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 Timișoara, Timișoara, 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 \( t \rightarrow^*_{\mathcal{R}} v \)), but rather to compute a set of pairs \( \langle \sigma, v \rangle \) made of a ground constructor substitution \( \sigma \) for the variables of \( t \) and a value \( v \), such that \( t\sigma \rightarrow^*_{\mathcal{R}} v \). We may denote the set of all such \( \sigma \) 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 \( \sigma \) is constructor substitution, \( v \) is a possibly non-ground constructor term, and \( t\sigma \rightarrow^*_{\mathcal{R}} v \). 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 \( \eta \) 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

$$e_1 \& \& \ldots \& \& e_n$$

(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: $$\sigma$$ is a solution of an equational goal $$G = s_1 \approx t_1 \& \& \ldots \& \& s_n \approx t_n$$ iff $$G\sigma \rightarrow^*_{\mathcal{R}'} \text{success}$$. (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 $$\mathcal{R}' = \mathcal{R} \cup \{x \approx x \rightarrow \text{success}\}$$, 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], LNCd [14], LCNC [8], LNC [10], and LCNCd [13]. Higher-order versions of narrowing calculi are also known, e.g., LN [17], HOLN [11], and LNff [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 [3], for weakly orthogonal TRSs, and the INS narrowing strategy [2] for overlappng 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 [9], 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.

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 ourselves 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 and 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 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 \( \Sigma \) partitioned into a set \( C \) of (data) constructors, and a set \( F \) 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 \( X \) to be a countably infinite set of sorted variables. The sets \( T(\Sigma, X) \), \( T(C, X) \) and \( T(C) \) of well-sorted terms, constructor terms, and ground constructor terms are defined as usual. The terms of \( T(C) \) are also called values.

We write \( \text{var}(t) \) for the set of variables which occur in a term \( t \). A term \( t \) is ground if \( \text{var}(t) = \emptyset \), and linear if it does not contain multiple occurrences of a variable. A pattern is a term \( f(t_1, \ldots, t_n) \) where \( f/n \in F \) and \( t_1, \ldots, t_n \in T(C, X) \). 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 \( l \rightarrow r \) where \( l \) is a pattern, and \( r \) is a term of the same sort as \( l \).

A conditional rewrite rule is of the form \( l \rightarrow r \Leftarrow c \) 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 \( l \rightarrow r \) (resp. \( l \rightarrow r \Leftarrow c \)) is \( \text{evar}(l \rightarrow r) := \text{var}(r) \setminus \text{var}(l) \) (resp. \( \text{evar}(l \rightarrow r \Leftarrow c) := (\text{var}(r) \cup \text{var}(c)) \setminus \text{var}(l) \)). 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 \( \sigma : X \rightarrow T(\Sigma, X) \) such that its domain \( \text{dom}(\sigma) := \{ x \in X \mid \sigma(x) \neq x \} \) is finite, and \( x \) and \( \sigma(x) \) are of the same sort for every \( x \in X \). It is common practice to denote a substitution \( \sigma \) by the finite set \( \{ x \mapsto \sigma(x) \mid x \in \text{dom}(\sigma) \} \). The restriction of a substitution \( \sigma \) to a set of variables \( V \subseteq X \) is the substitution \( \sigma|_V \) with \( \text{dom}(\sigma|_V) = \text{dom}(\sigma) \cap V \) and \( \sigma|_V(x) = \sigma(x) \) for all \( x \in \text{dom}(\sigma|_V) \). A (ground) constructor substitution is a substitution \( \sigma \) such that \( \sigma(x) \) is a (ground) constructor term for every \( x \in \text{dom}(\sigma) \). 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 \( \sigma \). Subsumption is the ordering on terms defined by \( s \leq t \) if \( \text{sat} = t \) for some substitution \( \sigma \). 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 \( \eta \) such that \( \sigma' = (\eta \circ \sigma)|_{\text{vars}(\sigma')} \).

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 \( \text{var}(t') \) consists of variables which did not occur in the expressions encountered so far.

A unifier of two terms \( s \) and \( t \) is a substitution \( \sigma \) such that \( s\sigma = t\sigma \). The unifier \( \sigma \) is a most general unifier (mgu for short) if \( \sigma \leq \sigma' \) for any other unifier \( \sigma' \) of \( s \) and \( t \). We write \( s < t \) for the restriction of an mgu of \( s \) and \( t \) to \( \text{vars}(s) \).

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

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

Rewriting with a CB-CTRS \( R \) is slightly more involved: A rewrite step of a term \( t \) w.r.t. \( R \) is a relation \( t \rightarrow_{p,l\rightarrow r,\sigma} t' \), abbreviated \( t \rightarrow_R t' \), which holds if there exist a position \( p \) in \( t \), a rule \( l \rightarrow r \leftarrow c \) from \( R \) with fresh variant \( l' \rightarrow r' \leftarrow c' \), and a substitution \( \sigma \) such that \( t|_p = l'|\sigma \) and \( t' = t[l'\sigma]_p \). \( \sigma \) is a solution of \( c' \), i.e., for every equation \( u \approx v \) from \( c' \), we have \( u\sigma \downarrow_R v\sigma \).

A narrowing step \( s \rightsquigarrow_R \sigma \) of a term \( s \) with respect to program presented to a TRS \( R \) is defined if there exist a non-variable position \( p \) of \( s \), a fresh variant \( l \rightarrow r \) of a rewrite rule from \( R \), and a unifier \( \sigma \) of \( s|_p \) and \( l \), such that \( t = s[l|p|\sigma] \). If \( R \) is a conditional TRS, the definition is slightly more involved: for a fresh variant \( l \rightarrow r \leftarrow c \) of a rewrite rule from \( R \) we also require to have \( \sigma \) solution of the conditional part \( c \). Here we indicate an equivalent definition which is more convenient for our further investigation: \( s \rightsquigarrow_R \sigma \) is a narrowing step of \( s \) w.r.t. a program \( R \) if \( \sigma \rightarrow_R t \) is a rewrite step at a non-variable position \( p \) of \( s \) (that is, \( s|_p \not\in X \)).

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

For any constructor substitution \( \sigma' \) and \( v' \in T(C, X) \) such that \( t\sigma' \rightarrow_R^* v' \), there exists \( \langle v, \sigma \rangle \in \text{Ans}(t) \) such that \( \sigma \leq \sigma' \) and \( v \leq v' \).

Narrowing-based computational models compute the elements \( \langle v, \sigma \rangle \) of \( \text{Ans}(t) \) by producing a set of narrowing derivations \( t \rightsquigarrow_R^* \sigma v \) with constructor term \( v \). Soundness follows from the observation that \( t \rightsquigarrow_R^* \sigma v \) implies \( t\sigma \rightarrow_R^* v \), and from the restriction to perform only narrowing steps \( t \rightsquigarrow_R^* \sigma, t_{i+1} \) with constructor substitution \( \sigma_i \). 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 non-variable position \( p \) of \( s \), a fresh variant of a rule \( \rho \in R \), and a constructor substitution.
such that we can perform a narrowing step \( s \leadsto_{p,\rho,\sigma} t \). The evaluation of a term \( s \) with \( S \) yields the set of computed answers \( \text{Ans}_R^S(s) = \{ (v, \sigma) \mid s \leadsto_{p,\rho,\sigma}^* v \text{ where } v \in T(C,A) \} \) and \((p, \rho, \sigma) \in S(t_i)\) for every component narrowing step \( t_i \leadsto_{p,\rho,\sigma} t_i' \).

### 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 \( T \) with nodes of one of the following three forms:

1. \( \text{branch}(t, p, T_1, \ldots, T_k) \) 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 \( T_i \) (1 \( \leq i \leq k \)) 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. \( \text{rule}(t, t \to r_1 ? \ldots ? r_k) \) 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. \( \text{exempt}(t) \).

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 \( T \) 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

\[
\mathcal{R} = \{ \text{ins}(x, \text{null}) \to \text{cons}(x, \text{null}), \\
\text{ins}(x, \text{cons}(y, z)) \to \text{cons}(x, \text{cons}(y, z)), \\
\text{ins}(x, \text{cons}(y, z)) \to \text{cons}(y, \text{ins}(x, z)) \}
\]

is OIS because its function symbol \( \text{ins} \) has the definitional tree

\[
T_{\text{ins}} = \text{branch}(\text{ins}(x, y), 2, \\
\text{rule}(\text{ins}(x, \text{null}) \to \text{cons}(x, \text{null})), \\
\text{rule}(\text{ins}(x, \text{cons}(y_1, z_1)) \to \text{cons}(x, \text{cons}(y_1, z_1)) \? \text{cons}(y_1, \text{ins}(x, z_1)))
\]

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 \( \text{INS}_\mathcal{R}(t) \) of tuples \( (p, l \to r, \sigma) \) which indicate the need to perform the narrowing step \( t \leadsto_{p,l \to r,\sigma} t' \), which has the same effect as the rewrite step \( t \sigma \leadsto_{p,l \to r} t' \). This strategy presupposes fixed a set \( \{ T_f \mid f \in \mathcal{F} \} \) 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 \( \text{INS}_\mathcal{R}(t) = \varphi_\mathcal{R}(t, T_f) \) where \( \varphi_\mathcal{R}(t, T) \) is defined by case distinction on the type of \( T \):

   a) If \( T = \text{branch}(\pi, p, T_1, \ldots, T_k) \), 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:
Finally, we recall the theoretical results already mentioned in the Introduction, which represent 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 \(l \rightarrow r \Leftarrow c\) where \(l\) is a nonlinear pattern is transformed into \(l' \rightarrow r \Leftarrow c \# k \& c'\) 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 \(x = y\) which express the fact that \(y\) is the fresh variable that replaced an occurrence of variable \(x\) in \(l\). For example, the linearization of \(f(c(x, c(x, y))) \rightarrow z \Leftarrow f(c(x, y)) \approx z\) is transformed into the unconditioned rewrite rule \(l \rightarrow if(c, r)\).

   - **Deconditionals**: Every left-linear conditional rewrite rule \(l \rightarrow r \Leftarrow c\) is transformed into the unconditional rewrite rule \(l \rightarrow if(c, r)\). To properly interpret the operation “if”, we add the rule \(if(success, x) \rightarrow x\).

2. The authors of [6] propose a mapping \(XEP\) that transforms an overlapping inductively sequential TRS \(R\) with extra variables into a set \(XEP(R)\) of rules \(l \rightarrow \langle r, \psi \rangle\) from terms to term/substitution pairs, and a term \(t\) into a term/substitution pair \((t, \chi)\). Also, they identify a rewrite relation \(\rightarrow_R\) induced by \(R' = XEP(R)\) between term/substitution pairs, such that \((v, \sigma) \in XEP(t) \rightarrow_R (v, \sigma)\) is a sound and complete set 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 \( \sigma \) is solution iff \( \sigma \) is a constructor substitution and \( s \sigma \rightarrow^* u \) and \( t \sigma \rightarrow^* u \) for some \( u \in T(C, X) \). We can achieve this interpretation for \( \approx \) by defining the TRS with extra variables \( R = R_{eq} \cup \{ x \approx x \rightarrow \text{success} \} \) and imposing the following restriction on the rewrite relation induced by \( R_{eq} \):

- A rewrite step \( t \rightarrow_{p,x\approx x\rightarrow \text{success},\sigma} t' \) is allowed iff \( \sigma \) is a constructor substitution.

The following result is obvious:

**Lemma 1.** \( \theta \) is a solution of \( G \) iff \( \theta \) is a constructor substitution and \( G\theta \rightarrow^* \text{success} \).

Unfortunately, strategy INS does not work well for \( R \) because \( R \) 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.
- \( T_s \approx = \text{branch}(x \approx y, 1, T_{1s}, \ldots, T_{ms}) \) 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 \( T_i = \text{branch}(\pi_i, 2, T_{i1}, \ldots, T_{im}) \) is the partial definitional tree of the linear pattern \( \pi_i = c_i(x_1, \ldots, x_n) \approx y \) where
    \[
    T_{i,j} = \begin{cases} 
    \text{exempt}(c_i(x_1, \ldots, x_n) \approx c_j(y_1, \ldots, y_n)) & \text{if } i \neq j, \\
    \text{rule}(\rho_c) & \text{if } i = j.
    \end{cases}
    \]

Next, we specialize INS to work for \( R \) in the following way:

1. ***INS\(_R\)(t_1 \approx t_2)*** is either
   - (a) \( \langle \lambda, x \approx x \rightarrow \text{success}, \{ t_1 \rightarrow t_2 \} \rangle \) if \( t_1, t_2 \in X \) and \( t_1 \neq t_2 \).
   - (b) \( \langle \lambda, x \approx x \rightarrow \text{success}, \{ \} \rangle \) if \( t_1 = t_2 \in X \).
   - (c) \( \varphi_R(t_1 \approx t_2, T_s) \) otherwise, where \( s \) is the sort of \( t_1 \).

2. ***INS\(_R\)(t)*** is defined like ***INS\(_{R_{eq}}\)(t)*** in all other cases.

The following result follows from the observation that, if \( s \rightarrow_{R,\sigma} t \) then \( \sigma \) is a constructor substitution and \( s \sigma \rightarrow_{R} t \).

**Lemma 2 (Soundness).** If \( \langle \text{success}, \theta \rangle \in \text{Ans}^{INS}_R(G) \) then \( \theta \) is a constructor substitution and \( G\theta \rightarrow_{R}^* \text{success} \).

Also, one can prove that ***INS\(_R\)*** is a complete narrowing strategy:

**Lemma 3 (Completeness).** If \( \theta \) is a constructor substitution and \( G\theta \rightarrow_{R}^* \text{success} \) then there exists \( \langle \text{success}, \theta' \rangle \in \text{Ans}^{INS}_R(G) \) such that \( \theta' \leq \theta \).

Thus, the narrowing strategy ***INS\(_R\)*** 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 \( INS_{\pi_q} \)-narrowing derivations. The insight behind the design of this strategy is based on following observation: Every successful search of a narrowing step \( \langle p, l \rightarrow r, \sigma \rangle \) with strategy \( INS_{\pi} \) 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 \( INS_{\pi} \), 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 \( INS_{\pi} \).

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 \( INS_{\pi} \), 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

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 Inductively sequential Rewriting Strategy:

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

- \( IRS_{\pi}(x \approx y) = IRS_{\pi_q}(y \approx x) = \{ x \rightarrow y \} \) if \( x, y \in \mathcal{X} \) and \( x \neq y \).
- \( IRS_{\pi}(y \approx y) = \{ (\lambda, x \approx x \rightarrow success) \} \) if \( y \in \mathcal{X} \).
- \( IRS_{\pi}(t_1 \approx t_2) = DT_{\pi}(t_1 \approx t_2, T_s) \) if \( \{ t_1, t_2 \} \not\subseteq \mathcal{X} \) and \( s \) is the sort of \( t_1 \).
- \( IRS_{\pi}(t) = DT_{\pi}(t, T_f) \) where \( IRS_{\pi}(t, T) \) 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 \( A = IRS_{\pi}(t|_p) \). Then

    \[
    IRS_{\pi}(t) = \begin{cases} 
    \{ (p \cdot q, \rho) \mid (q, \rho) \in A \} & \text{if} \ A \neq \emptyset \text{ is set of pairs,} \\
    A & \text{otherwise.}
    \end{cases}
    \]

The auxiliary method \( DR_{\pi}(t, T) \) is defined as follows:

1. If \( T = branch(\pi, p, T_1, \ldots, T_k) \), 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 \( \sigma_i = \{ x \rightarrow c_i(x_1, \ldots, x_n) \} \) be substitutions with \( x_1, \ldots, x_n \) fresh variables of appropriate sorts, for all \( 1 \leq i \leq n \). Then

     \[
     DT_{\pi}(t, T) = \{ x \rightarrow c_i(x_1, \ldots, x_n) \mid 1 \leq i \leq k \}.
     \]

   (b) If \( t|_p = c_i(s_1, \ldots, s_n) \) then \( DT_{\pi}(t, T) = DT_{\pi}(t, T_i) \).
(c) If \( t\rho = g(s_1, \ldots, s_m) \) with \( g \in \mathcal{F} \), let \( A = DT_{\mathcal{R}}(t\rho, T_0) \). Then
\[
DT_{\mathcal{R}}(t, T) = \begin{cases} 
\{ \langle p \cdot q, \rho \rangle \mid \{ q, \rho \} \in A \} & \text{if } A \neq \emptyset \text{ is set of pairs}, \\
A & \text{otherwise}. 
\end{cases}
\]

2. If \( T = \text{rule}(\pi, \pi \rightarrow r_1? \ldots ?r_k) \) then \( DT_{\mathcal{R}}(t, T) = \{ \langle \lambda, \pi \rightarrow r_i \rangle \mid 1 \leq i \leq k \} \).

3. Otherwise, \( T = \text{exempt}(\pi) \) and \( DT_{\mathcal{R}}(t, T) = \emptyset \).

An IRS-compliant rewrite step of a term \( t \) w.r.t. \( \mathcal{R} \) is a relation \( t \Rightarrow_{\mathcal{R}} t' \) which holds if either (1) \( \langle p, \rho \rangle \in IRS_{\mathcal{R}}(t) \). In this case \( \sigma = \{ \} \) and \( t' \) is the result of the rewrite step \( t \Rightarrow_{p, \rho} t' \), or (2) \( x \rightarrow t \) is a substitution rule from \( IRS_{\mathcal{R}}(t) \). In this case \( \sigma = \{ x \rightarrow t \} \) and \( t' \) is the result of maximal parallel rewriting with the rewrite rule \( x \rightarrow t \), as if \( x \) were a constant. An IRS-compliant rewrite derivation of a term \( t \) w.r.t. \( \mathcal{R} \) is a sequence \( t \Rightarrow_{\mathcal{R}} t_1 \Rightarrow_{\mathcal{R}} t_2 \ldots \Rightarrow_{\mathcal{R}} t_n \), abbreviated \( t \Rightarrow_{\mathcal{R}} t_n \) where \( \sigma = (\sigma_n \circ \ldots \circ \sigma_2 \circ \sigma_1)_{\text{var}(t)} \). For a given system of equations \( G \), the set of answers computed by our rewrite-based strategy is
\[
\text{Ans}_{\mathcal{R}}^{IRS}(G) = \{ \langle \text{success}, \sigma \rangle \mid G \Rightarrow_{\mathcal{R}}^* \text{ success} \}.
\]

\begin{theorem}
Let \( \mathcal{R} \) be an OIS. Then \( \text{Ans}_{\mathcal{R}}^{IRS}(G) = \text{Ans}_{\mathcal{R}}^{INS}(G) \).
\end{theorem}

Since strategy \( INS_{\mathcal{R}} \) is sound and complete, we conclude that strategy \( IRS_{\mathcal{R}} \) 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 \( x \approx x \rightarrow \text{success} \), 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 4). 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


