Goal-Directed Equation Solving

Nachum Dershowitz and G. Sivakumar
Department of Computer Science
University of Illinois at Urbana-Champaign
1304, W. Springfield Ave.
Urbana, Illinois 61801, U.S.A.

Abstract

Solving equations in equational Horn-clause theories is a programming paradigm that combines logic programming and functional programming in a clean manner. Languages like EQLOG, SLOG and RITE, express programs as rewrite rules and use narrowing to solve goals expressed as equations. In this paper, we express equational goal solving by means of a logic program that simulates rewriting of terms. Our goal-directed equation solving procedure is based on "directed goals", and combines narrowing with a more top-down approach. We also show how to incorporate a notion of operator derivability to prune some useless paths, while maintaining completeness of the method.

1. Equational Programming

Several proposed programming languages use (conditional) equations as a means of combining the main features of logic programming and functional programming; such languages include RITE [Dershowitz and Plaisted, 1985], SLOG [Fribourg, 1985], and EQLOG [Goguen and Meseguer, 1988]. Computing consists of finding values (substitutions) for the variables in a goal \( \varphi = t \) for which the equality holds. Efficient methods of solving equations are therefore very important, as is the ability to detect when an equation is unsatisfiable. In this paper, we concentrate on programs composed of unconditional rules though the ideas extend to conditional rules, as employed in the above languages. Solving equational goals and detecting unsatisfiability are also important for theorem-proving procedures (e.g. [Kaplan–88]) based on conditional equations.

Consider the following example of a system for reversing a list used in [Ullman and Van Gelder, 1985] to illustrate their scheme of top-down capture rules, where \( \text{rev} \) is reverse and \( \text{tcons} \) adds an element to the end of a list. (We use capital letters for variables in rules and terms.)

\[
\begin{align*}
\text{rev}(\text{nil}) & \rightarrow \text{nil} \\
\text{rev}(A \cdot X) & \rightarrow \text{tcons}(\text{rev}(X),A) \\
\text{tcons}(\text{nil},A) & \rightarrow A \cdot \text{nil} \\
\text{tcons}(B \cdot X,A) & \rightarrow B \cdot \text{tcons}(X,A)
\end{align*}
\]

A goal of the form \( X = \text{rev}(1 \cdot 2 \cdot \text{nil}) \) can be solved by rewriting the right-hand side of the goal to yield \( X = 2 \cdot 1 \cdot \text{nil} \); rewriting corresponds to the functional part of equational programming. On the other hand, a query like \( \text{rev}(X) = 1 \cdot 2 \cdot \text{nil} \), requires equation solving to produce the value(s) for \( X \) that satisfies the equation. This query has the answer, \( \{X \leftarrow 2 \cdot 1 \cdot \text{nil}\} \). Finding solutions corresponds to the logic programming capability.

Solving equations is, therefore, a basic operation in interpreters for such equational languages and efficient methods are of critical importance. In general, paramodulation can be used (as in resolution-based theorem provers to solve equations, but is highly inefficient. For equational theories that can be presented as a (ground) confluent rewrite system, better equation-solving methods have been devised, narrowing [Slagle–74, Fay–79, Hullot–80] being the most popular. Techniques for helping make a narrowing procedure efficient are given in [Josephson–Dershowitz–88]. An alternative approach to equation solving, based on decomposition and restructuring has been suggested in [Martelli, et al.–86]. We will refer to the latter as the decomposition procedure. In this paper, we combine the above two approaches in a goal-directed procedure. When an equation is unsatisfiable, none of these procedures are guaranteed to halt. Indeed, this is inherent to the semi-decidability of the (equational) satisfiability problem. Still, the ability to detect some unsatisfiable subgoals can save a lot of unnecessary computation.

We describe the narrowing and decomposition procedures in Section 2. Our new, goal-directed procedure is formulated as a Prolog program in Section 3; it captures the advantages of both narrowing and decomposition and incorporates pruning of certain unsatisfiable goals.

* This research was supported in part by the National Science Foundation under Grant DCR 85–13417.
2. Unconditional Equation Solving

In this section, we first review some basic notions of unconditional rewriting. Then, we describe the narrowing and decomposition procedures for solving equations—given a confluent rewrite system—and present examples to illustrate some drawbacks.

2.1. Rewriting

A rewrite rule is an oriented equation between terms, written $l \rightarrow r$; a rewrite system is a finite set of such rules. For a given system $R$, the rewrite relation $\rightarrow_R$ replaces any subterm that is an instance $l\sigma$ of a left-hand side $l$ by the corresponding instance $r\sigma$ of the right-hand side $r$ (where $\sigma$ is a substitution mapping variables to terms). We write $s \rightarrow_R t$, if $s$ rewrites to $t$ in one step; $s \rightarrow^*_R t$, if $t$ is derivable form $s$, i.e. if $s$ rewrites to $t$ in zero or more steps; $s \downarrow_R t$, if $s$ and $t$ join, i.e. if $s \rightarrow_R w$ and $t \rightarrow_R w$ for some term $w$. A term $s$ is irreducible, or in normal form, if there is no term $w$ such that $s \rightarrow w$.

A rewrite relation $\rightarrow_R$ is noetherian if there is no infinite chain of terms $t_1 \rightarrow_R t_2 \rightarrow_R \cdots \rightarrow_R t_k \rightarrow_R \cdots$. A rewrite relation is (ground) confluent if whenever two (ground, i.e. variable-free) terms, $s$ and $t$, are derivable from a term $u$, then a term $v$ is derivable from both $s$ and $t$. That is, if $u \rightarrow_R^* s$ and $u \rightarrow_R^* t$, then $s \rightarrow_R^* v$ and $t \rightarrow_R^* v$ for some term $v$. A rewrite system that is both (ground) confluent and noetherian is said to be (ground) convergent.

If $R$ is a convergent rewrite system and $E$ is the underlying equational theory (when rules in $R$ are taken as equations), then $s=t$ is a valid identity in $E$ iff $s \downarrow_R t$. An equational goal is given in the form $s \downarrow_R t$, where $s$ and $t$ are, in general, terms containing variables; a solution to such a goal is a substitution $\sigma$ such that $s\sigma \downarrow_R t\sigma$. This means that $s\sigma$ is equal to $t\sigma$ in the underlying $E$, for all substitutions of terms for variables in $s\sigma$ and $t\sigma$.

A solution is irreducible if each of the terms substituted for the variables in the equation are irreducible. Note that the terms $s$ and $t$ are interchangeable, since $s \downarrow_R t$ iff $t \downarrow_R s$; in this sense, equational goals are unoriented.

An equation solving procedure is complete if it can produce all solutions to any goal, up to equivalence in the underlying theory. That is, if $\sigma$ is a solution to $s \downarrow_R t$, then a complete procedure will produce a solution $\mu$ for the goal that is at least as general as $\sigma$. The more general a solution, the smaller it is under the following (quasi-) ordering $\leq$: $\mu \leq \sigma$ iff there exists a substitution $\tau$ such that $(X\mu)\tau \leftarrow_R X\sigma$, for all variables $X$ (where $\leftarrow_R$ is the reflexive, symmetric, and transitive closure of $\rightarrow_R$.)

2.2. The Narrowing Procedure

To solve goals using the narrowing method, given a confluent system $R$, two operations are applied to a goal:

Reflect

If $\sigma$ is the most general unifier of $s$ and $t$, then $\sigma$ is a solution of $s \downarrow_R t$.

Narrow

If $\sigma$ is a most general unifier of a nonvariable subterm $u$ of $s$ and a left-hand side $l$ of a rule $l \rightarrow r$, then $s \downarrow_R t$ has a solution if $s\sigma[r\sigma] \downarrow_R t\sigma$ does, where $s\sigma[r\sigma]$ is obtained by applying the substitution $\sigma$ to $s$ and rewriting the subterm $s\sigma$ to $r\sigma$.

Narrowing uses unification (instead of matching) to "apply" rules to terms that may contain variables. Since rule variables are universally quantified, one can always rename them so that the rule and term have no variable in common. For example, if $U + 0 \rightarrow U$ is a rule, then $(X + Y) + Z$ narrows to $X + Z$ via substitution $\{Y+0\}$.

For (ground) confluent systems, the narrowing procedure is complete with respect to irreducible solutions (even if $R$ is not noetherian). This is because for such solutions, all the rewrite steps in $s\sigma$ ($t\sigma$) can be simulated by narrowing steps. Narrowing can simulate any rewriting strategy (top-down, bottom-up, etc.); hence, it often produces duplicate solutions. For completeness, it is sufficient to simulate any one rewriting strategy.

Simple restrictions on narrowing, like narrowing at only outermost positions, are incomplete (outermost narrowing does not simulate every possible outermost rewriting). Our goal-directed method—presented in the next section—simulates innermost rewriting. Narrowing only at basic positions is one complete refinement [Hullot, 1980]. (A basic position is a nonvariable position of the original goal or one that was introduced into the goal by the nonvariable part of a right-hand side applied in a preceding narrowing step.) Another strategy (for noetherian systems) is to normalize all terms before any narrowing step [Fay, 1979]; in [Rety, 1987], the combination of both refinements is studied. In [Bosco et al., 1987], a strategy derived from simulating SLD-resolution on flattened equations is considered. While these restrictions are complete, they do not have the simplicity of our top-down goal-directed method.

If we solve the goal $\text{rev}(Y) = \text{l\_nil}$ using narrowing, we get the solution $\{Y = \text{\_nil}\}$. But the narrowing procedure does not halt after producing this unique solution. It generates infinitely many failing subgoals of the form $\text{rev}(\text{cons}(\text{cons}(...(\text{rev}(Y), A)...))) = \text{l\_nil}$.
2.3. The Decomposition Procedure

Using narrowing, one has no control over which (non-variable) narrowable subterm is used to produce new subgoals; all possibilities are explored. Martelli, et al. [1986] give a top-down equation-solving procedure, which ignores some narrowings, reducing the search space thereby. There are four basic operations:

**Decompose**

A goal of the form \( f(u_1, \ldots, u_n) \downarrow, f(v_1, \ldots, v_n) \) (with both terms having the same outermost operators), has a solution, if the \( n \) subgoals, \( u_1 \downarrow v_1, \ldots, u_n \downarrow v_n \), can be solved simultaneously.

**Restructure**

A goal \( f(t_1, \ldots, t_n) \downarrow t \) has a solution, if \( f(t_1, \ldots, t_n) \rightarrow v \) is a rule in \( R \) (the left-hand side of which has the same outermost operator as one side of the goal), and the \( n+1 \) subgoals, \( l_1 \downarrow u_1, \ldots, l_n \downarrow u_n \), and \( v \downarrow t \), can be solved simultaneously.

**Bind**

If the goal is of the form \( X \downarrow t \), where \( X \) is a variable, and \( X \) unifies with \( t \), then \( \{X \leftarrow t\} \) is a solution.

**Expand**

If the goal is of the form \( X \downarrow t \), where \( X \) is a variable, but \( X \) does not unify with \( t \) (because \( X \) occurs in \( t \)), then it has a solution if the \( n+1 \) subgoals, \( l_1 \downarrow u_1, \ldots, l_n \downarrow u_n \), can be solved simultaneously, where \( f(t_1, \ldots, t_n) \) is any subterm of \( t \), \( f(t_1, \ldots, t_n) \rightarrow v \) is a rule in \( R \) (with the same outermost operator), and \( t[v] \) is \( t \) with \( f(t_1, \ldots, t_n) \) replaced by \( v \).

A successful expansion amounts to narrowing \( t \).

The rule \( g(f(a)) \rightarrow a \) and goal \( X \downarrow f(g(X)) \) [Martelli, et al., 1986] demonstrates the need for expansion (what they call “full rewriting”) in the “occur check” case. Here, we can neither bind nor restructure, but by expanding at the subterm \( g(X) \), a solution \( \{X \leftarrow f(a)\} \) is obtained.

Though the decomposition method limits the search for solutions, where there are conflicting “constructor” symbols in the goal (a constructor is a symbol which is not outermost in any left-hand side), it introduces some new problems. Consider, for example:

\[
\begin{align*}
f(a(X), b(X)) & \rightarrow X \\
f(X, X) & \rightarrow X \\
a(e) & \rightarrow e \\
b(e) & \rightarrow e
\end{align*}
\]

To solve \( f(e, Y) \downarrow e \), narrowing would only use the second rule \( f(X, X) \rightarrow X \), giving the normalized solution \( \{Y \leftarrow e\} \). But the decomposition procedure also structures using the first rule \( f(a(X), b(X)) \rightarrow X \), to get the subgoals, \( a(X) \downarrow e, b(X) \downarrow Y, \) and \( X \downarrow e \); this gives another correct, but non-normalized, solution \( \{Y \leftarrow b(e)\} \). Thus, decomposition does not take full advantage of the fact that there is no way for \( e \) to rewrite to an instance of \( a(X) \) that enables the first rule to apply.

Moreover, there are unsatisfiable cases for which narrowing terminates with failure, but decomposition does not halt, as illustrated by the following example:

\[
\begin{align*}
f(a(X), b(X)) & \rightarrow a(X) \\
a(d(X)) & \rightarrow a(X) \\
b(d(X)) & \rightarrow b(X)
\end{align*}
\]

Consider solving the goal \( f(Y, Y) \downarrow Y \). Were one to try and narrow this, the search would stop immediately, as neither term is narrowable. The decomposition procedure, on the other hand, restructures the goal into \( Y \downarrow a(X) \) and \( Y \downarrow b(X) \), which in turn leads to attempts to solve \( a(X) \downarrow Y, b(X) \), with neither success nor failure. By using oriented goals, we show how to combine the advantages of this top-down approach with the elimination of narrowing subterms of left-hand sides.

8. Goal Directed Equation Solving

In this section, we introduce two new concepts, “operator derivability” and “oriented goals”, both of which are useful for pruning the search for solutions to goals. Rewriting, along with the pruning, is expressed as a set of rules (given here in pseudo-Prolog); by interpreting this as a logic program, equational goals may be solved.

8.1. Simulating Innermost Rewriting

Let the derivation \( t = f(t_1, \ldots, t_n) \rightarrow t_1 \rightarrow t_2 \cdots \rightarrow t_l \) be an bottom-up (innermost) rewriting sequence (if a rule is applied at some position then no lower position is rewritable). Derivations can be classified into two cases, depending on whether or not they contain a rewrite step at the outermost, root position:

**Directed Decompose**

If no rewrite step ever occurs at the top-level \( f \) of \( t \), then \( t \) also has \( f \) as its top operator. That is, \( t_l = f(t_1, \ldots, t_n) \) and there is a bottom-up derivation sequence of the \( t_l \) from \( t_1 \). This is expressed in the clause:

\[f(X_1, \ldots, X_n) \rightarrow f(Y_1, \ldots, Y_n) : X_1 \rightarrow Y_1, \ldots, X_n \rightarrow Y_n.\]

**Directed Restructure**

Suppose one rewrite step does take place at the top.
Then the instance of the rule of $R$ first applied at the top must be of the form $f(t_1, \ldots, t_n) \rightarrow v$ (with the same outermost operator $f$) and the subterms of $t = f(t_1, \ldots, t_n)$ must have been rewritten to make this rule applicable. We express this as:

$$f(X_1, \ldots, X_n) \rightarrow W := \begin{cases} f(Y_1, \ldots, Y_n) \rightarrow V & \text{if } R \in \mathcal{F} \\ X_1 \rightarrow Y_1 & \cdots & X_n \rightarrow Y_n & \sigma \rightarrow v & W. \end{cases}$$

These two clauses are actually schemas applying to all function symbols $f$; they can be coded in Prolog, using `functor` and `arg`. They constitute a complete program for finding the normal form of an input term using innermost rewriting.

### 3.2. Oriented Goals

The above clauses can also be used as a logic program that solves goals of the form $\sigma \rightarrow t$, where $\sigma$ and $t$ are (first-order) terms that may contain free, "logic" variables. Such goals are solved by finding substitutions $\sigma$ (for those variables) such that there is an innermost derivation $\sigma \rightarrow t$.

Unlike unoriented goals $\sigma \rightarrow t$ (which is symmetric in $\sigma$ and $t$), the goal $\sigma \rightarrow t$ is oriented, allowing rewritings only in $\sigma$. Unoriented goals can be re-expressed using oriented goals: replace $\sigma \rightarrow t$ by $\sigma \rightarrow Z \& \sigma \rightarrow t$, where $Z$ is a new variable. Consider again the example:

$$f(a(X), b(X)) \rightarrow X$$
$$f(X, X) \rightarrow X$$
$$a(e) \rightarrow e$$
$$b(e) \rightarrow e$$

To solve $f(e, Y) \rightarrow e$, we first replace it by the oriented goals $f(e, Y) \rightarrow Z \& e \rightarrow Z$. The directed decompose rule succeeds with the second conjunct and binds $Z$ to $e$, leaving the subgoal $f(e, Y) \rightarrow e$. The left-hand side $f(X_1, X_2) \rightarrow W$ of the directed restructuring rule for $f$ matches the new subgoal, and either of two rules in the above system match the condition $f(Y_1, Y_2) \rightarrow V$. If we pick $f(X, X) \rightarrow X$ we get subgoals $e \rightarrow X \& Y \rightarrow X$, which have a solution $\{Y=e, X=e\}$, obtained by decomposition. For the other $f$ rule, $f(a(X), b(X)) \rightarrow X$, the remainder of the condition fails, there being no way to solve $e \rightarrow a(X)$. The one successful solution, viz. $\{Y=e\}$, corresponds to the derivation $f(e, e) \rightarrow e$.

Note that no special rules (like `expand`) for the "occur-check" case are necessary. Consider solving the goal $g(f(X)) \rightarrow g(X)$ with rule $f(g(a)) \rightarrow a$. The decompose rule instantiates $X$ to $g(Z)$ and produces the subgoal $f(g(Z)) \rightarrow Z$, which can be solved by restructuring, yielding $(X \cdot g(a))$ as a solution.

Our two rule schemas serve as the basis of the goal-directed equation-solving procedure. Other than its simplicity, the main advantage of this formulation is that it allows one to easily incorporate additional rules that simplify and prune goals with no loss of completeness.

### 3.3. Operator Rewriting

Let $R$ be a rewrite system over terms constructed from a set $\mathcal{F}$ of function symbols. We consider a derived rewrite system $\mathcal{F}$ over $\mathcal{F}$, as follows: For each rule $f(t_1, \ldots, t_n) \rightarrow g(t_1, \ldots, t_n)$ in $R$, with $f \neq g$, we add a rule $f \rightarrow g$ to $\mathcal{F}$. For each rule $f(t_1, \ldots, t_n) \rightarrow X$ in $R$, where $X$ is a variable (sometimes referred to as a "collapsing" rule), we add rules $f \rightarrow g$, $g \rightarrow F$ for all function symbols $g$, other than $f$ in $\mathcal{F}$. Let $f$ and $g$ be two operators in $\mathcal{F}$. Operator $g$ is derivable from $f$ if $f \rightarrow g$. This (decidable) notion is similar to the "viability" criterion used in [Digricoli and Harrison, 1986] and allows us to prune subgoals during equation solving, since a goal $f(t_1, \ldots, t_n) \rightarrow g(t_1, \ldots, t_n)$ is satisfiable in $R$ only if $f \rightarrow g$.

For the reverse example of Section 1 we have:

<table>
<thead>
<tr>
<th>$R$</th>
<th>$F$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{rev}(\text{nil}) \rightarrow \text{nil}$</td>
<td>$\text{rev} \rightarrow \text{nil}$</td>
</tr>
<tr>
<td>$\text{rev}(A \cdot X) \rightarrow \text{tcons}(\text{rev}(X), A)$</td>
<td>$\text{rev} \rightarrow \text{tcons}$</td>
</tr>
<tr>
<td>$\text{tcons}(\text{nil}, A) \rightarrow A \cdot \text{nil}$</td>
<td>$\text{tcons} \rightarrow A \cdot \text{nil}$</td>
</tr>
<tr>
<td>$\text{tcons}(B \cdot X, A) \rightarrow B \cdot \text{tcons}(X, A)$</td>
<td>$\text{tcons} \rightarrow B \cdot X \cdot A$</td>
</tr>
</tbody>
</table>

Operator `nil` is derivable from `rev` but not from `tcons`.

Directed goals of the form $f(t_1, \ldots, t_n) \rightarrow g(t_1, \ldots, t_n)$, whose outermost operators do not satisfy the derivability criterion, can be pruned. That is, if $g$ is not derivable from $f$ in the corresponding rewrite system $\mathcal{F}$, then such goals will never be satisfiable. This can be expressed by the rule:

$$f(t_1, \ldots, t_n) \rightarrow g(t_1, \ldots, t_n) \leftarrow f \rightarrow g.$$

### 4. Conclusion

Putting all the above rules together, with some optimizations, we get the following Prolog program:

```
rile(X, Y) :- var(X), !, unify(X, Y).
rile(X, Y) :- not(derivable(X, Y)), !, fail.
rile(X, Y) :- functor(X, F, N), functor(Y, F, N), rile(N, X, Y).
               /*directed decompose*/
rile(X, Y) :- functor(X, F, N), functor(X, F, N), rile(X, Y).
               /*directed decompose*/
rile(X, Y) :- functor(X, F, N), functor(X, F, N), rule(L, R),
rile(N, X, L), rile(R, Y).
               /*directed restructure*/
rile(L, X, Y) :- arg(L, X, X), arg(Y, Y, Y), rile(X, Y).
               /*directed restructure*/
rile(0, X, Y).
```

Dershowitz and Sivakumar 169
where \( \text{RITE} \) is used for \( \rightarrow_{r} \), and \( \text{unify} \) and \( \text{derivable} \) predicates are defined in the natural way. The first rule, which checks if the query term is a variable, is used to not allow restructuring in variables. By extending this idea, we can also capture basic narrowing by keeping track of the non-variable positions where restructurings are necessary.

The procedure for solving (oriented) equational goals may be extended to handle conditional systems as well. Then, the procedure, itself a conditional rewrite program, serves as a meta-circular interpreter for conditional rewrite programs. While retaining the top-down approach of the decomposition procedure (looking at subterms only when necessary), we have been able to incorporate oriented goals (to prevent narrowing nonquery subterms) and pruning (for unsatisfiable goals)—both in a uniform manner.

There is still room for enhancements to the notion of operator rewriting, as can be seen from the following example:

\[
\begin{align*}
  a(d(X)) & \rightarrow b(X,X) \\
  a(f(e)) & \rightarrow f(e) \\
  b(f(X),Y) & \rightarrow b(X,f(Y))
\end{align*}
\]

Given the goal \( a(f(U)) \downarrow b(V,e) \), narrowing and decomposition produce infinitely many (nonsubsuming) equations when considering \( b(V,e) \). Our notion of operator derivability can be used to detect that the only way for a term headed by \( a \) to join a term headed by \( b \) is for the first to reach the form \( a(d(X)) \), whereas there is no way for the subterm \( f(U) \) of the left part of the goal to attain the form \( d(X) \); hence, the goal is unsatisfiable.

References


