Jeopardy:
Inverting Mildly Deep Definitions

Nachum Dershowitz
Department of Computer Science
Tel-Aviv University
Ramat Aviv, Tel-Aviv 69978, Israel

Subrata Mitra
Enterprise Component Technology
162 36th "A" Cross, 3rd Main, 7th Block
Jayanagar, Bangalore 560 082, India

April 8, 1999

Jeopardy? Isn't that a game show?
—Faye Kellerman, *Prayers for the Dead*

Abstract

We consider functions defined by ground-convergent left-linear rewrite systems. By restricting the depth of left sides and disallowing defined symbols at the top of right sides, we obtain an algorithm for function inversion.

1 Motivation

It is thought that some ancient cultures employed a solar calendar with a fixed year-length of 360 days and a simple scheme of 12 equal-length months. Imagine a 0-based version of such a calendar. Given a date $\langle d, m, y\rangle$ consisting of a year number y, month number m and day number d, it is trivial to calculate the number of elapsed days since the onset of the calendar on date $\langle 0, 0, 0\rangle$:

$$n(d, m, y) = 360 \times y + 30 \times m + d \tag{1}$$

To facilitate conversion of dates between calendars (see [Dershowitz and Reingold, 1997]), one also needs to compute the inverse of n to find the date $\langle d, m, y\rangle$ corresponding to a given number of elapsed days N. The appropriate function is not all that trivial:

$$n^-(N) = \langle N \bmod 30, \lfloor (N \bmod 360)/30 \rfloor, \lfloor N/360 \rfloor\rangle \tag{2}$$

The ideal of logic programming suggests that one should only need to specify the function n and simply let the programming language do the "dirty work" and solve for $\langle d, m, y \rangle$, given any N. That is, we want the machine to determine the appropriate question for a given answer, as in the popular game "Jeopardy". Narrowing (or any other complete semantic-unification procedure), given a goal $n(d, m, y) =^? 400$ (and some appropriate definitions of operations on natural numbers) would yield the sought-after solution $d = 10,\ m = 1,\ y = 1$. Unfortunately, it would also find numerous undesirable solutions, such as $d = 40,\ m = 12,\ y = 0$. Worse, unadulterated narrowing will continue forever seeking additional, nonexistent solutions for $y > 1$. To eliminate the undesired "solutions", one would have to add the missing parts of the specification in the form of constraints:

$$n(d, m, y) =^? N,\ d < 30 =^? T,\ m < 12 =^? T$$

To preclude nontermination, one could add "failure-causing" rules:

$$\begin{array}{rcl} s(x) =^? s(y) & \rightarrow & x =^? y \\ s(x) =^? 0 & \rightarrow & F \\ 0 =^? s(y) & \rightarrow & F \end{array} \tag{3}$$

Applied eagerly (as suggested in [Dershowitz and Plaisted, 1988]), these rules prune unsatisfiable inversion goals.

Thus, we are interested in the problem of solving sets of equations of the form $t = N$, where t is an arbitrary term containing defined function symbols, constructors (that is, undefined function symbols and constants), and variables, while N is a *value*, by which we mean a term containing constructors only, without defined symbols or variables. We will describe a broad class of functions that can be inverted in this manner. Failure rules, like (3), which hold in general for constructors in convergent systems, are built into the algorithm.

More generally, semantic matching is the process of generating a basis set of substitutions that, when applied to a "pattern" term, gives a term equal (in some theory) to a given "target" term. In other words, matching is the special ("one-way") case of (semantic) unification in which one of the two terms to be unified is ground (variable-free). Matching algorithms are required for pattern application in functional languages and have potential uses in logic-based languages. For example, given the usual definitions for *append* and *reverse* on lists, it is natural to implicitly define a predicate for checking if a list is a palindrome in the following manner:

$$\begin{array}{rcl} palindrome(append(x, reverse(x))) & = & T \\ palindrome(append(x, a : reverse(x))) & = & T \end{array} \tag{4}$$

To use such definitions within a functional pattern-directed language, it is necessary to match patterns of the form $append(x, reverse(x))$ against values like 1: 2: 2: 1: ϵ. To perform such matches—which is not possible in current functional languages—an inversion algorithm is required.

We restrict ourselves to equational theories that are presented as (rather typical) functional programs in the form of ground-convergent left-linear rewrite systems. Though for arbitrary linear systems, matching is unsolvable, by placing (not wholly unrealistic) syntactic restrictions on the sides of rules (left sides are restricted in depth and right sides may not have arbitrary defined symbols at the top), we can show termination of our inversion algorithm. We do not actually require sufficient completeness (only convergence), so some ground terms may have non-constructor normal forms.

For calendar computation from scratch, we also need a program for (unary) "natural arithmetic", such as

$$\begin{array}{rcl} x + 0 & \to & x \\ x + s(y) & \to & s(x + y) \\ x \times 0 & \to & 0 \\ x \times s(y) & \to & (x \times y) + x \end{array} \qquad \begin{array}{rcl} s(x) < s(y) & \to & x < y \\ 0 < s(y) & \to & T \\ 0 < 0 & \to & F \\ s(x) < 0 & \to & F \end{array} \tag{5}$$

Though this program does not meet our criteria, it can be massaged into shape; see Section 8.

On the other hand, the following (somewhat peculiar) gound-convergent version (with standard abbreviating conventions) of multiplication does satisfy the requirements we impose for invertibility:

$$\begin{array}{rcl} x \times 0 & \to & 0 \\ 0 \times x & \to & 0 \\ sx \times sy & \to & s(x \times sy + y) \end{array} \qquad \begin{array}{rcl} x + 0 & \to & x \\ x + sy & \to & s(x + y) \end{array} \tag{6}$$

Together with a definition of squaring:

$$x^2 \quad \to \quad x \times x \tag{7}$$

it allows us to compute square-roots by solving goals like $x^2 \to^? s^{100}0$.

In Section 6, we give an algorithm for inversion when certain syntactic conditions, enumerated in Section 5, are fulfilled. The algorithm, the correctness of which is proved in Section 7, is based on the more generally valid goal transformation rules of Section 4. From the theoretical point of view, we are interested in probing the borderline between decidability and undecidability, so the necessity of the conditions is also shown. Prior work

is summarized in Section 3 and future work is suggested in Section 8. First some preliminaries.

2 Nomenclature

We use standard notation and terminology for concepts in rewriting [Dershowitz and Jouannaud, 1990]. In particular, $s \rightarrow^! t$ means that the (first-order) term s rewrites (in zero or more steps using the system under question) to the normal-form (i.e. unrewritable term) t. For our purposes, any function symbol or constant that appears at the root of a left side of a rule is *defined*, while all others are *constructors*. A *constructor* term (or context) is composed of constructors and variables; a ground constructor term (context) is a *value* (context). A rule defining a symbol f will be called an f-rule.

A system is *left-linear* if no variable appears more than once on the left side; it is *linear* if no variable appears on either side more than once. The *depth* $\|t\|$ of a term t is the number of symbols in the longest path of its tree representation. This means that constants and variables have depth 1. A variable is *shallow* in a term if it does not appear below depth 1.

A matching *(sub)goal* takes the form $s \rightarrow^? t$, where s and t share no variables, and has a *solution* σ, assigning terms to variables in s, if and only if $s\sigma \rightarrow^! t\tau$, for some (ancillary) substitution τ of terms for variables in t. Clearly, if t itself is not in normal form, the goal has no solutions. There may of course be more than one solution to a goal. We need not compute them all: We can ignore more specific solutions than ones we do compute (e.g. $x \mapsto sz$ subsumes $x \mapsto sss0$); we can ignore solutions that are not normalized, since they must be equal to normal-form solutions (e.g. $x \mapsto 0 \times z$ is covered by $x \mapsto 0$ for the theory of multiplication).

In inversion problems $s =^? N$, the term N is a (variable-free) value. If the rewrite system R is ground convergent, then an equation $s =^? N$ has a ground solution γ in the equational theory of R if, and only if, γ is equivalent to some solution of the inversion goal $s \rightarrow^? N$. Hence, to find all solutions σ to $s =^? N$, one can look for a complete set of solutions to $s \rightarrow^? N$.

3 Background

Any complete procedure for semantic matching with respect to an arbitrary theory cannot always terminate, even if the theory is presented as a finite,

linear, convergent system for two reasons: matchability for some such theories is undecidable [Heibrunner and Hölldobler, 1987]; some have no finite set of most general unifiers [Fages and Huet, 1983].

Semantic unification in a theory supplied with a finite *ground convergent* (i.e. confluent for ground terms and terminating) rewrite system is known to be computable in the following special cases:

- Every non-ground right side is a variable [Hullot, 1980].
- Every non-ground right side is a constructor term [Dershowitz *et al.*, 1992].
- Every non-ground right side is a proper subterm of its left side [Narendran, Pfenning and Statman, 1997].
- Every non-ground right side is either a constructor term or a proper subterm of its left side [Mitra, 1994].
- Every right side is composed of constructors and proper subterms of its left side [Mitra, 1994].
- All variables are shallow on the left side [Christian, 1992].
- The system is linear and every variable that appears on both sides is shallow on both sides (convergence is unnecessary) [Nieuwenhuis, 1998].
- The system is linear and the right side of every f-rule is either a constructor term or a proper subterm of the left side, except for at most one right side that may be a value context with a single subterm $g(\ldots, r_i, \ldots)$, where every r_i is either a variable or a value [Dershowitz and Mitra, 1992].
- The system is linear and the right side of every f-rule is a constructor term, except for at most one right side that may be a constructor context with a single subterm $g(\ldots, r_i, \ldots)$, where every r_i is either a variable or a value [Mitra, 1994].

Only the last two special cases are powerful enough to capture even simple recursive functions like addition.

Under the assumption of ground convergence, one need only consider innermost computations, in which rules are always applied to terms whose proper subterms are in normal form. For that reason, one need only compute normal-form solutions; they are equivalent in the equational theory

R to all others. For left-linear systems, semantic matching can be simpler than unification, since we know the shape of the normal form of the instantiated pattern. In prior work [Dershowitz *et al.*, 1992], we showed decidability of semantic matching for certain variable-preserving or left-linear ground convergent systems satisfying a (noncomputable) semantic condition, called "decreasingness". Let $[\![t]\!]$ represent the depth (or other measure for which proper subterms are smaller than their superterms) of the (unique) normal form of t for a given system. The point is that:

Theorem 1 *For a ground convergent rewrite system, the normal-form solutions σ to an inversion goal $s \rightarrow^? N$ are bounded in depth, i.e. $\|x\sigma\| \leq \|N\|$, if for all defined symbols f and ground terms $[\![f(\ldots,t,\ldots)]\!] \geq [\![t]\!]$.*

Proof. If $s\sigma \rightarrow^! N$, then $[\![s\sigma]\!] = [\![N]\!]$. Hence, $\|x\sigma\| = [\![x\sigma]\!] \leq [\![s\sigma]\!] = [\![N]\!]$, since $x\sigma$ is a normal form. □

It follows that, for such systems, the (finite) set of (normal-form) solutions to inversion goals can be computed in finite time. The problem is that the condition precludes "erasing" rules that have a variable on the left that is not carried over to the right side. To get around this obstacle, one can distinguish between decreasing and non-decreasing defined symbols:

Theorem 2 ([Dershowitz *et al.*, 1992]) *The inversion problem is computable for a left-linear ground convergent rewrite system if no right side has a defined symbol at its root, nor a defined symbol that appears below a function symbol f that is "decreasing", in the sense that there are ground terms for which $[\![f(\ldots,t,\ldots)]\!] < [\![t]\!]$.*

In System (5), $\times$ and $<$ are decreasing (e.g. $[\![s^4 0 \times 0]\!] = [\![0]\!] = 1 < 5 = [\![s^4 0]\!]$), but one rule has a defined symbol at the top right. Indeed, the goal $0 \times x \rightarrow^? 0$ has infinitely many solutions $s^n 0$, and there is no more general term $s^i y$ for which $0 \times s^i y \rightarrow^! 0$. This is because $0 \times y = 0$ is only an inductive, but not an equational, theorem of (5). For this reason, we need to use convoluted rules for multiplication, as in (6).

4 Complete Inversion

For finite equational theories, satisfiability problems are recursively enumerable, and general-purpose semantic-unification procedures have been extensively studied. If we restrict ourselves to ground convergent rewrite systems that are left-linear, then the following transformation rules constitute a complete procedure for inversion of goals $s \rightarrow^? N$:

Decompose:	$\dfrac{f(s_1,\ldots,s_n) \to^? f(t_1,\ldots,t_n)}{s_1 \to^? t_1, \ldots, s_n \to^? t_n}$	
Mutate:	$\dfrac{f(s_1,\ldots,s_n) \to^? t}{s_1 \to^? l_1\rho, \ldots, s_n \to^? l_n\rho}$	ρ is a solution to $r \to^? t$; $f(l_1,\ldots,l_n) \to r$ is a (renamed) rule in R
Eliminate:	$\dfrac{x \to^? t}{x \equiv^? t}$	x is a variable
Ignore:	$\dfrac{s \to^? x}{}$	x is a variable

When Decompose and Mutate can both be applied to the same subgoal, both alternatives must be explored. The rules are applied until all that remains is a set of syntactic unification goals of the form $x_j \equiv^? t_j$. Any solution to the latter is a solution to the original goal. Variables in the left half of goals are never instantiated, giving a "basic" strategy.

We need to show that this system of goal transformations has the following properties:

Soundness Given a goal $s \to^? t$, if the procedure produces a solution σ, then $s\sigma \to \cdots \to t\tau$, for some substitution τ.

Completeness Given a goal $s \to^? t$, if $s\sigma \to^! t\tau$ for some σ and τ, the procedure will produce a solution μ such that there is a substitution ρ for which $R \models x\mu\rho = x\sigma$, for each variable x appearing in s.

Soundness should be clear except for the Ignore rule, for which we need the following lemmata:

Lemma 3 *For any left-linear rewrite system, if $s' \to^? t'$ is a current subgoal and x is a variable in t', then the set of current subgoals contains only that one occurrence of x.*

Proof. In the right half of the initial goal $s \to^? N$, there are no variables. When Decomposing $f(s_1,\ldots,s_n) \to^? f(t_1,\ldots,t_n)$, any variable that appeared once on the right is now in only one t_i. When Mutating, first $r \to^? t$ is solved, at which point the terms l_i do not yet appear in any subgoal. Later, when solving $s_i \to^? l_i\rho$, any variable y in $l_i\rho$ is either from the rule, in which case it is alone, since R is left-linear, or it was introduced by ρ, in which case it derives from eliminated goals $x_j \equiv^? t_j$ and appeared alone in the t_j. □

Lemma 4 *If a variable x only appears in the right half of a subgoal $s \to^? x$, that subgoal may be deleted (by Ignore).*

Proof. For each solution σ, the ancillary substitution τ includes $x \mapsto s\sigma$ to satisfy the eliminated subgoal. □

Were the system not left-linear, then there would be variables y in the right half of more than one goal. That would require an additional rule to transform a goal $f(s_1, \ldots, s_n) \to^? y$ into $y \equiv^? f(y_1, \ldots, y_n), s_1 \to^? y_1, \ldots, s_n \to^? y_n$, for new variables y_i.

The proof of completeness is by induction on the number of steps in any innermost normalizing derivation $s\sigma \to^! t\tau$, and, secondarily, on the depth of s.

1. If it has zero steps, then $s\sigma = t\tau$ and Decomposition and Eliminate will find a solution at least as general as σ.

2. If t is a variable, then Ignore generates the most general (trivial) solution.

3. If s is a variable x and σ provides a normal form N for it, then by confluence, $N \to^! t$ only if N is t, and Eliminate generates the right solution. If x also has a normal form (from a prior goal) looking like u, then t and u must unify.

4. Otherwise, Decomposition proceeds until a smaller subgoal that requires a rewrite at the top arises. Consider such a derivation

$$f(s_1, \ldots, s_n)\sigma \to \cdots \to f(l_1, \ldots, l_n)\rho = l\rho \overset{\text{top}}{\to} r\rho \to^! t\tau$$

   Mutation does the trick by first finding (something at least as general as) the substitution ρ for $r \to^? t$; the solution σ is then found from the $s_i \to^? l_i\rho$ subgoals.

5 Constructing Systems

In the next section we will prove that the following syntactic requirements suffice for solvability of matching goals:

(A) Each left side is of depth at most 3.

(B) If a right side is a variable or constant, then the left side of that rule is of depth at most 2.

(C) Whenever a right side is not a variable, it is headed by a constructor.

A left-linear, ground convergent system satisfying these three conditions will be called a *constructing system*. Only condition (C) is severe in practice.

Now we show that each of the above three restrictions is necessary: If one drops the requirement of left-linearity, we get undecidability using a rule $E(x,x) \to T$ to reduce an arbitrary unification problem $s =^? t$ to $E(s,t) \to^? T$.

In the remaining cases, we reduce unification in the theory of addition and multiplication over natural numbers (which is undecidable per Hilbert's Tenth Problem) to matching problems, for which we use the constructing rules in (6).

If one violates (C) and allows defined right-root symbols, we have the following counterexample:

$$\begin{array}{rcl} f0 & \to & 0 \\ E(0,0) & \to & 0 \\ E(sx,sy) & \to & fE(x,y) \end{array} \tag{8}$$

By induction it is easy to see that

$$E(x,y) \to^! 0 \text{ if and only if } x, y \to^! s^n 0 \text{ for some } n$$

Therefore, a goal of the form $E(t,u) \to^? 0$ would, in general, be unsolvable for arbitrary terms t and u involving $+$ and $\times$.

The following system illustrates the problem when a left side is of depth 3 and the right side of depth 1:

$$\begin{array}{rcl@{\qquad}rcl} f0 & \to & 0 & E(0,0) & \to & s0 \\ fs0 & \to & 0 & E(sx,sy) & \to & sfE(x,y) \end{array} \tag{9}$$

for which

$$E(x,y) \to^! s0 \text{ if and only if } x, y \to^! s^n 0 \text{ for some } n$$

Finally, we relax Condition (A) and allow depths greater than 2 below the left root. Consider:

$$\begin{array}{rcl@{\qquad}rcl} fsx & \to & s0 & E(0,0) & \to & ss0 \\ G(ss0,ssx) & \to & sssx & E(sx,sy) & \to & sfG(E(x,y),ss0) \end{array} \tag{10}$$

Since $fG(ss0,ss0) \to fsss0 \to s0$,

$$E(x,y) \to^! ss0 \text{ if and only if } x, y \to^! s^n 0 \text{ for some } n$$

and matching is undecidable.

6 Computable Inversion

We present in this section algorithm for function inversion for any theory presented by a constructing system R. For instance, the following is a constructing system for inserting a number in its correct place in an ordered list, and therefore it has a computable inverse:

$$\begin{array}{rcl@{\qquad}rcl}
min(x,0) & \to & 0 & max(x,0) & \to & x \\
min(0,x) & \to & 0 & max(0,x) & \to & x \\
min(sx,sy) & \to & s(min(x,y)) & max(sx,sy) & \to & s(max(x,y)) \\
\multicolumn{3}{r}{insert(x,nil) \to x : nil} & & & \\
\multicolumn{3}{r}{insert(x,y:z) \to min(x,y) : insert(max(x,y),z)} & & &
\end{array} \tag{11}$$

At each stage of the algorithm, we have a set of subgoals. We (don't care) non-deterministically choose one, $s \to^? t$, and consider the following cases.

1. If s and t are identical, the subgoal may be removed.
2. If t is a variable, just remove this subgoal.
3. Suppose s is a variable x:
   (a) If x is already bound to a term u, then bind it instead to the most general unification of u and t.
   (b) If u and t are not unifiable, fail.
   (c) If x is unbound but appears in t, fail.
   (d) If x does not appear in t, add the binding $x \mapsto t$.
4. If neither $s = f(s_1, \ldots, s_n)$ nor $t = g(t_1, \ldots, t_m)$ is a variable, try *both* of the following:
   (a) if $f = g$ (and $m = n$), replace the current goal with the multiset of goals, $s_1 \to^? t_1, \ldots, s_n \to^? t_n$.
   (b) For each rule $f(l_1, \ldots, l_n) \to r$ in R (with all its variables renamed apart from those in the goal), do one of the following:
      i. If r and t are identical, then replace the goal with subgoals $s_1 \to^? l_1, \ldots, s_n \to^? l_n$.

ii. If r is a variable x, then replace the goal with subgoals $s_1 \rightarrow^? l_1\rho, \ldots, s_n \rightarrow^? l_n\rho$, where ρ is $x \mapsto t$.

iii. If r is headed by a constructor that is not g, fail this path.

iv. If g is a constructor, first recursively solve the subgoals $r_1 \rightarrow^? t_1, \ldots, r_m \rightarrow^? t_m$ in succession. For each solution ρ to the variables of these m subgoals, solve the new subgoals $s_1 \rightarrow^? l_1\rho, \ldots, s_n \rightarrow^? l_n\rho$.

7 Correctness

To prove the correctness of the above inversion algorithm, we need to establish its soundness, completeness, and termination. It goes without saying that it can be exponential in cost.

Soundness will be easy, since each step of the algorithm is an application of one or more transformation rules of Section 4. For completeness, we need only check that every transformation that might lead to a solution is attempted by the algorithm.

For termination, we will use the bag (multiset) extension of the lexicographic measure $\langle \|t\|, \|s\| \rangle$ of a subgoal $s \rightarrow^? t$. We will also need the following invariant, which we show by computational induction:

> *For each mapping $x \mapsto u$ of a solution to a goal $s \rightarrow^? t$, we have $\|u\| \leq \|t\|$.*

Indeed, it is because most general solutions are bounded in size that the inversion problem is decidable for constructing systems.

Let the current goal $s \rightarrow^? t$ be called G. Let X signify the set of variables and γ, the current partial solution. We use $\equiv$ for identity of terms.

Consider each step of the algorithm in turn:

1: *If $s \equiv t$, remove G.*

> This is sound, since it is a composite of Decompose and Ignore and results in the trivial solution. There is no need to Mutate for completeness, since s must be normalized if t is.

2: *If $t \in X$, remove G.*

> This is just an application of Ignore and covers all its applicable cases. It, too, results in the trivial solution.

3b,3c: *If $s = x \in X$, but $s\gamma \equiv^? t$ is unsatisfiable, fail.*

In these cases, the syntactic goal $s \equiv^? t$ will fail.

3a,3d: *If $s = x \in X$ and $mgu(s\gamma, t)$ exists, remove G and replace $x \mapsto u$ in γ with $mgu(u, t)$.*

This is Eliminate combined with syntactic unification and covers the only successful case of Elimination.

Since the most general unifier of linear terms with disjoint variables (see Lemma 3) is bounded in depth by the maximum of their depths, all substitutions satisfy the invariant.

4a: *If $s = f(s_1, \ldots, s_n)$ and $t = f(\ldots, t_i, \ldots)$, replace G with subgoals $s_i \rightarrow^? t_i$, $i = 1, \ldots, n$.*

This is just Decompose and covers all cases needed for completeness that are not included in Step 1.

Termination follows from the fact that each subgoal is smaller, since $\|t_i\| < \|t\|$. By induction all solutions are bounded by $\|t\| - 1$.

4(b)i: *If $s = f(s_1, \ldots, s_n)$ and $f(\ldots, l_j, \ldots) \rightarrow t \in R$, replace G with subgoals $s_i \rightarrow^? l_i$, $i = 1, \ldots, n$.*

This is Mutate, when $r \equiv t$, and the subgoal $r \rightarrow^? t$ is trivial, yielding the identity substitution for ρ.

If r is a constant, then, by assumption (B), $\|l_i\| = 1 = \|r\| = \|t\|$ and solutions are bounded by $\|t\|$. Since $\|s_i\| < \|s\|$, the subgoals are smaller. If r is not a constant, then, by assumption (A), $\|l_i\| \leq 2 \leq \|r\| = \|t\|$.

4(b)ii: *If $s = f(s_1, \ldots, s_n)$, $f(\ldots, l_i, \ldots) \rightarrow x \in R$ and $x \in X$, replace G with subgoals $s_i \rightarrow^? l_i\{x \mapsto t\}$, $i = 1, \ldots, n$.*

This is Mutate for "collapsing" rules ($r \in X$).

By assumption (B), l_i is either a constant or a variable (possibly x). Thus, $\|l_i\{x \mapsto t\}\|$ is either 1 or $\|t\|$. In any case, the new subgoals are smaller by virtue of their shallower left half s_i.

4(b)iii: *If $t = g(\ldots, t_j, \ldots)$, but $l \rightarrow c(\ldots, r_k, \ldots) \in R$ has constructor $c \neq g$, this choice path fails.*

Mutation cannot succeed in this case, since $c(\ldots, r_k, \ldots)\sigma \rightarrow \cdots \rightarrow g(\ldots, t_j, \ldots)\tau$ is impossible.

4(b)iv: *If* $s = f(s_1, \ldots, s_n)$, $t = c(t_1, \ldots, t_m)$, $f(\ldots, l_i, \ldots) \rightarrow c(\ldots, r_j, \ldots) \in R$, *and* c *is a constructor, solve the set* $s_i \rightarrow^? l_i\rho$ $(i = 1, \ldots, n)$, *for each solution* ρ *of the* $r_j \rightarrow^? t_j$ $(j = 1, \ldots, m)$.

This is the only remaining case for Mutate, since assumption (C) is that the root of a non-variable right side is a constructor.

The $r_j \rightarrow^? t_j$ have shallower right halves, so the computation of ρ terminates.

By assumption (A), $\|l_i\| \leq 2$ and l_i can contribute at most 1 to the depth $\|l_i\rho\|$. By induction, $\|x\rho\| \leq \|t_j\| \leq \|t\| - 1$, so $\|l_i\rho\| \leq \|t\|$, and the second set of subgoals is smaller by virtue of their left halves. Also, their solution is bounded by $\|t\|$.

8 Extensions

A "symbolic definition" of the form

$$f(x_1, \ldots, x_n) \rightarrow e$$

which just defines a non-recursive function f that does not appear elsewhere (in e or R), like the calendar rule (1) for n or squaring rule $x^2 \rightarrow x \times x$ (7), can always be added to a decidable system, since those symbols can be immediately eliminated from any goal containing them.

Theorem 2 was refined in [Aguzzi and Modigliani, 1994] with a notion of "positional" increase. By using positional information, it is possible to handle certain systems that do not have leading constructors on the right-hand sides of rules. For example, the usual definition of $\times$ can be used instead of the three multiplication rules in (6). With such an extension, we should be able to handle insertion sort by adding $sort(\epsilon) \rightarrow \epsilon$ and $sort(x : y) \rightarrow insert(x, sort(y))$ to (11).

Looking at our proof of the correctness of the algorithm, it should be clear that one can allow left sides of depth $d > 3$ for rules such that the normal form of the right side r has no path of length less than $d + 1$. Checking normal forms of all instances is not a syntactic condition, but testing for ground reducibility is [Plaisted, 1985]. So we can replace Conditions (A) and (B) with the following:

(*) *For each rule $l \to r$, we have $\|l\| < \ell + 2$, where ℓ is the length of the shortest path from the root of r to a ground reducible position.*

Since the ground-normal form of any term is of depth at least 1, this condition guarantees that the subgoal $r \to^? t$ will result in variable bindings that are no deeper than $\|t\| - d$.

Returning to the calendar, the rules for $<$ needed to constrain the length of a month and a year violate even this condition. The calendar code, however, does not require solutions to arbitrary inequalities, only to inequalities of the form $x < N$, for ground term N. These cannot be handled by tabulated functions, with rules like $2 < 12 \to T$ (where 12 is just an abbreviation for $s^{12}0$), since the left side would be too deep. Instead, we can rephrase a constraint $x < N \to^? T$ as $x < N \to^? T^N\top$ and use:

$$\begin{array}{rcl@{\qquad}rcl} sx < sy & \to & T(x < y) & Lsx & \to & TLx \\ 0 < sx & \to & TLx & L0 & \to & \top \end{array} \tag{12}$$

in lieu of the rules for inequality in (5).

A more interesting situation is posed by months of unequal length, as in many archaic calendars having epagomenal days, which we treat as a thirteenth month of only five days (making for a 365-day year). That requires the constraint

$$m < 13 \wedge d < 30 \wedge (m < 12 \vee d < 5)$$

for which we use the goals

$$m < 13 \to^? T^{13}\top,\ d < 30 \to^? T^{30}\top,\ (m < 12 \vee d < 5) \to^? T^{17}\top$$

and compute disjunction as follows:

$$\begin{array}{rcl@{\qquad}rcl} Tx \vee y & \to & T(x \vee y) & \top \vee y & \to & \top \\ y \vee Tx & \to & T(x \vee y) & y \vee \top & \to & \top \end{array} \tag{13}$$

For example,

$$\begin{aligned} & 12 < 12 \vee 4 < 5 \to \cdots \to T^{12}(0 < 0) \vee T^4(0 < 1) \\ & \to T^{12}(0 < 0) \vee T^5 L0 \to T^{12}(0 < 0) \vee T^5\top \\ & \to \cdots \to T^{17}(0 < 0 \vee \top) \to T^{17}\top \end{aligned}$$

For non-left-linear systems, simple inversion goals of the form $f(\ldots, c_i, \ldots) \rightarrow^? N$, where the c_i are constructor terms, can be solved, provided no right side has a defined symbol at its root, nor a defined symbol that appears below a "decreasing" symbol [Mitra, 1994]. Syntactic criteria for this case are also possible.

In [Dershowitz and Mitra, 1992], we considered systems with potentially infinitely many solutions, which were captured as indexed terms, along the lines of [Comon, 1992]. (See Section 3.) Such an approach should work for some inversion problems with unbounded solutions.

The following constructing system for differentiation illustrates some of the subtleties of inversion problems:

$$\begin{array}{rcl} Dt & \rightarrow & s0 \\ D0 & \rightarrow & 0 \\ Dsx & \rightarrow & Dx + 0 \end{array} \qquad \begin{array}{rcl} D(x+y) & \rightarrow & Dx + Dy \\ D(x \times y) & \rightarrow & x \times Dy + y \times Dx \end{array} \tag{14}$$

The third rule has $+0$ to ensure that the right side is headed by the constructor $+$ (vis-a-vis this system, at least, it is a constructor). If we include our constructing rules for $\times$, inverting the goal $Dz \rightarrow^! t + t$ yields the indefinite integral $z \mapsto t \times t$, but, in the absence of simplifying rules for addition, we do not get more general solutions. And one cannot add addition, without turning $+$ into a defined function, at which point (14) would no longer be constructing.

Perhaps results (e.g. [Jacquemard, 1996]) on regularity of the normalizable terms, which have bearing on derivability, can help decide matching.

Finally, our algorithm can serve as the basis of a program to compile a logic program for computing the inverse of a given functional program.

Acknowledgement

We thank Claude Kirchner for his indispensable interest and gracious hospitality. A preliminary version of the inversion algorithm was presented at the Fifth Workshop on Logic, Language, Information and Computation (July 1998, São Paulo, Brazil).

References

[Aguzzi and Modigliani, 1994] Aguzzi, G., Modigliani, U.: A criterion to decide the semantic matching problem. Proc. Intl. Conf. on Logic and Algebra, Italy (1995).

[Christian, 1992] Christian, J.: Some termination criteria for narrowing and E-narrowing. Proc. 11th Intl. Conf. on Automated Deduction, Saratoga Springs, NY. Lecture Notes in Artificial Intelligence **607**:582–588, Springer-Verlag, Berlin (1992).

[Comon, 1992] Comon, H.: On unification of terms with integer exponents. Tech. Rep. 770, Université de Paris-Sud, Laboratoire de Réchérche en Informatique (1992).

[Comon *et al.*, 1991] Comon, H., Haberstrau, M., Jouannaud, J.P.: Decidable problems in shallow equational theories. Tech. Rep. 718, Université de Paris-Sud, Laboratoire de Réchérche en Informatique (1991).

[Dershowitz and Jouannaud, 1990] Dershowitz, N., Jouannaud, J.-P.: Rewrite systems. In J. van Leeuwen, ed., Handbook of Theoretical Computer Science **B**: 243–320, North-Holland, Amsterdam (1990).

[Dershowitz and Mitra, 1992] Dershowitz, N., Mitra, S.: Higher-order and semantic unification. Proc. 13th Conf. on Foundations of Software Technology and Theoretical Computer Science, Bombay, India. Lecture Notes in Artificial Intelligence **761**:139–150, Springer-Verlag, Berlin (1993).

[Dershowitz *et al.*, 1992] Dershowitz, N., Mitra, S., Sivakumar., G.: Decidable matching for convergent systems (Preliminary version). Proc. 11th Conference on Automated Deduction, Saratoga Springs, NY. Lecture Notes in Artificial Intelligence **607**:589–602, Springer-Verlag, Berlin (1992).

[Dershowitz and Plaisted, 1988] Dershowitz, N., Plaisted, D. A.: Equational programming. In: J. E. Hayes, D. Michie, J. Richards, eds., Machine Intelligence 11: The logic and acquisition of knowledge, 21–56. Oxford Press, Oxford (1988).

[Dershowitz and Reingold, 1997] Dershowitz, N., Reingold, E. M.: Calendrical Calculations. Cambridge University Press, Cambridge (1997).

[Fages and Huet, 1983] Fages, F., Huet, G.: Unification and matching in equational theories. Proc. 8th Colloq. on Trees in Algebra and Programming, L'Aquila, Italy. Lecture Notes in Computer Science **159**:205–220, Springer-Verlag, Berlin (1983).

[Heibrunner and Hölldobler, 1987] Heilbrunner, S., Hölldobler, S.: The undecidability of the unification and matching problem for canonical theories. Acta Informatica **24**(2):157–171 (1987).

[Hullot, 1980] Hullot, J.-M.: Canonical forms and unification. Proc. 5th Intl. Conf. on Automated Deduction, Les Arcs, France, Lecture Notes in Computer Science **87**:318–334, Springer-Verlag, Berlin (1980).

[Jacquemard, 1996] Jacquemard, F.: Decidable approximations of term rewriting systems. Proc. 7th Intl. Conf. on Rewriting Techniques and Applications, New Brunswick, NJ. Lecture Notes in Computer Science **1103**:362–376, Springer-Verlag, Berlin (1966).

[Mitra, 1994] Mitra, S.: Semantic unification for convergent systems. Ph.D. thesis, Dept. of Computer Science, University of Illinois, Urbana, IL, Tech. Rep. UIUCDCS-R-94-1855 (1994).

[Narendran, Pfenning and Statman, 1997] Narendran, P., Pfenning, F., Statman, R.: On the unification problem for Cartesian closed categories. J. Symbolic Logic **62**(2):636–647 (1997).

[Nieuwenhuis, 1998] Nieuwenhuis, R.: Decidability and complexity analysis by basic paramodulation. Information and Computation **147**:1–21 (1998).

[Plaisted, 1985] Plaisted, D. A.: Semantic confluence tests and completion methods. Information and Computation **65**:182–215 (1985).