Solving Goals in Equational Languages*

Nachum Dershowitz and G. Sivakumar

Department of Computer Science
University of Illinois at Urbana-Champaign
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 conditional rewrite rules and goals as equations to be solved. Procedures for completion of conditional equational theories, in a manner akin to that of Knuth and Bendix for unconditional theories, also require methods for solving equations appearing in conditions. Rewrite-based logic-programming uses (conditional) narrowing to solve equational goals. Recently a different, top-down equation solving procedure was proposed for unconditional rewrite systems. In this paper, we express equational goal solving using conditional rules. Some refinements are described: the notion of *operator derivability* is used to prune useless paths in the search tree and our use of *oriented goals* eliminates some redundant paths leading to non-normalized solutions. Our *goal-directed* method can also be extended to handle conditional systems.

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-Plaisted-85], SLOG [Fribourg-85], and EQLOG [Goguen-Meseguer-86]. In this paradigm, a program is a set of *rules*, that is, directed (conditional) equations, and a goal is the question whether an equation $s = t$ has a solution in the equational theory presented by the program. Computing consists of finding values (substitutions) for the variables in s and t for which the equality holds. Efficient methods of solving equations are therefore very important. So, too, is the ability to detect that equations are unsatisfiable.

* This research was supported in part by the National Science Foundation under Grant DCR 85-13417.

Consider the following example of a system for appending two lists:

$append(nil,X)$	$\rightarrow$	X
$append(U{\cdot}V,W)$	$\rightarrow$	$U{\cdot}append(V,W)$

A goal of the form $X{=}append(1{\cdot}nil,2{\cdot}nil)$ can be solved by *rewriting* the right-hand side of the goal to yield $X{=}1{\cdot}2{\cdot}nil$; rewriting corresponds to the functional part of equational programming. On the other hand, a query like $append(X,Y){=}1{\cdot}2{\cdot}nil$, requires *equation solving* to produce the different values for X and Y that satisfy the equation. This query has three answers, expressed as alternative bindings for the two variables in the goal: $\{X{\mapsto}nil,Y{\mapsto}1{\cdot}2{\cdot}nil\}$, $\{X{\mapsto}1{\cdot}nil,Y{\mapsto}2{\cdot}nil\}$, and $\{X{\mapsto}1{\cdot}2{\cdot}nil,Y{\mapsto}nil\}$. Finding these 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-86]. An alternative approach to equation solving, based on *decomposition* and *restructuring* has been suggested in [Martelli,*etal.*-86]. We will refer to the latter as the *decomposition* procedure. Both procedures are *complete*, in that they will find a solution if there exists one that substitutes unrewritable terms for the goal variables.

When, however, an equation is *unsatisfiable*, these procedures may not 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. Detecting unsatisfiability is also important in conditional completion procedures, such as the one in [Kaplan-88].

We describe the narrowing and decomposition procedures in Section 2. Our new, *goal-directed* procedure is formulated in Section 3 using conditional rules. Its behavior is illustrated and some optimizations are described that help capture the advantages of both narrowing and decomposition.

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 unconditional rewrite system—and present examples to illustrate some drawbacks.

2.1. Rewriting

A *rewrite rule* is an oriented equation, 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 an instance $l\sigma$ of a left-hand side l by the corresponding instance $r\sigma$ of the right-hand side r (where σ 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 said to be *irreducible*, or in *normal form*, if there is no term t such that $s \rightarrow_R t$.

A rewrite relation $\rightarrow_R$ is said to be *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) 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 confluent 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 $s \downarrow t$, where s and t are, in general, terms containing variables; a *solution* to such a goal is a substitution σ 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 $s \downarrow t$ iff $t \downarrow s$; 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 σ is a solution to $s \downarrow t$, then a complete procedure will produce a solution μ for the goal that is at least as general as σ. The more general a solution, the smaller it is under the following (quasi-) ordering $\leq$ on substitutions: $\mu \leq \sigma$ iff there exists a substitution τ such that $(X\mu)\tau \leftrightarrow_R^* X\sigma$, for all variables X (where $\leftrightarrow_R^*$ is the reflexive, symmetric, and transitive closure of $\rightarrow_R$.)

2.2. The Narrowing Procedure

To solve goals using the narrowing method, two operations are applied to a goal:

Reflect

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

Narrow

If σ is a most general unifier of a nonvariable subterm u of s and a left-hand side l of a rule $l \rightarrow r$ in R, then $s \downarrow t$ has a solution if $s\sigma[r\sigma] \downarrow t\sigma$ does, where $s\sigma[r\sigma]$ is obtained by applying the substitution σ 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. (We use capital letters for variables in rules and terms.) 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 $Y + 0 \rightarrow Y$ is a rule, then $(X + Y) + Z$ narrows to $X + Z$ via substitution $\{Y \mapsto 0\}$.

To solve a goal $s \downarrow t$, we try all possible ways of narrowing s and t. New subgoals produced thereby are checked for syntactic unifiability. Each sequence of narrowings leading to unifiable terms corresponds to a solution. A goal for which neither narrowing nor reflection applies is *unsatisfiable.* For (ground) confluent systems, the narrowing procedure is complete with respect to *irreducible* solutions (even if R is not noetherian).

Simple restrictions on narrowing, like an outermost narrowing strategy, are incomplete. Narrowing only at *basic* positions is one complete refinement [Hullot-80]. (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-79]; in [Rety-87], the combination of both refinements is studied. In [Bosco,*etal.*-87], a strategy derived from simulating SLD-resolution on flattened equations is considered.

To motivate our use of operator derivability later, let us examine a simple example, where narrowing does not halt for an unsatisfiable goal:

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

Clearly, the goal $a(Y) \downarrow b(Y)$ is unsatisfiable, because any term derived from $a(Y)$ will have a as its outermost operator (for any substitution for Y), while any term derived from $b(Y)$ will retain b as its outermost operator. But narrowing never stops: $a(Y)$ narrows to $a(Y_1)$, which narrows to $a(Y_2)$, ad infinitum, and similarly for $b(Y)$, producing an infinite sequence of instances of the same goal.

This particular example can be handled by the use of *subsumption* checking, as described in [Rety,*etal.*-85]. In general, though, the subsumption check cannot solve all the problems caused by infinitely narrowable terms.

2.3. The Decomposition Procedure

Using narrowing, one has no control over which (nonvariable) narrowable subterm is used 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 $f(u_1, \ldots, u_n) \downarrow t$ has a solution, if $f(l_1, \ldots, l_n) \rightarrow v$ 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 $\{X \mapsto t\}$ 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, $l_1 \downarrow t_1, \ldots, l_n \downarrow t_n$, and $X \downarrow t[v]$, can be solved simultaneously, where $f(t_1, \ldots, t_n)$ is any subterm of t, $f(l_1, \ldots, l_n) \rightarrow v$ 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.

Expansion amounts to "narrowing" at all possible subterms of t. The following example [Martelli,*etal.*-86] demonstrates the need for expansion in the "occur check" case: the rule is $g(f(a)) \rightarrow a$ and the goal is $X \downarrow f(g(X))$. Here, we can neither bind nor restructure, but by restructuring at the subterm $g(X)$, a solution $\{X \mapsto f(a)\}$ 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:

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

To solve $f(e,Y) \downarrow e$, narrowing would only use the second rule $f(X,X) \rightarrow X$, giving the normalized solution $\{Y \mapsto e\}$. But the decomposition procedure also restructures 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 $\{Y \mapsto b(e)\}$. 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:

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

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 $a(X) \downarrow b(X)$, with neither success nor failure.

3. Goal Directed Equation Solving

In this section, we formulate an equation solving procedure using conditional rules. Systems of conditional rules, as a programming paradigm that combines logic and functional programming, are described in [Dershowitz-Plaisted-87]. RITE [Josephson-Dershowitz-86] is an interpreter for such a language which uses conditional narrowing to solve goals. We briefly review this programming paradigm and then express the rewriting relation via a set of conditional rules. We introduce two new concepts, "operator derivability" and "oriented goals", which are useful for pruning the search for solutions to goals. Rewriting, along with the pruning rules, is expressed as a conditional rewrite program; by interpreting this program, equational goals may be solved.

3.1. Conditional Rewriting

A *conditional rule* is of the form $c : l \Rightarrow r$, meaning that the left-hand side l rewrites to the right-hand side r when the condition c holds. An example of a conditional rule for the $>$ predicate is $X>Y : s(X)>s(Y) \Rightarrow \mathbf{true}$. A *(conditional) rewrite program* is a system of such rules. Rewriting (narrowing), as defined earlier for unconditional systems, can be adapted to rules with conditions, by rewriting (narrowing) the condition to **true** before rewriting (narrowing) the term. For a given program P, we write $s \stackrel{*}{\Rightarrow} t$ is s rewrites, conditionally, to t. A condition is in general a conjunction of predicates $c_1 \,\&\, \cdots \,\&\, c_n$, and holds for substitution σ, if $c_i\sigma \stackrel{*}{\Rightarrow} \mathbf{true}$ for each conjunct c_i.

The RITE interpreter solves a goal (answers a query) g, by first using the conditional rules to rewrite g as much as possible and then performing one narrowing step, continuing in this way until **true** is reached. Narrowing steps are "don't know" nondeterministic; rewrite steps are "don't care"

nondeterministic. Assuming that the conditional program is (ground) convergent, then the interpreter will find a solution to g, if there is any substitution σ such that $g\sigma \Rightarrow^* \mathbf{true}$. A very useful feature of "eager" rewriting is that—by incorporating negative information—many narrowing paths can be pruned when a condition rewrites to **false**. Refer to [Dershowitz-Plaisted-85, Dershowitz-Plaisted-87] for complete definitions and examples.

3.2. Simulating Rewriting

Computing the rewrite relation $\rightarrow_R^*$ for an unconditional system can be broken down into three cases:

Reflect

Any term rewrites (in zero steps) to itself. This can be expressed as the following (unconditional) rule:

$$X \rightarrow^* X \;\Rightarrow\; \mathbf{true}.$$

Decompose

Suppose a term t is derivable from a term s, but no rule is applied at the outermost operator of s. Then t has the same outermost operator as s, and all rewriting takes place within proper subterms of s. This may be expressed as the conditional rule:

$$X_1 \rightarrow^* Y_1 \;\&\; \cdots \;\&\; X_n \rightarrow^* Y_n : \; f(X_1, \ldots, X_n) \rightarrow^* f(Y_1, \ldots, Y_n) \;\Rightarrow\; \mathbf{true}.$$

Restructure

The last possibility is that t is derivable from s and at least one rewrite step takes place at the top. If t is of the form $f(t_1, \ldots, t_n)$, then the instance of the rule of R first applied at the top must be of the form $f(s_1, \ldots, s_n) \rightarrow v$ (with the same outermost operator f). As a conditional rule we can express this as

$$f(Y_1, \ldots, Y_n) \rightarrow V \in R \;\&\; X_1 \rightarrow^* Y_1 \;\&\; \cdots \;\&\; X_n \rightarrow^* Y_n \;\&\; V \rightarrow^* W : \\ f(X_1, \ldots, X_n) \rightarrow^* W \;\Rightarrow\; \mathbf{true}.$$

Note that these are actually rule schemes, with one such rule per function symbol f.

3.3. Oriented Goals

The three conditional rules given above form a complete program for using a system R to rewrite terms. In other words, $s \rightarrow_R^* t$, for two terms s and t, iff $s \rightarrow^* t \Rightarrow^* \mathbf{true}$ using this program. The three conditional rules can also be used as a *logic* program that solves goals of the form $s \rightarrow^* t$, where s and t may contain "logic" variables. Such goals are solved by the RITE interpreter, which finds substitutions σ (for those variables) such that $s\sigma \rightarrow_R^* t\sigma$ is true.

Unlike unoriented goals $s \downarrow t$ (which is symmetric in s and t), the goal $s \rightarrow^* t$ is *oriented*, allowing rewritings only in s. Unoriented goals can be re-expressed using oriented goals: replace $s \downarrow t$ by $s \rightarrow^* Z \;\&\; t \rightarrow^* Z$, 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 $f(e,Y)\downarrow e$, it is first replaced by the oriented goals $f(e,Y) \rightarrow^* Z \ \& \ e \rightarrow^* Z$. The reflection 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 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 $\{Y \mapsto e, X \mapsto e\}$, obtained by reflection. 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. $\{Y \mapsto e\}$, corresponds to the derivation $f(e,e) \rightarrow_R^* e$.

We also do not need any special rules (like *expand*) for the "occur-check" case. Consider solving the goal $g(f(X))\downarrow X$ with rule $f(g(a)) \rightarrow a$. Converting this to oriented goals $g(f(X)) \rightarrow^* Y \ \& \ X \rightarrow^* Y$ and solving the second using reflection, leaves $g(f(X)) \rightarrow^* X$. This oriented goal can be solved by narrowing with the decompose rule. That instantiates X to $g(Z)$ and produces the subgoal $f(g(Z)) \rightarrow^* Z$, which can be solved by restructuring, yielding $\{X \mapsto g(a)\}$ as a solution.

The above three conditional rules are the basis of our goal-directed equation-solving procedure. The main advantage of this formulation, other than its simplicity, is that it allows us to incorporate additional conditional rules, which allow simplification and pruning of goals without losing completeness.

3.4. Basic Derivations

The goal-directed procedure retains the top-down approach of the decomposition procedure. One problem with its naive use is that it also "narrows" variable positions in the query. Consider, for example, a simple query of the form $X \rightarrow^* b$ with a rule $a \rightarrow b$. Though reflection produces the solution $X \mapsto b$, the restructuring rule also instantiates X with a, producing the equivalent solution $X \mapsto a$. Clearly, restructuring at a variable only produces superfluous reducible substitutions.

By using a "meta-predicate" *nonbasic*(X) to incorporate a version of *basic narrowing* [Hullot-80], this problem is avoided. Nonbasic positions are either variables in the original query or positions introduced by instantiating variables in query-terms with subterms of left-hand sides. To produce irreducible solutions one need not consider narrowing at nonbasic positions. This effect can be captured by forcing goals of the form $s \rightarrow^* t$, where s is *nonbasic*, to produce a solution by reflection or rewriting. This is expressed as the following conditional rule:

$$nonbasic(X): \ X \rightarrow^* Y \ \Rightarrow \ X = Y,$$

where $=$ is a new function symbol for which we also have the rule

$$X = X \ \Rightarrow \ \textbf{true}.$$

The first rule is used to simplify whenever the left-hand side of a goal is a variable or is a subterm of what was substituted for a variable in the original goal; it is not used to narrow. The second is used for narrowing, too.

3.5. 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 F over $\mathcal{F}$, as follows: For each rule $f(t_1,\ldots,t_n) \rightarrow g(s_1,\ldots,s_m)$ in R, we add a rule $f \rightarrow g$ to 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 $f \rightarrow g_i$ to F for all function symbols $g_i \in \mathcal{F}$. Let f and g be two operators in $\mathcal{F}$. Operator g is *derivable* from f if $f \rightarrow_F^* g$; operators f and g are *joinable*, if $f \downarrow_F g$. These (decidable) notions allow us to prune subgoals during equation solving, since a goal $f(t_1,\ldots,t_n) \downarrow g(s_1,\ldots,s_m)$ is satisfiable in R only if $f \downarrow_F g$.

A simple example is the following:

R			F
$a(e)$	$\rightarrow$	e	$a \rightarrow e$
$b(e)$	$\rightarrow$	e	$b \rightarrow e$
$g(Y,c)$	$\rightarrow$	$b(Y)$	$g \rightarrow b$
$f(e,e)$	$\rightarrow$	c	$f \rightarrow c$
$h(X)$	$\rightarrow$	X	$h \rightarrow e$
			$h \rightarrow a$
			$h \rightarrow b$
			$h \rightarrow c$
			$h \rightarrow f$
			$h \rightarrow g$

Note that $h(X) \rightarrow X$ gives rise to six rules for h in F; more rules are required if terms may contain symbols not in R. Operators g and h are joinable, but h is not derivable from g.

Goals of the form $f(s_1,\ldots,s_n) \rightarrow^* g(t_1,\ldots,t_m)$, 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 F, then such goals will never be satisfiable. This can be expressed by the conditional rule:

$$f \not\rightarrow_F^* g: \qquad f(X_1,\ldots,X_n) \rightarrow^* g(Y_1,\ldots,Y_m) \quad \Rightarrow \quad \textbf{false}.$$

Similarly, if we have two subgoals $f(X_1,\ldots,X_n) \rightarrow^* W$ and $g(Y_1,\ldots,Y_m) \rightarrow^* W$ then unless f and g are joinable, the whole goal may be pruned:

$$f \not\downarrow_F g: \qquad f(X_1,\ldots,X_n) \rightarrow^* W \;\&\; g(Y_1,\ldots,Y_m) \rightarrow^* W \quad \Rightarrow \quad \textbf{false}.$$

Any solution to a goal of the form $f(s_1,\ldots,s_n) \rightarrow^* v$, where there are no rules in R for f, can only be obtained by decomposition. That is, $f(s_1,\ldots,s_n) \rightarrow^* v$ iff v is of the form $f(t_1,\ldots,t_n)$ and $s_i \rightarrow^* t_i$, for each i. We have:

$$f(Z_1,\ldots,Z_n) \rightarrow W \notin R: \; f(X_1,\ldots,X_n) \rightarrow^* f(Y_1,\ldots,Y_n) \;\Rightarrow\; X_1 \rightarrow^* Y_1 \;\&\; \cdots \;\&\; X_n \rightarrow^* Y_n.$$

The notions of derivable and joinable operators are easily implementable and can be used to guide the search for solutions when deciding with which rule to restructure a query. These new rules are only used for eager rewriting, not to instantiate goal variables. Some rules refer to more than one subgoal; we assume that they apply regardless of the order in which the subgoals are written. RITE maintains an explicit stack of conditions, all of which may participate in pruning.

4. Conclusion

Putting all the above conditional rules together, we get the following procedure, where rules referring to operators (f or g) are actually schemata for all function symbols that can appear in terms (and m and n are their respective arities):

Goal-Directed Procedure			
	$U \rightarrow^* U$	$\Rightarrow$	**true**
$f(Y_1,\ldots,Y_n) \rightarrow V \in R$ & $X_1 \rightarrow^* Y_1$ & $\cdots$ & $X_n \rightarrow^* Y_n$ & $V \rightarrow^* W$:	$f(X_1,\ldots,X_n) \rightarrow^* W$	$\Rightarrow$	**true**
$X_1 \rightarrow^* Y_1$ & $\cdots$ & $X_n \rightarrow^* Y_n$:	$f(X_1,\ldots,X_n) \rightarrow^* f(Y_1,\ldots,Y_n)$	$\Rightarrow$	**true**
	$U = U$	$\Rightarrow$	**true**
nonbasic(U) :	$U \rightarrow^* V$	$\Rightarrow$	$U = V$
$f \not\rightarrow^*_F g$:	$f(X_1,\ldots,X_n) \rightarrow^* g(Y_1,\ldots,Y_m)$	$\Rightarrow$	**false**
$f \downarrow_F g$:	$f(X_1,\ldots,X_n) \rightarrow^* W$ & $g(Y_1,\ldots,Y_m) \rightarrow^* W$	$\Rightarrow$	**false**
$f(Z_1,\ldots,Z_n) \rightarrow W \notin R$:	$f(X_1,\ldots,X_n) \rightarrow^* f(Y_1,\ldots,Y_n)$	$\Rightarrow$	$X_1 \rightarrow^* Y_1$ & $\cdots$ & $X_n \rightarrow^* Y_n$

The first set of rules are used for (nondeterministically) narrowing goals; the other set is used eagerly (without backtracking) for simplifying subgoals, but not for narrowing.

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:

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

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.

Another useful notion that should be added, for convergent (terminating and confluent) systems, is *eager rewriting*—using rules in R to simplify goals as much as possible before applying goal-directed rules.

REFERENCES

[Bosco,*etal.*-87] Bosco, P. G., Giovanetti, E., and Moiso, C. "Refined strategies for semantic unification" *Proceedings of International Joint Conference on Theory and Practice of Software Development*, Pisa, Italy (March 1987), pp. 276-290. (Available as Vol. 250, Lecture Notes in Computer Science, Springer, Berlin.)

[Dershowitz-Plaisted-85] Dershowitz, N., and Plaisted, D. A. "Logic programming *cum* applicative programming". *Proceedings of the 1985 Symposium on Logic Programming*, Boston, MA (July 1985), pp. 54-66.

[Dershowitz-Plaisted-87] Dershowitz, N., and Plaisted, D. A. "Equational programming". In: *Machine Intelligence 11* (J. E. Hayes, D. Michie, and J. Richards, eds.), Oxford Press, Oxford, pp. 21-56, in press.

[Fay-79] Fay, M. "First-order unification in an equational theory". *Proceedings of the Fourth Workshop on Automated Deduction*, Austin, TX (February 1979), pp. 161-167.

[Fribourg-85] Fribourg, L. "SLOG: A logic programming language interpreter based on clausal superposition and rewriting". *Proceedings of the 1985 Symposium on Logic Programming*, Boston, MA (July 1985), pp. 172-184.

[Goguen-Meseguer-86] Goguen, J. A., and Meseguer, J. "EQLOG: Equality, types and generic modules for logic programming". In *Logic Programming: Functions, relations and equations* (D. DeGroot and G. Lindstrom, eds.), Prentice-Hall, Englewood Cliffs, NJ, pp. 295-363, 1986.

[Hullot-80] Hullot, J. M. "Canonical forms and unification". *Proceedings of the Fifth Conference on Automated Deduction*, Les Arcs, France (July 1980), pp. 318-334.

[Josephson-Dershowitz-86] Josephson, N. A., and Dershowitz, N. "An implementation of narrowing: The RITE way". *Proceedings of the Third IEEE Symposium on Logic Programming*, Salt Lake City, UT (September 1986), pp. 187-197. (Revised version to appear in *Journal of Logic Programming*.)

[Kaplan-88] Kaplan, S. "Simplifying conditional term rewriting systems: Unification, termination and confluence", *Journal of Symbolic Computation*, to appear.

[Kapur,*etal.*-86] Kapur, D., Sivakumar, G. and Zhang, H., "RRL: A Rewrite rule laboratory". *8th Intl. Conference on Automated Deduction*, Oxford, England (July 1986). (Available as Vol. 230, Lecture Notes in Computer Science, Springer, Berlin.)

[Martelli,*etal.*-86] Martelli, A., Moiso, C. and Rossi, G. F. "An algorithm for unification in Equational Theories". *Proceedings of the Third IEEE Symposium on Logic Programming*, Salt Lake City, UT (September 1986), pp. 180-186.

[Padawitz-87] Padawitz, P. "Strategy-controlled reduction and narrowing". *Proceedings of the Second International Conference on Rewriting Techniques and Applications*, Bordeaux, France (May 1987), pp. 242-255.

[Reddy-85] Reddy, U. S. "Narrowing as the operational semantics of functional languages". *Proceedings of the 1985 Symposium on Logic Programming*, Boston, MA (July 1985), pp. 138-151.

[Rety-87] Réty, P. "Improving basic narrowing techniques". *Proceedings of the Second International Conference on Rewriting Techniques and Applications*, Bordeaux, France (May 1987), pp. 228-241. (Available as Vol. 256, Lecture Notes in Computer Science, Springer, Berlin.)

[Rety,*etal.*-85] Réty, P., Kirchner, C., Kirchner, H., and Lescanne, P. "NARROWER: A new algorithm for unification and its application to logic programming". *Proceedings of the First International Conference on Rewriting Techniques and Applications*, Dijon, France (May 1985), pp. 141-157. (Available as Vol. 202, Lecture Notes in Computer Science, Springer, Berlin [September 1985].)