A Rewrite-based Computational Model for Functional Logic Programming

Mircea Marin[1], Temur Kutsia[2], and Besik Dundua[3]

[1] Department of Computer Science
West University of Timişoara, Timişoara, Romania
mircea.marin@e-uvt.ro
[2] Research Institute for Symbolic Computation,
Johannes Kepler University, Linz, Austria
kutsia@risc.jku.at
[3] Institute of Applied Mathematics
Tbilisi State University
bdundua@gmail.com

Abstract

Functional logic programming extends the functional programming style with two important features: the possibility to define nondeterministic operations with overlapping rules, and the usage of logic variables in both defining rules and expressions to evaluate. Conditional constructor-based term rewrite systems (CB-CTRSs) emerged as a suitable model for functional logic programs, because they can be easily transformed into an equivalent program in a core language where computations can be performed more efficiently.

We consider a recent proposal by Antoy and Hanus, of translating CB-CTRSs into an equivalent class of programs where computation can be performed by mere rewriting. His computational model has the limitation of computing only ground answer substitutions for equations with strict semantics interpreted as joinability to a value.

We propose two adjustments of their computational models, which are capable to compute non-ground answers.

1 Introduction

Functional logic programming (FLP for short) is a declarative programming style resulted from the successful amalgamation of the best features of functional programming with logic programming (see [7] for a recent survey). In comparison with pure functional languages, functional logic languages have more expressive power due to the possibility to define nondeterministic operations with overlapping rules (i.e., more than one rule can be applied to evaluate a function call), and the possibility to use logic variables in both defining rules and expressions to evaluate.

The presence of logic variables makes the evaluation of an expression t in the presence of a program $\mathcal{R}$ more challenging: We are not only interested to compute a value to which t reduces (that is, a ground constructor term v such that $t \to^*_{\mathcal{R}} v$), but rather to compute a set of pairs $\langle \sigma, v\rangle$ made of a ground constructor substitution σ for the variables of t and a value v, such that $t\sigma \to^*_{\mathcal{R}} v$. We may denote the set of all such σ by $GSol_{\mathcal{R}}(t)$ and call its elements *ground solutions* of t, but we consider more desirable to drop the restriction on groundness, and determine the elements of the set $Sol_{\mathcal{R}}(t)$ of pairs $\langle \sigma, v\rangle$ where σ is constructor substitution, v is a possibly non-ground constructor term, and $t\sigma \to^*_{\mathcal{R}} v$. A convenient representation of $Sol_{\mathcal{R}}(t)$ is by a set A of so called *computed answers*, with the following two properties: (1) $A \subseteq Sol_{\mathcal{R}}(t)$ (*soundness*), and (2) for every solution $\langle \varphi, v\rangle$ of t there exist $\langle \sigma, v'\rangle \in A$ and a constructor substitution η such that $v = v'\eta$ and $\varphi(x) = \eta(\sigma(x))$ for every variable x in t (*completeness*).

The computational models of FLP languages are designed to compute sound and complete sets of computed answers. When the interest is in solving systems of equations, we can model them by terms of the form

$$e_1 \,\&\&\, \ldots \,\&\&\, e_n \tag{1}$$

where every e_i is an equational term with a certain semantics, and `&&` is a predefined right-associative infix operation. In general, equational terms are of the form $s \approx t$, and defined to hold if s and t are reducible to the same (ground) constructor term. The nice thing when interpreting $\approx$ as joinability to a value is that we can extend $\mathcal{R}$ with rewrite rules for "$\approx$" and "`&&`", and obtain a TRS $\mathcal{R}'$ with the following property: σ is a solution of an equational goal $G = s_1 \approx t_1 \,\&\&\, \ldots \,\&\&\, s_n \approx t_n$ iff $G\sigma \to^*_{\mathcal{R}'}$ `success`. (See, e.g., [3].) This observation leads to the possibility to solve such systems of equations by *narrowing*, a mechanism that extends the concept of reduction from functional programming with unification and nondeterministic search from logic programming. Narrowing was designed initially as a sound and complete method for solving unification problems in equational theories presented by confluent term rewrite systems [10]: If $\mathcal{R}$ is such a TRS and $\mathcal{R}' = \mathcal{R} \cup \{x \approx x \to \texttt{success}\}$, then narrowing w.r.t. $\mathcal{R}'$ computes a complete set of $\mathcal{R}$-unifiers of s and t. Unfortunately, this way of computing answers is largely useless and hopelessly inefficient for FLP, because:

1. Unrestricted narrowing computes many $\mathcal{R}$-unifiers which are not solutions from the point of view of FLP.
2. $\mathcal{R}'$ ensures the interpretation of equality as joinability to *any* term. Nowadays, most FLP languages interpret equality as joinability to a common (ground) constructor term.
3. The search space of narrowing explores the possibility to unify *each* rule with *each* non-variable subterm of the input system. The resulting search space would be huge even for small TRSs.

Therefore, many narrowing strategies and narrowing calculi have been proposed, to reduce the search space for solutions without losing the completeness of the set of computed answers.

- Narrowing strategies are partial mappings from terms to narrowing steps, and their rôle is to tell us which subterms to select for narrowing, in order to compute a sound and complete set of answer substitutions.
- Narrowing calculi emerged as an alternative to narrowing strategies: Usually, they consist of a small set of elementary inference or transformation rules for equational goals, which can be used to simulate narrowing steps. Typical examples of narrowing calculi are LNC [15], LNC_d [14], LCNC [8], LCNC [16], and LCNC_d [13]. Higher-order versions of narrowing calculi are also known, e,g., LN [17], HOLN [11], and LN_ff [12].

The success of finding an adequate strategy or calculus for narrowing depends mainly on the class of TRSs chosen to represent programs, and on the way we define equality and the admissibility of solutions. Several interesting sound and complete narrowing strategies are already known for various classes of first-order left linear constructor-based TRS (CB-TRSs for short). To this category belong the following strategies: needed narrowing [4], for strongly sequential TRSs [9], parallel narrowing [5], for weakly orthogonal TRSs, and the INS narrowing strategy [2] for overlapping inductively sequential TRSs.

Programs presented by conditional CB-TRSs (CB-CTRSs for short) are desirable because they increase the conciseness and expressive power of rule-based definitions of operations of interest. In [6], Antoy and Hanus report a rewrite-based computational model for FLP with CB-CTRSs. Their main results can be summarised as follows:

1. They identify a transformation of CB-CTRSs into equivalent overlapping inductively sequential TRSs with extra variables (OISs for short). OISs are a simpler class of TRSs, for which we already know a sound and complete narrowing strategy [2]
2. They identify a sound and complete computational model for FLP for programs presented by OIS, based on a translation of OIS with extra variables into OIS without extra variables, and on rewriting of term/substitution pairs.

These results open the possibility to produce an efficient implementation of a sound and complete narrowing strategy for FLP with CB-CTRSs, by reducing all computations to rewrites of term/substitution pairs with respect to OISs. There are, however, some important limitations:

1. Equality is interpreted as reducibility to a common value.
2. The computed answers are ground constructor substitutions. As a result, the completeness property of the calculus implies that the set of computed answers is often infinite, and thus the solving process of a system of equations runs forever.

In this paper we overcome these limitations and propose a rewrite-based strategy for functional logic programming with programs presented by OISs. We argue that this strategy is sound and complete in the general sense, where we do not restrict ourself to ground solutions, and interpret equality as reducibility to a common, not necessarily ground, constructor term.

The paper is structured as follows. Section 2 reviews concepts an notations used in this paper, and previous results which are relevant for our investigation. Sections 3 and 4 describe the strategies proposed by us for solving systems of equations when programs are presented by OISs and equality is interpreted in the more general sense, mentioned above. The first one is a narrowing strategy obtained by an adjustment of the INS strategy, and the second one is a rewrite-based strategy. Both strategies proposed by us are sound and complete. Section 5 concludes.

2 Preliminaries

We consider a finite many-sorted signature Σ partitioned into a set $\mathcal{C}$ of (data) *constructors*, and a set $\mathcal{F}$ of (defined) *function symbols* or *operations*. We write f/n to indicate the fact that $f \in \Sigma$ is an n-ary constructor or operation. We also consider $\mathcal{X}$ to be a countably infinite set of sorted variables. The sets $\mathcal{T}(\Sigma, \mathcal{X})$, $\mathcal{T}(\mathcal{C}, \mathcal{X})$ and $\mathcal{T}(\mathcal{C})$ of well-sorted terms, constructor terms, and ground constructor terms are defined as usual. The terms of $\mathcal{T}(\mathcal{C})$ are also called *values*.

We write $var(t)$ for the set of variables which occur in a term t. A term t is *ground* if $var(t) = \emptyset$, and *linear* if it does not contain multiple occurrences of a variable. A *pattern* is a term $f(t_1, \ldots, t_n)$ where $f/n \in \mathcal{F}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. An *equation* is an expression $s \approx t$ where s, t are terms of the same sort. A *rewrite rule* is an expression of the form $l \to r$ where l is a pattern, and r is a term of the same sort as l.

A *conditional rewrite rule* is of the form $l \to r \Leftarrow c$ where l, r satisfy the same conditions as before, and c is a system of equations of the form $e_1 \,\&\&\, \ldots \,\&\&\, e_n$. The set of *extra variables* of $l \to r$ (resp. $l \to r \Leftarrow c$) is $evar(l \to r) := var(r) \setminus var(l)$ (resp. $evar(l \to r \Leftarrow c) := (var(r) \cup var(c)) \setminus var(l)$). An *constructor-based rewrite system* (CB-TRS) is a set of *rewrite rules*, and a conditional constructor-based TRS (CB-CTRS) is a set of conditional rewrite rules. CB-TRSs and CB-CTRSs are used to represent functional programs.

To formally define computations with respect to a given program, we shall introduce a few more notions. Positions in a term are denoted by sequences of natural numbers. We write $t|_p$ for the subterm of t at position p, and $t[s]_p$ for the result of replacing $t|_p$ with s at position p in t. A

substitution is a mapping $\sigma : \mathcal{X} \to \mathcal{T}(\Sigma, \mathcal{X})$ such that its domain $dom(\sigma) := \{x \in \mathcal{X} \mid \sigma(x) \neq x\}$ is finite, and x and $\sigma(x)$ are of the same sort for every $x \in \mathcal{X}$. It is common practice to denote a substitution σ by the finite set $\{x \to \sigma(x) \mid x \in dom(\sigma)\}$. The *restriction* of a substitution σ to a set of variables $V \subseteq \mathcal{X}$ is the substitution $\sigma|_V$ with $dom(\sigma|_V) = dom(\sigma) \cap V$ and $\sigma|_V(x) = \sigma(x)$ for all $x \in dom(\sigma|_V)$. A *(ground) constructor substitution* is a substitution σ such that $\sigma(x)$ is a (ground) constructor term for every $x \in dom(\sigma)$. Substitutions are extended to morphisms on terms in the obvious way, and we write $t\sigma$ for the image of t under the morphism induced by a substitution σ. *Subsumption* is the ordering on terms defined by $s \leq t$ if $s\sigma = t$ for some substitution σ. Also, we write $s < t$ if $s \leq t$ and $t \not\leq s$. The subsumption relation is defined on substitutions too: $\sigma \leq \sigma'$ if there exists a substitution η such that $\sigma' = (\eta \circ \sigma)|_{vars(\sigma')}$.

A *variant* of t is t' such that $t \leq t'$ and $t' \leq t$. t' is a *fresh variant* of t if it is a variant of t and $var(t')$ consists of variables which did not occur in the expressions encountered so far. A *unifier* of two terms s and t is a substitution σ such that $s\sigma = t\sigma$. The unifier σ is a most general unifier (*mgu* for short) if $\sigma \leq \sigma'$ for any other unifier σ' of s and t. We write $s \lhd t$ for the restriction of an mgu of s and t to $vars(s)$.

A *rewrite step* of a term t with respect to a CB-TRS $\mathcal{R}$ is a relation $t \to_{p,l\to r,\sigma} t'$, which is defined if there exist a position p in t, a rewrite rule $l \to r \in \mathcal{R}$ with a fresh variant $l' \to r'$, and a substitution σ such that $t|_p = l'\sigma$ and $t' = t[r'\sigma]_p$. We may write $t \to_{\mathcal{R}} t'$ instead of $t \to_{p,l\to r,\sigma} t'$. In general, if $\to$ is a binary relation on terms, we define the following relations:

- $\to^*$ for the reflexive-transitive closure of $\to$.
- $\downarrow$ for the strict joinability relation: $s \downarrow t$ if $s \to^* u$ and $t \to^* u$ for some $u \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Rewriting with a CB-CTRS $\mathcal{R}$ is slightly more involved: A rewrite step of a term t w.r.t. $\mathcal{R}$ is a relation $t \to_{p,l\to r\Leftarrow c,\sigma} t'$, abbreviated $t \to_{\mathcal{R}} t'$, which holds if there exist a position p in t, a rule $l \to r \Leftarrow c$ from $\mathcal{R}$ with fresh variant $l' \to r' \Leftarrow c'$, and a substitution σ such that $t|_p = l'\sigma$, $t' = t[r'\sigma]_p$, and σ is a *solution* of c', i.e., for every equation $u \approx v$ from c, we have $u\sigma \downarrow_{\mathcal{R}} v\sigma$, i.e., $u\sigma \to^*_{\mathcal{R}} w$ and $v\sigma \to^*_{\mathcal{R}} w$ for some $w \in \mathcal{T}(\mathcal{C}, \mathcal{X})$.

A narrowing step $s \rightsquigarrow_{\mathcal{R},\sigma} t$ of a term s with respect to program presented by a TRS $\mathcal{R}$ is defined if there exist a non-variable position p of s, a fresh variant $l \to r$ of a rewrite rule from $\mathcal{R}$, and a unifier σ of $s|_p$ and l, such that $t = s[r]_p\sigma$. If $\mathcal{R}$ is a conditional TRS, the definition is slightly more involved: for a fresh variant $l \to r \Leftarrow c$ of a rewrite rule from $\mathcal{R}$ we also require to have σ solution of the conditional part c. Here we indicate an equivalent definition which is more convenient for our further investigation: $s \rightsquigarrow_{\mathcal{R},\sigma} t$ is a narrowing step of s w.r.t. a program $\mathcal{R}$ if $s\sigma \to_{\mathcal{R}} t$ is a rewrite step at a non-variable position p of s (that is, $s|_p \notin \mathcal{X}$).

The evaluation of t in a FP language presented by a TRS $\mathcal{R}$ yields a value $v \in \mathcal{T}(\mathcal{C})$ such that $t \to^*_{\mathcal{R}} v$, whereas the evaluation of t in FLP is concerned with computing a set $Ans(t)$ of pairs $\langle v, \sigma\rangle$ with $v \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ and σ a constructor substitution, such that $t\sigma \to^*_{\mathcal{R}} v$ (soundness). In addition, we also wish the set to be complete in the following sense:

> For any constructor substitution σ' and $v' \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ such that $t\sigma' \to^*_{\mathcal{R}} v'$, there exists $\langle v, \sigma\rangle \in Ans(t)$ such that $\sigma \leq \sigma'$ and $v \leq v'$.

Narrowing-based computational models compute the elements $\langle v, \sigma\rangle$ of $Ans(t)$ by producing a set of narrowing derivations $t \rightsquigarrow^*_{\mathcal{R},\sigma} v$ with constructor term v. Soundness follows from the observation that $t \rightsquigarrow^*_{\mathcal{R},\sigma} v$ implies $t\sigma \to^*_{\mathcal{R}} v$, and from the restriction to perform only narrowing steps $t_i \rightsquigarrow_{\mathcal{R},\sigma_i} t_{i+1}$ with constructor substitution σ_i. Computational efficiency can be often achieved by using a *narrowing strategy*, which prescribes, for every term, what narrowing step(s) to perform next, without losing completeness. Formally, a narrowing strategy is a partial function $\mathcal{S}$ on terms, such that $\mathcal{S}(s)$ is either undefined, or a set of triples $\langle p, \rho, \sigma\rangle$ consisting of a nonvariable position p of s, a fresh variant of a rule $\rho \in \mathcal{R}$, and a constructor substitution

σ such that we can perform a narrowing step $s \rightsquigarrow_{p,\rho,\sigma} t$. The evaluation of a term s with $\mathcal{S}$ yields the set of computed answers $Ans^{\mathcal{S}}_{\mathcal{R}}(s) = \{\langle v, \sigma\rangle \mid s \rightsquigarrow^*_{\mathcal{R},\sigma} v$ where $v \in \mathcal{T}(\mathcal{C}, \mathcal{X})$ and $(p, \rho, \sigma) \in \mathcal{S}(t_i)$ for every component narrowing step $t_i \rightsquigarrow_{p,\rho,\sigma} t'_i\}$.

Overlapping inductively sequential rewrite systems

We define here the class of rewrite systems that concerns us most: overlapping inductively sequential TRSs with extra variables. They are defined via an auxiliary hierarchical data structure, called definitional tree. The following definitions are adapted from [1, 2].

A *partial definitional tree* (*pdt*) of a linear pattern t is a tree-like structure $\mathcal{T}$ with nodes of one of the following three forms:

1. $branch(t, p, \mathcal{T}_1, \ldots, \mathcal{T}_k)$ where $t|_p$ is a variable of a sort s with constructors $c_1/n_1, \ldots, c_k/n_k$ in standard ordering, and every $\mathcal{T}_i$ $(1 \le i \le k)$ is a pdt of the pattern $t[c_i(x_1, \ldots, x_{n_i})]_p$ and $x_1, \ldots, x_{n_i}$ are distinct fresh variables of appropriate sorts. The patterns $t[c_i(x_1, \ldots, x_{n_i})]_p$ are called the *children* of t in the pdt, and t their *parent*. Also, the variable $t|_p$ is referred to as *inductive variable* of t.
2. $rule(t, t \to r_1 ? \ldots ? r_k)$ where $t \to r_1, \ldots, t \to r_k$ are distinct variants of rewrite rules from $\mathcal{R}$. We say about this node that it contains the rule variants $t \to r_1, \ldots, t \to r_k$.
3. $exempt(t)$.

The first argument of every node is called the *pattern* of that node, and the set of patterns of a pdt consists of the patterns of its nodes. A *definitional tree* of an operation f/n is a pdt of a pattern $f(x_1, \ldots, x_n)$ where $x_1, \ldots, x_n$ are distinct variables. An operation f of $\mathcal{R}$ is *overlapping inductively sequential* if there exists a definitional tree $\mathcal{T}$ of f whose *rule*-nodes contain a variant for every rule which defines f in $\mathcal{R}$. A TRS is *overlapping inductively sequential* if every operation f is overlapping inductively sequential.

For example, the TRS

$$\begin{aligned}\mathcal{R} = \{\ & ins(x, null) \to cons(x, null),\\ & ins(x, cons(y, z)) \to cons(x, cons(y, z)),\\ & ins(x, cons(y, z)) \to cons(y, ins(x, z))\}\end{aligned}$$

is OIS because its function symbol ins has the definitional tree

$$\begin{aligned}\mathcal{T}_{ins} = branch(&ins(x, y), 2,\\ &rule(ins(x, null) \to cons(x, null)),\\ &rule(ins(x, cons(y_1, z_1)) \to cons(x, cons(y_1, z_1)) ? cons(y_1, ins(x, z_1))))\end{aligned}$$

It is obvious that every OIS is a left-linear constructor-based TRS, and every operation-rooted term $f(t_1, \ldots, t_n)$ is unifiable with some patterns from a definitional tree of f.

A sound and complete narrowing strategy for OIS is inductively sequential narrowing (INS): for every term t, it computes a (possibly empty) set $INS_{\mathcal{R}}(t)$ of triples $\langle p, l \to r, \sigma\rangle$ which indicate the need to perform the narrowing step $t \rightsquigarrow_{p,l\to r,\sigma} t'$, which has the same effect as the rewrite step $t\sigma \to_{p,l\to r} t'$. This strategy presupposes fixed a set $\{\mathcal{T}_f \mid f \in \mathcal{F}\}$ of definitional trees for all operations in $\mathcal{F}$, and is defined as follows:

1. If $t = f(t_1, \ldots, t_n)$ is an operation-rooted term, then $INS_{\mathcal{R}}(t) = \varphi_{\mathcal{R}}(t, \mathcal{T}_f)$ where $\varphi_{\mathcal{R}}(t, \mathcal{T})$ is defined by case distinction on the type of $\mathcal{T}$:
   (a) If $\mathcal{T} = branch(\pi, p, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, let s be the sort of $t|_p$ and $c_1/n_1, \ldots, c_k/n_k$ be the data constructors of sort s in canonical order. We distinguish 3 subcases:

i. If $t|_p = x \in \mathcal{X}$, let $\sigma_i = \{x \to c_i(x_1, \ldots, x_{n_i})\}$ be substitutions with $x_1, \ldots, x_{n_i}$ fresh variables of appropriate sorts, for all $1 \leq i \leq n$.
Then $\varphi_{\mathcal{R}}(t, \mathcal{T}) = \bigcup_{i=1}^{k} \{\langle q, l \to r, \sigma_i \circ \eta_i \rangle \mid \langle q, l \to r, \eta_i \rangle \in \varphi_{\mathcal{R}}(t\sigma_i, \mathcal{T}_i)\}$.

ii. If $t|_p = c_i(t_1, \ldots, t_{n_i})$ then $\varphi(t, \mathcal{T}) = \varphi(t, \mathcal{T}_i)$.

iii. Otherwise, the root of $t|_p$ is an operation. In this case

$$\varphi(t, \mathcal{T}) = \{\langle p \cdot q, l \to r, \eta \rangle \mid \langle q, l \to r, \eta \rangle \in \mathit{INS}_{\mathcal{R}}(t|_p)\}.$$

(b) If $\mathcal{T} = \mathit{rule}(\pi, \pi \to r_1 ? \ldots ? r_k)$ then $\varphi_{\mathcal{R}}(t, \mathcal{T}) = \{\langle \lambda, \pi \to r_i, \emptyset \rangle \mid 1 \leq i \leq k\}$.

(c) If $\mathcal{T} = \mathit{exempt}(\pi)$ then $\varphi_{\mathcal{R}}(t, \mathcal{T}) = \emptyset$.

2. If $t \in \mathcal{X}$ then $\varphi(t, \mathcal{T}) = \emptyset$,

3. Otherwise, $t = c(t_1, \ldots, t_n)$ with c/n a constructor symbol. In this case

$$\mathit{INS}_{\mathcal{R}}(t) = \begin{cases} \emptyset & \text{if } t \in \mathcal{T}(\mathcal{C}, \mathcal{X}), \\ \{\langle i \cdot q, l \to r, \sigma \rangle \mid \langle q, l \to r, \sigma \rangle \in \mathit{INS}_{\mathcal{R}}(t_i)\} & \text{if } i = \min\{j \mid \mathit{INS}_{\mathcal{R}}(t_j) \neq \emptyset\}. \end{cases}$$

The strategy *INS* can also be used to solve systems of equations $s_1 \approx t_1 \,\&\&\, \ldots s_n \approx t_n$ when "$\approx$" is interpreted as joinability to a value. The reason why this is so is that every IOS $\mathcal{R}$ can be extended to an OIS $\mathcal{R}_{eq} = \mathcal{R} \cup \{\rho_c \mid c \in \mathcal{C}\} \cup \{\mathtt{success} \,\&\&\, x \to x\}$, which ensures that $s \to^*_{\mathcal{R}} v$ and $t \to^*_{\mathcal{R}} v$ for some value v iff $s \approx t \to^*_{\mathcal{R}_{eq}}$: every rewrite rule ρ_c is of the form

$$c(x_1, \ldots, x_n) \approx c(y_1, \ldots, y_n) \to x_1 \approx y_1 \,\&\&\, \ldots \,\&\&\, x_n \approx y_n \,\&\&\, \mathtt{success} \tag{2}$$

Thus, we can use *INS* with $\mathcal{R}_{eq}$ to compute a sound and complete set of answers for every system of equations.

Well-known results

Finally, we recall the theoretical results already mentioned in the Introduction, which represented the starting point of our further investigation:

1. Every CB-CTRS can be transformed into an equivalent overlapping inductively sequential TRS with extra variables [3]. This transformation is achieved in two stages:
   - *Linearization:* Every rule $l \to r \Leftarrow c$ where l is a nonlinear pattern is transformed into $l' \to r \Leftarrow c \,\&\&\, c'$ where l' is obtained from l by replacing all its variable occurrences with distinct fresh variables, and c' is a conjunction of equations of the form $x \doteq y$ which express the fact that y is the fresh variable that replaced an occurrence of variable x in l. For example, the linearization of $f(c(x, c(x, y))) \to z \Leftarrow f(c(x, y)) \approx z$ is $f(c(x_1, c(x_2, x_3))) \to z \Leftarrow f(c(x, y)) \approx z \,\&\&\, x_1 \approx x \,\&\&\, x_2 \approx y \,\&\&\, x_3 \approx z$.
   - *Deconditionalization:* Every left-linear conditional rewrite rule $l \to r \Leftarrow c$ is transformed into the unconditional rewrite rule $l \to \mathtt{if}(c, r)$. To properly interpret the operation "$\mathtt{if}$", we add the rule $\mathtt{if}(\mathtt{success}, x) \to x$.
2. The authors of [6] propose a mapping *XEP* that transforms an overlapping inductively sequential TRS $\mathcal{R}$ with extra variables into a set $\mathit{XEP}(\mathcal{R})$ of rules $l \to \langle r, \psi \rangle$ from terms to term/substitution pairs, and a term t into a term/substitution pair $\langle t, \chi \rangle$. Also, they identify a rewrite relation $\to_{\mathcal{R}'}$ induced by $\mathcal{R}' = \mathit{XEP}(\mathcal{R})$ between term/substitution pairs, such that $\{\langle v, \sigma \rangle \mid \mathit{XEP}(t) \to^*_{\mathcal{R}'} \langle v, \sigma \rangle\}$ is a sound and complete of answers for t.

3 Narrowing without the restriction to groundness

The first problem addressed by us is to see how INS can be adjusted to solve equational goals if we drop the restriction on groundness, i.e., we wish to solve systems of equations $s \approx t$ for which σ is solution iff σ is a constructor substitution and $s\sigma \to^*_{\mathcal{R}} u$ and $t\sigma \to^*_{\mathcal{R}} u$ for some $u \in \mathcal{T}(\mathcal{C}, \mathcal{X})$. We can achieve this interpretation for "$\approx$" by defining the TRS with extra variables $\overline{\mathcal{R}} = \mathcal{R}_{eq} \cup \{x \approx x \to \mathtt{success}\}$ and imposing the following restriction on the rewrite relation induced by $\mathcal{R}_{eq}$:

- A rewrite step $t \to_{p,x\approx x\to\mathtt{success},\sigma} t'$ is allowed iff σ is a constructor substitution.

The following result is obvious:

Lemma 1. *θ is a solution of G iff θ is a constructor substitution and $G\theta \to^*_{\overline{\mathcal{R}}} \mathtt{success}$.*

Unfortunately, strategy *INS* does not work well for $\overline{R}$ because $\overline{R}$ is not OIS. To fix this problem, we will make the following general assumptions:

- For every given sort s, we assume predefined an enumeration of all its constructors, and call it *canonical order*.
- $\mathcal{T}^s_{\approx} = branch(x \approx y, 1, \mathcal{T}_1, \ldots, \mathcal{T}_m)$ is the definitional tree for the operation $\approx$ between terms of sort s, with the following structure:
  - $c_1/n_1, \ldots, c_m/n_m$ is the enumeration of all constructors of sort s in canonical order,
  - Every $\mathcal{T}_i = branch(\pi_i, 2, \mathcal{T}_{i,1}, \ldots, \mathcal{T}_{i,m})$ is the partial definitional tree of the linear pattern $\pi_i = c_i(x_1, \ldots, x_{n_i}) \approx y$ where

$$\mathcal{T}_{i,j} = \begin{cases} exempt(c_i(x_1, \ldots, x_{n_i}) \approx c_j(y_1, \ldots, y_{n_j})) & \text{if } i \neq j, \\ rule(\rho_c) & \text{if } i = j. \end{cases}$$

Next, we specialize *INS* to work for $\overline{R}$ in the following way:

1. $INS_{\overline{\mathcal{R}}}(t_1 \approx t_2)$ is either
   (a) $\langle \lambda, x \approx x \to \mathtt{success}, \{t_1 \to t_2\}\rangle$ if $t_1, t_2 \in \mathcal{X}$ and $t_1 \neq t_2$.
   (b) $\langle \lambda, x \approx x \to \mathtt{success}, \{\}\rangle$ if $t_1 = t_2 \in \mathcal{X}$.
   (c) $\varphi_{\overline{\mathcal{R}}}(t_1 \approx t_2, \mathcal{T}^s_{\approx})$ otherwise, where s is the sort of t_1.
2. $INS_{\overline{\mathcal{R}}}(t)$ is defined like $INS_{\mathcal{R}_{eq}}(t)$ in all other cases.

The following result follows from the observation that, if $s \rightsquigarrow_{\overline{\mathcal{R}},\sigma} t$ then σ is a constructor substitution and $s\sigma \to_{\overline{\mathcal{R}}} t$.

Lemma 2 (Soundness)**.** *If $\langle \mathtt{success}, \theta\rangle \in Ans^{INS}_{\overline{\mathcal{R}}}(G)$ then θ is a constructor substitution and $G\theta \to^*_{\overline{\mathcal{R}}} \mathtt{success}$.*

Also, one can prove that $INS_{\overline{\mathcal{R}}}$ is a complete narrowing strategy:

Lemma 3 (Completeness)**.** *If θ is a constructor substitution and $G\theta \to^*_{\overline{\mathcal{R}}} \mathtt{success}$ then there exists $\langle \mathtt{success}, \theta'\rangle \in Ans^{INS}_{\overline{\mathcal{R}}}(G)$ such that $\theta' \leq \theta$.*

Thus, the narrowing strategy $INS_{\overline{\mathcal{R}}}$ is indeed sound and complete.

4 A rewrite-based strategy

In this section we introduce our second contribution: a rewrite-based strategy capable to simulate $INS_{\mathcal{R}_{eq}}$-narrowing derivations. The insight behind the design of this strategy is based on following observation: Every successful search of a narrowing step $\langle p, l \to r, \sigma\rangle$ with strategy $INS_{\overline{R}}$ can be decomposed into a sequence of elementary steps of three kinds:

T1. Binding steps, which bind a variable to a constructor term. To this category belong case 1.(a).i from the unspecialized definition of $INS_{\overline{\mathcal{R}}}$, and case 1.(a) from the definition of the specialized version.

T2. Lookup steps, which simply traverse the term in depth to detect a suitable narrowing position. To this category belong cases 1.(a).ii, 1.(a).iii, and 3 from the unspecialized definition of $INS_{\overline{\mathcal{R}}}$.

T3. Final search steps, which detect the possibility to rewrite an instance of a term of the traversed term with a rewrite rule at a certain position. To this category belong cases 1.(b) from the unspecialized definition of $INS_{\overline{\mathcal{R}}}$, and case 1.(b) in the definition of the specialized version.

Starting from this observation, we can "sequentialize" strategy INS to stop as soon as it performs the first step of type T1 or type T3, and return the corresponding variable binding (if the step type is T1) or position/rule (if the step type is T3). We call IRS the "sequentialized" version of INS, because it is intended to stand for **I***nductively sequential* **R***ewriting* **S***trategy*:

- When it returns a binding of the form $x \to t$, it indicates the need to perform maximum parallel rewriting with the rewrite rule $x \to t$ as if x were a constant. Note that such a parallel rewrite step has the same effect as applying the substitution $\{x \to t\}$.
- When it returns a pair $\langle p, l \to r\rangle$, it indicates the need to perform the rewrite step with rule $l \to r$ at position p.

In order to ensure the fact that IRS simulates INS, we must guarantee that every call of the IRS strategy proceeds from the place where the previous call of IRS stopped the computation of an INS-step. Based on these considerations, we came up with the following definition of $IRS_{\overline{\mathcal{R}}}$:

- $IRS_{\overline{\mathcal{R}}}(x \approx y) = IRS_{\mathcal{R}_{eq}}(y \approx x) = \{x \to y\}$ if $x, y \in \mathcal{X}$ and $x \neq y$.
  $IRS_{\overline{\mathcal{R}}}(y \approx y) = \{\langle \lambda, x \approx x \to \texttt{success}\rangle\}$ if $y \in \mathcal{X}$.
  $IRS_{\overline{\mathcal{R}}}(t_1 \approx t_2) = DT_{\overline{\mathcal{R}}}(t_1 \approx t_2, \mathcal{T}^s_{\approx})$ if $\{t_1, t_2\} \not\subset \mathcal{X}$ and s is the sort of t_1.
- $IRS_{\overline{\mathcal{R}}}(t) = DT_{\overline{\mathcal{R}}}(t, \mathcal{T}_f)$ where $IRS\_DT(t, \mathcal{T})$ is defined as follows:
- Otherwise, there exists a leftmost outermost position $p \neq \lambda$ of t such that $t|_p = f(t_1, \ldots, t_n)$ with $\in \mathcal{F}$. Let $A = IRS_{\overline{\mathcal{R}}}(t|_p)$. Then
  $$IRS_{\overline{\mathcal{R}}}(t) = \begin{cases} \{\langle p \cdot q, \rho\rangle \mid \langle q, \rho\rangle \in A & \text{if } A \neq \emptyset \text{ is set of pairs,} \\ A & \text{otherwise.} \end{cases}$$

The auxiliary method $DR_{\overline{\mathcal{R}}}(t, \mathcal{T})$ is defined as follows:

1. If $\mathcal{T} = branch(\pi, p, \mathcal{T}_1, \ldots, \mathcal{T}_k)$, let s be the sort of $t|_p$ and $c_1/n_1, \ldots, c_k/n_k$ be the data constructors of sort s in canonical order. We distinguish 3 subcases:
   (a) If $t|_p = x \in \mathcal{X}$, let $\sigma_i = \{x \to c_i(x_1, \ldots, x_{n_i})\}$ be substitutions with $x_1, \ldots, x_{n_i}$ fresh variables of appropriate sorts, for all $1 \leq i \leq n$. Then
   $DT_{\overline{\mathcal{R}}}(t, \mathcal{T}) = \{x \to c_i(x_1, \ldots, x_{n_i}) \mid 1 \leq i \leq k\}$.
   (b) If $t|_p = c_i(s_1, \ldots, s_{n_i})$ then $DT_{\overline{\mathcal{R}}}(t, \mathcal{T}) = DT_{\overline{\mathcal{R}}}(t, \mathcal{T}_i)$.

(c) If $t|_p = g(s_1, \ldots, s_m)$ with $g \in \mathcal{F}$, let $A = DT_{\overline{\mathcal{R}}}(t|_p, \mathcal{T}_g)$. Then
$$DT_{\overline{\mathcal{R}}}(t, \mathcal{T}) = \begin{cases} \{\langle p \cdot q, \rho\rangle \mid \langle q, \rho\rangle \in A\} & \text{if } A \neq \emptyset \text{ is set of pairs,} \\ A & \text{otherwise.} \end{cases}$$

2. If $\mathcal{T} = rule(\pi, \pi \to r_1 ? \ldots ? r_k)$ then $DT_{\overline{\mathcal{R}}}(t, \mathcal{T}) = \{\langle \lambda, \pi \to r_i\rangle \mid 1 \leq i \leq k\}$.
3. Otherwise, $\mathcal{T} = exempt(\pi)$ and $DT_{\overline{\mathcal{R}}}(t, \mathcal{T}) = \emptyset$.

An *IRS-compliant rewrite step* of a term t w.r.t. $\overline{\mathcal{R}}$ is a relation $t \Rightarrow_\sigma t'$ which holds if either (1) $\langle p, \rho\rangle \in IRS_{\overline{\mathcal{R}}}(t)$. In this case $\sigma = \{\}$ and t' is the result of the rewrite step $t \to_{p,\rho} t'$, or (2) $x \to t$ is a substitution rule from $IRS_{\overline{\mathcal{R}}}(t)$. In this case $\sigma = \{x \to t\}$ and t' is the result of maximal parallel rewriting with the rewrite rule $x \to t$, as if x were a constant. An *IRS-compliant rewrite derivation* of a term t w.r.t. $\overline{\mathcal{R}}$ is a sequence $t \Rightarrow_{\sigma_1} t_1 \Rightarrow_{\sigma_2} \ldots \Rightarrow_{\sigma_n} t_n$, abbreviated $t \Rightarrow^*_\sigma t_n$ where $\sigma = (\sigma_n \circ \ldots \circ \sigma_2 \circ \sigma_1)|_{var(t)}$. For a given system of equations G, the set of answers computed by our rewrite-based strategy is

$$Ans^{IRS}_{\overline{\mathcal{R}}}(G) = \{\langle \texttt{success}, \sigma\rangle \mid G \Rightarrow^*_\sigma \texttt{success}\}.$$

Theorem 1. *Let $\mathcal{R}$ be an OIS. Then $Ans^{IRS}_{\overline{\mathcal{R}}}(G) = Ans^{INS}_{\overline{\mathcal{R}}}(G)$.* □

Since strategy $INS_{\overline{\mathcal{R}}}$ is sound and complete, we conclude that strategy $IRS_{\overline{\mathcal{R}}}$ is sound and complete too.

5 Concluding remarks

Overlapping inductively sequential TRSs with extra variables have emerged as a convenient core language for FLP computations. Unfortunately, the INS strategy can not be applied directly to solve systems of equations where equality is interpreted as reducibility to a common constructor term. The reason for this is that there is no way to define this kind of equality with rewrite rules of this kind. We have identified a simple way to overcome this limitation: to extend the underlying OIS with the nonlinear rewrite rule $x \approx x \to \texttt{success}$, and to impose restrictions on the way it is used in rewrite and narrowing derivations. By exploring this idea, we have identified a simple adjustment of strategy INS, which is sound and complete for our generalized interpretation of equality (Section 3). Moreover, we found a way to simulate the narrowing computations of interest to us with a strategy for rewrite derivations (Section 4). The efficiency of both strategies stems from the fact that both of them are controlled by definitional trees.

References

[1] Sergio Antoy. Definitional trees. In *In Proc. of the 3rd International Conference on Algebraic and Logic Programming*, pages 143–157. Springer LNCS, 1992.

[2] Sergio Antoy. Optimal non-deterministic functional logic computations. In Michael Hanus, Jan Heering, and Karl Meinke, editors, *Algebraic and Logic Programming*, volume 1298 of *Lecture Notes in Computer Science*, pages 16–30. Springer Berlin Heidelberg, 1997.

[3] Sergio Antoy. Constructor-based conditional narrowing. In *Proceedings of PPDP 2001*, pages 199–206. ACM Press, 2001.

[4] Sergio Antoy, Rachid Echahed, and Michael Hanus. A needed narrowing strategy. In *Journal of the ACM*, pages 268–279. ACM Press, 1994.

[5] Sergio Antoy, Rachid Echahed, and Michael Hanus. Parallel Evaluation Strategies for Functional Logic Languages. In *Proc. of the Fourteenth International Conference on Logic Programming (ICLP'97)*, pages 138–152. MIT Press, 1997.

[6] Sergio Antoy and Michael Hanus. Overlapping rules and logic variables in functional logic programs. In *22nd International Conference on Logic Programming*, pages 87–101. Springer LNCS, 2006.

[7] Sergio Antoy and Michael Hanus. Functional logic programming. *Communications of the ACM*, 53(4):74–85, 2010.

[8] J.C. Gonzalez-Moreno, M.T. Hortala-Gonzalez, F.J. Lopez-Fraguas, and M. Rodriguez-Artalejo. An approach to declarative programming based on a rewriting logic. *The Journal of Logic Programming*, 40(1):47–87, 1999.

[9] Michael Hanus, Salvador Lucas, and Aart Middeldorp. Strongly sequential and inductively sequential term rewriting systems. *Information Processing Letters*, 67(1):1–8, 1998.

[10] Jean-Marie Hullot. Canonical forms and unification. In Wolfgang Bibel and Robert Kowalski, editors, *5th Conference on Automated Deduction Les Arcs, France, July 8-11, 1980*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer Berlin Heidelberg, 1980.

[11] Tetsuo Ida, Mircea Marin, and Taro Suzuki. Higher-Order Lazy Narrowing Calculus: A Solver for Higher-Order Equations. In *Computer Aided Systems Theory - EUROCAST 2001-Revised Papers*, pages 479–493, London, UK, 2001. Springer-Verlag.

[12] Mircea Marin. *Functional Logic Programming with Distributed Constraint Solving*. PhD thesis, RISC-Linz Institute, Schloss Hagenberg, Austria, 2000.

[13] Mircea Marin and Aart Middeldorp. New completeness results for lazy conditional narrowing. In *Proceedings of the 6th ACM SIGPLAN Conference on Principles and Practice of Declarative Programming*, pages 120–131, Verona, 2004. ACM Press.

[14] Aart Middeldorp and Satoshi Okui. A deterministic lazy narrowing calculus. *Journal of Symbolic Computation*, 25(6):733–757, 1998.

[15] Aart Middeldorp, Satoshi Okui, and Tetsuo Ida. Lazy narrowing: Strong completeness and eager variable elimination. *Theoretical Computer Science*, 167(1,2):95–130, 1996.

[16] Aart Middeldorp, Taro Suzuki, and Mohamed Hamada. Complete selection functions for a lazy conditional narrowing calculus. *Journal of Functional and Logic Programming*, 2002(3), March 2002.

[17] Christian Prehofer. *Solving higher-order equations: from logic to programming*. Birkhauser, 1998.