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, 1986]. Computing consists of finding values (substitutions) for the variables in a goal $s=t$ 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 *rev* is reverse and *tcons* adds an element to the end of a list. (We use capital letters for variables in rules and terms.)

$rev(nil)$	$\rightarrow$	nil
$rev(A \cdot X)$	$\rightarrow$	$tcons(rev(X),A)$
$tcons(nil,A)$	$\rightarrow$	$A \cdot nil$
$tcons(B \cdot X,A)$	$\rightarrow$	$B \cdot tcons(X,A)$

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

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 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 *irreducible*, or in *normal form*, if there is no term t such that $s \rightarrow_R t$.

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 $s \downarrow_R t$.

n *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 $t\sigma$.

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 $t \downarrow_? s$; 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 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 $\lesssim$ on substitutions: $\mu \lesssim \sigma$ 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, given a confluent system R, 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. 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 $\{Y \mapsto 0\}$.

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 $rev(Y)=1 \cdot nil$ using narrowing, we get the solution $\{Y \mapsto 1 \cdot nil\}$. But the narrowing procedure does not halt after producing this unique solution. It generates infinitely many failing subgoals of the form $rev(tcons(tcons(...(rev(Y_n),A)...)))=1 \cdot nil$.

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, $u_1 \downarrow_? v_1$, . . . , $u_n \downarrow_? v_n$, 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, $l_1 \downarrow_? t_1$, . . . , $l_n \downarrow_? t_n$, and $X \downarrow_? t[v]$, can be solved simultaneously, where $f(t_1,..,t_n)$ 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 $f(t_1,..,t_n)$ 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 $\{X \mapsto f(a)\}$ 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:

$$\begin{array}{rcl} f(a(X),b(X)) & \rightarrow & X \\ f(X,X) & \rightarrow & X \\ a(e) & \rightarrow & e \\ b(e) & \rightarrow & e \end{array}$$

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:

$$\begin{array}{rcl} f(a(X),b(X)) & \rightarrow & a(X) \\ a(d(X)) & \rightarrow & a(X) \\ b(d(X)) & \rightarrow & b(X) \end{array}$$

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. By using oriented goals, we show how to combine the advatages of this top–down approach with the elimination of narrowing subterms of left–hand sides.

3. Goal Directed Equation Solving

In this section, we 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.

3.1. Simulating Innermost Rewriting

Let the derivation $t = f(t_1,..,t_n) \rightarrow t_1 \rightarrow t_2 \cdots \rightarrow t!$ 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, $t! = f(t_1!, \ldots, t_n!)$ and there is a bottom–up derivation sequence of the $t_i!$ from t_i. This is expressed in the clause:

$f(X_1,..,X_n) \rightarrow_? f(Y_1,..,Y_n) :- X_1 \rightarrow_? Y_1 \,\&\, \cdots \,\&\, X_n \rightarrow_? Y_n.$

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 $f(s_1, \ldots, s_m) \rightarrow v$ (with the same outermost operator f) and the subterms of $t = f(t_1,..,t_n)$ must have been rewritten to make this rule applicable. We express this as:

$$f(X_1,..,X_n) \rightarrow_? W \;:-\; f(Y_1,..,Y_n) \rightarrow V \in R \;\&\; X_1 \rightarrow_? Y_1 \;\&\; \cdots \;\&\; X_n \rightarrow_? Y_n \;\&\; V \rightarrow_? W.$$

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 $s \rightarrow_? t$, where s and t are (first–order) terms that may contain free, "logic" variables. Such goals are solved by finding substitutions σ (for those variables) such that there is an innermost derivation $s\sigma \rightarrow_R^* t\sigma$.

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 $f(e,Y) \downarrow_? e$, 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 $\{Y \mapsto e, X \mapsto e\}$, 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. $\{Y \mapsto e\}$, corresponds to the derivation $f(e,e) \rightarrow_R^* e$.

Note that no special rules (like *expand*) for the "occur–check" case are necessary. Consider solving the goal $g(f(X)) \rightarrow_? X$ 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 $\{X \mapsto g(a)\}$ 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 F over $\mathcal{F}$, as follows: For each rule $f(t_1,..,t_n) \rightarrow g(s_1, \ldots, s_m)$ in R, with $f \neq g$, we add a rule $f \rightarrow g$ to F. For each rule $f(t_1,..,t_n) \rightarrow X$ 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 other than f in $\mathcal{F}$. Let f and g be two operators in $\mathcal{F}$. Operator g is *derivable* from f if $f \rightarrow_F^* g$. 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 $f(t_1,..,t_n) \rightarrow_? g(s_1, \ldots, s_m)$ is satisfiable in R only if $f \rightarrow_F^* g$. For the reverse example of Section 1 we have:

R			F
$rev(nil)$	$\rightarrow$	nil	$rev \rightarrow nil$
$rev(A \cdot X)$	$\rightarrow$	$tcons(rev(X),A)$	$rev \rightarrow tcons$
$tcons(nil,A)$	$\rightarrow$	$A \cdot nil$	$tcons \rightarrow \cdot$
$tcons(B \cdot X,A)$	$\rightarrow$	$B \cdot tcons(X,A)$	$tcons \rightarrow \cdot$

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

Directed goals of the form $f(s_1, \ldots, s_m) \rightarrow_? g(t_1,..,t_n)$, 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 F, then such goals will never be satisfiable. This can be expressed by the rule: $f(X_1,..,X_n) \not\rightarrow_? g(Y_1, \ldots, Y_m) :- f \not\rightarrow_F^* g$.

4. Conclusion

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

```
rite(X,Y) :- var(X), !, unify(X,Y).
rite(X,Y) :- not(derivable(X,Y)), !, fail.
rite(X,Y) :- functor(X,F,N), functor(Y,F,N), rites(N,X,Y).
                /*directed decompose*/
rite(X,Y) :- functor(X,F,N), functor(L,F,N), rule(L,R),
                rites(N,X,L), rite(R,Y).
                /*directed restructure*/
rites(I,X,Y):-arg(I,X,Xi), arg(I,Y,Yi), rite(Xi,Yi),
                I1 is I-1, rites(I1,X,Y).
rites(0,X,Y).
```

where *rite* is used for $\rightarrow_{\mathcal{P}}$, and *unify* and *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:

$$\begin{array}{lcl} a(d(X)) & \rightarrow & b(X,X) \\ a(f(e)) & \rightarrow & f(e) \\ b(f(X),Y) & \rightarrow & b(X,f(Y)) \end{array}$$

Given the goal $a(f(U)) \downarrow_{\mathcal{P}} b(V,e)$, 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

[Bosco *et al.*, 1987] 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.

[Dershowitz and Plaisted, 1985] 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 and Plaisted, 1987] 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.

[Digricoli and Harrison, 1986] Digricoli, V. J., and Harrison, M. C., "Equality–based binary resolution" *JACM*, Vol. 33, No. 2, April 1986, pp. 253–289.

[Fay, 1979] 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, 1985] 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 and Meseguer, 1986] 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, 1980] Hullot, J. M. "Canonical forms and unification". *Proceedings of the Fifth Conference on Automated Deduction*, Les Arcs, France (July 1980), pp. 318–334.

[Josephson and Dershowitz, 1986] 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.

[Kaplan, 1987] Kaplan, S. "Simplifying conditional term rewriting systems: Unification, termination and confluence", *Journal of Symbolic Computation*, Vol. 4, No.3, pp. 295–334.

[Knuth and Bendix, 1970] Knuth, D. E., and Bendix, P. B. "Simple word problems in universal algebras". In: *Computational Problems in Abstract Algebra*, J. Leech, ed. Pergamon Press, Oxford, U. K., 1970, pp. 263–297.

[Martelli,*et al.*, 1986] 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, 1987] 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.

[Rety, 1987] 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.)

[Ullman and Van Gelder, 1985] Ullman, J. D. and Van Gelder, A. "Testing applicability of top–down capture rules". STAN–CS–85–1046, Dept. of Computer Science, Stanford University.