PROCEEDINGS OF THE INTERNATIONAL CONFERENCE
ON FIFTH GENERATION COMPUTER SYSTEMS 1988,
edited by ICOT. 



# CONDITIONAL EQUATIONAL PROGRAMMING AND THE THEORY OF CONDITIONAL TERM REWRITING*

Nachum Dershowitz and Mitsuhiro Okada

Department of Computer Science
University of Illinois
Urbana, IL 61801, USA

Department of Computer Science
Concordia University
Montreal, Quebec H3G 1M8, CANADA

## ABSTRACT

Conditional (directed) equations provide a paradigm of computation that combines the clean syntax and semantics of both PROLOG–like logic programming and LISP–like functional programming in a uniform manner. For functional programming, equations are used as rules for left–to–right "rewriting"; for logic programming, the same rules are used for "conditional narrowing". Increased expressive power is obtained by combining both rewriting and narrowing.

In this paper, we discuss the theory behind conditional rewriting. We provide criteria for the two most important correctness properties for conditional rewrite programs: termination and (ground and general) confluence. We present reasonable conditions for ensuring the completeness of the conditional rewriting and narrowing mechanisms. In particular, we address the situation where conditions contain existentially quantified variables.

## 1. INTRODUCTION

Various proposals have been set forth for combining features of functional programming and logic (relational) programming. [See the collection in (DeGroot and Lindstrom 1986).] The simplest provide a convenient interface between resolution–based goal reduction and rewrite–based term evaluation, without fully integrating the two. Such languages normalize terms (i.e. rewrite them to terms than cannot be rewritten further) before attempting unification, but do not use function definitions to instantiate free variables during goal reduction. Consequently, they are incomplete, in the sense that a solution to a goal will not necessarily be found whenever one provably exists. An early example of such a language is QLOG (Komorowski 1982).

An alternative approach is to treat function definitions of the form

$$f(\bar{x}) \Leftarrow \mathit{if}\ \ p[\bar{x}]\ \mathit{then}\ r[\bar{x}]\ \mathit{else}\ s[\bar{x}]$$

(where $\bar{x}$ is a list of variables and the square brackets indicate that the variables may appear anywhere in the indicated term) as a pair of implications

$$\begin{array}{rcrcl} p[\bar{x}] & \Rightarrow & f(\bar{x}) & = & r[\bar{x}] \\ \neg p[\bar{x}] & \Rightarrow & f(\bar{x}) & = & s[\bar{x}] \end{array}$$

and use paramodulation (unifying one side of an equation with a non–variable subterm of a clause and replacing with the other side) as an inference rule within a resolution–based interpreter. Uniform (Kahn 1981) is an early combination of LISP and PROLOG, incorporating such an equality rule. For example, the *append* function could be defined by the following set of clauses:

$$\begin{array}{rcrcl} & & \mathit{null}(\mathit{nil}) & & \\ \neg\mathit{null}(\mathit{cons}(x,y)) & & & & \\ & & \mathit{car}(\mathit{cons}(x,y)) & = & x \\ & & \mathit{cdr}(\mathit{cons}(x,y)) & = & y \\ \mathit{null}(x) & \Rightarrow & \mathit{append}(x,y) & = & y \\ \neg\mathit{null}(x) & \Rightarrow & \mathit{append}(x,y) & = & \\ & & \mathit{cons}(\mathit{car}(x),\mathit{append}(\mathit{cdr}(x),y)). & & \end{array}$$

*This research was supported in part by the National Science Foundation (United States) under Grant DCR 85–13417, the Natural Science and Engineering Research Council (Canada) under Grant OGP3663, ICR39976, Fonds pour la Formation de Chercheurs et l'Aide a la Recherche (Quebec) under Grant EQ41840, and the Committee on Aid to Research Activities (Concordia Univ.) under Grant N42.

Though completeness is achievable in such a manner, the resultant language requires non–linear forward reasoning and lacks the sense of direction that distinguishes computing from theorem proving.

Another alternative is to use function definitions

$$f(\bar{x}) \Leftarrow \textit{if } p[\bar{x}] \textit{ then } r[\bar{x}] \textit{ else } s[\bar{x}]$$

as one–way rewriting rules

$$f(\bar{x}) \rightarrow \textit{if } p[\bar{x}] \textit{ then } r[\bar{x}] \textit{ else } s[\bar{x}].$$

Rewrite rules are used to replace "equals–by–equals", but only in the left–to–right direction. That is, a rule $l \rightarrow r$ may be applied to a term $t$ if a subterm $s$ of $t$ matches (by "one–sided" unification) the left–hand side $l$ with some substitution $\sigma$ of terms for the variables in $l$. The rule is applied by replacing the subterm $s\sigma = l\sigma$ in $t$ with the right–hand side $r\sigma$. Two additional rules for simplification are needed:

$$\begin{aligned} \textit{if } \textit{true} \textit{ then } r \textit{ else } s &\rightarrow r \\ \textit{if } \textit{false} \textit{ then } r \textit{ else } s &\rightarrow s. \end{aligned}$$

Terms are rewritten until no rule applies; when (and if) that situation occurs, the resultant irreducible term, called a *normal form*, is considered the "value" of the initial term. Often, the two cases defined by a condition $p[\bar{x}]$ can be better expressed as mutually–exclusive left–hand–side patterns. For example, the following set of rules is all that is needed to define *append*:

$$\begin{aligned} \textit{append}(\textit{nil},y) &\rightarrow y \\ \textit{append}(\textit{cons}(x,y),z) &\rightarrow \textit{cons}(x,\textit{append}(y,z)). \end{aligned}$$

[For a survey of the theory of rewriting, see (Huet and Oppen 1980) or (Dershowitz and Jouannaud 1989).]

To use rewrite rules for logic programming, i.e. to find values for variables that satisfy an equational goal like $\textit{append}(x,y)=x$, a "linear" restriction on paramodulation analogous to the SLD– or SPU–strategy in Horn–clause logic, can be used. *Narrowing* (Slagle 1974) is like rewriting, except that unification is used in place of pattern matching: a rule $l \rightarrow r$ may be applied to a term $t$ if a *nonvariable* subterm $s$ of $t$ unifies with the left–hand side $l$ with some substitution of terms for the variables in $l$. (Variables in $l$ and $t$ are treated as disjoint.) The result is $t\sigma$ with $s\sigma$ replaced by $r\sigma$, where $\sigma$ is the most general unifier of $l$ and $s$. A programming language with narrowing–like operational semantics was first suggested by Dershowitz (1983). Other, independent suggestions of narrowing (or variants) as a way of incorporating functions and goal reduction are: "clausal superposition" for Horn clauses with equality in SEC (Fribourg 1984); the interleaving in EQLOG (Goguen and Meseguer 1984) of narrowing for solving equations with linear resolution for Horn clauses; the "equality rule" within TABLOG (Malachi et al. 1984); unification with conditional expressions in QUTE (Sato and Sakurai 1984); the use of decomposed equations within PROLOG by Tamaki (1984); the use of "logical variables" in the deterministic functional language FGL+LV (Lindstrom 1985); and "object refinement" in EQL (Jayaraman and Silbermann 1986). Related proposals include (Darlington et al. 1985; Kanamori 1985; McCabe 1985; Reddy 1985; Smolka 1985; Barbuti et al. 1986; You and Subrahmanyam 1986; Robinson 1988). Narrowing, like paramodulation, can be simulated (in PROLOG) by decomposing terms (Deransart 1983).

For completeness of narrowing, "ground confluence" of the system of oriented equations is required. Ground confluence implies that a variable–free term can have at most one normal form. With ground confluence, any irreducible solution to a goal can be found by narrowing. *Regular* systems, i.e. rewriting systems in which (a) no variable appears more than once on any left–hand side, (b) only variables that appear on the left appear on the right, (c) no left–hand side is unifiable with a nonvariable (not necessarily proper) subterm of another left–hand side, and (d) no left–hand side unifies with (a renamed) nonvariable proper subterm of itself, are confluent (Huet 1980). The above system for *append*, for instance, meets all these requirements. Semantic methods for establishing ground confluence are given in (Plaisted 1985; Zhang and Rémy 1985).

The main problem with using conditional (if–then–else) terms is that the resultant rules are usually not terminating. In other words, uninhibited rewriting need not halt (with an irreducible term). Thus, to guarantee that normal forms of terms and irreducible solutions to goals will be found (whenever they exist) requires a lazy, outermost evaluation strategy. [See, for example, (Reddy 1985).] Furthermore, rather strict syntactic conditions (as above) are necessary for completeness. And, without termination, full advantage cannot be taken of the evaluation mechanism available to functional languages.

These considerations suggest the use of *conditional* rewrite rules as a means of expressing function definitions. Each definition

$$f(\bar{x}) \Leftarrow \textit{if } p[\bar{x}] \textit{ then } r[\bar{x}] \textit{ else } s[\bar{x}]$$

translates into two conditional rules:

$$\begin{array}{lll} p[\bar{x}]=true: & f(\bar{x}) & \rightarrow \quad r[\bar{x}] \\ p[\bar{x}]=false: & f(\bar{x}) & \rightarrow \quad s[\bar{x}]. \end{array}$$

Using conditional equations, allows one to program with rules that can never lead to infinite sequences of rewrites. Proposed languages along these lines include RITE (Dershowitz and Plaisted 1985), SLOG (Fribourg 1985), and EQLOG (Goguen and Meseguer 1986).

In this paper, we concentrate on narrowing–based programming languages and terminating conditional systems. Section 2 deals with the completeness of conditional rewriting and Section 3 with establishing termination and confluence. Section 4 addresses the completeness of conditional narrowing and Section 5 considers what happens when conditions contain new (existentially quantified) variables. The results we report on are summarized in two tables; they have implications for program verification in many of the above–mentioned languages.

## 2. CONDITIONAL REWRITING

We use standard notations from equational theory and rewrite theory: $s = t$ stands for the usual sense of equality in systems; $s \rightarrow t$ stands for one rewrite step in a given rewriting system; $s \rightarrow^* t$ is the reflexive–transitive closure of the rewrite relation $\rightarrow$; $s \rightarrow^+ t$ is the transitive closure of $\rightarrow$; $s \downarrow t$ stands for $s \rightarrow^* u \ ^*\!\leftarrow$ for some $u$; $s \longleftrightarrow^* t$ means that there exist $u_1,\ldots, u_n$ such that $s \downarrow u_1 \downarrow \cdots \downarrow u_n \downarrow t$. We will assume some familiarity with the main notions in rewriting, viz. "termination", "confluence", and "critical pair".

By a *conditional equational system*, we mean a set of Horn clauses of the form

$$s_1=t_1 \wedge \cdots \wedge s_n=t_n \quad \Rightarrow \quad l \ = \ r.$$

A *standard conditional rewriting system* is a set of rules of the form

$$s_1 \downarrow t_1 \wedge \cdots \wedge s_n \downarrow t_n: \quad l \ \rightarrow \ r,$$

meaning that an instance $l\sigma$ of $l$ rewrites to $r\sigma$ only if each $s_i\sigma$ can be reduced (by zero or more rewrites) to the same term as the corresponding $t_i\sigma$. A *natural conditional rewriting system* has rules of the form

$$s_1 \longleftrightarrow^* t_1 \wedge \cdots \wedge s_n \longleftrightarrow^* t_n: \quad l \ \rightarrow \ r.$$

Here a rule applies if there exists a proof $s_i \longleftrightarrow^* t_i$ for each $i$ (using any number of rewrites in *either* direction).

For example, if $R$ is the standard system

$$\begin{array}{rrl} & null(nil) & \rightarrow \quad true \\ & null(cons(x,y)) & \rightarrow \quad false \\ & car(cons(x,y)) & \rightarrow \quad x \\ & cdr(cons(x,y)) & \rightarrow \quad y \\ null(x)\downarrow true: & append(x,y) & \rightarrow \quad y \\ null(x)\downarrow false: & append(x,y) & \rightarrow \\ & cons(car(x),append(cdr(x),y)). & \end{array}$$

then $append(cons(a,cons(b,nil)),cons(c,nil)) \rightarrow^* cons(a,cons(b,cons(c,nil)))$.

Note that a natural rewriting system is not very different from the underlying equational system (replacing $\rightarrow$ with $=$, $\downarrow$ with $=$, and : with $\Rightarrow$). That is,

**Theorem 1.** *For any natural rewriting system $R$ and underlying equational system $E$,*

$$R \vdash p \longleftrightarrow^* q \quad \Leftrightarrow \quad E \vdash p=q$$

*for all terms $p$ and $q$.*

If $p = q$ is provable in $E$, there is a proof in SPU (*Selected Positive Unit*) form, i.e., a proof in which each condition $s_i\sigma = t_i\sigma$ is proved before a substitution instance

$$s_1\sigma=t_1\sigma \wedge \cdots \wedge s_n\sigma=t_n\sigma \quad \Rightarrow \quad l\sigma \ \rightarrow \ r\sigma$$

of a conditional rule is used to replace equals. (In this section, we only consider universal theorems, not existential ones. Hence, we need not consider unification.) Thus, $R$ can prove $p \longleftrightarrow^* q$ by imitating the proof in $E$. In fact, the rewriting proof has exactly the same proof structure as the SPU one, where a right–to–left use of a rule in the proof of $p \longleftrightarrow^* q$ in $R$ corresponds to use of the symmetric axiom in the proof of $p = q$ in $E$. The SPU–proof strategy is complete with respect to first–order equational validity and provability and so is conditional replacement of equals for natural rewriting systems. Conversely, any proof in $R$ can readily be interpreted as a proof in $E$. For this reason we can identify (SPU–) proofs in an equational system with the corresponding proofs in the corresponding natural system. A proof $p \longleftrightarrow^* q$ in a natural rewriting system is therefore called an *equational proof.*

For a standard system $R$, we will let $R^=$ denote the underlying equational system and $R^*$ denote the corresponding natural system (replacing $\downarrow$ with $\longleftrightarrow^*$). The following implications are obvious:

**Theorem 2.** *For any standard conditional rewriting system $R$,*

$$R\vdash p\downarrow q \;\Rightarrow\; R^*\vdash p\downarrow q \;\Rightarrow\; R^=\vdash p=q.$$

For a comparison of these formulations of conditional rewriting with several others, see (Dershowitz et al. 1988). The question is, under what conditions can the implications be turned into equivalences, allowing equational proofs to be replaced by rewrite proofs.

From classical (unconditional) rewriting theory, we know that the *confluence* property ($s\rightarrow^* v\ ^*\!\leftarrow t$ for some $v$ whenever $s\ ^*\!\leftarrow u\rightarrow^* t$ for some $u$), is equivalent to the *Church–Rosser* property, namely that any equational proof $p \longleftrightarrow^* q$ can be replaced by a rewrite proof $p \downarrow q$. It follows that:

**Theorem 3.** *For any confluent natural conditional rewriting system $R^*$,*

$$R^*\vdash p\downarrow q \;\Leftrightarrow\; R^=\vdash p=q.$$

Furthermore, an inductive argument shows that:

**Theorem 4.** *For any confluent standard conditional rewriting system $R$,*

$$R\vdash p\downarrow q \;\Leftrightarrow\; R^=\vdash p=q.$$

It is not hard to see that if $R$ is confluent, then so is the natural system $R^*$, but the converse does not hold, in general, since the enabling conditions in a natural proof need not have proofs that are transformable into "downarrow" ones. What we need is something stronger than confluence. Let $s \downarrow^* t$ mean that there exists a normal proof $s \downarrow t$ such that every subproof $s_i\sigma \longleftrightarrow^* t_i\sigma$ used in establishing the conditions needed for $s \downarrow t$ is also of the "fully normal" form $s_i$ sigma $\downarrow^* t_i\sigma$. If for every proof of $s \longleftrightarrow^* t$ in a natural system $R^*$ there exists a fully normal proof $s \downarrow^* t$ in $R^*$, then we say that $R^*$ has the *strong Church–Rosser property.* With this notion, we have

**Theorem 5.** *If a natural conditional rewriting system $R^*$ has the strong Church–Rosser property, then*

$$R\vdash p\downarrow q \;\Leftrightarrow\; R^*\vdash p\downarrow q.$$

## 3. CONDITIONAL CONVERGENCE

A convergent (conditional or unconditional) rewriting system is one with both the confluence and termination properties. (A system is *terminating* if all reduction sequences are finite.) Convergent systems give unique irreducible forms for any given term. Thus, for a convergent system $R$, $R^= \vdash s=t$ if, and only if, the normal forms of $s$ and $t$ are identical. A *ground convergent* system is one that is both terminating and ground confluent. (Ground confluence means that $s\rightarrow^* v\ ^*\!\leftarrow t$ for some $v$ whenever $s\ ^*\!\leftarrow u\rightarrow^* t$ for a variable–free term $u$.) Ground convergent rewriting may be used as the evaluation mechanism for first–order functional programs and lends itself easily to parallel evaluation schemes.

For terminating unconditional systems, the Critical Pair Lemma (Knuth and Bendix, 1970) provides an effective test for confluence. Also, for such systems (assuming a finite number of rules), the joinability ($\downarrow$) relation is decidable. Thus, validity is decidable for convergent undecidable systems. Unfortunately, with conditional systems, we are faced with two new phenomena: (a) Decidability of joinability is not necessarily decidable, even for (finite) terminating conditional systems (Kaplan, 1987). (b) Contrary to what had been surmised (Kaplan, 1987), the critical pair test does not guarantee confluence for standard systems (Dershowitz et al. 1987). To overcome these difficulties, additional constraints on conditional systems are required.

Clearly, a rewrite system is terminating if, and only if, the reduction ordering $\rightarrow^*$ defined by the system is well–founded. In practice, one usually provides an embedding $o$ such that $s \rightarrow^+ t$ implies $o(s) > o(t)$ for a suitable well–founded partial ordering $>$. We have shown elsewhere (Dershowitz and Okada 1988) that the usual well–founded partial–order structures used in rewriting termination arguments are simple substructures of the system of Ackermann's ordinal numbers, a system of proof theoretic ordinals in logic. [See (Okada 1988) for some examples of the use of even stronger proof theoretic ordinals.] For conditional systems, termination arguments are complicated by the need to take conditions into account, since $o(l\sigma) > o(r\sigma)$ does not usually hold for $\sigma$ that do not satisfy the conditions of the rule.

As mentioned above, For decidability of joinability, well–foundedness of the reduction ordering is insufficient. A conditional system is called *decreasing* if there exists a well–founded extension $>$ of the reduction ordering $\rightarrow$ which satisfies two additional properties: (a) $>$ the proper subterm relation $\rhd$ (i.e. if $s$ is a proper subterm of $t$ then $t > s$) and (b) for each rule $s_1\downarrow t_1\wedge\cdots\wedge s_n\downarrow t_n$: $l \rightarrow r$, $l\sigma > s_i\sigma$, $t_i\sigma$ for all substitutions $\sigma$ and indices $i$ $(1\leq i\leq n)$.

**Lemma.** *For decreasing systems, all the basic notions are decidable, i.e., the rewrite relation ($\rightarrow$), derivability relation ($\rightarrow^*$), the joinability relation ($\downarrow$), and normal form attribute are all recursive.*

This can be proved by transfinite induction with respect to $>$.

One can readily confirm that decreasing systems are strictly more general than "simplifying systems" (Kaplan 1987) or "reducing systems" (Jouannaud and Waldmann 1986), but enjoy the same nice properties. (See Dershowitz et al. 1988.) In fact, decreasing systems exactly capture the finiteness of recursive evaluation of terms, in the following technical sense: For given conditional system $R$, let $:\!\!-$ be the relation defined by $s :\!\!- p$ if there is a rule $s_1\downarrow t_1 \wedge \cdots \wedge s_n\downarrow t_n : l \rightarrow r$ in $R$ such that $l\sigma$ is a subterm of $s$ and $p$ is one of the $s_i\sigma$ or $t_i\sigma$.

**Lemma.** *For any decreasing natural conditional rewriting system $R$, $(\rightarrow \cup :\!\!-)^+$ is well-founded if, and only if, $R$ is decreasing.*

The "if" direction follows directly from the definition of "decreasing"; the "only if" direction follows from the fact that $(\rightarrow \cup :\!\!- \cup \rhd)^+$ is well-founded if $(\rightarrow \cup :\!\!-)^+$ is.

It follows from results in the previous section, that in order for a standard rewriting system to be complete with respect to provability in the underlying equational system (or, equivalently, with respect to validity in the sense of first-order logic with identity), we need either to directly establish its confluence or to establish the strong Church-Rosser property for the corresponding natural system. We consider now conditions under which a terminating conditional system is confluent whenever its critical pairs are joinable.

If $c: l \rightarrow r$ and $p: s \rightarrow t$ are rules in a conditional system $R$ and $l$ unifies via most general unifier $\mu$ with a nonvariable subterm of $s$, then the conditional equation $c\mu \wedge p\mu \Rightarrow s\mu[r\mu] = t\mu$ is a *critical pair* of $R$, where $s\mu[r\mu]$ is $s\mu$ with its subterm $l\mu$ replaced by $r\mu$. It can be verified that the critical pair test does hold for terminating natural systems, i.e. if $s\sigma \downarrow t\sigma$ for every critical pair $c \wedge p \Rightarrow s = t$ and substitution $\sigma$ such that $c\mu\sigma$ and $p\mu\sigma$ hold, then the system is confluent. But standard systems require an additional constraint: no left hand side may unify with a proper nonvariable subterm of any left-hand side. Such systems will be called *overlay* systems.

**Theorem 6.** *A terminating overlay standard conditional rewriting system is confluent if every critical pair is joinable.*

See (Dershowitz et al. 1987) for a proof.

We should remark that interpreting Horn clauses as conditional rewrite rules (with right-hand side *true*) leads to an overlay system, because predicate symbols are never nested in the "head" of a clause. Furthermore, all critical pairs are joinable, since all right-hand sides are the same. This theorem also applies to pattern-directed functional languages in which defined functions may not be nested on left-hand sides.

When one side of each condition in a standard system is an irreducible ground term (like *true*), we call the system *normal*. Bergstra and Klop (1986) extended Huet's (1980) confluence result for regular unconditional systems to normal conditional systems. They showed that any left-linear (possibly nonterminating) normal system with no critical pairs is confluent. Since our interest here is solely in terminating systems, we can relax the "no critical pair" part. A critical pair $c\mu \wedge p\mu \Rightarrow s\mu[r\mu] = t\mu$, obtained from rules $c: l \rightarrow r$ and $p: s \rightarrow t$, is said to be *shallow joinable* if there exist a term $v$, a derivation $t\mu \rightarrow^* v$ with depth less than or equal to the depth of the rewrite $l\mu \rightarrow r\mu$, and a derivation $s\mu[r\mu] \rightarrow^* v$ with depth less than or equal to that of $s\mu \rightarrow t\mu$. By "depth" we mean the maximum depth of recursion in the evaluation of conditions.

**Theorem 7.** *A left-linear terminating normal conditional rewriting system is confluent, if every critical pair is shallow joinable.*

See (Dershowitz et al. 1987) for proofs of this theorem and of the necessity for each of the requirements.

For the last result of this section, we have:

**Theorem 8.** *A decreasing natural conditional system rewriting has the strong Church-Rosser property if every critical pair is joinable.*

The proof proceeds by replacing each proof level with a rewrite ($\downarrow$) proof. More precisely, first we normalize the surface proof $s \leftrightarrow^* t$ to a normal form $s \downarrow t$ in the given natural system. Then we consider the immediate conditions $c_1 \leftrightarrow^* d_1, \ldots, c_n \leftrightarrow^* d_n$ used for the proof $s \downarrow t$, and normalize the surface proof of each of those to $c_i \downarrow d_i$, exactly as in proof simplification for unconditional systems (Bachmair et al. 1985). This normalization process is repeated until all proofs and subproofs are rewrite proofs. The successive subproof normalizations stop after a finite number of steps on account of the decreasingness property. See (Okada 1987).

By virtue of this theorem, one can establish confluence of a standard system $R$ by showing that the corresponding natural system $R^*$ is decreasing and all its critical pairs are joinable.

## 4. CONDITIONAL NARROWING

The previous two sections concerned the use of conditional rewrite systems to reduce terms to their value and to answer universal queries of the form $\forall \overline{x}\ s[\overline{x}] = t[\overline{x}]$. In this and the following section, we discuss the application of rewriting techniques to solving existential queries of the form $\exists \overline{x}\ s[\overline{x}] = t[\overline{x}]$. This corresponds to the logic-programming capability of resolution-based languages like PROLOG.

Narrowing has been proposed as an extension for solving goals in rewriting-based languages. *Conditional narrowing* may be defined as follows: Let $s \downarrow t$ be a goal. If $s$ and $t$ are unifiable, then the goal is said to "narrow to true" via their most general unifier. Alternatively, if there is a conditional rule $c$: $l \rightarrow r$ such that $l$ unifies with a nonvariable subterm of $s$ (or $t$) via most general unifier $\mu$ (the variables in the rule are renamed so that they are disjoint from those in $s$), then all the conditions in $c\mu$ are narrowed in tandem until they are solved, say via substitution $\rho$. Then we say that the top-level goal narrows to $s\mu\rho \downarrow t\mu\rho$ via the composed substitution $\mu\rho$. Thus, narrowing is a "linear" process: rules are overlapped only on goals, not on other rules.

For example, given the standard conditional system for *append* in Section 2, the goal $append(x,y) \downarrow x$ narrows using the last rule if $null(x) \downarrow false$ narrows to true. By using (the renamed rule) $null(cons(u,v)) \rightarrow false$, we can solve the condition, narrowing the original goal to $cons(car(cons(u,v)), append(cdr(cons(u,v)),y)) \downarrow cons(u,v)$. Rewriting is a special case of narrowing; it reduces the above goal to $cons(u,append(v,y)) \downarrow cons(u,v)$. This, in turn, is narrowable by the first rule for *append* if $null(v) \downarrow true$ narrows to true. Solving, by letting $v$ be *nil*, we narrow to a new goal $cons(u,y) \downarrow cons(u,v)$. Since the two terms are now unifiable (letting $y = v$), narrowing has produced the solution $x = cons(u,v) = cons(u,nil)$ and $y = v = nil$.

In the unconditional case, it has been shown that narrowing is complete for any (ground) convergent system (Hullot 1980). By "complete", we mean that if there exists a substitution $\sigma$ such that $s\sigma \longleftrightarrow^* t\sigma$, then $s \downarrow t$ narrows to true. Similarly, the variant of narrowing in which only irreducible terms are narrowed is complete (Dershowitz 1983). For conditional systems the analogous result is that (under the same assumptions) any equationally satisfiable goal can be solved by conditional narrowing.

**Theorem 9** (Dershowitz and Plaisted 1988). *Narrowing is complete for ground convergent standard conditional rewriting systems (with no extra variables in conditions).*

The restriction that all variables occurring in conditions also appear on the left will be lifted in the next section.

For ground convergent systems, all goals may be reduced to normal form before any narrowing step. Simplification, that is reduction via terminating rules, is a very powerful feature, particularly when defined function symbols are allowed to be arbitrarily nested in left-hand sides. Assuming ground confluence and termination, any strategy can be used for simplification. Furthermore, negation can be handled by incorporating negative information in the form of rewrite rules, which are then used to simplify subgoals to *false*. Combined with eager simplification, this approach has the advantage of allowing unsatisfiable goals to be pruned, thereby avoiding some potentially infinite narrowing paths. See Dershowitz and Plaisted 1985. Normalizing before narrowing is not necessary, however, and other language proposals employ different strategies. Some superfluous paths (that cannot lead to solutions) can be avoided by making a distinction between constructor symbols and defined ones (assuming that terms built entirely from constructors are irreducible). Two terms headed by different constructors can never be equal; when headed by the same constructor, they are equal if, and only if, their respective arguments are equal. See, for example, (Dershowitz and Plaisted 1985; Fribourg 1985; Kanamori 1985; Reddy 1985). Other restrictions and variations of narrowing which preserve completeness include (Hullot 1980; Martelli et al. 1986; Dershowitz and Sivakumar 1988).

Even if a program is ground convergent, alternative narrowing derivations must be explored if completeness is to be assured. Thus, narrowing-based languages that deterministically choose one possible narrowing over others cannot guarantee that solutions will be found. Preprocessing and structure-sharing techniques for rewriting and narrowing are explored in (Josephson and Dershowitz 1988).

## 5. EXTRA VARIABLES

Traditional rewriting theory (e.g. Huet 1980) usually has a constraint on occurrences of variables, namely that every variable occurring on a right-hand side of a rule also occur on the corresponding left-hand side. A natural extension of this constraint for a conditional rules is this that every variable occurring either in a condition or on the right-hand side also occur on the left. But if conditional rules are to

generalize Horn–clause programming, such a constraint is unacceptable, since even very simple relations, such as transitivity, require extra variables in conditions.

Accordingly, we can redefine rewriting in the extra–variable case as follows: $u[l\sigma] \rightarrow u[r\sigma\tau]$ for a rule $c$: $l \rightarrow r$ if there exists a substitution $\tau$ for the new variables such that $c\sigma\tau$ holds. As an example, let's replace the rules for *append* with:

$$\begin{array}{rcl} append(nil,y) & \rightarrow & y \\ x{\downarrow}cons(u,v) \wedge append(v,y){\downarrow}z: & & \\ append(x,y) & \rightarrow & cons(u,z), \end{array}$$

where $u$, $v$, and $z$ are extra variables. The last rule is applicable if there *exist* substitutions for the extra variables that make the conditions hold. Now we have $append(cons(a,nil),nil) \rightarrow cons(a,nil)$, since $u = a$, $v = nil$, and $z = nil$ is a solution to the conditions. Operationally, narrowing may be used to solve conditions with extra variables; the definition of narrowing is unchanged.

Even with extra variables, Theorems 6 and 7 hold as stated. Allowing extra variables, however, does introduce a problem: ground confluence no longer guarantees the completeness of the narrowing mechanism (Giovannetti and Moiso 1988). The following theorem allows for extra variables, at the expense of a stronger confluence condition, called *level–confluence*. A standard system $R$ is (ground) level–confluent if there exists a term $v$ such that whenever $s \,{}^{*}\!\leftarrow u \rightarrow^{*} t$ with a maximum depth of $n$, there is a rewrite proof $s \downarrow t$ of depth no greater than $n$.

**Theorem 10** (Giovannetti and Moiso 1988). *Narrowing is complete for terminating ground level–confluent standard conditional rewriting systems.*

The proof of this theorem (as well as the previous theorem) is based on the following:

**Lemma.** *Let $R$ be a standard conditional rewrite system (possibly having extra variables). If $s\sigma \rightarrow^{*} t$, $\sigma$ is an irreducible substitution, and all the instances of rewrite rules use in the proof (not only the surface proof but also subproofs for conditions) are irreducible, then there exist a term $u$ and substitutions $\eta$ and $\tau$ such that $s$ narrows to $u$ via $\eta$, $u\tau = t$, and $\eta\tau = \sigma$.*

By "irreducible substitution", we mean that the substitution maps all variables to irreducible terms; by "irreducible instance", we mean that the rule is applied to a subterm having irreducible terms matching left–hand side variables and any extra variables appearing in conditions.

The lemma is proved by double induction on the depth and length of the derivation $s\sigma \rightarrow^{*} t$. If the derivation is empty (has zero steps), then $s\sigma = t$, and the result is obvious. Suppose then that $s\sigma$ is first reduced using some rule $p$: $l \rightarrow r$ in $R$. Since $\sigma$ is irreducible, it must be that $s$ has a nonvariable subterm $u$ such that $u\sigma$ is an instance $l\theta$ of $l$, i.e. $s\sigma[u\sigma] = s\sigma[l\theta] \rightarrow s\sigma[r\theta] \rightarrow^{*} t$ and $p\theta$ narrows to true where $\theta$ is a substitution for variables in $l$ and/or $p$. Let $\mu$ be the most general unifier of $u$ and $l$. Then, for some irreducible substitution $\tau$, we have $\sigma = \mu\tau$ when $\mu$ is restricted to the variables in $s$, and $\theta = \mu\tau$ when $\mu$ is restricted to variables in $l$. Since $p\theta = p\mu\tau$, by induction $p\mu$ narrows to true via some $\rho$ such that $\tau = \rho\eta$ for some irreducible $\eta$. By the definition of a single narrowing step, $s[u]$ narrows to $s\mu\rho[r\mu\rho]$ via $\mu\rho$. Since $s\sigma[r\theta] = s\mu\rho\eta[r\mu\rho\eta] \rightarrow^{*} t$, by induction $s\mu\rho[r\mu\rho]$ narrows to $u$ via some $\phi$, $\eta = \phi\psi$, and $t = u\psi$ for some $\psi$. Hence, we have $s[u]$ narrows to $u$ via $\mu\rho\phi$, $\sigma = \mu\rho\phi\psi$, and $t = u\psi$, as desired.

Unfortunately, level–confluence of critical pairs does not ensure level–confluence of the system, as can be seen from the following counterexample:

$$\begin{array}{rrcl} & h(f(a)) & \rightarrow & c \\ d{\downarrow}d: & h(x) & \rightarrow & j(x) \\ d{\downarrow}d: & c & \rightarrow & j(f(a)) \\ d{\downarrow}d: & a & \rightarrow & b \\ & c & \rightarrow & d \\ & j(g(b)) & \rightarrow & d \\ d{\downarrow}h(f(x)): & f(x) & \rightarrow & g(x) \end{array}$$

This normal system is terminating and every critical pair is level–joinable, but despite the fact that $f(b) \leftarrow f(a) \rightarrow g(a) \rightarrow g(b)$, narrowing cannot solve the goal $f(b) \downarrow g(x)$. However, since shallow–joinable critical pairs are level–joinable, we can apply Theorem 7, thereby ensuring completeness of narrowing for terminating left–linear shallow–joinable normal systems. The following, similar counterexample demonstrates the need for left–linearity:

$$\begin{array}{rrcl} k(a,a){\downarrow}c: & h(f(a)) & \rightarrow & p(a) \\ k(a,a){\downarrow}c: & h(x) & \rightarrow & j(x) \\ & p(b) & \rightarrow & j(f(a)) \\ k(a,a){\downarrow}c: & a & \rightarrow & b \\ k(x,a){\downarrow}c: & p(x) & \rightarrow & q(x) \\ & q(b) & \rightarrow & j(g(a)) \\ h(f(x)){\downarrow}c: & f(x) & \rightarrow & g(x) \\ & j(g(b)) & \rightarrow & c \\ & k(x,x) & \rightarrow & c \end{array}$$

An alternative approach to new variables can be based on the notion of decreasing systems. There is of course no way one can insist that left–hand sides be greater than *all* instances of a condition containing a new variable. Instead, we revise our definition of rewriting in the extra–variable case, and define $u[l\sigma] \longrightarrow u[r\sigma\tau]$ for a rule $c: l \longrightarrow r$ only if there exists an *irreducible* substitution $\tau$ for the new variables such that $c\sigma\tau$ holds. Then we can say that a system $R$ is decreasing if there exists a well–founded ordering containing the (new) rewrite relation $\longrightarrow$ and the proper subterm relation $\rhd$, and for which $l\sigma$ is greater than both terms of each condition in $c\sigma\tau$, for any irreducible substitution $\tau$.

For example, the above *append* system is decreasing in this sense. Note that the joinability of the conditions must take the form $x \longrightarrow^* cons(u,v) \wedge append(v,y) \longrightarrow^* z$, if the new variables are irreducible.

Employing the above lemma again, it can be shown that

**Theorem 11.** *Narrowing is complete for decreasing ground–canonical standard conditional rewriting systems.*

## 6. CONCLUSION

The confluence results of Section 3 are summarized in the following table:

| *System type* | *Confluence conditions* |
|---|---|
| unconditional | left–linear<br>no critical pairs<br>(Huet 1980) |
| unconditional<br>terminating | joinable critical pairs<br>(Knuth and Bendix, 1970) |
| conditional<br>normal | left–linear<br>no critical pairs<br>(Bergstra and Klop 1986) |
| conditional<br>normal<br>terminating | left–linear<br>shallow–joinable<br>critical pairs |
| conditional<br>normal<br>terminating | overlay<br>joinable critical pairs |
| conditional<br>decreasing | joinable critical pairs<br>(cf. Kaplan, 1987) |

For an atom (in our case, an equation) $P$, let $P'$ be any substitution instance of it. A closed formula (sentence) $P'$ is said to be in $\Pi_0^0$ (ground) form; a closed formula $\forall \overline{x}\, P'$ is said to be in (universal) $\Pi_1^0$–form; a closed formula $\exists \overline{x}\, P'$, is said to be in (existential) $\Sigma_1^0$–form; a closed formula $\forall \overline{x}\, \exists \overline{y}\, P'$ is said to be in (universal–existential) $\Pi_2^0$–form. Using this notation, and bearing in mind that that solved goals hold for all instances of variables not instantiated by narrowing, the main completeness results of this paper appear in the following table:

| *Mechanism* | *Complete for...* |
|---|---|
| First–order<br>Functional programming | essentially<br>$\Pi_0^0$–theorems |
| Ground–confluent rewriting | $\Pi_0^0$–theorems |
| Horn–clause programming | $\Sigma_1^0$–theorems |
| Confluent rewriting | $\Pi_1^0$–theorems |
| Ground–convergent narrowing<br>(no extra variables) | $\Sigma_1^0$–theorems |
| Ground–level–convergent<br>narrowing | $\Sigma_1^0$–theorems |
| Decreasing ground–convergent<br>narrowing | $\Sigma_1^0$–theorems |
| Convergent narrowing<br>(no extra variables) | $\Pi_2^0$–theorems |
| Level–convergent narrowing | $\Pi_2^0$–theorems |
| Decreasing convergent<br>narrowing | $\Pi_2^0$–theorems |

## REFERENCES


L. Bachmair, N. Dershowitz, and J. Hsiang, "Orderings for equational proofs," *Proceedings of the Symposium on Logic in Computer Science*, Cambridge, MA, pp. 346–357, June 1986.

R. Barbuti, M. Bellia, G. Levi, and M. Martelli, "LEAF: A language which integrates logic, equations and functions," in D. DeGroot and G. Lindstrom, ed., *Logic Programming*, pp. 201–238, 1986.

J. A. Bergstra and J. W. Klop, "Conditional rewrite rules: Confluency and termination," *J. of Computer and System Sciences* **32**, pp. 323–362, 1986.

J. Darlington, A. J. Field, and H. Pull, "The unification of functional and logic languages," in D. DeGroot and G. Lindstrom, ed., *Logic Programming*, pp. 37–70, 1986.

D. DeGroot and G. Lindstrom, eds., *Logic Programming: Functions, Relations, and Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1986.

P. Deransart, "An operational algebraic semantics of PROLOG programs," Internal report, Institut National Recherche en Informatique et Automatique, LeChesnay, France, 1983.

N. Dershowitz, "Computing with rewrite systems," Technical Report ATR–83(8478)–1, Information Sciences Research Office, The Aerospace Corp., El Segundo, CA, January 1983. (Appeared in *Information and Control* **64**(2/3), pp. 122–157 [May/June 1985].)

N. Dershowitz and J.–P. Jouannaud, "Rewriting systems," in A. Meyer, M. Nivat, M. Paterson, D. Perrin, ed., *Handbook of Theoretical Computer Science*, North–Holland, Amsterdam, 1989. (To appear.)

N. Dershowitz and M. Okada, "Proof–theoretic techniques and the theory of rewriting," *Proceedings of the Third Symposium on Logic in Computer Science*, Edinburgh, Scotland, pp. 104–111, July 1988.

N. Dershowitz, M. Okada, and G. Sivakumar, "Confluence of conditional rewrite systems," *Proceedings of the First International Workshop on Conditional Rewriting*, Orsay, France, July 1987. (Available as Vol. 308, *Lecture Notes in Computer Science*, Springer, Berlin [1988].)

N. Dershowitz, M. Okada, and G. Sivakumar, "Canonical conditional rewrite systems," *Proceedings of the Ninth Conference on Automated Deduction*, Argonne, IL, pp. 538–549, May 1988. (Available as Vol. 310, *Lecture Notes in Computer Science*, Springer, Berlin.)

N. Dershowitz and D. A. Plaisted, "Logic programming *cum* applicative programming," *Proceedings of the IEEE Symposium on Logic Programming*, Boston, MA, pp. 54–66, July 1985.

N. Dershowitz and D. A. Plaisted, "Equational programming," in J. Richards, ed., *Machine Intelligence 11: The logic and acquisition of knowledge*, pp. 21–56, Oxford Press, Oxford, 1988. (In Press.)

N. Dershowitz and G. Sivakumar, "Goal–directed equation solving," *Proceedings of the Seventh National Conference on Artificial Intelligence*, St. Paul, MN, pp. 166–170, August 1988.

L. Fribourg, "Oriented equational clauses as a programming language," *J. of Logic Programming* **1**(2), pp. 179–210, August 1984.

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

E. Giovannetti and C. Moiso, "A completeness result for conditional narrowing," *Proceedings of the First International Workshop on Conditional Rewriting*, Orsay, France, July 1987. (Available as Vol. 308, *Lecture Notes in Computer Science*, Springer, Berlin [1988].)

J. A. Goguen and J. Meseguer, "Equality, types, modules and (why not?) generics for logic programming," *Logic Programming* **1**(2), pp. 179–210, 1984.

J. A. Goguen and J. Meseguer, "EQLOG: Equality, types, and generic modules for logic programming," in D. DeGroot and G. Lindstrom, ed., *Logic Programming*, pp. 295–363, 1986.

G. Huet, "Confluent reductions: Abstract properties and applications to term rewriting systems," *J. of the Association for Computing Machinery* **27**(4), pp. 797–821, October 1980.

G. Huet and D. C. Oppen, "Equations and rewrite rules: A survey," in R. Book, ed., *Formal Language Theory: Perspectives and Open Problems*, pp. 349–405, Academic Press, New York, 1980.

J.–M. Hullot, "Canonical forms and unification," *Proceedings of the Fifth Conference on Automated Deduction*, Les Arcs, France, pp. 318–334, July 1980.

B. Jayaraman and F. S. K. Silbermann, "Equations, sets, and reduction semantics for functional and logic programming," *Proceedings of the ACM Conference on LISP and Functional Programming*, Cambridge, MA, pp. 320–331, August 1986.

N. A. Josephson and N. Dershowitz, "An implementation of narrowing," *Logic Programming*, 1988. (In press.)

J.–P. Jouannaud and B. Waldmann, "Reductive conditional term rewriting systems," *Proceedings of the Third IFIP Working Conference on Formal Description of Programming Concepts*, Ebberup, Denmark, 1986.

K. M. Kahn, "Uniform — A language based upon unification which unifies much of Lisp, Prolog and Act 1," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, B.C., pp. 933–939, August 1981.

T. Kanamori, "Computation by meta-unification with constructors," Report TR-152, Institute for New Generation Computer Technology, Tokyo, Japan, 1985.

S. Kaplan, "Simplifying conditional term rewriting systems: Unification, termination and confluence," *J. of Symbolic Computation* **4**(3), pp. 295–334, December 1987.

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

H. J. Komorowski, "QLOG—The programming environment for PROLOG in LISP," in S.-A. Tärnlund, ed., *Logic Programming*, pp. 315–322, Academic Press, 1982.

G. Lindstrom, "Functional programming and the logical variable," *Proceedings of the Twelfth ACM Symposium on Principles of Programming Languages*, New Orleans, LA, pp. 266–280, January 1985.

Y. Malachi, Z. Manna, and R. J. Waldinger, "TABLOG: The deductive tableau programming language," *Proceedings of the ACM Symposium on LISP and Functional Programming*, Austin, TX, pp. 323–330, August 1984.

A. Martelli, C. Moiso, and G. F. Rossi, "An algorithm for unification in equational theories," *Proceedings of the IEEE Symposium on Logic Programming*, Salt Lake City, UT, pp. 180–186, September 1986.

F. G. McCabe, "Lambda PROLOG," Report, Department of Computing, Imperial College, London, England, 1985.

M. Okada, "A logical analysis for the theory of conditional rewriting," *Proceedings of the First International Workshop on Conditional Rewriting*, Orsay, France, July 1987. (Available as Vol. 308, *Lecture Notes in Computer Science*, Springer, Berlin [1988].)

M. Okada, "Note on a proof of the extended Kirby–Paris game for labeled finite trees," *European Journal of Combinatorics*, 1988. (To appear.)

D. A. Plaisted, "Semantic confluence tests and completion methods," *Information and Control* **65**(2/3), pp. 182–215, May/June 1985.

U. S. Reddy, "Narrowing as the operational semantics of functional languages," *Proceedings of the IEEE Symposium on Logic Programming*, Boston, MA, pp. 138–151, July 1985.

J. A. Robinson, "Beyond LOGLISP – Combining functional and relational programming in a reduction setting," in J. Richards, ed., *Machine Intelligence 11: The logic and acquisition of knowledge*, pp. 1–20, Oxford Press, Oxford, 1988. (In press.)

M. Sato and T. Sakurai, "QUTE: A functional language based on unification," *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, Japan, pp. 157–165, November 1984.

J. R. Slagle, "Automated theorem-proving for theories with simplifiers, commutativity, and associativity," *J. of the Association for Computing Machinery* **21**(4), pp. 622–642, 1974.

G. Smolka, "FRESH: A higher-order language with unification and multiple results," in D. DeGroot and G. Lindstrom, ed., *Logic Programming*, pp. 469–524, 1986.

P. A. Subrahmanyam and J.- H. You, "FUNLOG: A computational model integrating logic programming and functional programming," in D. DeGroot and G. Lindstrom, ed., *Logic Programming*, pp. 157–198, 1986.

H. Tamaki, "Semantics of a logic programming language with a reducibility predicate," *Proceedings of the IEEE Symposium on Logic Programming*, Atlantic City, NJ, pp. 259–264, February 1984.

H. Zhang and J.-L. Rémy, "Contextual rewriting," *Proceedings First International Conference on Rewriting Techniques and Applications*, Dijon, France, pp. 46–62, May 1985. (Available as Vol. 202, *Lecture Notes in Computer Science*, Springer, Berlin [September 1985].)