CANONICALIZATION AND DEMODULATION

by

Robert L. Veroff

Applied Mathematics Division

February 1981
TABLE OF CONTENTS

ABSTRACT.................................................................5

INTRODUCTION............................................................6

I. $X^3 = X$ PROBLEM.......................................................7

II. CANONICALIZATION.....................................................9
    A. MOTIVATION.........................................................9
    B. IMPLEMENTATION.................................................10
    C. COMMENTS........................................................12

III. ADDED LEMMAS.....................................................13

IV. GROUPING SYMBOL..................................................14
    A. MOTIVATION.......................................................14
    B. IMPLEMENTATION.................................................14

V. SUBTRACTION........................................................16
    A. MOTIVATION.......................................................16
    B. IMPLEMENTATION.................................................17
    C. COMMENTS........................................................20

VI. RECOGNIZING CYCLIC EXPRESSIONS..............................20
    A. MOTIVATION.......................................................21
    B. IMPLEMENTATION.................................................21

VII. USING THE DENIAL................................................21
    A. GENERAL MOTIVATION............................................21
    B. FIRST METHOD..................................................21
    C. SECOND METHOD................................................22
    D. COMMENTS........................................................22

CONCLUSION.............................................................22

APPENDIX A. TERM WEIGHTING.........................................23

APPENDIX B. LEXICAL ORDERING OF TERMS...........................25
CANONICALIZATION AND DEMODULATION

by

Robert L. Veroff

ABSTRACT

This paper discusses mechanisms that were developed for the Argonne National Laboratory - Northern Illinois University theorem proving system. By defining special input clauses and demodulators, it is possible to simulate mathematical processes such as canonicalization of polynomials with no special programming. The mechanisms presented in this paper resulted from a study of the $X^3 = X$ problem in ring theory. The use of the mechanisms allowed this problem to be solved for the first time by our automated theorem proving system.
INTRODUCTION

A general goal in the development of an automated theorem proving system is the design and implementation of algorithms that will increase the capabilities of the system without significantly increasing the demands for computer resources (time and/or memory). Increased capabilities imply being able to solve problems that were previously unsolvable by the automated system. Included would be problems considered difficult because of the complex nature of the axiom set, such as set theory or certain database problems, and problems considered difficult because of the length of the proof or the nature of the inferencer required in the proof, such as the \( X^3 = X \) problem in ring theory:

**Theorem:** In a ring, if for all \( X \), \( X^3 = X \), then the ring is commutative.

The \( X^3 = X \) has been considered to be a benchmark problem in the field [1]. It is of interest because it has a relatively long proof (by current automated theorem proving standards) which requires some rather cumbersome polynomial manipulations.

This paper describes mechanisms that were developed for the Argonne National Laboratory - Northern Illinois University (ANL-NIU) theorem proving system during the study of the \( X^3 = X \) problem. These involve the addition of special input clauses and demodulators that induce some canonicalization of the clause space and allow the simulation of some processes that might otherwise require special programming.

The mechanisms are general in that they simulate common mathematical processes which are applicable to wide classes of problems in abstract algebras. Although the mechanisms are presented in the context of the ANL-NIU theorem proving system, the major features of the program that are required could easily be incorporated into most resolution and paramodulation based systems that have demodulation.

The paper is divided into seven sections. Section I introduces the setting (terminology, conventions, etc.) for the remainder of the paper. In particular, the \( X^3 = X \) problem is discussed in detail. Sections II through IV introduce mechanisms for canonicalizing and simplifying a clause space. Sections V through VII introduce mechanisms that aid the proof search.

Four appendices are also provided. Appendices A, B and C review, respectively, the notions of weighting, lexical ordering, and demodulation in the context of the ANL-NIU theorem proving system. Appendix D contains the statement and proof of a theorem about the canonicalization discussed in Section II.
1. \( X^2 = X \) PROBLEM

Because the mechanisms presented in this paper were developed during a study of the \( X^3 = X \) problem, we first present the mathematical formulation and proof of the problem, followed by the corresponding clause formulation.

Let \( X, Y, \) and \( Z \) be arbitrary elements of a ring, and let \( 0 \) denote the ring identity. Then the ring axioms are as follows:

1. \( X+Y = Y+X \) (commutativity of addition)
2. \( (X+Y)+Z = X+(Y+Z) \) (associativity of addition)
3. \( (XY)Z = X(YZ) \) (associativity of multiplication)
4. \( (X+Y)Z = XZ+YZ \) (right distributivity)
5. \( X(Y+Z) = XY+XZ \) (left distributivity)
6. \( X+0 = X \) (right identity)
7. \( 0+X = X \) (left identity)
8. \( X+(-X) = 0 \) (right inverse)
9. \( (-X)+X = 0 \) (left inverse)

Then the \( X^3 = X \) theorem is

**Theorem:** In a ring, \( XXX = X \) implies \( XY = YX \) (or \( XY - YX = 0 \)).

The proof is usually presented by way of the following three lemmas:

**Lemma 1:** In a ring, \( XXX = X \) implies \( 6X = 0 \).

**Proof:**
(1.1) \( XXX = X \) (hypothesis)
(1.2) \( (X+Y)(X+Y)(X+Y) = (X+Y) \) (substitute)
(1.3) \( XXX = X \) (hypothesis)
(1.4) \( 6XXX = 0 \) (identify \( X \) with \( Y \))
(1.5) \( 6X = 0 \) (from hypothesis)

**Lemma 2:** In a ring, \( XXX = X \) implies \( 3XY + 3YX = 0 \).

**Proof:**
(2.1) \( XXX = X \) (hypothesis)
(2.2) \( (X+XX)(X+XX)(X+XX) = X+XX \) (substitute)
(2.3) \( 3X + 3XX = 0 \) (simplify)
(2.4) \( 3(X+Y) + 3(X+Y)(X+Y) = 0 \) (substitute)
(2.5) \( (3X+3XX) + (3Y+3YY) + (3XY+3YX) = 0 \) (simplify)
(2.6) \( 3XY + 3YX = 0 \) (simplify with 2.3)

**Lemma 3:** In a ring, \( XXX = X \) implies \( 2XY - 2YX = 0 \).

**Proof:**
(3.1) \( XXX = X \) (hypothesis)
(3.2) \( XXY + XYY + XYY + YXX + YXY + YYX = 0 \) (see 1.3)
(3.3) \( XXY + XYY - XYY - YXX - YYX - YYX = 0 \) (\( -X \) for \( X \))
(3.4) \( 2XYY + 2YXY + 2YXX = 0 \) (3.2 minus 3.3)
(3.5) \( 2XY + 2YXX + 2YXY = 0 \) (\( Y \) times 3.4 and hypothesis)
(3.6) \( 2XY + 2YXY + 2YXX = 0 \) (3.4 times \( Y \) and hypothesis)
(3.7) \( 2XY - 2YX = 0 \) (3.6 minus 3.5)
The proof of the theorem then easily follows.

**Proof of Theorem:**

1. \( XY + 5YX = 0 \) (Lemma 2 minus Lemma 3)
2. \( XY - YX = 0 \) (1 minus Lemma 1)

The mathematical statement of the problem must be given a clause formulation for the automated theorem proving system [2]. The clause formulation that we used for the \( X^3 = X \) problem is given below. Although the clause formulation of a problem, in general, is not unique, the following represents a straightforward translation of mathematical functions and relations to function and predicate symbols in the theorem proving language.

**Notation:**

1. For each statement, names beginning with W,X,Y,Z are variables.
2. All other names denote constants, functions, or predicates.
3. E denotes the ring identity.
4. \( \text{EQUAL}(X,Y) \) denotes the fact \( X=Y \) (equality predicate).
5. \( \text{ADD}(X,Y) \) denotes \( X+Y \) (addition) in a ring.
6. \( \text{MULT}(X,Y) \) denotes \( XY \) (multiplication) in a ring.
7. \( \text{INV}(X) \) denotes \(-X\) (additive inverse) in a ring.

For this paper, all clauses that are intended to be input clauses to the theorem proving system will be given in the form

\[ \text{CL} \ <\text{clause number}> <\text{clause}> ; \]

The clauses will often be referred to by their clause numbers to facilitate discussion.

With this notation, the basic ring axioms become (in clause form)

\[ \begin{align*}
\text{CL 1.} & \quad \text{EQUAL}(\text{ADD}(X,Y),\text{ADD}(Y,X)) ; \\
\text{CL 2.} & \quad \text{EQUAL}(\text{ADD}(\text{ADD}(X,Y),Z),\text{ADD}(X,\text{ADD}(Y,Z))) ; \\
\text{CL 3.} & \quad \text{EQUAL}(\text{MULT}(\text{MULT}(X,Y),Z),\text{MULT}(X,\text{MULT}(Y,Z))) ; \\
\text{CL 4.} & \quad \text{EQUAL}(\text{MULT}(\text{ADD}(X,Y),Z),\text{ADD}(\text{MULT}(X,Z),\text{MULT}(Y,Z))) ; \\
\text{CL 5.} & \quad \text{EQUAL}(\text{MULT}(X,\text{ADD}(Y,Z)),\text{ADD}(\text{MULT}(X,Y),\text{MULT}(X,Z))) ; \\
\text{CL 6.} & \quad \text{EQUAL}(\text{ADD}(X,E),X) ; \\
\text{CL 7.} & \quad \text{EQUAL}(\text{ADD}(E,X),X) ; \\
\text{CL 8.} & \quad \text{EQUAL}(\text{ADD}(\text{INV}(X),X),E) ; \\
\text{CL 9.} & \quad \text{EQUAL}(\text{ADD}(X,\text{INV}(X)),E) ; \\
\text{CL 10.} & \quad \text{EQUAL}(X,X) \ (\text{for closure}) ; \\
\end{align*} \]

From the clause formulation just described, the \( X^3 = X \) theorem becomes

**Theorem:** In a ring, \( \text{EQUAL}(\text{MULT}(X,\text{MULT}(X,X)),X) \) implies

\[ \text{EQUAL}(\text{MULT}(X,Y),\text{MULT}(Y,X)) \]

(or \( \text{EQUAL}(\text{ADD}(\text{MULT}(X,Y),\text{INV}(\text{MULT}(Y,X))),E) \).)

The corresponding input to the theorem proving system (oriented towards proof by contradiction [2]) would be
**II. CANONICALIZATION**

This section describes mechanisms for partially canonicalizing a clause space. Subsection A describes a canonical form. Subsection B illustrates one way to automate the canonicalization process with demodulation. Subsection C contains some general comments and observations.

A. Motivation

The use of canonical forms can greatly reduce the size of a clause space without significantly blocking the derivation of desired clauses. For example, saving the facts \(A+B+C=E, A+C+B=E, B+A+C\ldots\) etc. all in the canonical form \(A+B+C=E\) clearly has its advantages. Several different canonicalizations are implemented in the \(X^3 = X\) experiments.

In choosing canonical forms for the \(X^3 = X\) experiments, we considered especially ease of manipulation of the canonical forms as well as ease of implementation of the canonicalization process. The chosen canonical forms emphasize the notion of a polynomial as a sum of (possibly negative) products. The general canonical form for a polynomial was chosen to be fully distributed (sum of products) and fully right associated with respect to addition and multiplication. To
have a unique canonical form for a polynomial, some canonical ordering of the summands of the polynomial is needed. Under the assumption that the space of all terms is well ordered with respect to some complexity measure, we chose the general canonical form for a polynomial such that less complex terms would always be to the left of more complex terms.

Another possible canonicalization would be to keep all polynomials in the form \( \text{EQUAL}(\text{poly}, E) \) (all terms on one side of the equality) by subtracting from both sides of an equality whenever necessary. Experience shows, however, that it is not desirable to completely eliminate the "nonzero" polynomial (non canonical) form. Polynomials with exactly one sum (two terms) are often more useful as demodulators or as "from" clauses in paramodulation in the form \( \text{EQUAL}(t_1, t_2) \) than in the form \( \text{EQUAL}(\text{ADD}(t_1, \text{INV}(t_2)), E) \). For example, \( \text{EQUAL}(\text{MULT}(X, \text{MULT}(X, X)), X) \) is generally more applicable than \( \text{EQUAL}(\text{ADD}(\text{MULT}(X, \text{MULT}(X, X)), \text{INV}(X)), E) \).

Another reason for not eliminating the "nonzero" polynomial form is its use as an intermediate clause. That is, the generation of a "nonzero" polynomial followed by a cancellation step is often the most effective way of generating a useful lemma. For example, say the special hypothesis for some problem in ring theory is \( \text{EQUAL}(\text{MULT}(X, X), X) \). Then the lemma \( \text{EQUAL}(\text{ADD}(\text{MULT}(X, Y), \text{MULT}(Y, X)), E) \) is found by paramodulating the special hypothesis into one of the distributive laws (clauses 4 and 5 above) and then cancelling \( \text{ADD}(X, Y) \) from both sides of the equality (see Section V on subtraction).

Therefore, rather than automatically keeping all polynomials in the "zero" polynomial form, a preference will be created for "zero" polynomials in the proof search strategy. That is, "zero" polynomials will tend to be inspected (and used in an inference) before "nonzero" polynomials. As a result, polynomials that are in a canonical form are used almost exclusively, without completely eliminating the other forms from consideration. This procedure can be thought of as a form of canonicalization.

Other forms of canonicalization will be noted below as they are introduced.

B. Implementation

The theorem proving WEIGHT statement and DEPTH constant (in the ANL-NIU system) have been chosen so that clauses 1 through 9 (defined in Section I) become demodulators. (See Appendix A for a discussion on term weighting, and Appendix C for a discussion on the demodulation process.) In particular, clause 1 is assumed to be a DEPTH=0 demodulator and clauses 2 through 9 are assumed not to be DEPTH=0 demodulators.

Because the distributivity axioms (clauses 4 and 5) are chosen to be demodulators, canonicalization with respect to distributivity is now achieved by always multiplying out products.

Canonicalization with respect to associativity is captured with clauses 2 and 3 above, which are demodulators. Note that these two
demodulators imply that all polynomials will be right associated with respect to addition and multiplication at all times.

Canonicalization with respect to commutativity of addition is more complicated. Recall that EQUAL(ADD(X,Y),ADD(Y,X)) is a DEPTH=O demodulator. Unlike other demodulators, DEPTH=0 demodulators are not applied unconditionally. A DEPTH=0 demodulator is applied to term \( t_1 \) to produce term \( t_2 \) if and only if \( t_2 \) is lexically earlier than \( t_1 \).

The lexical ordering of symbols and terms is determined by the DEFINE statement and by the function LEXPFVAR (in the ANL-NIU system). (See Appendix B for a discussion of the lexical ordering of terms and Appendix C for a discussion of the demodulation process.) For simplicity, the value for LEXPFVAR is assumed to be small. That is, variables will always be the lexically earliest terms (and \( V_i \) will be earlier than \( V_j \) for \( i < j \)).

In view of the above, it becomes evident that the convention of right associating polynomials is chosen with good reason. Say polynomials are chosen to be left associated (by reversing the order of the arguments of the equality unit that represents associativity of addition and redefining the weights of the arguments so that the new clause becomes a demodulator), and consider the two terms:

1. ADD(X,ADD(X,X))
2. ADD(ADD(X,X),X).

The first term would demodulate to the second by an application of associativity. But then because \( X < \text{ADD} \) in the lexical ordering (single variables are the earliest), it follows that the second term would demodulate to the first by an application of commutativity of addition. This particular demodulation loop is avoided by choosing to canonicalize with right association.

The symbol ADD must be late in the lexical ordering of symbols for any applications of commutativity of addition to take place (other than between the two innermost terms of a polynomial). Note, for example, that the term \( \text{ADD}(t_1,\text{ADD}(t_2,t_3)) \) can demodulate to the term \( \text{ADD}(\text{ADD}(t_2,t_3),t_1) \) only if the symbol ADD is lexically earlier than the term \( t_1 \). There is, however, a serious complication unless the symbol ADD is very late in the lexical ordering of symbols. Consider the two terms:

1. \( \text{ADD}(t_1,\text{ADD}(t_1,t_1)) \)
2. \( \text{ADD}(\text{ADD}(t_1,t_1),t_1) \),

where \( \text{ADD} \) is lexically earlier than \( t_1 \). It follows that the first term demodulates to the second by an application of commutativity, and the second demodulates to the first by the application of associativity that right associates. Thus we have a demodulation loop.

This complication is avoided by adding the "cycle" demodulator,

\[ \text{CL 11. EQUAL(ADD(X,ADD(Y,Z)),ADD(Y,ADD(X,Z)))} \]
which is a DEPTH=0 demodulator. The "cycle" demodulator has the effect of bubbling terms through the polynomial without disturbing the association. Because with the "cycle" demodulator, \( \text{EQUAL(ADD(X,Y),ADD(Y,X))} \) need only be applied to the innermost terms of a polynomial for effective implementation of canonicalization with respect to commutativity of addition, the symbol \( \text{ADD} \) can be late in the lexical ordering of symbols, and the looping difficulty is avoided. (See the example that follows.)

In addition to solving the looping problem above, the "cycle" demodulator provides an effective shortcut to a potentially tedious problem. For example, consider the term \( \text{ADD(t2,ADD(t1,ADD(t3,t4)))} \) where \( t_1 < \text{ADD} < t_2 < t_3 < t_4 \) in the lexical ordering. The following sequence of demodulators would be generated in the no-cycle implementation (if we assume that the looping problem described above has been dealt with in some other way):

<table>
<thead>
<tr>
<th>Demodulant</th>
<th>Demodulator Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD(ADD(t1,ADD(t3,t4)),t2)</td>
<td>commutativity (because ADD &lt; t2)</td>
</tr>
<tr>
<td>ADD(t1,ADD(ADD(t3,t4),t2))</td>
<td>associativity</td>
</tr>
<tr>
<td>ADD(t1 ADD(t3,ADD(t4,t2)))</td>
<td>associativity</td>
</tr>
<tr>
<td>ADD(t1,ADD(t3,ADD(t2,t4)))</td>
<td>commutativity (because t2 &lt; t4)</td>
</tr>
<tr>
<td>ADD(t1,ADD(ADD(t2,t4),t3))</td>
<td>commutativity (because ADD &lt; t3)</td>
</tr>
<tr>
<td>ADD(t1,ADD(t2,ADD(t4,t3)))</td>
<td>associativity</td>
</tr>
<tr>
<td>ADD(t1,ADD(t2,ADD(t3,t4)))</td>
<td>commutativity (because t3 &lt; t4)</td>
</tr>
</tbody>
</table>

The "cycle" demodulator generates the last demodulator in one step.

A negated term will be considered more complex than a term that is not negated. Therefore, to preserve the notion of keeping less complex terms to the left of more complex terms, \( \text{MULT} \) is chosen to be lexically earlier than \( \text{INV} \). With this ordering, the term \( \text{ADD(INV(MULT(t1,t2))),MULT(t3,t4))} \), for example, will demodulate to the preferred canonical form, \( \text{ADD(MULT(t3,t4),INV(MULT(t1,t2)))} \).

Using the "cycle" demodulator and having variables \( < \) constants \( < \) \( \text{MULT} < \text{INV} < \text{ADD} \) in the lexical order ensure that polynomials will be "nicely" canonicalized. Note that because of right associativity, single variables will always be the outermost terms in the canonical form.

Appendix D presents a proof that the canonicalization process just described always terminates and produces a unique canonical form. Because the interaction between sets of demodulators can be very subtle and complex to analyze, the proof is given in the context of a simple clause space (no other demodulators). The more general proof has not been formalized.

C. Comments

Note that an inference can conceivably be blocked by the demodulation of terms to a canonical form. Consider, for example, the associative law, \( \text{EQUAL(ADD(ADD(X,Y),Z),ADD(X,ADD(Y,Z)))} \) (clause 2). A
desirable lemma might be found by paramodulating with some equality into the \( \text{ADD}(Y,Z) \) term of this clause, and then paramodulating into the resulting \( \text{ADD}(X,Y) \) term with the left inverse law, \( \text{EQUAL}(\text{ADD}(\text{INV}(X),X),E) \) (clause 8). Because this associative law is a demodulator that forces all polynomials to be right associated with respect to addition, the second paramodulation would be blocked by the application of the demodulator to the first result. That is, the term corresponding to the \( \text{ADD}(\text{ADD}(X,Y),Z) \) term after the first paramodulation would be demodulated to a term corresponding to \( \text{ADD}(X,\text{ADD}(Y,Z)) \). The trade-off between the benefits of canonicalization and the harm of blocked inferences has not been completely analyzed.

A new inference rule, hyper-paramodulation, has been developed that may allow some canonicalization while avoiding some of the corresponding undesirable side effects [7].

The above discussion has introduced the notion of using the demodulation process to canonicalize clauses. The classical function of demodulation, however, has been to simplify terms. This suggests that there are two types of demodulators: those that canonicalize and those that simplify. The distinction between these two types is not well defined. The terminology is used only to help motivate the usefulness of certain input clauses. The canonicalizers tend to be the demodulators which have little effect on the complexity of terms. These are often demodulators in which the ordering of the arguments of the equality literal (and therefore the order of the demodulation) is chosen by some convention (possibly arbitrary). The distributive laws as they are written above, for example, are canonicalizers. Demodulators such as \( \text{EQUAL}(\text{ADD}(X,E),X) \), however, are clearly simplifiers.

In addition to clauses 1 through 10 above (representing the ring axioms), there are several different sets of input clauses grouped for special purposes described in the following sections. Some of the clauses are axioms that can be used both in inference rules and as demodulators. Some, however, are highly special demodulators that are added to lists that cannot be used in an inference.

### III. ADDED LEMMAS

Sometimes the addition of certain simple, commonly known lemmas to the set of input clauses can increase efficiency:

CL 12. \( \text{EQUAL}(\text{INV}(\text{INV}(X)),X) \);  
CL 13. \( \text{EQUAL}(\text{INV}(E),E) \);  
CL 14. \( \text{EQUAL}(\text{MULT}(E,X),E) \);  
CL 15. \( \text{EQUAL}(\text{MULT}(X,E),E) \);  
CL 16. \( \text{EQUAL}(\text{MULT}(X,\text{INV}(Y)),\text{INV}(\text{MULT}(X,Y))) \);  
CL 17. \( \text{EQUAL}(\text{MULT}(\text{INV}(X),Y),\text{INV}(\text{MULT}(X,Y))) \);  
CL 18. \( \text{EQUAL}(\text{INV}(\text{ADD}(X,Y)),\text{ADD}(\text{INV}(X),\text{INV}(Y))) \);  

With appropriate choice of the theorem proving WEIGHT statement and DEPTH constant (in the ANL-NIU) system, all of these clauses will become
demodulators. It is often beneficial to make these clauses available for use in inferences as well. Note that clauses 16, 17 and 18 are canonicalizers in the sense that they act more as rewrite rules than as simplifiers. The choice of the canonical form (order of the arguments of the equality literal) is consistent with the sum of products orientation of polynomials. That is, negation (additive inverse) is distributed across addition (\(\text{ADD}(\text{INV}(X),\text{INV}(Y)) \)) rather than \(\text{INV}(\text{ADD}(X,Y))\)) and is collected from multiplication (\(\text{INV}(\text{MULT}(X,Y))\)) rather than \(\text{MULT}(\text{INV}(X),Y)\) or \(\text{MULT}(X,\text{INV}(Y))\)).

IV. GROUPING SYMBOL

This section introduces the notion of a grouping symbol. Subsection A describes the need for such a symbol, and subsection B describes a mechanism for implementing the symbol with special functions and demodulators.

A. Motivation

To aid the manipulation of complicated polynomials, a mechanism to recognize and combine like terms of a polynomial was desired. That is, a mechanism was needed that would associate an integral coefficient with the terms of a polynomial. One effect would be that polynomials could be written in a condensed form. For example, \(A+A+A+A+B+B\) could be written as \(4A+2B\). Another, more significant effect concerns the subtraction of polynomials. This effect will be discussed in the next section, when the implementation of a subtraction algorithm is presented.

B. Implementation

A technique for integrating integer arithmetic into the theorem proving system is being developed, but the current implementation uses a strictly functional representation.

Define:

- \(H\) - a two place function denoting the grouping symbol
- \(S\) - a unary function denoting the successor function
- \(\text{CO}\) - a constant denoting the integer 0
- \(\text{IADD}\) - a two place function denoting integer addition

Then \(H(\text{term } t, \text{ integer } n)\) denotes \(n\) occurrences of term \(t\).

In other words, \(3AB\) is denoted \(H(\text{MULT}(A,B),S(S(\text{CO})))\)).

Several clauses are needed to implement this strategy. The theorem proving \(\text{WEIGHT}\) statement and \(\text{DEPTH}\) constant will be chosen so that all of the clauses become demodulators. Because these clauses are needed only to implement the grouping symbol strategy, the clauses will be put on special lists so that they are not available for use in inferences.
The following clauses define the grouping symbol:

CL 19. \( \text{EQUAL}(\text{ADD}(X,X),\text{H}(X,\text{S}(\text{S}(\text{CO})))) \);
CL 20. \( \text{EQUAL}(\text{ADD}(X,\text{ADD}(X,Y)),\text{ADD}(\text{H}(X,\text{S}(\text{S}(\text{CO}))),Y)) \);
CL 21. \( \text{EQUAL}(\text{ADD}(\text{H}(X,Y),\text{H}(X,Z)),\text{H}(X,\text{IADD}(Y,Z))) \);
CL 22. \( \text{EQUAL}(\text{ADD}(\text{H}(X,Y),\text{ADD}(\text{H}(X,Z),V)),\text{ADD}(\text{H}(X,\text{IADD}(Y,Z)),V)) \);
CL 23. \( \text{EQUAL}(\text{ADD}(X,\text{H}(X,Y)),\text{H}(X,\text{S}(Y))) \);
CL 24. \( \text{EQUAL}(\text{ADD}(X,\text{ADD}(\text{H}(X,Y),Z)),\text{ADD}(\text{H}(X,\text{S}(Y)),Z)) \);

The following clauses define the integer arithmetic necessary to implement the grouping symbol strategy:

CL 25. \( \text{EQUAL}(\text{IADD}(X,\text{CO}),X) \);
CL 26. \( \text{EQUAL}(\text{IADD}(\text{CO},X),X) \);
CL 27. \( \text{EQUAL}(\text{IADD}(X,\text{S}(Y)),\text{S}(\text{IADD}(X,Y))) \);
CL 28. \( \text{EQUAL}(\text{IADD}(\text{S}(X),Y),\text{S}(\text{IADD}(X,Y))) \);

The following clauses are simplifiers and canonicalizers. Note that the canonicalizations implemented by clauses 36 through 39 are consistent in spirit with the canonicalizations described in Section II.

CL 29. \( \text{EQUAL}(\text{H}(E,X),E) \);
CL 30. \( \text{EQUAL}(\text{H}(X,\text{CO}),E) \);
CL 31. \( \text{EQUAL}(\text{H}(X,\text{S}(\text{CO})),X) \);
CL 32. \( \text{EQUAL}(\text{ADD}(\text{INV}(X),\text{H}(X,\text{S}(Y))),\text{H}(X,Y)) \);
CL 33. \( \text{EQUAL}(\text{ADD}(\text{INV}(X),\text{S}(Y)),\text{H}(\text{INV}(X),Y)) \);
CL 34. \( \text{EQUAL}(\text{ADD}(\text{INV}(X),\text{S}(Y)),\text{H}(X,\text{S}(Z))),\text{ADD}(\text{H}(\text{INV}(X),Y),\text{H}(X,Z))) \);
CL 35. \( \text{EQUAL}(\text{ADD}(\text{INV}(X),\text{S}(Y)),\text{ADD}(\text{H}(X,\text{S}(Z))),\text{ADD}(\text{H}(X,Y),\text{H}(\text{INV}(X),Z))) \);
CL 36. \( \text{EQUAL}(\text{INV}(\text{H}(X,Y)),\text{H}(\text{INV}(X),Y)) \);
CL 37. \( \text{EQUAL}(\text{MULT}(X,\text{H}(Y,Z)),\text{H}(\text{MULT}(X,Y),Z)) \);
CL 38. \( \text{EQUAL}(\text{MULT}(\text{H}(X,Z),\text{Y}),\text{H}(\text{MULT}(X,Y),Z)) \);
CL 39. \( \text{EQUAL}(\text{H}(\text{ADD}(X,Y),Z),\text{ADD}(\text{H}(X,Z),\text{H}(Y,Z))) \);

Note that because terms may occur with or without the grouping symbol (1AB is represented as AB - see clause 31), it is often useful to add special demodulators to account for the different combinations that are possible. For example, clauses 32 through 35 are special corollaries to the lemmas \( \text{EQUAL}(\text{ADD}(X,\text{INV}(X)),E) \) and \( \text{EQUAL}(\text{ADD}(\text{INV}(X),X),E) \).

Note also that the grouping process does not necessarily group all like terms. Like terms are recognized only if they come next to each other in the polynomial. For example, the like terms of \( \text{ADD}(t_1,\text{ADD}(t_2,t_1)) \) would not be grouped by the above demodulators. It might be useful to add some special grouping demodulators to account for the different possible distances between like terms.

The set of clauses chosen above (clauses 19 through 39) was sufficient to solve the \( X^3 = X \) problem.
V. SUBTRACTION

This section introduces a mechanism for subtracting polynomials. Subsection A describes a restricted algorithm for polynomial subtraction. Subsection B describes an implementation of the algorithm with special clauses and demodulators. Subsection C contains some observations.

A. Motivation

The need for having a mechanism for subtracting polynomials should be clear. However, because the obvious implementations of polynomial subtraction produce a prohibitive number of generated clauses, a set of rules was developed to limit the application of subtraction as an inference.

One rule is to allow only subtractions between polynomials of the form $\text{EQUAL}(\text{poly}, E)$ (the preferred canonical form). Another rule is to subtract only polynomials that have a nontrivial like term (that cancels). These rules are reasonable because subtraction, with these restrictions, can in some sense now be thought of as a simplification process.

The rule about identifying like terms is worth emphasizing in the context of the theorem proving environment (resolution, unification, etc.). With no common term established between polynomials, no cancellation of terms would be possible because variables of one clause would remain distinct from the variables of the other clauses. For example, if the polynomial $XX = E$ were subtracted from the polynomial $XX + X = E$, the result would be $XX + X - YY = E$ and not the desired polynomial $X = E$.

When this rule about like terms was implemented, the effective decrease in the size of the clause space (and the corresponding increase in the efficiency of the theorem proving system) was significant, but not profound. We determined that terms in two polynomials were being considered the "same" (i.e., a subtraction took place) if the terms had a common instance. For example, the term $\text{MULT}(X, \text{MULT}(Y, Z))$ can be unified with the term $\text{MULT}(V, W)$ by letting $W$ take the value $\text{MULT}(Y, Z)$.

A restriction was therefore added requiring that two terms be considered like if and only if they really were the same. In other words, the unification of the two terms was non length expanding (allowing unifications of variables with constants and other variables only). This implementation turned out to be very effective and is now being used.

It does not suffice merely to identify and cancel a single like term in two polynomials to get the most benefit from the subtraction. To maximize the possibility of cancelling all like terms in the two polynomials, we compare the negation of each term of the subtracted polynomial with each term of the first polynomial for possible cancellation.
B. Implementation

Three sets of clauses are used to implement subtraction: the subtraction template (a hyper-resolution nucleus), a set of special predicates for identifying like terms in polynomials, and a set of special demodulators (not available for use in an inference) for "bubbling" the negated terms of one polynomial through the other. Bubbling is the process by which terms of the two relevant polynomials are compared for possible cancellation. See the example that follows clause 59 below.

It will be assumed in this discussion that the grouping symbol discussed in the previous section has been implemented and must be accounted for.

The obvious method for identifying like terms in polynomials would be to have a subtraction template for the i-th term of one polynomial identifying with the j-th term of another for every i and j. This is prohibitive, as there would be 64 templates when polynomials are restricted to being of order 8 or less, even without accounting for the possible variations of having terms with and without the grouping symbol.

A Q predicate has been added as a general term identifier. The original concept is due to Steve Winker [5]. The Q predicate is a three place predicate Q(p1,t,p2) that identifies term t as a term in polynomial p1 leaving residual p2 when it is cancelled. There are two Q units (clauses with exactly one literal) for each possible position (with respect to the outermost term) in a polynomial. One occurs with and one without the grouping symbol. For example, the Q units,

\[ Q(\text{ADD}(X,\text{ADD}(Y,Z)),Y,\text{ADD}(X,Z)) \]
\[ \text{and } Q(\text{ADD}(X,\text{ADD}(H(Y,S(W)),Z)),Y,\text{ADD}(X,\text{ADD}(H(Y,W),Z))), \]

capture all possible terms in the second position of an arbitrary polynomial. Note that the Z term may be an arbitrarily large polynomial itself. This is why positions are given with respect to the outermost term.

With this specification for term identification, there is further motivation for the grouping symbol described in the previous section. Consider the following two polynomials:

1. \( A + A + A = E \)
2. \( A + A + B = E. \)

The second polynomial can be subtracted from the first in six distinct ways to produce the polynomial \( A - B = E \) by identifying each occurrence of 'A' in one with an occurrence of 'A' in the other (because polynomials will be subtracted upon identification of a like term that cancels). Consider now, however, the corresponding "grouped" polynomials
3. \(3A = E\) 
and 4. \(2A + B = E\).

The second polynomial can be subtracted from the first in only one way by identifying the 'A' term in one polynomial with the 'A' term in the other.

Because a term of a polynomial might be a summand of the polynomial or the term of a grouping symbol that is the summand of the polynomial, two Q units must be input for each possible position in the polynomial. The effect is that there might be some duplication of effort even with the grouping symbol. Consider the two polynomials

1. \(3A + B = E\) 
and 2. \(3A + C = E\).

The second polynomial may be subtracted from the first to obtain \(B - C = E\) in two distinct ways, by either identifying the '3A' or the 'A' terms in the two polynomials as like terms.

The third argument of the Q predicate identifies the residual that remains when the term identified by the second argument is cancelled from the polynomial identified by the first argument. The use of the residual is twofold. First, because we subtract only polynomials when a common term cancels, why not eliminate the need for the cancellation by combining only the residuals from the start? Second, if \(\text{EQUAL}(\text{poly}, \text{term})\) should be a demodulator, then efforts to subtract \(\text{poly}\) (as opposed to a residual) would actually be blocked. (\(\text{poly1} - \text{poly} = \text{poly2}\) would demodulate to \(\text{poly1} - \text{term} = \text{poly2}\) before the subtraction could take place.)

Although the following set of Q units was sufficient for solving the \(X^3 = X\) problem, it is not intended to be a complete set. A user would, in general, have to decide about the size of polynomials he is willing to deal with.

CL 40. \(Q(\text{ADD}(X,V), X, V)\) 
CL 41. \(Q(\text{ADD}(X1,\text{ADD}(X,V)), X, \text{ADD}(X1, V))\) 
CL 42. \(Q(\text{ADD}(X1,\text{ADD}(X2,\text{ADD}(X,V))), X, \text{ADD}(X1, \text{ADD}(X2,V)))\) 
CL 43. \(Q(\text{ADD}(X1,\text{ADD}(X2,\text{ADD}(X3,\text{ADD}(X,V)))), X, \text{ADD}(X1,\text{ADD}(X2,\text{ADD}(X3,V))))\) 
CL 44. \(Q(\text{ADD}(H(X,S(Z)), X, H(X,Z)))\) 
CL 45. \(Q(\text{ADD}(H(X,S(Z)), V), X, \text{ADD}(H(X,Z), V))\) 
CL 46. \(Q(\text{ADD}(X1, \text{ADD}(H(X,S(Z)), V)), X, \text{ADD}(X1, \text{ADD}(H(X,Z), V)))\) 
CL 47. \(Q(\text{ADD}(X1, \text{ADD}(X2, \text{ADD}(H(X,S(Z)), V))), X, \text{ADD}(X1, \text{ADD}(X2, \text{ADD}(H(X,Z), V))))\)

The following demodulators permit the bubbling process:

CL 48. \(\text{EQUAL}(\text{ADD}(Y, \text{H1}(X)), \text{ADD}(Y, X))\) 
CL 49. \(\text{EQUAL}(\text{ADD}(\text{H1}(\text{INV}(X)), \text{ADD}(X, Y)), Y)\) 
CL 50. \(\text{EQUAL}(\text{ADD}(\text{H1}(\text{INV}(X)), \text{ADD}(\text{H}(X, S(V)), Y)), \text{ADD}(\text{H}(X, V), Y))\) 
CL 51. \(\text{EQUAL}(\text{ADD}(\text{H1}(\text{INV}(X), S(V))), \text{ADD}(X, Y)), \text{ADD}(\text{H}(\text{INV}(X), V), Y))\)
CL 52. EQUAL(ADD(H1(H(INV(X),S(V))),ADD(H(X,S(Z)),Y)),
ADD(H1(H(INV(X),V)),ADD(H(X,Z),Y)))
CL 53. EQUAL(ADD(ADD(H2(C),ADD(I2(W),Z)),?),
ADD(ADD(H2(C),Z),ADD(H1(INV(W)),Y)))
CL 54. EQUAL(ADD(ADD(H2(C),I2(W)),Y),ADD(H1(INV(W)),Y))
CL 55. EQUAL(I2(E),E)
CL 56. EQUAL(I2(ADD(X,Y)),ADD(I2(X),I2(Y)))
CL 57. EQUAL(ADD(H2(C),E),E)

The unary functions H1 and H2 are place holders that allow the
bubbling process to proceed without interference from extraneous
demodulations. The unary function I2 is a temporary negation symbol
that serves a similar purpose. More specifically, H2(C) is used to
"hold" the negated polynomial so that one term at a time can be bubbled
through the first polynomial. H1, by being late in the lexical
ordering, carries the term that is to be bubbled (with applications of
the "cycle" demodulator and commutativity of addition) against each term
of the first polynomial to the innermost term, where it is removed. I2
guarantees that each term of the negated polynomial will be selected for
bubbling. (It does not suffice to use the negation symbol, INV, for I2
because the demodulator EQUAL(INV(INV(X)),X) would interfere if the
negated polynomial had any negative terms.)

Clauses 48 through 51 reflect the four ways that the bubbling
process can terminate. Clause 52 cancels like terms that are opposite
in sign when they come next to each other in the bubbling process.
Clause 53 begins the bubbling of a new negated term that has been
"held". Clause 54 begins the bubbling of the last negated term.

Note that two I2 canonicalizers (clauses 55 and 56) have been
added. Although clause 55 is merely a convenient simplifier, clause 56
is absolutely necessary for the proper completion of the process.

In the special case that the residual from a subtracted polynomial
demodulates to the ring identity, E, clause 57 acts as a demodulator to
prevent clauses with the place holder, H2(C), from being kept.

The above discussion motivates the use of the following subtraction
and cancellation (which is just a special form of subtraction)
templates.

CL 58. ~EQUAL(W,E) ~EQUAL(V,E) ~Q(W,X,W1) ~Q(V,X,V1)
EQUAL(ADD(ADD(H2(C),I2(W1)),V1),E) ;
CL 59. ~EQUAL(W,V) ~Q(W,X,W1) ~Q(V,X,V1)
EQUAL(ADD(ADD(H2(C),I2(W1)),V1),E) ;

The unification restriction is implemented by not allowing length
increasing unifications on the satellites of the subtraction nucleus.
This option is available in the ANL-NIU theorem prover (See the
discussion of EQRD in [4]).
Example:

Given the two polynomials \((t_1 + (t_2 + t_3))\) and \((t_2 + t_5)\), the following sequence indicates (in algebraic form) the sequence of demodulants that will be generated by the bubbling process if the second polynomial is subtracted from the first:

1. \((- (t_2 + t_5)) + (t_1 + (t_2 + t_3))\)
2. \((- (t_2)) + (- (t_5)) + (t_1 + (t_2 + t_3))\)
3. \((- (t_5)) + ((- (t_2)) + (t_1 + (t_2 + t_3)))\)
4. \((- (t_5)) + (t_1 + ((- (t_2)) + (t_2 + t_3)))\)
5. \((- (t_5)) + (t_1 + t_3)\)
6. \((- (t_5)) + (t_1 + t_3)\)
7. \((t_1 + ((- (t_5)) + t_3))\)
8. \((t_1 + (t_3 + (- t_5)))\)
9. \((t_1 + (t_3 + (- t_5)))\)

In clause form, the same sequence becomes:

<table>
<thead>
<tr>
<th>Term</th>
<th>Demodulator Applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ADD(ADD(H_2(C), I_2(ADD(t_2,t_5))),</td>
<td></td>
</tr>
<tr>
<td>ADD(t_1,ADD(t_2,t_3)))</td>
<td>(from inference)</td>
</tr>
<tr>
<td>2. ADD(ADD(H_2(C),ADD(I_2(t_2),I_2(t_5))),</td>
<td></td>
</tr>
<tr>
<td>ADD(t_1,ADD(t_2,t_3)))</td>
<td>clause 56</td>
</tr>
<tr>
<td>3. ADD(ADD(H_2(C),I_2(t_5)),</td>
<td></td>
</tr>
<tr>
<td>ADD(H_1(INV(t_2)),ADD(t_1,ADD(t_2,t_3))))</td>
<td>clause 53</td>
</tr>
<tr>
<td>4. ADD(ADD(H_2(C),I_2(t_5)),</td>
<td></td>
</tr>
<tr>
<td>ADD(t_1,ADD(H_1(INV(t_2)),ADD(t_2,t_3))))</td>
<td>clause 11</td>
</tr>
<tr>
<td>5. ADD(ADD(H_2(C),I_2(t_5)),ADD(t_1,t_3))</td>
<td></td>
</tr>
<tr>
<td>6. ADD(H_1(INV(t_5)),ADD(t_1,t_3))</td>
<td>clause 49</td>
</tr>
<tr>
<td>7. ADD(t_1,ADD(H_1(INV(t_5)),t_3))</td>
<td></td>
</tr>
<tr>
<td>8. ADD(t_1,ADD(t_3,H_1(INV(t_5))))</td>
<td></td>
</tr>
<tr>
<td>9. ADD(t_1,ADD(t_3,INV(t_5)))</td>
<td></td>
</tr>
</tbody>
</table>

C. Comments

The demodulator that represents associativity of addition (clause 2) could conceivably apply to some of the terms above (1 through 5) and seriously interfere with the bubbling process. This interference has been avoided by proper choice of rules that determine which demodulator to apply when more than one can apply. See Appendix D for the rules for applying demodulators in the ANL-NIU system.

VI. RECOGNIZING CYCLIC EXPRESSIONS

In a ring, a cyclic expression is an expression of the form \(i \cdot t = E\), where \(i\) is an integer, \(t\) is some expression, and \(E\) is the ring identity.
In this section a mechanism for recognizing cyclic expressions is introduced. Motivation for the mechanism is presented in Subsection A, and an implementation is described in Subsection B.

A. Motivation

If the cyclic expression \( i^*t = E \) is derived for some integer \( i \) and some term \( t \), it is reasonable to expect to be able to demodulate the expression \( (i+j)^*t \) to \( j^*t \) for arbitrary integer \( j \). Note that it does not suffice to make the clause \( i^*t = E \) into a demodulator. For example, say the lemma \( \text{EQUAL}(H(A,S(S(S(CO)))),E) \) (3A = E) is generated and becomes a demodulator. Because of the way the S functions are nested, the term \( H(A,S(S(S(S(S(CO)))))) \) (5A) would not be simplified by this demodulator. It does, however, suffice to have the demodulator \( \text{EQUAL}(H(A,S(S(S(Z)))),H(A,Z)) \), as we do here.

B. Implementation

The following resolution nucleus allows the generation of the demodulator with the desired property:

\[
\text{CL 60. } \sim \text{EQUAL}(H(X,Y),E) \ \text{EQUAL}(H(X,IADD(Y,Z)),H(X,Z)) ;
\]

The \( IADD(Y,Z) \) term immediately demodulates to the appropriate \( S(S(...S(Z)...)) \) term (using clauses 25 through 28).

VII. USING THE DENIAL

Two mechanisms for using the denial of a theorem in the proof search are introduced here. Subsection A contains some general motivation. Subsections B and C describe the two mechanisms. Subsection D contains a final observation.

A. General Motivation

Proofs found by the theorem prover (at least when the equality formulation is stressed) tend to be completely "forward." That is, they proceed forward from the axioms and special hypothesis until a contradiction with the denial of the theorem is found. This procedure partially defeats one of the nice properties of proof by contradiction. It would seem much more efficient to allow the denial of the theorem to play an active role in the proof search. In other words, it seems desirable to have the theorem prover "back up" the denial and meet the "forward" proof in the middle.

B. First Method

Given that the conclusion of the theorem to be proved is of the form \( \text{EQUAL}(poly,E) \), then the denial is \( \sim \text{EQUAL}(poly',E) \), where \( poly' \) is a ground instance of \( poly \). It seems reasonable to "notice" occurrences of \( poly \) in the clauses of the "forward" proof search. This is done by adding the demodulator \( \text{EQUAL}(poly,\text{DENIAL}(X1,X2,...Xk)) \), where the \( X \)'s are
the distinct variables in poly. The denial of the theorem is then input
as \( \neg \text{EQUAL} (\text{DENIAL}(A_1,A_2,\ldots A_k),E) \), where the A's are distinct constants.
For example, if as in the \( X^3 = X \) problem we are trying to prove
\( \text{EQUAL}(\text{ADD}(\text{MULT}(X,Y),\text{INV}(\text{MULT}(Y,X))),E) \), then the following clauses are added:

\[
\begin{align*}
\text{CL P4.} & \quad \text{EQUAL}(\text{ADD}(\text{MULT}(X,Y),\text{INV}(\text{MULT}(Y,X))),\text{DENIAL}(X,Y)) \\
\text{CL P5.} & \quad \neg\text{EQUAL}(\text{DENIAL}(A,B),E) \\
\text{CL 61.} & \quad \text{EQUAL}(\text{DENIAL}(X,X),E)
\end{align*}
\]

The special clause numbers P4 and P5 have been used as a reminder
that these two clauses are direct consequences of the problem being
considered and are not part of the general axiom set. Clause P4 is the
demodulator that "notices" occurrences of poly as described above.
Clause P5 is the new denial of the theorem. Clause 61 is input in this
case to prevent (with demodulation and subsumption) the generation of
many useless clauses.

C. Second Method

The general fact that in a ring \( A = 0 \) and \( B \neq 0 \) implies \( A - B \neq 0 \)
is used to expand the set of inequalities that might clash against
equalities derived in the "forward" proof search.

The NEQUAL predicate has been introduced to allow the
implementation of this process by way of the subtraction mechanism
described in Section V. The following clauses define the NEQUAL
predicate, give the denial of the \( X^3 = X \) theorem in the NEQUAL form, and
give the relevant subtraction template.

\[
\begin{align*}
\text{CL 64.} & \quad \neg\text{NEQUAL}(X,X) \\
\text{CL 65.} & \quad \text{EQUAL}(X,Y) \quad \neg\text{NEQUAL}(X,Y) \\
\text{CL 66.} & \quad \text{NEQUAL}(\text{ADD}(\text{MULT}(A,B),\text{INV}(\text{MULT}(B,A))),E) \\
\text{CL 67.} & \quad \neg\text{NEQUAL}(W,E) \quad \neg\text{EQUAL}(V,E) \quad \neg\text{Q}(W,X,W1) \quad \neg\text{Q}(V,X,V1) \\
& \quad \text{NEQUAL}(\text{ADD}(\text{H2}(C),\text{I2}(W1)),V1),E)
\end{align*}
\]

D. Comments

While the first method has had some effect on efficiency, the
effect of the second method has been profound. Note that the first
method partially interferes with the second method (by demodulating
terms in the NEQUAL literals), indicating that both methods should not
be implemented in one experiment.

CONCLUSION

Each of the mechanisms described in this paper has been designed to
simulate some desired mathematical or logical operation on a clause
space. Mechanisms such as these can be the building blocks of a
powerful automated theorem proving system.
APPENDIX A: TERM WEIGHTING

Appendix A contains a formalization of the term weighting mechanism. In particular, a function WT is described that assigns an integer value (weight) to any term (well formed formula) that can occur in a given theorem proving run.

Definition A1. A weight template is defined recursively as a string of symbols \( F(s_1, s_2, \ldots, s_n) \) where \( F \) is an \( n \)-ary function symbol and the \( s_i \) are either weight templates or unsigned integer coefficients.

Examples: Weight Templates

\[
\begin{align*}
A \\
F(3,1) \\
F(1,G(1)) \\
G(F(2,A))
\end{align*}
\]

Definition A2. The generalization of a weight template is the term constructed from the template as follows: Each integer coefficient is replaced by a distinct variable. Note that generalizations are unique up to the names of the variables.

Examples: Generalizations

\[
\begin{align*}
A, \\
F(V0,V1), \\
F(V0,G(V1)), \\
G(F(V0,A))
\end{align*}
\]

Definition A3. A weight template matches a term \( T \) if \( T \) is an instance of the generalization of the template. Note that there is a direct correspondence between the integer coefficients of the weight template and subterms of \( T \).

Example: Matching and Correspondence of Coefficients

The weight template \( F(2,G(1)) \) matches any term \( T \) that is an instance of the term \( F(V0,G(V1)) \). The coefficient 2 corresponds to the subterm of \( T \) that would be an instance of the variable \( V0 \), and the coefficient 1 corresponds to the subterm of \( T \) that would be an instance of the variable \( V1 \).

The theorem proving WEIGHT statement consists of an ordered list of weight templates with unsigned integer offsets:

\[
\text{WEIGHT template+offset template+offset ... template+offset ;}
\]

The theorem proving parameter, VARWT, assigns an integer value (weight) to single variables. The parameter behaves like a function in the sense that each variable \( V_i \) may be assigned a different value.

Let \( T \) be an arbitrary term to be weighed by the WEIGHT statement. Then the weight of \( T \), denoted WT(\( T \)), is computed recursively as follows: If \( T \) is a single variable \( V_i \), then WT(\( T \)) = VARWT(\( V_i \)). Otherwise, the list of templates is searched backwards (last template tested first)
until a template is found that matches $T$. If no template is found, then $WT(T) = 0$. Otherwise let $i_1, i_2, \ldots i_k$ be the coefficients of the template, and let $t_1, t_2, \ldots t_k$ be the corresponding subterms of $T$. Then $WT(\text{term}) = i_1 \ast WT(t_1) + i_2 \ast WT(t_2) + \ldots + i_k \ast WT(t_k) + \text{the offset for the relevant weight template.}$

In the following examples, $A$ will denote a constant, $G$ will denote a function with one argument, and $F$ will denote a function with two arguments. Also, it will be assumed that $VARWT(V_i) = 1$ for all $i$.

Example: $\text{WEIGHT A+1 F(1,1)+1 G(1)+1 ;}$

This is one of the simplest and most common WEIGHT statements. In this example, $WT(T) =$ the number of symbols in $T$.

Example: $\text{WEIGHT A+2 F(1,5)+1 G(1)+2 G(G(1))+3 ;}$

\[
\begin{align*}
\text{WT( F(A,A) )} & \quad = \text{WT(A) + 5 * WT(A) + 1} \quad \text{(template F(1,5)+1)} \\
& \quad = 2 + 5 * 2 + 1 \quad \text{(A+2 twice)} \\
& \quad = 13 \\
\text{WT( G(F(A,V0) )} & \quad = \text{WT(F(A,V0)) + 2} \quad \text{(G(1)+2)} \\
& \quad = \text{WT(A) + 5 * WT(V0) + 1 + 2} \quad \text{(F(1,5)+1)} \\
& \quad = 2 + 5 * 1 + 1 + 2 \quad \text{(A+2 and VARWT(V0))} \\
& \quad = 10 \\
\text{WT( G(F(V0,A) )} & \quad = \text{WT(F(V0,A)) + 2} \quad \text{(G(1)+2)} \\
& \quad = \text{WT(V0) + 5 * WT(A) + 1 + 2} \quad \text{(F(1,5)+1)} \\
& \quad = 1 + 5 * 2 + 1 + 2 \quad \text{(VARWT(V0) and A+2)} \\
& \quad = 14 \\
\text{WT( G(G(F(V0,G(A)))) )} & \quad = \text{WT(G(F(V0,G(A)))) + 3} \quad \text{(G(G(1)+3)} \\
& \quad = \text{WT(F(V0,G(A)))+ 2 + 3} \quad \text{(G(1)+2)} \\
& \quad = \text{WT(V0) + 5 * WT(G(A)) + 1 + 2 + 3} \quad \text{(F(1 5)+1)} \\
& \quad = 1 + 5 * (WT(A) + 2) + 1 + 2 + 3 \quad \text{(VARWT(V0) and G(1)+2)} \\
& \quad = 1 + 5 * (2 + 1) + 1 + 2 + 3 \quad \text{(A+2)} \\
& \quad = 27
\end{align*}
\]
APPENDIX B: LEXICAL ORDERING OF TERMS

Appendix B contains a formalization of the mechanism for lexically ordering terms. In particular, a function LEX is described that assigns an integer value (lexical position) to any term (well formed formula) that can occur in a given theorem proving run.

Definition B1. A function definition is a string of symbols $F(f_1, f_2, \ldots, f_k) = A_n, L_m$ where the $f_i$ are n-ary functions which are assigned a lexical value $m$ (which is used to compute the function LEX described below). The function definition $F(\ldots) = A_n$ is equivalent to the function definition $F(\ldots) = A_n, L_0$.

Definition B2. A predicate definition is a string of symbols $P(p_1, p_2, \ldots, p_k) = A_n$ where the $p_i$ are n-ary predicates.

The theorem proving DEFINE statement consists of a list of function and predicate definitions:

DEFINE definition definition definition ... definition ;

The theorem proving parameter, LEXPFVAR, assigns an integer value (lexical position) to single variables.

Let the theorem prover alphabet consist of the function names defined in the DEFINE statement and a finite number of variable names, $V_0$ through $V_n$. Also include in the alphabet the special symbols '(', ')', and ','.

A word over the alphabet is a term (well formed formula) consisting of a finite string of "letters" from the alphabet.

The DEFINE statement and the LEXPFVAR parameter determine a total lexical order for the letters of the alphabet:

1. The letter defined by function or variable name $N_1$ is lexically earlier than the letter defined by $N_2$ if the lexical value assigned to $N_1$ by the relevant function definition in the DEFINE statement or by LEXPFVAR is strictly less than the corresponding value for $N_2$.

If these values are equal, then a distinction is made between three types of names: constants (functions with no arguments), mappings (functions with at least one argument), and variables.

2. Within the class of all names that are assigned the same lexical value, all constants are ordered earlier than all mappings which are ordered earlier than all variables.

3. Within the class of all names of constants that are assigned the same lexical value, the letter defined by the name $N_1$ is lexically earlier than the letter defined by the name $N_2$ if the name $N_1$ is alphabetically earlier than the name $N_2$ in the usual dictionary sense (with letters earlier than digits).
(4) Within the class of all names of mappings that are assigned the same lexical value, the letter defined by the name $N_1$ is lexically earlier than the letter defined by the name $N_2$ if the name $N_1$ is alphabetically earlier than the name $N_2$ in the usual dictionary sense (with letters earlier than digits).

(5) Within the class of all names of variables that are assigned the same lexical value, the letter defined by the name $V_i$ is lexically earlier than the letter defined by the name $V_j$ if the integer $i$ is less than the integer $j$.

(6) The special symbols are lexically later than all function and variable names, and ‘(' is earlier than ',' which is earlier than ')'.

Because the alphabet is totally ordered lexically, it follows that the space of all terms (which are words over the alphabet) is totally ordered lexically. That is, for all terms $t_1$ and $t_2$, either $t_1$ is earlier than $t_2$ or $t_2$ is earlier than $t_1$ or $t_1$ is identical to $t_2$.

**Lemma B1.** Given a complete lexical ordering (determined by the DEFINE statement and the LEXPFVAR parameter), there exists an integer valued function $LEX$ such that for any two terms $t_1$ and $t_2$

(1) $LEX(t_1) = LEX(t_2)$ iff $t_1$ is identical to $t_2$

and (2) $LEX(t_1) < LEX(t_2)$ iff $t_1$ is lexically earlier than $t_2$.

**Proof.** There is no infinite (lexically) descending chain of finite length words over a finite alphabet.
APPENDIX C: DEMODULATION

Appendix C describes the demodulation process. See [6] for a more formal treatment of the subject.

Definition C1. A clause is an equality unit if it satisfies all of the following:

(1) It is a unit clause (contains exactly one literal).

(2) The predicate of the literal represents a binary relation.

(3) The predicate name begins with the string of symbols 'EQUAL'.

Every equality unit that is added to the clause space is tested for possible addition to the list of demodulators. The selection mechanism is based on the theorem proving DEPTH parameter and on the WT function described in Appendix A.

Definition C2. Equality unit EQUAL...(t_1, t_2) is a demodulator if it satisfies the following:

(1) t_1 is not a variable.

(2) The relation WT(t_1) - WT(t_2) ≥ DEPTH is satisfied.

Definition C3. Demodulator EQUAL...(t_1, t_2) is a DEPTH=0 demodulator if it satisfies the relation, WT(t_1) - WT(t_2) = 0. No other demodulators are DEPTH=0 demodulators.

Definition C4. Demodulator EQUAL...(t_1, t_2) matches term T if T is an instance of t_1.

Definition C5. Let S = F(s_1, s_2, ..., s_n) and T = F(t_1, t_2, ..., t_n) be the left arguments respectively of two demodulators D_1 and D_2 that both match some common term. That is, the term is an instance of both S and T. Demodulator D_1 is a better match than D_2 if the following condition is satisfied: If the arguments s_i and t_i of S and T, respectively, are compared for i = 1 to n, then there exists a k such that

(1) s_k is not a variable.

(2) t_k is a variable.

(3) For i = 1 to k-1, either s_i and t_i are both variables or neither is a variable.

Note that this is not a recursive definition. That is, subterms of the arguments s_i and t_i are not relevant to the better match property.
Examples: Better Matches

\begin{align*}
\text{EQUAL}(F(A,VO,V1),B) & \text{ is a better match than EQUAL}(F(VO,C,D),E). \\
\text{EQUAL}(F(VO,A,VO),B) & \text{ is a better match than EQUAL}(F(VO,V1,C),D). \\
\text{EQUAL}(G(G(VO)),A) & \text{ and EQUAL}(G(G(B)),C) \text{ are not comparable with respect to the better match property.} \\
\text{EQUAL}(C,A) & \text{ and EQUAL}(C,VO) \text{ are not comparable with respect to the better match property.}
\end{align*}

The rule for demodulating a term has three parts: a rule for selecting a subterm to demodulate, a rule for selecting a demodulator to apply when more than one can apply to the selected subterm, and a rule for applying the selected demodulator to the selected term.

The rule for selecting a subterm is as follows: A subterm of a term is always demodulated before the term that contains it. If two terms are at the same level (are arguments of the same function), then the leftmost term is demodulated first.

The rule for selecting demodulators is as follows: If only one demodulator applies then apply it. If more than one demodulator applies, then consider the subset of these that are best matches in the sense of definition C5. If only one such demodulator exists, then apply it. If more than one such demodulator exists (not all demodulators are comparable with respect to the better match property), then apply the best match demodulator that has the highest clause number (was the last added to the clause space).

The rule for applying a demodulator \( \text{EQUAL}... (t_1, t_2) \) to a term \( T \) that it matches is as follows: Recall that \( T \) must be an instance of the term \( t_1 \). That is, there exists a substitution \( \sigma \) such that \( T \) and \( t_1(\sigma) \) are identical. The general demodulation step is to replace \( T \) with the corresponding instance, \( t_2(\sigma) \), of \( t_2 \). If the demodulator is \( \text{DEPTH}=0 \) demodulator, then this replacement is done unconditionally. If, however, the demodulator is a \( \text{DEPTH}=0 \) demodulator, then this replacement is done conditionally. That is, \( T \) is replaced by \( t_2(\sigma) \) only if \( \text{LEX}(t_2(\sigma)) < \text{LEX}(T) \) (see Appendix B).
APPENDIX D

Appendix D contains a proof that the canonicalization process described in Section I of the paper always terminates and produces a unique canonical form. The proof is given in the context of a simple clause space (no other demodulators). The more general proof has not been formalized.

Definition D1. The total weight of a term T is an ordered pair \((WT(T), LEX(T))\), where WT calculates the weight of T (as described in Appendix A), and LEX assigns a value that reflects the lexical order of T (as described in Appendix B).

Let the total weight of T be less than the total weight of T' if and only if

1. \(WT(T) < WT(T')\)
2. \(WT(T) = WT(T')\) and \(LEX(T) < LEX(T')\).

For this discussion, let ADD and MULT denote addition and multiplication as in the body of this paper. Let \(V_0, V_1, V_2, ... V_n\) denote variables. And let \(C_0, C_1, C_2, ... C_m\) denote constants. These are the only symbols considered.

Also, assume the following values for the theorem proving WEIGHT and DEFINE statements and the parameters VARWT, LEXPFVAR and DEPTH:

\[
\begin{align*}
\text{WEIGHT} & \quad \text{ADD}(1,1)+1 \quad \text{MULT}(1,1)+1 \\
& \quad \text{ADD}(\text{ADD}(1,1),1)+100 \quad \text{MULT}(\text{MULT}(1,1),1)+100 \\
& \quad \text{MULT}(1,\text{ADD}(1,1))+100 \quad \text{MULT}(\text{ADD}(1,1),1)+100 \\
& \quad \text{CO}+1 \quad \text{C}1+1 \quad \text{C}2+1 \quad \ldots \quad \text{C}m+1;
\end{align*}
\]

\[
\begin{align*}
\text{DEFINE} & \quad F(\text{EQUAL})=A_2 \quad F(\text{ADD})=A_2,L_3 \quad F(\text{MULT})=A_2,L_2 \\
& \quad F(\text{CO},\text{C}1,...,\text{C}m)=A_0,L_1;
\end{align*}
\]

\[
\begin{align*}
\text{VARWT}(\text{V}i) = 1 \quad \text{for all } i \\
\text{LEXPFVAR} = 0 \\
\text{DEPTH} = 0
\end{align*}
\]

The DEFINE statement and the LEXPFVAR parameter determine the following lexical order of the symbols (see Appendix B):

\[
V_0 < V_1 < ... < V_n < C_0 < C_1 < ... < C_m < \text{MULT} < \text{ADD}
\]

Definition D2. A term (polynomial) is in canonical form if

1. it is fully distributed (sums of products);
2. it is fully right associated with respect to ADD and MULT;
and (3) term \(t_1\) is to the left of term \(t_2\) iff \(LEX(t_1) \leq LEX(t_2)\).

Note that Lemma B1 (Appendix B) implies the uniqueness of the canonical form.
Consider the following set of equality unit clauses:

(D1) \( \text{EQUAL}(\text{ADD}(X,Y),\text{ADD}(Y,X)) \)
(D2) \( \text{EQUAL}(\text{ADD}(X,\text{ADD}(Y,Z)),\text{ADD}(Y,\text{ADD}(X,Z))) \)
(D3) \( \text{EQUAL}(\text{ADD}(\text{ADD}(X,Y),Z),\text{ADD}(X,\text{ADD}(Y,Z))) \)
(D4) \( \text{EQUAL}(\text{MULT}(\text{MULT}(X,Y),Z),\text{MULT}(X,\text{MULT}(Y,Z))) \)
(D5) \( \text{EQUAL}(\text{MULT}(\text{ADD}(X,Y),Z),\text{ADD}(\text{MULT}(X,Z),\text{MULT}(Y,Z))) \)
(D6) \( \text{EQUAL}(\text{MULT}(X,\text{ADD}(Y,Z)),\text{ADD}(\text{MULT}(X,Y),\text{MULT}(X,Z))) \)

The WEIGHT statement and the DEPTH parameter determine that all of the above equality units will be demodulators (see Appendix C). These are the only demodulators considered in this discussion.

The demodulators are divided into two classes. D1 and D2 are DEPTH=0 demodulators that apply conditionally based on the lexical order of a term. That is, \( t_1 \) demodulates to \( t_2 \) iff \( \text{LEX}(t_2) < \text{LEX}(t_1) \). D3 through D6 are not DEPTH=0 demodulators and they apply unconditionally. See Appendix C for a description of the demodulation process.

The rest of this discussion is restricted to the environment defined by the clause space and parameter values described above.

**Lemma D1.** A term is in canonical form iff no demodulator applies.

**Proof.** Consider the three conditions in the definition of canonical form:

Demodulators D1 and D2 apply iff condition (3) does not hold.
Demodulators D3 and D4 apply iff condition (2) does not hold.
Demodulators D5 and D6 apply iff condition (1) does not hold.

**Lemma D2.** The application of any demodulator strictly lowers the total weight of a term.

**Proof.** The following observations can be made from an inspection of the demodulators D1 through D6, the WEIGHT statement and the lexical ordering of symbols:

(1) If a DEPTH=0 demodulator is applied to term \( t_1 \) to produce term \( t_2 \) then \( \text{WT}(t_2) = \text{WT}(t_1) \) and \( \text{LEX}(t_2) < \text{LEX}(t_1) \).

(2) If a demodulator (not DEPTH=0) is applied to term \( t_1 \) to produce term \( t_2 \) then \( \text{WT}(t_2) < \text{WT}(t_1) \).

**Theorem.** The demodulation process always terminates, and the final demodulated form is canonical.

**Proof.** Lemma D2 and the fact that total weight \( > (0,0) \) implies termination. Then Lemma D1 implies that the final form is canonical.
REFERENCES


