{i}{\to}^* t_1$, $r\sigma_2 \overset{i}{\to}^* t_2$, etc., are successful innermost reductions from the right-hand side of the rule $l \to r$;
- then, a substitution which is more general than $\sigma_1, \sigma_2, \ldots$, say σ, is computed;
- finally, one could say that $(l \to r)\sigma$ is a more specific version of $l \to r$.

While this method seems clearly correct, it is generally undecidable. Therefore, in order to approximate substitutions $\sigma_1, \sigma_2, \ldots$ above, we consider *narrowing* [25]. In particular, we will consider *innermost basic* narrowing, denoted by $\overset{i}{\leadsto}$, the counterpart of innermost rewriting. The reader is referred to [7, 10, 15] for a formal definition of innermost basic narrowing. To graphically specify *non-narrowable* subterms, we strike them out.[2] Essentially, innermost basic narrowing is defined as the smallest relation satisfying the following transition rules:

$$\text{(innermost narrowing)} \quad \frac{\varphi_{\mathsf{inn}}(t) = p \ \wedge \ l \to r \in \mathcal{R} \ \wedge \ \sigma = \mathsf{mgu}(t|_p, l)}{t \overset{i}{\leadsto}_\sigma (t[r]_p)\sigma}$$

$$\text{(reflection)} \quad \frac{\varphi_{\mathsf{inn}}(t) = p}{t[s]_p \overset{i}{\leadsto}_{\mathsf{ref}} t[\cancel{s}]_p}$$

Intuitively speaking, given a term t, innermost basic narrowing first selects the position p of an innermost narrowable term $t|_p$ using the narrowing strategy φ_{inn};[3] then, we should

- perform a narrowing step on $t|_p$ using all applicable rules (innermost narrowing) and
- mark $t|_p$ as *non-narrowable*, denoted by $t[\cancel{s}]_p$, so that it is considered a "normal form" in subsequent narrowing steps (reflection).

Note that the innermost narrowable term selected by φ_{inn} may contain some (in principle) narrowable proper subterms that are marked as non-narrowable by previous reflection steps. In the rest of the paper, we will consider the same leftmost innermost strategy in both rewriting and narrowing for simplicity.

Definition 8. *We say that an innermost basic narrowing derivation $s \overset{i}{\leadsto}{}^*_\sigma t$ is* successful *if t is a constructor term. Also, we say that it is a* failing *derivation if t is not a constructor term but it cannot be further narrowed.*

Example 9. Let us consider the following TRS:

$$\mathcal{R} = \left\{ \begin{array}{r} \mathsf{f}(x, y) \to y \\ \mathsf{g}(0) \to 0 \end{array} \right\}$$

Given the initial term $\mathsf{f}(\mathsf{g}(w), \mathsf{g}(w))$, we have the following successful innermost basic narrowing derivations (selected redexes are underlined):

$$\mathsf{f}(\underline{\mathsf{g}(w)}, \mathsf{g}(w)) \overset{i}{\leadsto}_{\mathsf{ref}} \mathsf{f}(\cancel{\mathsf{g}(w)}, \underline{\mathsf{g}(w)}) \overset{i}{\leadsto}_{\{w \mapsto 0\}} \underline{\mathsf{f}(\cancel{\mathsf{g}(0)}, 0)} \overset{i}{\leadsto}_{id} 0$$
$$\mathsf{f}(\underline{\mathsf{g}(w)}, \mathsf{g}(w)) \overset{i}{\leadsto}_{\{w \mapsto 0\}} \mathsf{f}(0, \underline{\mathsf{g}(0)}) \overset{i}{\leadsto}_{\{w \mapsto 0\}} \underline{\mathsf{f}(0, 0)} \overset{i}{\leadsto}_{id} 0$$

The computation of all narrowing derivations starting from a given term is usually represented by means of a narrowing *tree*:

Definition 10 (innermost basic narrowing tree)**.** *Let $\mathcal{R}$ be a TRS and t be a term. A (possibly incomplete) innermost basic narrowing tree for t in $\mathcal{R}$ is a (possibly infinite) directed rooted node- and edge-labeled graph τ built as follows:*

[2] A more elaborated formulation based on representing terms using a pair (skeleton, substitution) can be found in [15].

[3] Given a term t, φ_{inn} selects the position of an innermost narrowable subterm $t|_p$ of t such that $t|_p$ is not struck out; as in the case of innermost rewriting, we do not need to fix a particular selection strategy.

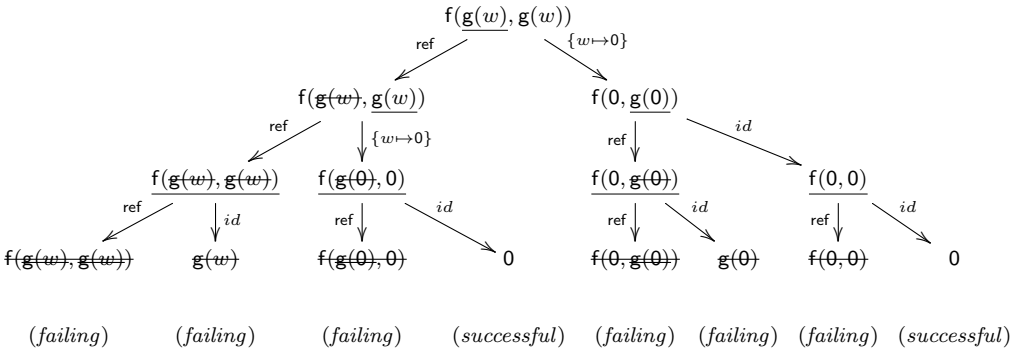

Figure 1: An innermost basic narrowing tree for $\mathsf{f}(\mathsf{g}(w), \mathsf{g}(w))$

- *the root node of τ is labeled with t;*
- *every node s is either a leaf (a node with no output edge) or it is unfolded as follows: there is an output edge from node s to node s' labeled with σ for all innermost basic narrowing steps $s \overset{i}{\leadsto}_\sigma s'$ for the selected innermost narrowable term (including reflection);*
- *at least the root node should be unfolded.*

By abuse of notation, we will denote a finite innermost basic narrowing tree τ (and its subtrees) for a term t as a finite set with the narrowing derivations starting from t in this tree, i.e., $t \overset{i}{\leadsto}^*_\theta s \in \tau$ if there is a root-to-leaf path from t to s in τ labeled with substitutions $\theta_1, \theta_2, \ldots, \theta_n$ such that $t \overset{i}{\leadsto}_{\theta_1} t' \overset{i}{\leadsto}_{\theta_2} \ldots \overset{i}{\leadsto}_{\theta_n} s$ is an innermost basic narrowing derivation and $\theta = \theta_n \circ \cdots \circ \theta_1$.

Example 11. Consider again the TRS $\mathcal{R}$ of Example 9. Given the initial term $\mathsf{f}(\mathsf{g}(w), \mathsf{g}(w))$, innermost basic narrowing constructs the narrowing tree τ shown in Fig. 1. Here, the search space is finite and the tree includes all narrowing derivations, so we can denote τ as follows:

$$\begin{aligned}\tau = \{\ & \mathsf{f}(\underline{\mathsf{g}(w)}, \mathsf{g}(w)) \overset{i}{\leadsto}_{\mathsf{ref}} \mathsf{f}(\mathsf{g}(\cancel{w}), \underline{\mathsf{g}(w)}) \overset{i}{\leadsto}_{\mathsf{ref}} \underline{\mathsf{f}(\mathsf{g}(\cancel{w}), \mathsf{g}(\cancel{w}))} \overset{i}{\leadsto}_{\mathsf{ref}} \mathsf{f}(\mathsf{g}(\cancel{w}), \mathsf{g}(\cancel{w})), \\ & \mathsf{f}(\underline{\mathsf{g}(w)}, \mathsf{g}(w)) \overset{i}{\leadsto}_{\mathsf{ref}} \mathsf{f}(\mathsf{g}(\cancel{w}), \underline{\mathsf{g}(w)}) \overset{i}{\leadsto}_{\mathsf{ref}} \underline{\mathsf{f}(\mathsf{g}(\cancel{w}), \mathsf{g}(\cancel{w}))} \overset{i}{\leadsto}_{id} \mathsf{g}(\cancel{w}), \\ & \ldots \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \}\end{aligned}$$

We note that reflection steps are required when functions are not defined over all possible constructor terms. To be precise, a typical *lifting lemma*—which is used to prove the correspondence between rewriting and narrowing derivations—requires proving that, for all reductions $s\theta \overset{i}{\to}^* t$ with θ a normalized substitution, there exists a narrowing derivation $s \overset{i}{\leadsto}^*_\sigma t'$ with $\sigma' \circ \sigma = \theta$ and $t'\sigma' = t$. This property, however, does not hold without the reflection steps: consider, e.g., the TRS of Example 11 above and the innermost reduction $\mathsf{f}(\mathsf{g}(1), 0) \overset{i}{\to}_{\mathcal{R}} 0$; this reduction cannot be lifted to innermost basic narrowing without a reflection step: $\mathsf{f}(\mathsf{g}(x), y) \overset{i}{\leadsto}_{\mathsf{ref}} \mathsf{f}(\mathsf{g}(\cancel{x}), y) \overset{i}{\leadsto}_{id} y$.

Note that an innermost basic narrowing tree can be *incomplete* in the sense that we do not require all unfoldable nodes to be unfolded (i.e., not all finite derivations should be either successful or failing). Nevertheless, if a node is selected to be unfolded, it should be unfolded in all possible ways using both innermost narrowing and reflection steps on the selected narrowable

term (i.e., one cannot *partially* unfold a node by ignoring some matching rules or dismissing the reflection step, which would give rise to incorrect results). In order to keep the tree finite, one can introduce a heuristics that determines when the construction of the tree should terminate. We consider the definition of a particular strategy for ensuring termination out of the scope of this paper; nevertheless, one could use a simple depth-k strategy (i.e., all narrowing derivations are not further unfolded after k narrowing steps but left incomplete) or some more elaborated strategies based on well-founded or well-quasi orderings [8] (as in narrowing-driven partial evaluation [1]).

Let us now recall the notion of *least (general) generalization* [23], which will be required to compute a common generalization of all instances of a term by a set of narrowing derivations.

Definition 12 (least general generalization [23], *lgg*)**.** *Given two terms s and t, we say that w is a generalization of s and t if $w \leq s$ and $w \leq t$; moreover, it is called the least general generalization of s and t, denoted by $lgg(s,t)$, if $w' \leq w$ for all other generalizations w' of s and t. This notion is extended to sets of terms in the natural way: $lgg(\{t_1,\ldots,t_n\}) = lgg(t_1, lgg(t_2,\ldots lgg(t_{n-1},t_n)\ldots))$ (with $lgg(\{t_1\}) = t_1$ when $n = 1$).*

An algorithm for computing the least general generalization can be found, e.g., in [9]. Let us recall this algorithm for completeness. In order to compute $lgg(s,t)$, this algorithm starts with a tuple $\langle\{s \sqcap_x t\},\ x\rangle$, where x is a fresh variable, and applies the following rules until no rule is applicable:

$$\begin{array}{rcl}
\langle\{f(s_1,\ldots,s_n) \sqcap_x f(t_1,\ldots,t_n)\} \cup P,\ w\rangle & \Rightarrow & \langle\{s_1 \sqcap_{x_1} t_1,\ldots,s_n \sqcap_{x_n} t_n\} \cup P,\ w\sigma\rangle \\
 & & \text{where } \sigma \text{ is } \{x \mapsto f(x_1,\ldots,x_n)\} \\
 & & \text{and } x_1,\ldots,x_n \text{ are fresh variables} \\
\langle\{s \sqcap_x t, s \sqcap_y t\} \cup P,\ w\rangle & \Rightarrow & \langle\{s \sqcap_y t\} \cup P,\ w\sigma\rangle \\
 & & \text{where } \sigma \text{ is } \{x \mapsto y\}
\end{array}$$

Then, the second element of the final tuple is the computed least general generalization.

For instance, the computation of $lgg(\mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{a})),\mathsf{f}(\mathsf{b},\mathsf{g}(\mathsf{b})))$ proceeds as follows:

$$\begin{array}{rcl}
\langle\{\mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{a})) \sqcap_x \mathsf{f}(\mathsf{b},\mathsf{g}(\mathsf{b}))\},\ x\rangle & \Rightarrow & \langle\{\mathsf{a} \sqcap_{x_1} \mathsf{b}, \mathsf{g}(\mathsf{a}) \sqcap_{x_2} \mathsf{g}(\mathsf{b})\},\ \mathsf{f}(x_1,x_2)\rangle \\
 & \Rightarrow & \langle\{\mathsf{a} \sqcap_{x_1} \mathsf{b}, \mathsf{a} \sqcap_{x_3} \mathsf{b}\},\ \mathsf{f}(x_1,\mathsf{g}(x_3))\rangle \\
 & \Rightarrow & \langle\{\mathsf{a} \sqcap_{x_3} \mathsf{b}\},\ \mathsf{f}(x_3,\mathsf{g}(x_3))\rangle
\end{array}$$

Therefore, $lgg(\mathsf{f}(\mathsf{a},\mathsf{g}(\mathsf{a})),\mathsf{f}(\mathsf{b},\mathsf{g}(\mathsf{b}))) = \mathsf{f}(x_3,\mathsf{g}(x_3))$.

We now introduce a constructive algorithm to produce a more specific version of a rule:

Definition 13 (*msv* algorithm)**.** *Let $\mathcal{R}$ be a TRS and let $l \to r \in \mathcal{R}$ be a rewrite rule such that r is not a constructor term. Let τ be a finite (possibly incomplete) innermost basic narrowing tree for r in $\mathcal{R}$ and let $\tau' \subseteq \tau$ be the tree obtained from τ by excluding the failing derivations. We define $msv(\mathcal{R}, l \to r, \tau)$ as follows:*

$$msv(\mathcal{R}, l \to r, \tau) = \begin{cases} (l \to r)\sigma & \text{if } \tau' \neq \varnothing \text{ and } r\sigma = lgg(\{r\theta \mid r \overset{i}{\leadsto}{}^*_\theta\ t \in \tau'\}), \\ & \text{with } \mathcal{D}\mathsf{om}(\sigma) \subseteq \mathcal{V}\mathsf{ar}(r) \\ \bot & \text{otherwise (i.e., } \tau \text{ only contains failing derivations)} \end{cases}$$

where $\bot$ is used to denote that the rule is useless for successful reductions starting from $l \to r$ (i.e., no successful reduction starting from $l \to r$ can use it).

Note that a rule $l \to r$ in a constructor TRS $\mathcal{R}$ with $r \in \mathcal{T}(\mathcal{C},\mathcal{V})$ is a most specific version of itself.

Example 14. Consider the following TRS:

$$\mathcal{R} = \left\{ \begin{array}{ll} (1) & \mathsf{f}(x) \to \mathsf{c}(\mathsf{g}(x), \mathsf{h}(x)) \\ (2) & \mathsf{g}(\mathsf{a}) \to \mathsf{a} \\ (3) & \mathsf{h}(x) \to x \end{array} \right\}$$

We consider that τ follows a depth-2 strategy to compute $msv(\mathcal{R}, (1), \tau)$ (which, in this example, builds the complete tree), so it contains the following derivations:

$$\begin{array}{ll} \mathsf{c}(\underline{\mathsf{g}(x)}, \mathsf{h}(x)) \overset{i}{\leadsto}_{\{x \mapsto \mathsf{a}\}} \mathsf{c}(\mathsf{a}, \underline{\mathsf{h}(\mathsf{a})}) \overset{i}{\leadsto}_{id} \mathsf{c}(\mathsf{a}, \mathsf{a}) & \\ \mathsf{c}(\underline{\mathsf{g}(x)}, \mathsf{h}(x)) \overset{i}{\leadsto}_{\{x \mapsto \mathsf{a}\}} \mathsf{c}(\mathsf{a}, \underline{\mathsf{h}(\mathsf{a})}) \overset{i}{\leadsto}_{\mathsf{ref}} \mathsf{c}(\mathsf{a}, \cancel{\mathsf{h}(\mathsf{a})}) & \text{(failing)} \\ \mathsf{c}(\underline{\mathsf{g}(x)}, \mathsf{h}(x)) \overset{i}{\leadsto}_{\mathsf{ref}} \mathsf{c}(\cancel{\mathsf{g}(x)}, \underline{\mathsf{h}(x)}) \overset{i}{\leadsto}_{id} \mathsf{c}(\cancel{\mathsf{g}(x)}, x) & \text{(failing)} \\ \mathsf{c}(\underline{\mathsf{g}(x)}, \mathsf{h}(x)) \overset{i}{\leadsto}_{\mathsf{ref}} \mathsf{c}(\cancel{\mathsf{g}(x)}, \underline{\mathsf{h}(x)}) \overset{i}{\leadsto}_{\mathsf{ref}} \mathsf{c}(\cancel{\mathsf{g}(x)}, \cancel{\mathsf{h}(x)}) & \text{(failing)} \end{array}$$

Notice that we have no other derivation due to the leftmost innermost selection strategy. Hence, we have only one non-failing derivation and, thus, $lgg(\{\mathsf{c}(\mathsf{g}(\mathsf{a}), \mathsf{h}(\mathsf{a}))\}) = \mathsf{c}(\mathsf{g}(\mathsf{a}), \mathsf{h}(\mathsf{a}))$ and $msv(\mathcal{R}, (1), \tau) = (1)\{x \mapsto \mathsf{a}\} = \mathsf{f}(\mathsf{a}) \to \mathsf{c}(\mathsf{g}(\mathsf{a}), \mathsf{h}(\mathsf{a}))$.

In the above example, *msv* provides a more specific version w.r.t. innermost reduction. Unfortunately, this is not the case in general.

Example 15. Consider the following constructor TRS:

$$\mathcal{R} = \left\{ \begin{array}{ll} (1) & \mathsf{f}(x) \to \mathsf{g}(\mathsf{h}(x)) \\ (2) & \mathsf{g}(\mathsf{s}(x)) \to \mathsf{b} \\ (3) & \mathsf{h}(x) \to \mathsf{s}(\mathsf{i}(x)) \\ (4) & \mathsf{i}(\mathsf{a}) \to \mathsf{b} \end{array} \right\}$$

We consider the complete innermost basic narrowing tree τ to compute $msv(\mathcal{R}, (3), \tau)$, so it contains the following derivations:

$$\begin{array}{ll} \mathsf{s}(\underline{\mathsf{i}(x)}) \overset{i}{\leadsto}_{\{x \mapsto \mathsf{a}\}} \mathsf{s}(\mathsf{b}) & \\ \mathsf{s}(\underline{\mathsf{i}(x)}) \overset{i}{\leadsto} \mathsf{s}(\cancel{\mathsf{i}(x)}) & \text{(failing)} \end{array}$$

Hence, we have only one non-failing derivation and, thus, $lgg(\{\mathsf{s}(\mathsf{i}(\mathsf{a}))\} = \mathsf{s}(\mathsf{i}(\mathsf{a}))$ and

$$msv(\mathcal{R}, (3), \tau) = (3)\{x \mapsto \mathsf{a}\} = \mathsf{h}(\mathsf{a}) \to \mathsf{s}(\mathsf{i}(\mathsf{a}))$$

However, the derivation $\mathsf{f}(\mathsf{b}) \overset{i}{\to}^*_{\mathcal{R}} \mathsf{b}$ is not possible in $\mathcal{R}' = \{(1), (2), \mathsf{h}(\mathsf{a}) \to \mathsf{s}(\mathsf{i}(\mathsf{a})), (4)\}$. Thus, $\mathcal{R}'$ is not a more specific version of $\mathcal{R}$ (although every successful derivation in $\mathcal{R}$ starting from rule (3) is successful in $\mathcal{R}'$).

Observe that an innermost basic tree τ constructed from a rule $l \to r \in \mathcal{R}$ covers successful derivations starting from $l \to r$ only, i.e., derivations from $l\sigma$ with a constructor substitution σ to a constructor term. However, this is not sufficient for other successful derivations starting from a rule other than $l \to r$. To be more precise, given a unsuccessful derivation $l\sigma \overset{i}{\to}_{\mathcal{R}} r\sigma \overset{i}{\to}^*_{\mathcal{R}} t \in NF_{\mathcal{R}} \setminus \mathcal{T}(\mathcal{C}, \mathcal{V})$, it might happen that there exists a successful derivation of the form $s \overset{i}{\to}^*_{\mathcal{R}} s'[l\sigma]_p \overset{i}{\to}_{p, l \to r} s'[r\sigma]_p \overset{i}{\to}^*_{p \leq, \mathcal{R}} s'[t]_p \overset{i}{\to}^*_{\mathcal{R}} u \in \mathcal{T}(\mathcal{C}, \mathcal{V})$. Therefore, one cannot conclude that a failing derivation is indeed useless for *all* successful derivations (but only for those starting from the considered rule).

A naive solution would consist in considering all innermost basic narrowing derivations non-failing. Unfortunately, in this case the notion of *msv* would be useless (generally producing trivial instances).

A better solution to this problem is to restrict reductions to so called *constructor-based* ones $\xrightarrow{c}_{\mathcal{R}}$: $C[l\sigma] \xrightarrow{c}_{\mathcal{R}} C[r\sigma]$ iff $l \to r \in \mathcal{R}$ and σ is a constructor substitution. Constructor-based reduction $\xrightarrow{c}_{\mathcal{R}}$ of a constructor system $\mathcal{R}$ is a special case of innermost reduction $\xrightarrow{i}_{\mathcal{R}}$. Compared with $\xrightarrow{i}_{\mathcal{R}}$, the reduction $\xrightarrow{c}_{\mathcal{R}}$ of constructor system $\mathcal{R}$ has the following property: for any successful derivation $s \xrightarrow{c}{}^*_{\mathcal{R}} s'[l\sigma]_p \xrightarrow{c}_{p,l\to r} s'[r\sigma]_p \xrightarrow{c}{}^*_{p\leq,\mathcal{R}} s'[t]_p \xrightarrow{c}{}^*_{\mathcal{R}} u$ with $t \in NF_{\mathcal{R}}$, we have that t is a constructor term, i.e., $l\sigma \xrightarrow{c}{}^*_{\mathcal{R}} t$ is a successful derivation.

In this context, the reflection steps of innermost basic narrowing are no longer needed to lift constructor-based reductions, as we will show below. A nice property of innermost narrowing is that only constructor substitutions are computed for constructor TRSs (since both the arguments of the left-hand sides and the arguments of the selected redex must be constructor terms; otherwise, the derivation fails).

The correctness of the *msv* algorithm for constructor-based reduction is stated as follows:

Theorem 16. *Let $\mathcal{R}$ be a constructor TRS, $l \to r \in \mathcal{R}$ be a rewrite rule with $r \notin \mathcal{T}(\mathcal{C}, \mathcal{V})$, and τ be a finite (possibly incomplete) innermost narrowing tree for r in $\mathcal{R}$. Then,*

- *If $msv(\mathcal{R}, l \to r, \tau) = (l \to r)\sigma$, then $(l \to r)\sigma$ is a more specific version of $l \to r$ in $\mathcal{R}$ w.r.t. $\xrightarrow{c}$. Moreover, if σ is not a constructor substitution, then $l \to r$ is not used in any successful reduction in $\mathcal{R}$ w.r.t. $\xrightarrow{c}$.*
- *If $msv(\mathcal{R}, l \to r, \tau) = \bot$, then $l \to r$ is not used in any successful reduction in $\mathcal{R}$ w.r.t. $\xrightarrow{c}$.*

Note that in the case that $l \to r$ is not used in any successful reduction in $\mathcal{R}$, to preserve constructors (i.e., $\mathcal{C}_{\mathcal{R}} = \mathcal{C}_{\mathcal{R}'}$), we do not remove $l \to r$ from $\mathcal{R}$. In order to prove this result, we first need a *lifting lemma* that relates constructor-based (innermost) reductions with innermost (basic) narrowing derivations. For this purpose, we first prove the following lifting lemma (a simple extension of [19, Lemma 3.4]):

Lemma 17. *Let $\mathcal{R}$ be a TRS, s, t terms, θ a normalized substitution, and V a set of variables such that $\mathcal{V}ar(s) \cup \mathcal{D}om(\theta) \subseteq V$ and $t = s\theta$. If $t \xrightarrow{i}{}^*_{\mathcal{R}} t'$, then there exist a term s' and substitutions θ', σ such that $s \overset{i}{\leadsto}{}^*_{\sigma} s'$, $s'\theta' = t'$, $\theta' \circ \sigma = \theta$ $[V]$, and θ' is a normalized substitution.*

Proof. This lemma is an easy consequence of [19, Lemma 3.4]. By applying [19, Lemma 3.4] to $t \xrightarrow{i}{}^*_{\mathcal{R}} t'$, we have that there exists a narrowing derivation $s \leadsto^*_{\sigma} s'$ with $\theta' \circ \sigma = \theta$ $[V]$ such that $s'\theta' = t'$ for some normalized substitution θ'. Moreover, $s \leadsto^*_{\sigma} s'$ and $s\theta \xrightarrow{i}{}^* t'$ employ the same rewrite rules at the same positions. Hence we can consider that every rewrite step on a redex $f(t_1, \ldots, t_n)$ with $t_1, \ldots, t_n$ constructor terms gives rise to an innermost narrowing step, and every rewrite step on a redex $f(t_1, \ldots, t_n)$ where some t_i are not constructor terms (but normal forms) gives rise to a sequence of zero or more reflection steps, followed by an innermost narrowing step (note the equivalence between normal forms in $t \xrightarrow{i}{}^*_{\mathcal{R}} t'$ and non-narrowable terms produced by reflection steps in $s \leadsto^*_{\sigma} s'$). Therefore, the narrowing derivation $s \leadsto^*_{\sigma} s'$ is indeed an innermost basic narrowing derivation and will be denoted by $s \overset{i}{\leadsto}{}^*_{\sigma} s'$. □

In the following, we denote by $\overset{i^-}{\leadsto}$ the pure innermost narrowing relation (i.e., without the reflection transition rule). The following lemma is an easy extension of the previous one:

Lemma 18. *Let $\mathcal{R}$ be a TRS, s,t terms, θ a constructor substitution, and V a set of variables such that $\mathcal{V}ar(s) \cup \mathcal{D}om(\theta) \subseteq V$ and $t = s\theta$. If $t \xrightarrow{c}{}^*_{\mathcal{R}} t'$, then there exist a term s' and constructor substitutions θ', σ such that $s \overset{i}{\rightsquigarrow}{}^*_{\sigma} s'$, $s'\theta' = t'$, and $\theta' \circ \sigma = \theta$ $[V]$.*

Now, by using this lifting lemma, we can prove the correctness of Theorem 16 above:

Proof. Let $msv(\mathcal{R}, l \to r, \tau) = (l \to r)\sigma$ and $\mathcal{R}' = (\mathcal{R} \setminus \{l \to r\}) \cup \{(l \to r)\sigma\}$. First, we note that if $s \xrightarrow{c}{}^*_{\mathcal{R}'} t$ is successful in $\mathcal{R}'$ then $s \xrightarrow{c}{}^*_{\mathcal{R}} t$ is successful in $\mathcal{R}$: it follows from the definition of $\xrightarrow{c}$ that the set of rules used in $s \xrightarrow{c}{}^*_{\mathcal{R}'} t$ is a constructor system, and hence $\xrightarrow{c}_{\mathcal{R}'} \subseteq \xrightarrow{c}_{\mathcal{R}}$. Thus, we only show that if $s \xrightarrow{c}{}^*_{\mathcal{R}} t$ is successful in $\mathcal{R}'$ w.r.t. $\xrightarrow{c}$ then $s \xrightarrow{c}{}^*_{\mathcal{R}} t$ is successful in $\mathcal{R}$ w.r.t. $\xrightarrow{c}$. To prove this claim, it suffices to show the following claim:

> *Let $l_0 \to r_0 \in \mathcal{R}$, θ be a constructor substitution, and s,t be terms such that $r_0\theta \xrightarrow{c}{}^*_{\mathcal{R}} s$ and $r_0\theta \xrightarrow{c}{}^*_{\mathcal{R}'} s$. If $s \xrightarrow{c}{}^*_{\mathcal{R}} t \in \mathcal{T}(\mathcal{C},\mathcal{V})$, then $l_0\theta \xrightarrow{c}_{\mathcal{R}'} r_0\theta$ and $s \xrightarrow{c}{}^*_{\mathcal{R}'} t$. Moreover, $\tau' \neq \varnothing$ and σ is a constructor substitution whenever a rewrite step of $l \to r$ appears in $l_0\theta \xrightarrow{c}_{\mathcal{R}} r_0\theta$ or $s \xrightarrow{c}{}^*_{\mathcal{R}} t$.*

We prove this claim by induction on the length of $s \xrightarrow{c}{}^*_{\mathcal{R}} t$.

We first show that $l_0\theta \xrightarrow{c}_{\mathcal{R}'} r_0\theta$. We make a case distinction depending on whether $l_0 \to r_0$ is equal to $l \to r$ or not.

- If $(l_0 \to r_0) \neq (l \to r)$, then we have that $l_0\theta \xrightarrow{c}_{\mathcal{R}'} r_0\theta$ since $l_0 \to r_0 \in \mathcal{R}'$.
- Otherwise, we have that $r\theta \xrightarrow{c}{}^*_{\mathcal{R}} t$. Thus, it follows from Lemma 18 that $r \overset{i}{\rightsquigarrow}{}^*_{\delta} t$ and $\delta \leq \theta$ for some constructor substitution δ. By the construction of τ, there exists a term r' and constructor substitutions σ', σ'' such that $r \overset{i}{\rightsquigarrow}{}^*_{\sigma'} r' \overset{i}{\rightsquigarrow}{}^*_{\sigma''} t$ and $r \overset{i}{\rightsquigarrow}{}^*_{\sigma'} r' \in \tau$, i.e., $\delta = \sigma' \circ \sigma''$. By the construction of σ, we have that $\sigma \leq \sigma'$, and hence $\sigma \leq \theta$. Thus, we have that $l_0\theta \xrightarrow{c}_{(l\to r)\sigma} r_0\theta$.

Next we show that $s \xrightarrow{c}{}^*_{\mathcal{R}'} t$. Since the case that $s \xrightarrow{c}{}^*_{\mathcal{R}\setminus\{l\to r\}} t$ is trivial, we consider the remaining case that $s \xrightarrow{c}{}^*_{\mathcal{R}} t$ contains a rewrite step of $l \to r$. By the definition of $\xrightarrow{c}$, we can assume that

$$s \xrightarrow{c}{}^*_{\mathcal{R}\setminus\{l\to r\}} C[l\theta_1]_p \xrightarrow{c}_{p,l\to r} C[r\theta_1]_p \xrightarrow{c}{}^*_{p\leq,\mathcal{R}} C[t_1]_p \xrightarrow{c}{}^*_{\mathcal{R}} t$$

for some context $C[\,]$, a constructor substitution θ_1 and a term $t_1 \in \mathcal{T}(\mathcal{C},\mathcal{V})$. By the induction hypothesis, we have that $l\theta_1 \xrightarrow{c}{}^*_{\mathcal{R}'} t_1$, and hence $r\theta \xrightarrow{c}{}^*_{\mathcal{R}'} s \xrightarrow{c}{}^*_{\mathcal{R}'} C[l\theta_1] \xrightarrow{c}{}^*_{\mathcal{R}'} C[t_1]$. Again, by the induction hypothesis, we have that $C[t_1] \xrightarrow{c}{}^*_{\mathcal{R}'} t$, and hence $s \xrightarrow{c}{}^*_{\mathcal{R}'} t$. Thus, the claim holds.

Finally, we show that $\tau' \neq \varnothing$ and σ is a constructor substitution if a rewrite step of $l \to r$ appears in $l_0\theta \xrightarrow{c}_{\mathcal{R}} r_0\theta$ or $s \xrightarrow{c}{}^*_{\mathcal{R}} t$. If $(l_0 \to r_0) = (l \to r)$, then $\tau' \neq \varnothing$ and σ is a constructor substitution since a constructor substitution θ for a successful derivation exists and $\sigma \leq \theta$. Otherwise, by the induction hypothesis, $\tau' \neq \varnothing$ and σ is a constructor substitution whenever $s \xrightarrow{c}{}^*_{\mathcal{R}} t$ contains a rewrite step of $l \to r$.

Therefore, $(l \to r)\sigma$ is a more specific version of $l \to r$ in $\mathcal{R}$ w.r.t. $\xrightarrow{c}$, and moreover, $\tau' \neq \varnothing$ and σ is a constructor substitution if $l \to r$ is used in a successful reduction in $\mathcal{R}$ w.r.t. $\xrightarrow{c}$. □

Now, we consider an alternative for the correctness of the *msv* algorithm without restricting to constructor-based reduction. For this purpose, we consider non-erasing constructor TRSs, which enjoy the following nice property:

Lemma 19. *Let $\mathcal{R}$ be a non-erasing constructor TRS and s, t be terms. If $s \xrightarrow{i}{}^*_{\mathcal{R}} t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ then $s \xrightarrow{c}{}^*_{\mathcal{R}} t$. Furthermore, one may assume that the innermost derivation $s \xrightarrow{i}{}^*_{\mathcal{R}} t$ and the constructor-based derivation $s \xrightarrow{c}{}^*_{\mathcal{R}} t$ employ the same rewrite rules at the same positions.*

Proof. We prove this lemma by induction on the length of the derivation. Consider the derivation $C[l\sigma] \xrightarrow{i}_{\mathcal{R}} C[r\sigma] \xrightarrow{i}{}^*_{\mathcal{R}} t$ with a normalized substitution σ. Suppose that σ is not a constructor substitution. Then, $l\sigma$ contains a normal form containing a defined symbol g. Since $\mathcal{R}$ is a non-erasing constructor system, the defined symbol g in $l\sigma$ is not consumed by the application of rules during the reduction $s \xrightarrow{i}{}^*_{\mathcal{R}} t$, and hence g appears in t. This contradicts that $t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$. Thus, σ is a constructor substitution and hence $C[l\sigma] \xrightarrow{c}_{\mathcal{R}} C[r\sigma]$. By the induction hypothesis, we have that $C[r\sigma] \xrightarrow{c}{}^*_{\mathcal{R}} t$. Therefore, we have that $s \xrightarrow{c}{}^*_{\mathcal{R}} t$. □

Moreover, it follows from Lemma 19 that Theorem 16 also holds for non-erasing constructor TRSs and $\xrightarrow{i}$.

Theorem 20. *Let $\mathcal{R}$ be a non-erasing constructor TRS, $l \to r \in \mathcal{R}$ be a rewrite rule such that r is not a constructor term, and τ be a finite (possibly incomplete) innermost basic narrowing tree for r in $\mathcal{R}$. Then,*

- *If $msv(\mathcal{R}, l \to r, \tau) = (l \to r)\sigma$, then $(l \to r)\sigma$ is a more specific version of $l \to r$ in $\mathcal{R}$ w.r.t. $\xrightarrow{i}$. Moreover, if σ is not a constructor substitution, then $l \to r$ is not used in any successful reduction in $\mathcal{R}$ w.r.t. $\xrightarrow{c}$.*
- *If $msv(\mathcal{R}, l \to r, \tau) = \bot$, then $l \to r$ is not used in any successful reduction in $\mathcal{R}$ w.r.t. $\xrightarrow{i}$.*

Proof. Let $msv(\mathcal{R}, l \to r, \tau) = (l \to r)\sigma$ and $\mathcal{R}' = (\mathcal{R} \setminus \{l \to r\}) \cup \{(l \to r)\sigma\}$. We only show that if $s \xrightarrow{i}{}^*_{\mathcal{R}'} t$ is successful in $\mathcal{R}'$ then $s \xrightarrow{i}{}^*_{\mathcal{R}} t$ is successful in $\mathcal{R}$ since the remaining part can be proved similarly to Theorem 16 by using Lemma 19.

Suppose that σ is a constructor substitution. It follows from Lemma 19 that $s \xrightarrow{c}{}^*_{\mathcal{R}'} t$. It follows from Theorem 16 that $(l \to r)\sigma$ is a more specific version of $l \to r$ in $\mathcal{R}$, and hence $s \xrightarrow{c}{}^*_{\mathcal{R}} t$. Therefore, it follows from $\xrightarrow{c}_{\mathcal{R}} \subseteq \xrightarrow{i}_{\mathcal{R}}$ that $s \xrightarrow{i}{}^*_{\mathcal{R}} t$.

Suppose that σ is not a constructor substitution. Since $\mathcal{R}$ is non-erasing, so is $\mathcal{R}'$. Thus, a term rooted by a defined symbol appears in a proper subterm of $l\sigma$ appears in $r\sigma$. This means that a normal form rooted by a defined symbol does not disappear in reductions of $\mathcal{R}'$, i.e., $(l \to r)\sigma$ is not used in any successful derivation in $\mathcal{R}'$. Therefore, $s \xrightarrow{i}{}^*_{\mathcal{R}'} t \in \mathcal{T}(\mathcal{C}, \mathcal{V})$ implies $s \xrightarrow{i}{}^*_{\mathcal{R}' \setminus \{(l \to r)\sigma\}} t$, and hence $s \xrightarrow{i}{}^*_{\mathcal{R}} t$. □

In program inversion for injective functions, the resulting systems are basically non-erasing [22]. Thus, *msv* is often useful in improving the resulting systems of the inversion, e.g., inc^{-1} in Section 1.

Clearly, computing *most* specific versions is generally undecidable since it would require the computation of all (possibly infinite) narrowing derivations starting from the right-hand side of this rule. Nevertheless, analogously to [18], we can at least state the conditions under which the computed more specific version is actually a most specific version. Intuitively speaking, the computed more specific version is a *most* specific version when no further narrowing steps might further instantiate the result of the *lgg* operator.

Theorem 21. *Given the conditions of Theorem 16 (resp. Theorem 20), If* $lgg(\{r\sigma \mid r \overset{i}{\leadsto}{}^*_\sigma r' \in \tau\})$ *and* $lgg(\{r\sigma \mid r \overset{i}{\leadsto}{}^*_\sigma r' \in \tau_{succ}\})$ *are variants, then* $msv(\mathcal{R}, l \to r, \tau)$ *is a most specific version of* $l \to r$ *in* $\mathcal{R}$ *w.r.t.* $\overset{c}{\to}$ *(resp.* $\overset{i}{\to}$*).*

Proof. First, let us not that $\tau_{succ} \subseteq \tau$ and, thus, $\{r\sigma \mid r \overset{i}{\leadsto}{}^*_\sigma r' \in \tau_{succ}\} \subseteq \{r\sigma \mid r \overset{i}{\leadsto}{}^*_\sigma r' \in \tau\}$. Assume $S = \{r\sigma \mid r \overset{i}{\leadsto}{}^*_\sigma r' \in \tau_{succ}\}$ and $\{r\sigma \mid r \overset{i}{\leadsto}{}^*_\sigma r' \in \tau\} = S \cup T$ for some set T (the instances of r with the substitutions of incomplete, non-failing derivations). Then, the proof is an immediate consequence of the following property: if $lgg(S)$ is a variant of $lgg(S \cup T)$, then $lgg(S)$ is also a variant of $lgg(S \cup T\sigma)$ for all substitution σ (since only incomplete derivations can produce further variable bindings).

Now, let us prove the above property, where we consider singleton sets $S = \{s\}$ and $T = \{t\}$ for simplicity. First, by definition of the *lgg* operator, we have $lgg(\{s\}) = s$. Since $lgg(s, t)$ is a variant of s, we have $s \leq t$. Therefore, s is also more general than $t\sigma$ for all σ and, thus, $lgg(s, t) = lgg(s, t\sigma)$ for all σ, which proves the claim. □

Actually, Theorem 21 provides the basis for refining the *msv* algorithm: one could safely stop unfolding an incomplete innermost basic narrowing derivation $s \overset{i}{\leadsto}{}^*_\sigma t$ if there exists a successful innermost basic narrowing derivation $s \overset{i}{\leadsto}{}^*_\theta s'$ such that $\theta \leq \sigma$ $[\mathcal{V}\mathsf{ar}(s)]$.

5 Conclusion and Future Work

We have introduced the notion of a *more specific* rule, which allows us to replace a rule of a rewriting system by an instance of this rule such that the reductions of interest—the so called *successful* reductions from a constructor instance of a left-hand side to a constructor term—are still preserved. We have also introduced an algorithm for computing more specific versions and have proved its correctness as well as some basic properties. In general, the MSV transformation might help to make some properties of a TRS explicit, which might ease the static analysis of the program's properties (as in [18], where a similar transformation was introduced in the context of logic programming).

As for future work, we plan to extend our results to the conditional case. Another avenue of future work is the use of the MSV transformation to improve the accuracy of termination analyses.

References

[1] E. Albert and G. Vidal. The narrowing-driven approach to functional logic program specialization. *New Generation Computing*, 20(1):3–26, 2002.

[2] J. M. Almendros-Jiménez and G. Vidal. Automatic partial inversion of inductively sequential functions. In *Proceedings of the 18th International Symposium on Implementation and Application of Functional Languages*, volume 4449 of *Lecture Notes in Computer Science*, pp. 253–270, Springer, 2006.

[3] S. Antoy and M. Hanus. Overlapping rules and logic variables in functional logic programs. In *Proceedings of the 22nd International Conference on Logic Programming*, volume 4079 of *Lecture Notes in Computer Science*, pp. 87–101, Springer, 2006.

[4] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[5] A. Bondorf. Towards a self-applicable partial evaluator for term rewriting systems. In *Proceedings of the International Workshop on Partial Evaluation and Mixed Computation*, pp. 27–50, North-Holland, Amsterdam, 1988.

[6] A. Bondorf. A self-applicable partial evaluator for term rewriting systems. In *Proceedings of International Conference on Theory and Practice of Software Development, Barcelona, Spain*, volume 352 of *Lecture Notes in Computer Science*, pp. 81–95, Springer, 1989.

[7] P. Bosco, E. Giovannetti, and C. Moiso. Narrowing vs. SLD-resolution. *Theor. Comput. Sci.*, 59:3–23, 1988.

[8] N. Dershowitz. Termination of rewriting. *J. Symb. Comput.*, 3(1&2):69–115, 1987.

[9] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pp. 243–320, Elsevier, Amsterdam, 1990.

[10] L. Fribourg. 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.

[11] R. Glück and M. Kawabe. A program inverter for a functional language with equality and constructors. In *Proceedings of the first Asian Symposium on Programming Languages and Systems*, volume 2895 of *Lecture Notes in Computer Science*, pp. 246–264, Springer, 2003.

[12] R. Glück and M. Kawabe. A method for automatic program inversion based on LR(0) parsing. *Fundam. Inform.*, 66(4):367–395, 2005.

[13] M. Hanus. The integration of functions into logic programming: From theory to practice. *J. Log. Program.*, 19&20:583–628, 1994.

[14] P. G. Harrison. Function inversion. In *Proceedings of the International Workshop on Partial Evaluation and Mixed Computation*, pp. 153–166, North-Holland, Amsterdam, 1988.

[15] S. Hölldobler. *Foundations of Equational Logic Programming*, volume 353 of *Lecture Notes in Artificial Intelligence*, Springer, 1989.

[16] J.-M. Hullot. Canonical forms and unification. In *Proceedings of the 5th International Conference on Automated Deduction*, volume 87 of *Lecture Notes in Computer Science*, pp. 318–334, Springer, 1980.

[17] H. Khoshnevisan and K. M. Sephton. InvX: An automatic function inverter. In *Proceedings of the 3rd International Conference of Rewriting Techniques and Applications*, volume 355 of *Lecture Notes in Computer Science*, pp. 564–568, Springer, 1989.

[18] K. Marriott, L. Naish, and J.-L. Lassez. Most specific logic programs. *Ann. Math. Artif. Intell.*, 1:303–338, 1990.

[19] A. Middeldorp and E. Hamoen. Completeness results for basic narrowing. *Appl. Algebra Eng. Commun. Comput.*, 5:213–253, 1994.

[20] N. Nishida, M. Sakai, and T. Sakabe. Generation of inverse computation programs of constructor term rewriting systems. *IEICE Trans. Inf. & Syst.*, J88-D-I(8):1171–1183, 2005 (in Japanese).

[21] N. Nishida, M. Sakai, and T. Sakabe. Partial inversion of constructor term rewriting systems. In *Proceedings of the 16th International Conference on Rewriting Techniques and Applications*, volume 3467 of *Lecture Notes in Computer Science*, pp. 264–278, Springer, 2005.

[22] N. Nishida and G. Vidal. Program inversion for tail recursive functions. In *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications*, volume 10 of *LIPIcs*, pp. 283–298, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

[23] G. Plotkin. Building-in equational theories. *Machine Intelligence*, 7:73–90, 1972.

[24] J. G. Ramos, J. Silva, and G. Vidal. Fast Narrowing-Driven Partial Evaluation for Inductively Sequential Systems. In *Proceedings of the 10th ACM SIGPLAN International Conference on Functional Programming*, pp. 228–239, ACM Press, 2005.

[25] J. R. Slagle. Automated theorem-proving for theories with simplifiers, commutativity and associativity. *J. ACM*, 21(4):622–642, 1974.