RAL-88-043

Science and Engineering Research Council

Rutherford Appleton Laboratory

Chilton DIDCOT Oxon OX11 0QX

RAL-88-043

# Automated Equational Reasoning and the Knuth-Bendix Algorithm: An Informal Introduction

A J J Dick

June 1988

# AUTOMATED EQUATIONAL REASONING AND THE KNUTH-BENDIX ALGORITHM: AN INFORMAL INTRODUCTION

*A. J. J. Dick*

Rutherford Appleton Laboratory
Chilton, Didcot
OXON OX11 0QX

## 1. INTRODUCTION

Why reason with equations? The answer is that equality is a fundamental logic concept of universal importance; equations are often a very natural way of expressing mathematical knowledge, and the properties of the equality relation allow us to reason by "replacing equals by equals", a very powerful and general technique.

Particularly interesting to formal methods in computer science is the use of equations for defining the properties of objects, relations or functions, and the use of equations as rewrite rules to model computation.

A comprehensive survey of the use of equations in computer science has been published in [HueOpp80b].

Let us consider a typical example of equational reasoning.

Given the equations

$$\begin{aligned} 0+x &= x && (1)\\ x+0 &= x && (2)\\ (-x)+x &= 0 && (3)\\ (x+y)+z &= x+(y+z) && (4) \end{aligned}$$

prove that $(-(-a))=a$ for any $a$.

We may use the ability to freely interchange equal expressions in whatever context they may appear, to explore the properties and implications of the given equations, and prove the desired result. A typical human presentation of such a proof might be as follows :-

*Figure 1.*

| | | | | | |
|---|---|---|---|---|---|
| Proof :- | i) | $(-(-a))$ | = | $(-(-a))+0$ | by (2) |
| | ii) | | = | $(-(-a))+((-a)+a)$ | by (3) |
| | iii) | | = | $((-(-a))+(-a))+a$ | by (4) |
| | iv) | | = | $0+a$ | by (3) |
| | v) | | = | $a$ | by (1) |

How can we set about automating such reasoning?

A close examination of the reasoning process embodied in this human proof reveals that there is a great deal going on behind the scenes of an apparently simple proof. To illustrate just one aspect, consider the application of equation (3) in step ii), in which 0 is replaced by $-a+a$. At this stage of the proof, the motivation for binding the variable $x$ to $a$ is not obvious.[1] It's purpose is only revealed at step iv), where the second

[1] One can envisage a strategy in which variables are only bound to existing elements of the original expressions. Such a technique would, it seems, work satisfactorily in this case, but it does not provide a solution to the binding problem in general.

application of equation (3) requires such a binding. Perhaps it is misleading to represent the proof as a sequence of such steps when in reality the author of the proof may have used a quite different intuition to derive it (from step iii) outwards, for instance). However, a more accurate rendering of the step-wise proof may be:-

*Figure 2.*

| | | | | | |
|---|---|---|---|---|---|
| Proof :- | i) | $(-(-a))$ | = | $(-(-a))+0$ | by (2) |
| | ii) | | = | $(-(-a))+((-x)+x)$ | by (3) |
| | iii) | | = | $((-(-a))+(-x))+x$ | by (4) |
| | iv) | | = | $0+a$ | by (3) (with $x$ bound to $a$) |
| | v) | | = | $a$ | by (1) |

Such observations suggest that, in considering the automation of such reasoning, the following basic processes are necessary:-

☐ *Unification*: Finding variable bindings that will unify an expression with one side of an axiom. From the brief discussion above, it is clear that simple matching of an axiom to an expression is not sufficient. Embodied in unification is the ability to rename variables to avoid confusion.

☐ *Rewriting*: Replacing one side of an axiom by the other side within the context of an expression. (Replacing equals by equals).

☐ *Strategy*: This is where the major difficulty lies. Finding the sequence of axiom applications that show two expressions to be equal, requires effectively a search of an infinite network representing the closure of the equality relation. This search space is ridden with infinite sequences and loops requiring a very cautious approach. At the very least, one must

- ☐ record the path followed through the search space, and avoid repetition.
- ☐ adopt a ''breadth first'' strategy which favours ''less complex'' expressions, to avoid getting lost in ever divergent paths.

A naive strategy such as this one is totally *non-deterministic*, and requires full backtracking. For any non-trivial proof, a vast amount of searching is likely to be required before a proof is found. It is desirable that any strategy chosen should be

- ☐ *sound*, i.e. any solution found must be correct; and
- ☐ *complete*, i.e. must be capable of traversing the entire search space, so that if there is a solution, it will be found.

A large part of these notes will be concerned with a technique called the *Knuth-Bendix algorithm*, which is a complete and sound strategy for solving the equality problem expressed by a given set of axioms.

## 2. REWRITE RULES

A key idea in equational reasoning is to treat a set of equations as rules for rewriting expressions. A *rewrite rule*, written $E \Rightarrow E'$, is an equation which is used in only one direction, i.e. $E$ may be replaced by $E'$, but not vice versa. Thus in the context of the example above, we can write the equations as

*Figure 3.*

$$0+x \Rightarrow x \qquad \text{(R1)}$$

$$x+0 \Rightarrow x \qquad \text{(R2)}$$

$$-x+x \Rightarrow 0 \qquad \text{(R3)}$$

$$(x+y)+z \Rightarrow x+(y+z) \qquad \text{(R4)}$$

A set of such rewrite rules we call a *rewrite system.*

We write $E \Rightarrow^* F$ if $E$ can be rewritten to $F$ by a sequence of zero or more rules applied one after the other in any order. We write $E \Rightarrow^1 F$ if $E$ can be rewritten to $F$ by the application of a single rule. If none of a set of rewrite rules apply to an expression, $E$, then $E$ is said to be in *normal form* with respect to that set.

Using axioms in only one direction greatly decreases the number of possible applications. Whereas before the number of possible repeated applications of axioms was infinite (reflecting the infinite nature of the search space), in many cases (the most interesting ones!), there is only a finite way of repeated applying rewrite rules. We shall examine how we can ensure this finiteness in a moment. The problem with restricting our use of the axioms in this way, however, is that, although our deductions remain *sound*, we risk losing *completeness*. Can we still ensure that it is possible to traverse the entire search space? In relation to these ideas/problems, we now discuss two fundamentally important properties of rewrite systems.

## 2.1. Finite Termination

A rewrite system is said to be *finitely terminating* or *nœtherian* if there are no infinite rewriting sequences $E \Rightarrow E' \Rightarrow E'' \Rightarrow \cdots$.

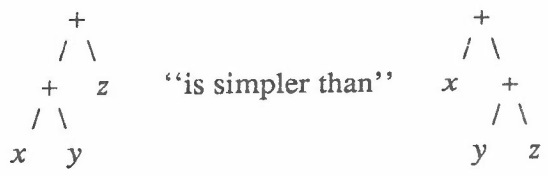
The importance of this property is that an algorithm that repeatedly applies rules to expressions whenever possible will always terminate, leaving the *normal form* of the original expression. It guarantees that every expression has a normal form. We have no need to worry about loops or infinite sequences in the search space; they are all avoided.

How then can we ensure finite termination? Examine for a moment the rewrite rules R1 to R3 above. It is clear that the right-hand side of each of them is in some respect ''simpler'' than the left-hand side. Thus any application of these rules is going to ''simplify'' the expression they are applied to.

Let us therefore define an ordering on expressions, », which describes their ''simplicity''. For example,

$0+x \gg x$ ($x$ ''is simpler than'' $0+x$ )
$x+0 \gg x$ ($x$ ''is simpler than'' $x+0$ )
$-x+x \gg 0$ ( 0 ''is simpler than'' $-x+x$ ).

It is easy to see how ''simplicity'' could somehow be related to size. The associativity axiom A4, however, is more difficult, since both sides appear to be of exactly the same size. An expression could be regarded as being simpler if, viewing expressions as trees, it has a bigger right than left subtree. i.e.




or $(x+y)+z \gg x+(y+z)$.

So for each rewrite rule, $E \Rightarrow E'$, we can insist not only that $E = E'$, but also that $E \gg E'$. But we must also require that, for every substitution $\sigma$, $\sigma(E) \gg \sigma(E')$, because in the process of rewriting, we may may use any instance of the rule $E \Rightarrow E'$.

If we insist on these properties, then we can say that the rewrite system is finitely terminating if and only if there is no infinite sequence $E \gg E' \gg E'' \cdots$. In the terminology associated with partial orderings, if « has this property, it is called a *well-founded ordering*.

Thus we can ensure *finite termination* of a rewrite system by associating a *well-founded ordering* on expressions such that for every rule $E \Rightarrow E'$, and all substitutions $\sigma$, $\sigma(E) \gg \sigma(E')$.

An ordering on terms is said to be *stable* or *compatible with term structure* if $E \gg E'$ implies that

i) $\sigma(E) \gg \sigma(E')$ for all substitutions $\sigma$; and

ii) $f(\cdots E \cdots) \gg f(\cdots E' \cdots)$ for all contexts $f(\cdots)$.

With a stable, well-founded ordering, we can ensure finite termination by checking so see that $E \gg E'$ for all rules $E \Rightarrow E'$.

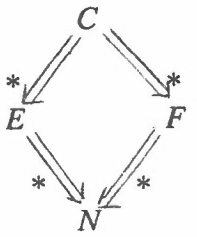
There are, of course, many ways of defining such orderings. The reader is referred to [HueOpp80b], [Der82] and[DerMan79] for a survey of various methods.

## 2.2. Unique Termination

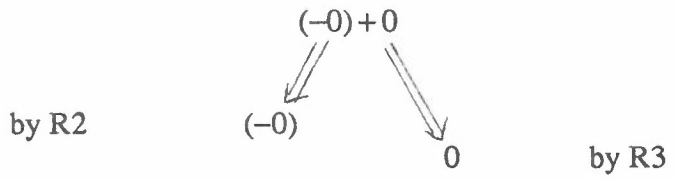
Finite termination ensures that every expression has a normal form. Unique termination together with finite termination ensures that every expression has just one normal form. This is a very important property, becuase it ensures *completeness*; in fact, it turns out that such a rewrite system provides a complete *decision procedure* for the equational theory.

Consider a set of rules which rewrites an expression, $C$, to $E$ or to $F$ by different sequences of applications. If for every such situation, there exist sequences of rules that allow $E$ and $F$ to be rewritten to the same normal form, $N$, then the set of rules is uniquely terminating. This property is illustrated in Figure 5, and is often called *confluence*, which means literally "flowing together"; intuitively, all rewriting sequences that diverge from the same place eventually flow together again.

*Figure 4.* **Confluence.**




The following example shows that the set of rules R1 to R4 above are not confluent :-




No further rules can be applied to 0 or (–0), showing that the expression (–0)+0 has more than one normal form.

A rewrite system that is both confluent and nœtherian is said to be *canonical*. Canonical rewrite systems enjoy the following Church-Rosser property, which characterizes the completeness of reasoning with rewrite rules:-

> A rewrite system is *Church-Rosser* if and only if, for all expressions $E$ and $F$, $E$ is equal to $F$ if and only if there exists an $N$ such that $E \Rightarrow^* N$ and $F \Rightarrow^* N$.

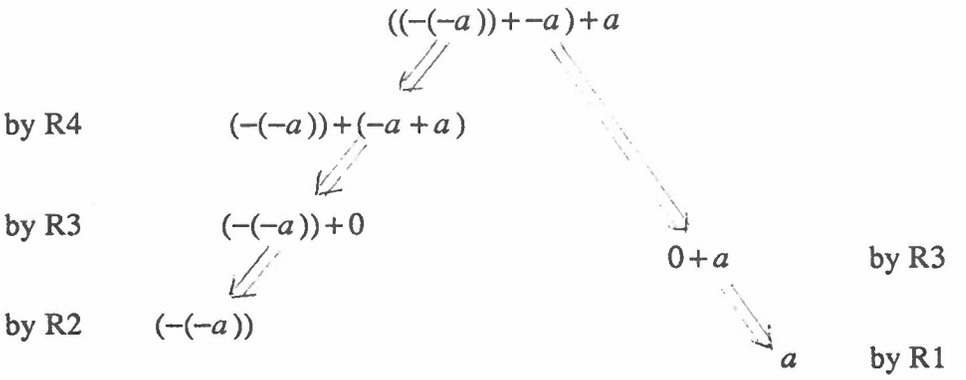
In other words, if a rewrite system is canonical, we can determine whether or not any two expressions are equal in the equational system by seeing if those expressions have the same normal form or not. What is more, confluence ensures that this is a decision procedure, in the sense that we can apply rules in any order and still get the same result; no backtracking or ''undoing'' of rule applications is required.

Later on we will consider how to check whether a rewrite system is canonical. In essence, the Knuth-Bendix algorithm is a method of adding new rules to a nœtherian rewrite system to make it canonical.

## 2.3. Proof Revisited

Now let us rearrange the proof of Figure 2 in a way that demonstrates changes in expression complexity :-

*Figure 5.* **Proof showing changes in expression complexity.**




Viewed in this light, the whole essence of the proof seems to be the top, most complex expression $((-(-a))+-a)+a$. Starting from $(-(-a))$, the complexity of the expression builds up to a peak, and is then simplified in another direction. All (non-trivial) proofs contain at least one peak expression, called a *critical expression*, which can be rewritten in two different ways. A proof that does not contain a peak expression is trivial, in that both sides of the theorem simplify to the same normal form by simple application of the rewrite rules. Note that the theorem being proved is represented by the normal form expressions, $(-(-a))$ and $a$.

The ''eureka'' step of devising a proof would seem to be the discovery of one or more appropriate critical expressions, from which proofs could be constructed by simplifying these expressions to alternative normal forms.

Here are some more examples :-

*Figure 6.* **Proof of commutativity of a group in which all non-neutral elements are of order two.**

Given the axioms

$$e \,.\, x \Rightarrow x \qquad \text{(R5)}$$

$$x \,.\, e \Rightarrow x \qquad \text{(R6)}$$

$$x \,.\, x \Rightarrow e \qquad \text{(R7)}$$

$$(x \,.\, y) \,.\, z \Rightarrow x \,.\, (y \,.\, z) \qquad \text{(R8)}$$

Proof :-

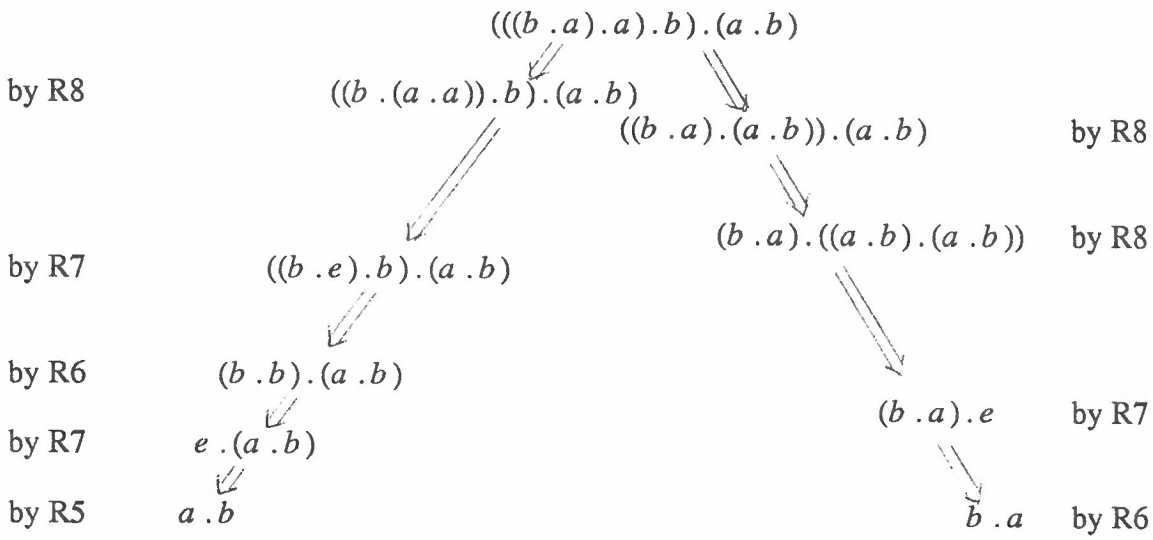
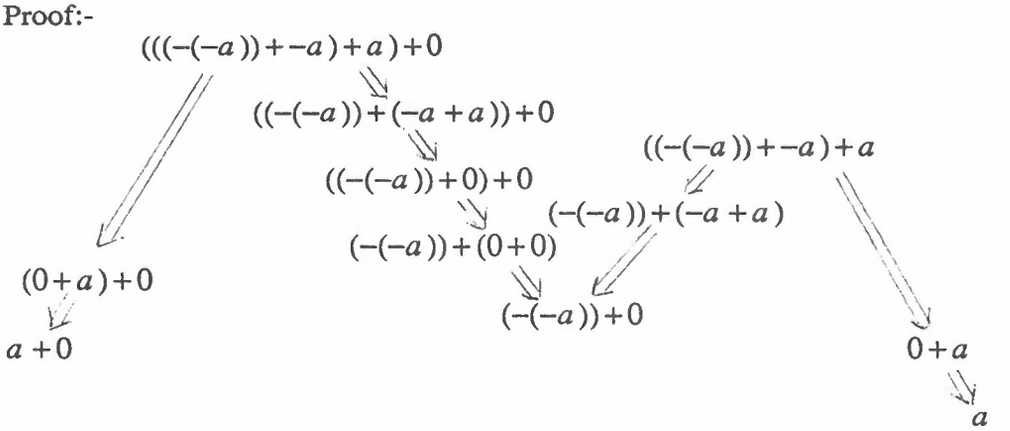


*Figure 7.*

Given axioms R1, R3 and R4,

prove axiom R2, i.e. that $x+0=x$.

Proof:-




## 3. RULE APPLICATION

The rewrite rule $L \Rightarrow R$, in effect, specifies that any instance $\sigma(L)$ of $L$ can be replaced by the corresponding instance $\sigma(R)$ of $R$. So in applying the rule to an expression, $E$, we must attempt to make substitutions in $L$ which make it identical to any sub-expression of $E$. If we represent the substitutions by $\sigma$, applying the rule is the same as

i) finding $\sigma$ such that $\sigma(L)=E'$ where $E'$ is a sub-expression of $E$; then

ii) replacing $E'$ by $\sigma(R)$ in $E$.

For example, take $L \Rightarrow R$ to be $x+0 \Rightarrow x$ and $E$ to be $y+(0+0)$. By letting $\sigma$ be $[x \leftarrow 0]$, (i.e. the substitution of 0 for $x$), we have $\sigma(L) = 0+0$, a sub-expression of $y+(0+0)$. Then $\sigma(R) = 0$ and the rewritten expression is $y+0$.

Now the rule can be applied a second time to the (sub-)expression $y+0$ by letting $\sigma$ be $[x \leftarrow y]$, rewriting the expression again to $y$. No more applications are possible.

## 4. CRITICAL EXPRESSIONS

We have observed that one of the keys to reasoning with non-confluent sets of rules is the discovery of appropriate *critical expressions*, which can be rewritten in two different ways.

How can we generate these critical expressions? For an expression to be rewritten in two different ways, there must be two rules that apply to it (or one rule that applies in two distinct ways). In other words, a critical expression must contain two occurrences of left-hand sides of rewrite rules.

Unification is the process of finding the most general common instance of two expressions. The unification of the left-hand sides of two rules (if successful) would therefore give us a critical expression to which both rules would be applicable.

Simple unification, however, is not sufficient to find all critical expressions, because a rule may be applied to *any part* of an expression, not just the whole of it. For this reason, we must unify the left-hand side of each rule with all possible sub-expressions[1] of

[1] In practice, we do not unify with a sub-expression if it is a simple variable, since this yields a critical expression of no practical value. The two instances must overlap in the superposed form.

left-hand sides. This process is called *superposition*.

For example, the critical expression $((-(-a))+-a)+a$ in the proof of Figure 6 can be generated by superposing rules R3 and R4; more particularly, by unifying the left-hand side of R3 with a sub-expression of R4, so that

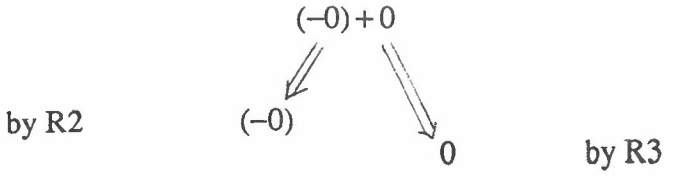
$$((-(-a))+(-a))+a \text{ is } ((x+y)+z)\,[x\leftarrow(-(-a))]\,[y\leftarrow(-a)]\,[z\leftarrow a]$$

and $(-(-a))+(-a)$ is $((-x')+x')\,[x'\leftarrow(-a)]$.

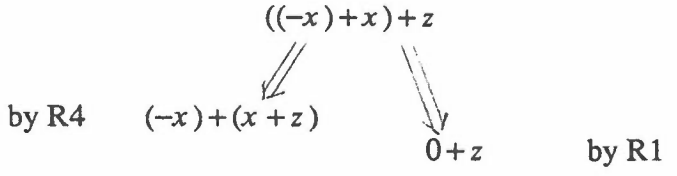
It is easy to see that the set of critical expressions associated with a set of rewrite rules may well be infinite. Generating and processing an infinite set of critical expressions does not seem to offer any advantage over a "trial and error" strategy. However, it is a well-known fact that, if the unification of two first order expressions is possible, it yields a most general unified form, unique to within renaming [Rob65]. This most general unified form is the one that subsumes all other unified forms. Superposition, being a form of multiple unification, likewise yields a finite set of most general critical expressions. Given a finite set of rules, then, we are able to generate a finite set of critical expressions which subsume the entire infinite set. For the rewrite rules R1 to R4 above, this set is :-

| | from | | and | |
|---|---|---|---|---|
| $0+0$ | | R1 | | R2 |
| $-0+0$ | | R2 | | R3 |
| $(0+y)+z$ | | R1 | | R4 |
| $(x+0)+z$ | | R2 | | R4 |
| $(x+y)+0$ | | R2 | | R4 |
| $(-x+x)+z$ | | R3 | | R4 |
| $((x+y)+z)+u$ | | R4 | | R4 |

Note that the critical expression $((-(-a))+(-a))+a$ used in the example proof is not found in the above set, but is subsumed by $((-x)+x)+z$. Thus we will not be able to find a proof for $(-(-a))=a$ directly. However, application of rules to these critical expressions enables us to derive two other simple theorems:-




i.e. $(-0)=0$.




i.e. $(-x)+(x+z)=z$.

Now let us consider these theorems as new rewrite rules:-

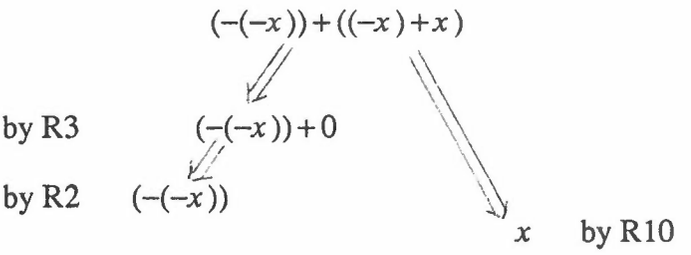
$$(-0) \Rightarrow 0 \qquad \text{(R9)}$$

$$(-x)+(x+z) \Rightarrow z \qquad \text{(R10)}$$

We must expand our set of critical expressions by

| | from | | and | |
|---|---|---|---|---|
| $(-0)+0$ | | R3 | | R9 |
| $(-0)+(0+z)$ | | R1 | | R10 |
| $(-x)+(x+0)$ | | R2 | | R10 |
| $(-(-x))+((-x)+x)$ | | R3 | | R10 |
| $(-(x+y))+((x+y)+z)$ | | R4 | | R10 |
| $(-0)+(0+z)$ | | R9 | | R10 |
| $(-(-x))+((-x)+(x+y))$ | | R10 | | R10. |

Simplification of these new expressions allows us to derive more theorems, including the one we want :-



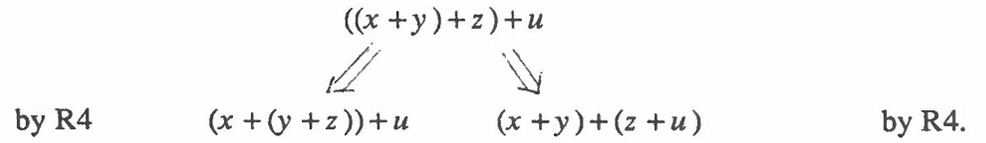

i.e. $(-(-x))=x$.

We have derived this proof, at best, after 2 successful superpositions, and 6 successful rules applications; in practical terms, more probably after 12 successful superpositions, and 10 successful rule applications.

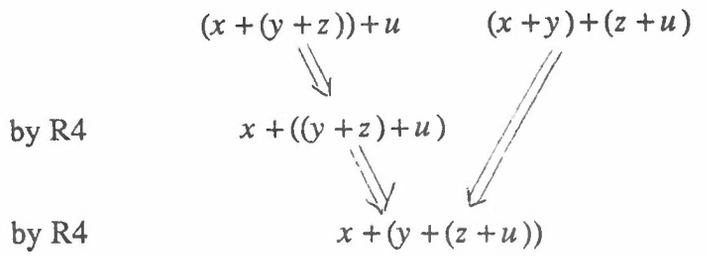
## 5. CREATING A CONFLUENT SET

What exactly are we doing when we introduce new rewrite rules in this way? The properties of equality ensure that these derived rules are true equations within the closure of the equality relation defined by the original axioms. We are in some way extending the repertoire of rewrite rules, and producing a more powerful simplifying machine.

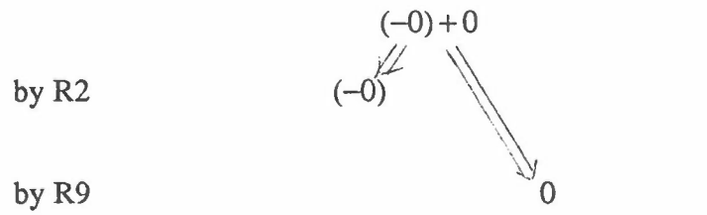
Consider for a moment the critical expression $((x+y)+z)+u$ derived from superposing R4 of on itself. It can be rewritten by R4 in two ways, as follows :-




We shall call $(x+(y+z))+u$ and $(x+y)+(z+u)$ a *critical pair*. The normal forms derived from the critical pair are the same, showing no necessity for a new rule :-

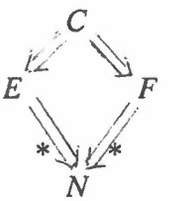



Now consider the critical expression $(-0)+0$ derived from rules R2 and R3, with critical pair $(-0)$ and 0, which formed the new rule R9, $(-0) \Rightarrow 0$ . When R9 was superposed on R3, the same critical expression was derived. On processing $(-0)$ and 0 the second time, the new rule caused the pair to have the same normal form:-



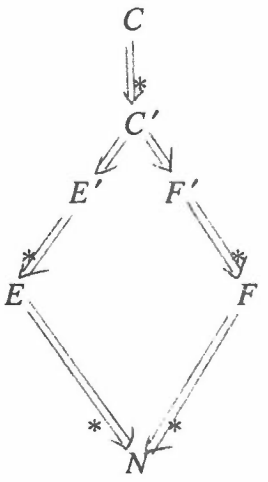

If all critical pairs are reduced to unique normal forms in this way, we produce no new rules, and the existing set of rules is said to be *locally confluent*.

*Figure 8.* **Local confluence.**




Here $C'$ is a critical expression with critical pair $<E', F'>$, and it is easy to see that confluence is equivalent to local confluence as expressed in Figure 10.

*Figure 9.* **Detail of confluence**




The equivalence of confluence and local confluence in nœtherian rewriting systems was first proved in [New42]. A key theorem by Knuth and Bendix[KnuBen70], section 5] showed that to test for local confluence it is sufficient to consider only those critical expressions found by the superposition of the left-hand sides of the rules. The reason for this is clear when it is realised that superposition yields expressions of minimum interaction between rules, and that all other critical expressions are subsumed by these.

## 6. KNUTH-BENDIX COMPLETION.

The Knuth-Bendix completion algorithm attempts to transform a set of axioms into a canonical set of rewrite rules. It uses precisely the method described above for finding new rules from critical pairs. The algorithm may be summarized as follows :-

*Figure 10.* **The Knuth Bendix Algorithm.**

(Initially, the axiom set contains the initial axioms,
and the rule set is empty).

A **while** the axiom set is not empty **do**

B **begin** Select and remove an axiom from the axiom set;

C Normalise the axiom;

D **if** the axiom is not of the form $x=x$ **then**
**begin**

E Order the axiom using the simplification ordering, «, to form a new rule (stop with failure if not possible);

F Place any rules whose left-hand side is reducible by the new rule back into the set of axioms;

G Superpose the new rule on the whole set of rules to find the set of critical pairs;

H Introduce a new axiom for each critical pair;

**end**

**end.**

The algorithm may behave in one of the following ways :-

- **terminate with success.** A finite canonical set has been found.
- **terminate with failure.** This occurs at step E if a particular axiom cannot be ordered by «. For example, the ordering we described in section 2 cannot order the commutative axiom, $a+b=b+a$. If such an axiom is generated, the algorithm fails.
- **loop without terminating.** Certain canonical sets are infinite, and in attempting to complete them, the algorithm never terminates. .LP Step F ensures that all rewrite rules are normalised with respect to each other. This means that the introduction of a new rule may cause existing rules to disappear. Thus the set of rules does not grow consistently with every iteration.

In the examples below, we show

a) the successful application of the Knuth-Bendix algorithm to a set of axioms describing a semi-group.

b) a case where the algorithm terminates with failure.

*Figure 11.* **Operation of the Knuth-Bendix algorithm on semi-group axioms.**

*Given axioms :-* *A1:* $0+x=x$
*A2:* $(-x)+x=0$
*A3:* $(x+y)+z=x+(y+z)$

*Operation of the algorithm:-*

| *Derived rules :-* | *Derivation :-* *Critical expression* | *From* | *Rules applied :-* *LHS* | *RHS* |
|---|---|---|---|---|
| R1: $0+x \Rightarrow x$ | | *A1* | | |
| *R2:* $-x+x \Rightarrow 0$ | | *A2* | | |
| *R3:* $(x+y)+z \Rightarrow x+(y+z)$ | | *A3* | | |
| *R4:* $-x+(x+y) \Rightarrow y$ | $((-x)+x)+y$ | *R3 R2* | *R3* | *R2,R1* |
| *R5:* $-0+x \Rightarrow x$ | $(-0)+(0+x)$ | *R1 R4* | *R1* | *R4* |
| *R6:* $-(-x)+0 \Rightarrow x$ | $-(-x)+((-x)+x)$ | *R2 R4* | *R2* | *R4* |
| *R7:* $-(-x)+y \Rightarrow x+y$ | $-(-x)+((-x)+(x+y))$ | *R4 R4* | *R4* | *R4* |
| *R6 becomes R8* | | | | |
| R8: $x+0 \Rightarrow x$ | $—x+((-x)+x)$ | R2 R4 | R2,R7 | R4 |
| R9: $(-0) \Rightarrow 0$ | $(-0)+0$ | R8 R2 | R8 | R2 |
| *R5 disappears* | | | | |
| R10: $-(-x) \Rightarrow x$ | $-(-x)+0$ | R8 R7 | R8 | R7,R8 |
| *R7 disappears* | | | | |
| R11: $x+(-x) \Rightarrow 0$ | $-(-x)+(-x)$ | R7 R2 | R7 | R2 |
| R12: $x+((-x)+y) \Rightarrow y$ | $-(-x)+((-x)+y)$ | R7 R4 | R7 | R4 |
| R13: $x+(y+-(x+y)) \Rightarrow 0$ | $(x+y)+-(x+y)$ | R3 R11 | R3 | R11 |
| R14: $x+-(y+x) \Rightarrow -y$ | $-y+(y+(x+-(y+x)))$ | R4 R13 | R4 | R13,R8 |
| *R13 disappears* | | | | |
| R15: $-(x+y) \Rightarrow -y+(-x)$ | $-y+(y+-(x+y))$ | R4 R14 | R4 | R14 |
| *R14 disappears* | | | | |
| *Terminates with success.* | | | | |

Complete set :-

| Derived rules :- | Derivation :- Critical expression | From | Rules applied :- LHS | RHS |
|---|---|---|---|---|
| R1: $0+x \Rightarrow x$ | | A1 | | |
| R2: $-x+x \Rightarrow 0$ | | A2 | | |
| R3: $(x+y)+z \Rightarrow x+(y+z)$ | | A3 | | |
| R4: $-x+(x+y) \Rightarrow y$ | $((-x)+x)+y$ | R3 R2 | R3 | R2,R1 |
| R8: $x+0 \Rightarrow x$ | $--x+((-x)+x)$ | R2 R4 | R2,R7 | R4 |
| R9: $(-0) \Rightarrow 0$ | $(-0)+0$ | R8 R2 | R8 | R2 |
| R10: $-(-x) \Rightarrow x$ | $-(-x)+0$ | R8 R7 | R8 | R7,R8 |
| R11: $x+(-x) \Rightarrow 0$ | $-(-x)+(-x)$ | R7 R2 | R7 | R2 |
| R12: $x+((-x)+y) \Rightarrow y$ | $-(-x)+((-x)+y)$ | R7 R4 | R7 | R4 |
| R15: $-(x+y) \Rightarrow -y+(-x)$ | $-y+(y+-(x+y))$ | R4 R14 | R4 | R14 |

*Figure 12.* **Operation of the Knuth-Bendix algorithm on a commutative group.**

Given axioms :- A1: $e.x = x$
A2: $x.e = x$
A3: $x.x = e$
A4: $(x.y).z = x.(y.z)$

| Derived rules :- | Derivation :- Critical expression | From | Rules applied :- LHS | RHS |
|---|---|---|---|---|
| R1: $e.x \Rightarrow x$ | | A1 | | |
| R2: $x.e \Rightarrow x$ | | A2 | | |
| R3: $x.x \Rightarrow e$ | | A3 | | |
| R4: $(x.y).z \Rightarrow x.(y.z)$ | | A4 | | |
| R5: $x.(x.y) \Rightarrow y$ | $(x.x).y$ | R4 R3 | R4 | R3,R1 |
| R6: $x.(y.(x.y)) \Rightarrow e$ | $(x.y).(x.y)$ | R4 R3 | R4 | R3 |
| R7: $y.(x.y) \Rightarrow x$ *R6 disappears* | $x.(x.(y.(x.y)))$ | R5 R6 | R5 | R6,R2 |
| R8: $y.x$ ?=? $x.y$ *Terminates with failure* | $x.(x.(y.x))$ | R5 R7 | R5 | R7 |

At this stage, it is found that $y.x$ and $x.y$ cannot be ordered, and the algorithm terminates with failure. We have, however, succeeded in proving that the group is commutative.

## 7. FAIR STRATEGIES IN KNUTH-BENDIX COMPLETION

Efficiency is affected considerably by the order in which axioms are selected from the axiom set for the formation of a new rule. Our choice has been to select the simplest or shortest axiom first. This strategy is consistently better than selection on a first-come first-served basis.

There are some strategies that would prevent a canonical rewrite systems from being found, even if a finite system existed. For this reason, whatever strategy used should be *fair* in the sense that

i) no axiom should be ignored indefinitely for consideration as a rewrite rule;

ii) no possible overlap between rules should be delayed indefinitely.

In the form of the algorithm described above, the actions of selecting an axiom and performing superpositions are connected events. So part i) of the fairness hypothesis above covers part ii). In other forms of the completion algorithm, e.g.[Hue81], the formation of rules from axioms is separated from the process of superposition, and fairness parts i) and ii) must be ensured separately.

## 8. LIMITATIONS OF THE KNUTH-BENDIX ALGORITHM.

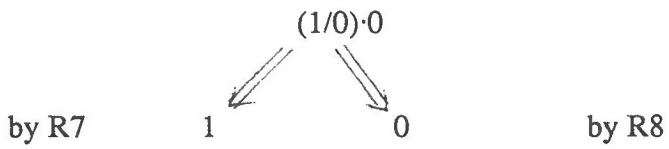
The limitations of the completion algorithm are in four main areas:-

- The simplification ordering, , is required to be a total ordering; i.e. expressions cannot be considered as having the same order of simplicity. In practice, it may be very hard to find a total ordering on expressions to suit particular needs. It would be very useful to be able to say ''I don't know which way to order this axiom'' and still be able to reason with it.
- Permutative axioms such as commutativity are of themselves themselves non-nœtherian. For example, either side of the axiom $a+b=b+a$ can be matched with the other, and whichever way the rule is first applied, it is immediately applicable again, thus generating an infinite rewriting sequence. Special treatment is required for such axioms.
- Many important algebras have canonical sets of rewrite rules which are infinite. We need a way of representing such sets finitely.
- Algebraic structures involving partial functions require some form of conditional axiom to prevent the generation of meaningless expressions. For example, given the axioms describing part of a division ring

$$(1/x)\cdot x \Rightarrow 1 \qquad \text{(R7)}$$

$$x\cdot 0 \Rightarrow 0 \qquad \text{(R8)}$$

the critical pair $(1/0)\cdot 0$ is found, which contains a division by zero. The critical pair, as may be expected, causes problems :-




The new equality, $1=0$ causes the equational theory to collapse.

Recent work has attempted to meet some of these problems in a variety of ways. For the

treatment of permutative axioms, three distinct approaches have developed:-

- ☐ by defining reduction on congruence classes of terms [LanBal77a]
- ☐ by defining critical pairs modulo special (commutative/associative) unification algorithms; here distributive lattices and Boolean algebras, for example, can be completed finitely[PetSti81]
- ☐ by using confluence of reduction modulo the congruence formed by the permutative axioms; however, rules are required to left-linear.[Heu80a].

For the treatment of certain infinite confluent sets,[Ric83] uses two sets of rules with slightly different simplification orderings.

Conditional rewriting systems have been developed by[LanBal79], [Rem82] and others, and used in [Gog78] for handlingly partial algebras.

Special unification which takes into account ordered sorts is used in ERIL[Dic87] as means of treating certain partial algebras.

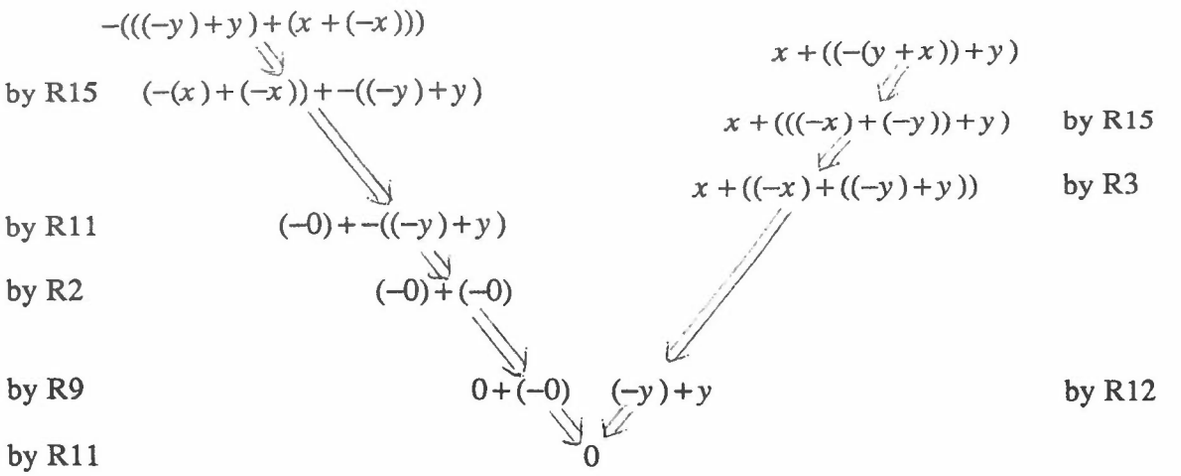
## 9. THE KNUTH-BENDIX ALGORITHM AND THEOREM PROVING

In the case where a finite canonical set of rules can be generated, the importance to equational reasoning is clear: an efficient decision procedure is found for solving the identity problem. A possible strategy for proving theorems is in two parts. Firstly, the given axioms are used to find a canonical set of rewrite rules (if possible). Secondly, new equations are shown to be theorems by reducing both sides to normal form. If the normal forms are the same, the theorem is shown to be a consequence of the given axioms; if different, the theorem is proven false.

The complete set of rules in Figure 12 can be used to prove, for instance, that

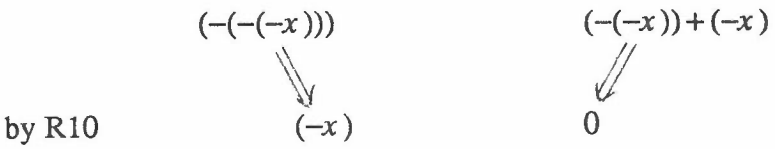
$$-(((-y)+y)+(x+(-x))) = x+((-(y+x))+y).$$

The rewrite rules are used to find the normal forms of both expressions :-




Since the normal forms are the same, the theorem has been shown to be a consequence of the given axioms after only 9 successful rule applications. Note that, due to confluence, the normal forms found are independent of the order of rule application.

Here is an attempt to prove the theorem $(-(-(-x))) = (-(-x))+(-x)$ :-




The different normal forms indicate that the theorem is not a consequence of the given axioms.

Where, however, the canonical set is infinite, we have no decision procedure. At best we can use the completion algorithm as a semi-decision procedure, because, given sufficient time and space, it is possible to prove any valid identity by generating a sufficient number of rules. That this is a complete process has been shown by[Hue81]. The invalidity of an identity can never be proven.

A major problem is that the ''proof'' procedure is not goal oriented. The generation of critical pairs is based on the rules formed, and not motivated by the identity we are seeking to prove. For this reason, the completion algorithm may not represent an efficient proof method.

Interesting comparisons have been made [Kuc83] between the algebraic completion process and resolution theorem-proving, and recent work[Pau85a], [HsiDer83] has proposed ways of using superposition and the completion process to prove theorems in first-order predicate logic. At heart of these techniques is the realisation that any clause $P$ can be made into an equality axiom of the form $P = true$. These methods seem to be considerably more efficient than resolution in many cases, especially where user defined functions and equality are imbedded in the logic. Two types of proof seem to be possible: constructive proofs in which the completion process is used in an attempt to generate the required result, and proofs by refutation in which the negation of the desired clause is assumed ($C = false$) and included in the completion process in an attempt to generate a contradiction ($true = false$). The advantage of the constructive proof is that the equational theory is not disturbed by the proving process, and further proofs can be attempted without having to repeat the work already done. By contrast, proof by refutation, in effect, destroys the theory by generating consequences of a false assumption, and every proof must recommence from the start; however, the proof is to a certain extent goal oriented, and experience in[Pau85a] suggests that contradictions are found very quickly if the theorem to be proved is true. Both methods are likely to behave in an infinitary manner if the theorem to be proved is false.

It is the implicit use of the properties of equality that make the Knuth-Bendix method of superposition eminently more suitable for reasoning about equality than methods that rely on resolution by unification, in which the axioms of equality, and in particular those describing the well-definedness of every function and predicate symbol used, must be explicitly stated.

## 10. REFERENCES

[DerMan79]. N. Dershowitz and Z. Manna, ''Proving Termination with Multiset Orderings,'' *Com. of the ACM* 22(8), pp. 465-476 (1979).

[Der82]. N. Dershowitz, ''Orderings for Term Rewriting Systems,'' *J. of Theoretical Computer Science* **17**, pp. 279-301 (1982).

[Dic87]. A. J. J. Dick, ''Order-Sorted Equational Reasoning and Rewrite Systems,'' PhD Thesis (to appear), Imperial College, London (1987).

[Gog78]. J. A. Goguen, ''Order sorted algebra,'' Tech. Report UCLA Computer Science Dept., Semantics and Theory of Computation, Report No. 14 (1978).

[HsiDer83]. J. Hsiang and N. Dershowitz, ''Rewrite Methods for Clausal and Non-clausal Theorem Proving,'' pp. 331-346 in *Proc. 10th ICALP* (July 1983).

[Heu80a]. G. Huet, ''Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems,'' *Journal of the ACM* **27** (4), pp. 797-821 (Oct. 1980 ).

[HueOpp80b]. G. Huet and D. C. Oppen, ''Equations and Rewrite Rules - A Survey,'' STAN-CS-80-785 (1980).

[Hue81]. G. Huet, ''A Complete Proof of Correctness of the Knuth-Bendix Completion Algorithm,'' *J. of Computer and System Science* **23**(1), pp. 11-21 (Aug. 1981).

[KnuBen70]. D. E. Knuth and P. B. Bendix, ''Simple Word Problems in Universal Algrebras,'' pp. 263-297 in *Computational Problems in Abstract Algebra*, ed. J. Leech , Pergamon Press (1970).

[Kuc83]. W. Kuchlin, ''A Theorem-proving Approach to the Knuth-Bendix Completion Algorithm,'' *Lecture Notes in Computer Science* **144** (1983).

[LanBal77a]. D. S. Lankford and A. M. Ballantyne, ''Decision procedures for simple equational theories with permutative axioms: Complete sets of permutative reductions,'' Rep. ATP-37, Univ. of Texas, Austin: Dep. Math. Comp. Sci. (1977).

[LanBal79]. D. S. Lankford, ''Some new approaches to the theory and applications of conditional term rewriting systems,'' Rep. MTP-6, Louisiana Tech. Univ., Ruston, Math. Dept. (1979).

[New42]. M. H. A. Newman, ''On Theories with a Combinatorial Definition of 'Equivalence','' *Annals of Mathematics* **43,2** , pp. 223-243 (1942 ).

[Pau85a]. E. Paul, ''On Solving the Equality Problem in Theories Defined by Horn Clauses,'' Proc. of EUROCAL'85, Linz, Austria, Springer Verlag LNCS Vol. 203 (Apr. 1985).

[PetSti81]. G. E. Peterson and M. E. Stickel, ''Complete Sets of Reductions for Some Equational Theories,'' *Journal of the ACM* **28**(2), pp. 233-264 (1981).

[Rem82]. J-L. Remy, ''Etude des Systemes de Reecriture Conditionnels et Applications aux Types Abstraits Algebriques,'' Thesis, Centre de Recherche en Informatique de Nancy, Nancy, France (July 1982).

[Ric83]. M. M. Richter, ''Complete and Incomplete Systems of Reductions,'' Informatik Fachlerichte 57, Springer GI-12 Jehreshagung (1983).

[Rob65]. J. A. Robinson, ''A Machine-Oriented Logic Based on the Resolution Principle,'' *Journal of the ACM* **12** , pp. 32-41 (1965).