Higher-Order and Semantic Unification*

Nachum Dershowitz, Subrata Mitra

Department of Computer Science
University of Illinois
1304 West Springfield Avenue
Urbana, IL 61801, U.S.A.
{nachum, mitra}@cs.uiuc.edu

Abstract. We provide a complete system of transformation rules for semantic unification with respect to theories defined by convergent rewrite systems. We show that this standard unification procedure, with slight modifications, can be used to solve the satisfiability problem in combinatory logic with a convergent set of algebraic axioms R, thus resulting in a complete higher-order unification procedure for R. Furthermore, we use the system of transformation rules to provide a syntactic characterization for R which results in decidability of semantic unification.

1 Introduction

Equation solving is the process of finding a substitution that makes two terms equal in a given theory, while *semantic unification* is the process which generates a basis set of such unifying substitutions. For example, considering the function definitions

$$even(x+x) = true, even((x+x)+1) = false,$$

$$length([\,]) = 0, length([x\,|\,y]) = length(y)+1,$$

if we could unify with respect to + and *length*, then we could solve queries of the form $even(length(x)) \overset{?}{=} true$ in logic-programming languages. In this paper, we provide a complete system of transformation rules for semantic unification with respect to theories defined by convergent (confluent and terminating) rewrite systems.

There have been different proposals to combine higher-order features with first order equational reasoning (including [Bre88, Dou91], and others). Most of these deal with the combination of lambda-calculus and a first-order equational theory. Recently, Dougherty and Johann ([Dou93, DJ92]) proposed a method for higher-order reasoning by transforming lambda-calculus terms to combinatory logic, that is, they use a combination of combinatory-logic with an equational theory as the formulation of higher-order reasoning. In [DJ92], they also provide a complete set of transformations for solving the satisfiability problem in such a

* This research was supported in part by the U. S. National Science Foundation under Grants CCR-90-07195 and CCR-90-24271.

combined system. Some of the main advantages (pointed out in [DJ92]) of using combinatory logic as the basis of higher-order unification (as opposed to the traditional lambda-calculus based methods such as [Sny90, NQ91], etc.) are: it eliminates some technical problems associated with bound variables, allows easy incorporation of type-variables, and facilitates the use of substitution like in the first-order case. We extend the first-order unification procedure to develop a complete method for solving the satisfiability problem in combinatory logic with a set of algebraic axioms R which can be presented as a convergent rewrite system.

Our approach enjoys the following advantages:

- It provides more control on positions where rules get applied. In general, we apply rules only to the top-most position in goals.
- It is possible to incorporate additional pruning rules (for example, reachability analysis [DS87, CR91]) directly.

Finally, we use the system of transformation rules to derive syntactic restrictions on rewrite systems that result in decidable semantic unification problems.

In general, we use standard terminology and notation for rewrite systems. Most of the notations that we use for equational theories are consistent with [DJ90], while we borrow notation from [DJ92] for the higher-order aspects.

Types are formed by closing a set of *base types* (for example, integer and boolean) under the type forming operation $\alpha_1 \rightarrow \alpha_2$ (for types α_1 and α_2). We assume the constants $\mathcal{I}, \mathcal{K}$ and $\mathcal{S}$ (called *redex atoms*), given types as usual. A term is *linear* if every variable occurs exactly once.

For $\mathcal{CL}$ (the simply typed combinatory-logic terms), the convergent rewrite system $\{\mathcal{I} \rightarrow x, (\mathcal{K}x)y \rightarrow x, ((\mathcal{S}x)y)z \rightarrow (xz)(yz)\}$ (henceforth denoted C) defines *weak reduction.* Note that this rewrite system is terminating only for typed combinatory logic. We define *combinatory-R reduction* as $\rightarrow_C \cup \rightarrow_R$. Whenever any such relation is terminating (convergent) we can talk of the (unique) normal forms it assigns to terms.

It is well-known that using combinatory reductions is not enough to capture equivalence of lambda terms. For example, $\mathcal{SK}$ and $\mathcal{KI}$ are distinct normal forms with respect to $\rightarrow_C$, though their translations to lambda-calculus are both equal to $\lambda y \lambda z.z$. However, it is possible to extend combinatory-R equality to capture equivalence of functional terms, by using the following rule of *extensionality*:

$$\text{Infer } s = t \text{ if } sD = tD, \text{ where } D \text{ is a new constant.}$$

A term is said to be *pure* if it does not contain any constant introduced by the extensionality axiom. We use the notation $s =_{RC} t$ (or say that s and t are RC-equal) to denote the equality of the lambda-calculus translations of s and t with respect to $\beta\eta R$ convertibility (which, by virtue of the above discussion, is identical to the equality induced by $C \cup R$ with extensionality).

In formulating rules for validity and higher-order unification, we deal with unordered-pairs of terms. A pair $s \overset{?}{=} t$ is *trivial* if $s \equiv t$, and is *RC-valid* if $s =_{RC} t$. A term-pair $s \overset{?}{=} t$ has a *solution* σ if $s\sigma =_{RC} t\sigma$. These notions can be extended to collections of pairs in the usual way; for example, we say that a collection is valid iff each of its pairs is valid.

2 Semantic Unification

In this section we provide a complete system of transformations for semantic unification with respect to a convergent rewrite system. The main idea behind using convergent rewrite systems for semantic unification is that if θ is a solution to $s \overset{?}{=} t$, then there must be a common normal form w such that $s\theta \to^! w$ and $t\theta \to^! w$. An equational goal $s \overset{?}{=} t$ can therefore be converted into two *directed* goals $s \to^? x$ and $t \to^? x$, where x is a variable not in t or s. Furthermore, since the rewrite system is convergent, it is sufficient to use any one complete rewriting strategy. Therefore, we will henceforth be interested in enumerating those solutions θ which correspond to innermost reductions in the derivation $s\theta \to^! w$ and $t\theta \to^! w$. We do not actually demand all solutions for completeness; rather, if for every variable x, $x\sigma$ and $x\tau$ are equal (with respect to R) then (at least) one of σ or τ is deemed redundant. We can use the transformation rules of Table 1 to solve the semantic unification problem with a convergent R.

Some explanations are in order:

- Each transformation rule consists of an antecedent (the first line), a consequent (the third line) and, optionally, a condition (the fourth line, whenever present). Whenever a subgoal matches the pattern of the antecedent of a transformation rule, we can replace it with the corresponding consequent, provided the condition holds.
- The transformation rules given in Table 1 are non-deterministic, that it, for completeness all possibilities have to be tried. Thus, for any initial goal, we generate a tree (which we will call the *solution tree*) of such possibilities.
- We use expressions of the form $x \mapsto t$, where x is a variable, to keep track of partial solutions. An "unbound variable" is one that does not occur in the domain of the partial solution generated so far.
- For convenience, we do not apply resulting substitutions back into left-hand sides of goals. This gives basic-narrowing [Hul80, Ret87] like capabilities without having to keep explicit markers for basic positions. However, we now need additional transformation rules to handle different possible right-hand sides of goals.

The transformations of Table 1 suffice:

Theorem 1 (Completeness). *Let R be a convergent rewrite system, and $G \equiv \{s \to^? x, t \to^? x\}$ be a set of goals which admit a solution θ. Then, there is a sequence of transformations of the form $G \leadsto^! \mu$, such that the generated substitution μ is at least as general as θ $(\mu \leq \theta)$.*

Eliminate	$\{x \rightarrow^? t\} \rightsquigarrow \{x \mapsto t\}$ where x is an unbound variable that does not occur in t
Bind	$\{x \rightarrow^? s, x \mapsto t\} \rightsquigarrow \{x \mapsto s\} \cup mgu(s,t)$
Mutate	$\{f(s_1,\ldots,s_n) \rightarrow^? t\} \rightsquigarrow \{s_1 \rightarrow^? l_1,\ldots,s_n \rightarrow^? l_n, r \rightarrow^? t\}$ where $f(l_1,\ldots,l_n) \rightarrow r$ is a renamed rule in R
Decompose	$\{f(s_1,\ldots,s_n) \rightarrow^? f(t_1,\ldots,t_n)\} \rightsquigarrow \{s_1 \rightarrow^? t_1,\ldots,s_n \rightarrow^? t_n\}$
Imitate	$\{f(s_1,\ldots,s_n) \rightarrow^? x\} \rightsquigarrow \{s_1 \rightarrow^? x_1,\ldots,s_n \rightarrow^? x_n, x \mapsto f(x_1,\ldots,x_n)\}$ where x is an unbound variable, and $x_1,\ldots,x_n$ are new variables
Apply	$\{s \rightarrow^? t\} \cup \sigma \rightsquigarrow \{s \rightarrow^? t\sigma\} \cup \sigma$

Table 1. Transformation Rules for Semantic Unification

3 Higher-Order Unification

In this section we formulate a RC-unification procedure based on the transformation rules given in Table 1. Given two combinatory-logic terms s and t, we want to find a complete set of their RC-unifiers; each RC-unifier being a solution σ as indicated before, that is, we want to enumerate all substitutions σ such that $s\sigma =_{RC} t\sigma$ (with the understanding that whenever two substitutions are RC-equal, at least one of them is redundant). Since $=_{RC}$ has no known presentation as a convergent rewrite system (thus, transformations from Section 2 cannot be used verbatim) we use the idea of extending the convergent relation $C \cup R$ with extensionality, as discussed in Section 1. Roughly, to decide if $s\sigma =_{RC} t\sigma$, we reduce $s\sigma$ and $t\sigma$ (using innermost reductions) with respect to $C \cup R$ as far as possible, and then invoke extensionality; repeating this process until all term-pairs are trivial.

Our top-down method for combining combinatory-logic with a first-order rewrite system therefore proceeds as follows:

- Given a goal $s \overset{?}{\underset{RC}{=}} t$, break it into a set of two directed goals of the form: $\{s \rightarrow^? x, t \rightarrow^? y\}$, and solve for x and y, which are new variables. The interpretation of a goal of the form $u \rightarrow^? v$ is: Find a solution σ such that $u\sigma \rightarrow^!_{C \cup R} v\sigma$.
- If the terms bound to x and y are syntactically unifiable with most general unifier θ, return one solution as the composition of the partial substitution collected so far with θ (details given below).
- If unification fails, s and t may still be RC-equivalent, by virtue of extensionality. We simply apply the extensionality axiom as required, and continue goal-solving with respect to $C \cup R$.

We now provide details of the different aspects. For the first part, we simply use the system of transformations given in Table 1, with the understanding that the only non-zeroary function (for example, f in Decompose, Mutate, etc., in Table 1) is "function application." In the rest of our discussion on RC-unification, we will refer to this system of transformation rules as WR, for weak-R transformations. When using this system of transformations for higher-order unification we do not deal explicitly with types. Nevertheless, every transformation rule must be type preserving. We also insist that only pure terms are bound to variables (in Eliminate and Bind). Furthermore, we would restrict our attention to left-linear rewrite systems alone. The more general case can be handled using ideas developed in [DJ92].

The transformation rules in Table 1 form the crux of the higher-order semantic unification procedure, since they provide a strategy of unification with respect to weak-combinatory equality. However, weak combinatory equality is not identical to $=_{RC}$. We need to extend the rules for weak-equality to handle extensionality. Towards this end we provide a new set of transformation rules (called EXT) shown in Table 2. The rules are non-deterministic (for example, both Bind and Extend would apply to the subgoal $x \overset{?}{=} t$), and, as usual, all possibilities have to be tried for completeness.

We are now ready to specify the complete goal-solving procedure:

Definition 1. (RCU) To solve $s \overset{?}{=} t$ for $\mathcal{CL}$ terms s and t, we proceed as follows:

1. Use WR to obtain a partial solution $\{x \mapsto u, y \mapsto v\} \cup \sigma$ to the goal $\{s \rightarrow^? x, t \rightarrow^? y\}$.
2. Use EXT to reduce $u\sigma \overset{?}{=} v\sigma; \phi$ to subgoals $\theta; \{l_i \overset{?}{=} r_i \mid i = 1, \ldots, m\}$.
3. Recursively, find solutions τ_i to $l_i \overset{?}{=} r_i, i = 1, \ldots, m$.
4. Return $\{x \mapsto u, y \mapsto v\} \cup \sigma \cup \theta \cup \tau_1 \cup \ldots \cup \tau_m$, provided it is a substitution (that is, it passes the "occur check").

Example 1. Let R be the rule $F0x \rightarrow x$, where 0 and x are, respectively, a constant and a variable of type integer.

We look for solutions to the goal $z \overset{?}{=} F0$. Since both z (a variable) and $F0$ are in $C \cup R$ normal form, we have $u\sigma \equiv z$ and $v\sigma \equiv F0$ in step 2 of RCU.

Bind	$\{x \stackrel{?}{=} t\} \cup E; G$ $\rightsquigarrow$ $\{x \mapsto t\} \cup E\{x \mapsto t\}; G\{x \mapsto t\}$ if x does not occur in t
Decompose	$\{Fs_1 \ldots s_n \stackrel{?}{=} Ft_1 \ldots t_n\} \cup E; G$ $\rightsquigarrow$ $\{s_1 \stackrel{?}{=} t_1, \ldots, s_n \stackrel{?}{=} s_n\} \cup E; G$ where F is an atom
Extend	$\{s \stackrel{?}{=} t\} \cup E; G$ $\rightsquigarrow$ $E; G \cup \{sD \stackrel{?}{=} tD\}$ where D is a new constant, provided sD and tD are type-correct

Table 2. Transformation Rules for Extension

Therefore, one solution to the goal is $\{z \mapsto F0\}$. However, there are other solutions, since we can add an argument to the two sides, and thereafter attempt to solve $zD \stackrel{?}{=} F0D$, that is, $zD \stackrel{?}{=} D$ (here D is a new constant of type integer). Some possible derivations with this goal are:

$$zD \stackrel{?}{=} D \rightsquigarrow_{\mathbf{Mutate}_{\mathcal{I}}} z \mapsto \mathcal{I}$$

$$\begin{array}{rll} zD \stackrel{?}{=} D & \rightsquigarrow_{\mathbf{Mutate}_{\mathcal{K}}} & z_1 \stackrel{?}{=} D, z \mapsto \mathcal{K}z_1 \\ & \rightsquigarrow & \mathbf{Fail} \end{array}$$

$$\begin{array}{rll} zD \stackrel{?}{=} D & \rightsquigarrow_{\mathbf{Mutate}_{\mathcal{S}}} & z_1D(z_2D) \stackrel{?}{=} D, z \mapsto \mathcal{S}z_1z_2 \\ z_1D(z_2D) \stackrel{?}{=} D & \rightsquigarrow_{\mathbf{Mutate}_{\mathcal{K}}} & z_1 \mapsto \mathcal{K} \\ z_1D(z_2D) \stackrel{?}{=} D & \rightsquigarrow_{\mathbf{Mutate}_{\mathcal{S}}} & z_3(z_2D)(D(z_2D)) \stackrel{?}{=} D, z_1 \mapsto \mathcal{S}z_3 \\ & \rightsquigarrow & \mathbf{Fail} \end{array}$$

In the figure we have shown three different ways of mutating the goal $zD \stackrel{?}{=} D$, corresponding to the three rules in C. The first possibility yields a solution immediately ($\{z \mapsto \mathcal{I}\}$). In the second case, we get a subgoal of the form $z_1 \stackrel{?}{=} D$, which has no solution (we cannot use elimination, since it would not give a pure substitution, and we cannot add further arguments, since D is of type integer). In the last case, we have further branches, only one of which yield a solutions, namely $\{z \mapsto \mathcal{SK}z_2\}$, which, in fact, is the same as the previous solution, since $\mathcal{SK}x =_{RC} \mathcal{I}$ for any variable x. We get failure in the last branch, since we have a subterm of the form $D(z_2D)$ which is not type correct, D being of integer type. Finally, we could mutate zD using the rule from R; this would repeat the solution $\{z \mapsto F0\}$. Therefore, we have exhausted all possibilities, and have a finite solution tree in this case.

Our main result is completeness of the transformation rules for finding solutions to higher-order equations:

Theorem 2 (Completeness). *Let R be a left-linear convergent rewrite system. If θ is a RC-unifier for two terms s and t, then there exists a RCU-sequence that enumerates a solution σ that is at least as general as θ.*

Note that any RCU-sequence in general would consist of alternate applications of WR and EXT. Therefore, the completeness proof is a combination of the following lemmata:

Lemma 3. *Let R be a left-linear convergent rewrite system. Suppose θ is a solution to the goal $s \rightarrow^{?} t$ (i.e., $s\theta \rightarrow^{!}_{C \cup R} t\theta$), then there is a sequence of WR steps starting with the goal $s \rightarrow^{?} t$ that generates a substitution σ which is at least as general as θ.*

Lemma 4. *Let R be a convergent left-linear rewrite system. If s and t are terms in $C \cup R$ normal form, such that θ is a solution to $s \overset{?}{=} t$, then there exists an EXT derivation of the form $s \overset{?}{=} t; \phi \rightsquigarrow^{!} \sigma; G$, such that $\sigma \cup \tau$ is at least as general as θ, where τ is a solution to the set of goals G.*

Any innermost reduction steps can be simulated using WR (with respect to $C \cup R$). Furthermore, we explicitly add arguments (Extend) and use decomposition to expose inner positions where arguments need to be added.

4 Decidable Semantic Unification

It is well-known that semantic unification is undecidable, even when the theory under consideration admits a convergent presentation. In this section, we describe a syntactic restriction on rewrite systems which results in decidable first-order semantic unification. In the following we say that a function f is a *defined function* if it appears at the root of a left-hand side of R; any other function symbol is a *constructor*.

Definition 2 (Flat Term). A term is said to be *flat* if every path from a variable to its root contains at most one defined function, and the variable appears immediately below the defined function.

Definition 3 (Singular). A flat term is *singular* if all its variables appear under a single defined function.

For example, if f is a defined function and s, a and 0 are constructors, then $s(f(x, y))$ and $s(f(f(0,0), x))$ are singular, while $f(s(x), y)$ (x does not appear directly below f) is not. Finally, $a(f(x,y), f(z,0))$ is flat but not singular, since the variables appear under two different defined functions.

Theorem 5. *Let R be a (left- and right-) linear convergent rewrite system. If for every function f defined by R there is at most one rule with a right-hand side that is neither a constructor term nor a subterm of the corresponding left-hand side and that is singular, then the semantic unification problem is decidable for R.*

Proof. The main idea is to show that for any goal of the form $r \rightarrow^? t$, where r is a variant of some right-hand side of R, a complete set of solutions can be expressed as a linear recurrence.

Consider the goal $r \rightarrow^? t$. The interesting case is when we have to apply the transformation rule for mutation. (Without mutation and with r singular, we get a finite branch of the solution tree, which is not a problem.) Furthermore, mutation using rules for which the right-hand sides are either constructor terms or subterms of the corresponding left-hand sides leads to finite branches (see [DMS92] for details). If r has any leading constructors, then we can use Decompose and Imitate as required, and end up with a goal of the form $f(r_1, \ldots, r_n) \rightarrow^? t'$, where f is a defined function. Consider mutation of this goal using the rule $f(l_1, \ldots, l_n) \rightarrow r'$. By the assumptions of the theorem, each r_i is either a variable or a ground term. Therefore, all the subgoals of the form $r_i \rightarrow^? l_i$ can be solved without mutation, which leaves a single subgoal of the form $r' \rightarrow^? t'$. Furthermore, because of flatness and linearity, no variable in r' could have been bound when solving the $r_i \rightarrow^? l_i$ subgoals. Also, along any path in the solution tree, the right-hand sides of goals cannot get any more complicated (mutation keeps the right-hand side intact; decomposition reduces the structure).

Therefore, if there is an infinite path in the solution tree it must contain a repetition of the form:

$$g_1 \equiv \{r_1 \rightarrow^? t_1\} \rightsquigarrow_{\mathbf{Mutate}^*} \{r_2 \rightarrow^? t_2\} \equiv g_2,$$

where r_2 and t_2 are renamed versions of r_1 and t_1, respectively. In this situation we can express finitely a complete set of solutions to g_1 without having to explore g_2 further. Let σ denote the substitution generated in the path from g_1 to g_2, and σ' be any solution along some other finite path to g_1. (Since there is a single singular rule, there can be no other infinite paths.) Then, corresponding to σ', we get a complete set of solutions, which can be expressed as $\sigma^n \circ \sigma', n \geq 0$ (refer to the example below for details).

Therefore, we can express all solutions to the goal under consideration as a set of linear equations. Finally, we can reduce any general goal $s \overset{?}{=} t$ to finitely many sets of goals of the form $\{r_i \rightarrow^? t_i\}$, where each r_i is a variant of a right-hand side of R; thus, each such subgoal has a recurrence pattern (as discussed above) as its solution. Since terms are expressible using a regular language and a system of linear equations, there is a unification algorithm (using intersections of regular languages) for such terms, which can be used in case of repeated variables in s and t.

Example 2. Consider the rewrite system for addition:

$$0 + x \rightarrow x \qquad (1)$$

$$s(x) + y \to s(x + y) \tag{2}$$

The goal

$$y + (y + z) \overset{?}{=} z + z$$

is transformed into two directed goals, $\{y + (y + z) \to^? x', z + z \to^? x'\}$. Some of the derivation steps are:

$$\left\{ \begin{array}{ll} y + (y + z) & \to^? x' \\ z + z & \to^? x' \end{array} \right\} \rightsquigarrow_{\mathbf{Mutate(2)}} \left\{ \begin{array}{c} y \to^? s(x_1), y + z \to^? y_1, \\ s(x_1 + y_1) \to^? x', z + z \to^? x' \end{array} \right\}$$

We can solve the goal $z + z \to^? x'$ by changing it to $z_1 + z_2 \to^? x', z_1 = z_2$. (This is how we can handle non-linear variables in the original goal in general.) The steps involved in solving the goal $z_1 + z_2 \to^? x'$ are:

$$\begin{array}{lll} \{z_1 + z_2 \to^? x'\} & \rightsquigarrow_{\mathbf{Mutate(1)}} & \{z_1 \to^? 0, z_2 \to^? z_0, z_0 \to^? x'\} \\ & \rightsquigarrow & \{z_1 \mapsto 0, z_2 \mapsto x'\} \\ \{z_1 + z_2 \to^? x'\} & \rightsquigarrow_{\mathbf{Mutate(2)}} & \{z_1 \to^? s(x_2), z_2 \to^? y_2, s(x_2 + y_2) \to^? x'\} \\ & \rightsquigarrow & \{x_2 + y_2 \to^? x'', \theta\} \end{array}$$

where $\theta \equiv \{z_1 \mapsto s(x_2), z_2 \mapsto y_2, x' \mapsto s(x'')\}$. We now have a subsuming pattern of the form $\{z_1 + z_2 \to^? x'\} \rightsquigarrow^* \{x_2 + y_2 \to^? x''\}$, producing the solution $z_1 = s^n(0), z_2 = z_0, x' = s^n(z_0)$, which, together with the constraint $z_1 = z_2$, simplifies to $z = s^n(0), x' = s^{2n}(0)$. Similarly, we can simplify the remaining goals to get the following: $x_1 = s^m(0), y_1 = y_0, x' = s^{m+1}(y_0)$ (as a general solution to the goal $s(x_1 + y_1) \to^? x'$), $y = s^j(0), z = z_a, y_1 = s^j(z_a)$ (from the goal $y + z \to^? y_1$), and $y = s(x_1)$. We solve these equations (with respect to n, m and j) to get $y = s^j(0), z = s^n(0)$, where $x' = s^{2j+n}(0) = s^{2n}(0)$, which gives $2j = n$ after simplification.

In the above example, we handled non-linear variables in the original goal by solving additional constraints, which was simple because we had a unary function (s) to deal with. We now briefly indicate how solutions can be represented and such constraints solved in general. According to the requirements of Theorem 5, if x is a variable in the original goal, then along the infinite branch of the solution tree, we have the following:

$$x \mapsto u[x_1], x_1 \mapsto v[x_2], x_2 \mapsto v[x_3], \ldots,$$

where v is the context which gets repeated due to the subsuming pattern. Thus, in effect, the variable x gets bound to a term-pattern, which we represent as $x \mapsto u[v^n[x_{n+1}]]$. Therefore, in solving a constraint of the form $x = y$, where both x and y could be bound to such term patterns, we have to "unify" these patterns. We will consider the case where the term pattern contains a single infinite branch, since multiple branches would be at independent positions, and therefore could be solved independently. This path itself can be represented as a simple regular expression containing function symbols. (For example, for the binding of x as shown above, such a path is of the form $\{l_1\}\{l_2\}^*\{l_3\}$, where

l_1 is the path in the context u, l_2 is in v and l_3 is in the substitution used for x_{n+1} along any finite path for solving x.) Therefore, in order to solve the equality constraint $x = y$ (or, to unify such patterns), we find the intersection of the regular expressions which represent these two paths. It can be shown that this intersection is a similar regular expression, which, if infinite, represents the infinite path in the unified term. Once we get this regular path, we have to unify the finite branches along it, which is not difficult.

5 Discussion

Higher-order semantic unification is of interest in automated theorem proving, type inferencing, higher-order extensions of logic-programming, etc. Our higher-order unification procedure performs better than one based on narrowing. In particular, we provide an example showing how infinitely many branches of the solution tree can be eliminated.

Example 3. Let R be the convergent rewrite system:

$$\begin{aligned} A(Sx) &\rightarrow AZ \\ B(Sx) &\rightarrow BZ \end{aligned}$$

Here A, B, Z and S are constants, while x is a variable. We also assume that the C rules are available as usual.

Note that a term with A at its head can never be RC-unified with a term with B as its head, since there are no rules to change one to the other. Now consider the goal: $A(xZ) \overset{?}{=} BZ$. Using the approach of [DJ92], the sub-term xZ could be narrowed indefinitely:

$$\begin{aligned} xZ &\rightsquigarrow x_1Z(y_1Z), x \mapsto (Sx_1)y_1 \\ &\rightsquigarrow x_2Z(y_2Z), x_1 \mapsto (Sx_2)y_2. \end{aligned}$$

However, using the approach outlined in this paper, we first transform the goal to get $\{A(xZ) \rightarrow^? x', x'' \mapsto (BZ)\}$, since (BZ) is in ground normal form. We therefore have the following possible derivation sequences:

$$\begin{aligned} \{A(xZ) \rightarrow^? x'\} &\rightsquigarrow_{\textbf{Imitate}} \{xZ \rightarrow^? x_1, x' \mapsto Ax_1\} \\ &\rightsquigarrow \textbf{Fail} \end{aligned}$$

$$\begin{aligned} \{A(xZ) \rightarrow^? x'\} &\rightsquigarrow_{\textbf{Mutate}} \{xZ \rightarrow^? Sx_1, AZ \rightarrow^? x'\} \\ &\rightsquigarrow \{xZ \rightarrow^? Sx_1, x' \mapsto AZ\} \\ &\rightsquigarrow \textbf{Fail} \end{aligned}$$

Whenever we bind x' to any term which has A as the head, we can prune the corresponding branch (because of the observation about R made before); thus, the initial goal is unsatisfiable.

This example shows that for RC-unification, the top-down approach works better than one based on narrowing. Similar pruning capabilities of the top-down approach for the first-order case alone has been mentioned in [DS87, DMS90], and they carry over to higher-order solving also. More elaborate pruning mechanisms have been studied for the first order case by [CR91], where a graph of terms based on R and the goals under question has been used. Their approach is top-down, so we believe that it is possible to combine it with the higher-order capabilities developed here.

Finally, as far as the decidability results are concerned, it may be possible to handle non-linearity and multiple singular right-hand sides by generalizing the definition of subsuming goals.

References

[Bre88] V. Breazu-Tannen. Combining Algebra and Higher-order Types. In *Proc. of the Third Annual IEEE Symposium on Logic in Computer Science*, pages 82–90, 1988.

[CR91] J. Chabin and P. Réty. Narrowing Directed by a Graph of Terms. In *Proc. of the Fourth Int. Conference on Rewriting Techniques and Applications*, Como, Italy, April, 1991. Vol. 488, pages 112–123, of *Lecture Notes in Computer Science*, Springer Verlag, 1990.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320, North-Holland, Amsterdam, 1990.

[DMS90] Nachum Dershowitz, Subrata Mitra and G. Sivakumar. Equation Solving in Conditional AC-Theories. In *Proceedings of the Second International Conference on Algebraic and Logic Programming*, Nancy, France, October 1990. Vol. 463, pages 283–297, of *Lecture Notes in Computer Science*, Springer Verlag (1990).

[DMS92] Nachum Dershowitz, Subrata Mitra, and G. Sivakumar. Decidable matching for convergent systems. In *Proceedings of the Eleventh Conference on Automated Deduction*, pages 589–602, Saratoga Springs, NY, June 1992. Vol. 607 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, Berlin.

[DS87] Nachum Dershowitz and G. Sivakumar. Solving Goals in Equational Languages. In *Proceedings of the First International Workshop Conditional Term Rewriting System*, Orsay, France, July 1987. Vol. 308, pages 45–55, of *Lecture Notes in Computer Science*, Springer Verlag (1987).

[Dou91] D. J. Dougherty. Adding Algebra to the Untyped Lambda Calculus. In *Proc. of the Fourth Int. Conference on Rewriting Techniques and Applications*, Como, Italy, April, 1991. Vol. 488, pages 37–48, of *Lecture Notes in Computer Science*, Springer Verlag, 1990.

[Dou93] D. J. Dougherty. Higher-order unification via combinators. Theoretical Computer Science 114, pages 273–298.

[DJ92] Daniel J. Dougherty and Patricia Johann. A Combinatory Logic Approach to Higher-order E-unification. In *Proc. of the Eleventh Conference on Automated Deduction*, Saratoga Springs, New York, June, 1992. Vol. 607, pages 79–93, of *Lecture Notes in Computer Science*, Springer Verlag, 1992.

[Hul80] Jean-Marie Hullot. Canonical forms and unification. In R. Kowalski, editor, *Proc. of the Fifth International Conference on Automated Deduction*, pages 318–334, Les Arcs, France, July 1980. Vol. 87 of *Lecture Notes in Computer Science*, Springer, Berlin.

[NQ91] T. Nipkow and Z. Qian. Modular Higher-order E-Unification. In *Proc. of the Fourth Int. Conference on Rewriting Techniques and Applications*, Como, Italy, April, 1991. Vol. 488, pages 200–214, of *Lecture Notes in Computer Science*, Springer Verlag, 1990.

[Ret87] Pierre Réty. Improving Basic Narrowing Techniques. In P. Lescanne, editor, *Proc. of the Second International Conference on Rewriting Techniques and Applications*, pages 228–241, Bordeaux, France, May 1987. Vol. 256 of *Lecture Notes in Computer Science*, Springer, Berlin.

[Sny90] Wayne Snyder. Higher Order E-Unification. In *Proceedings of the Tenth Int. Conference on Automated Deduction*, Kaiserslautern, FRG, July 1990. Vol. 449, pages 573–587, of *Lecture Notes in Computer Science*, Springer Verlag, 1990.