Comp. & Maths. with Appls. Vol. 2, pp. 1–16. Pergamon Press, 1976. Printed in Great Britain.

COMPLEXITY AND RELATED ENHANCEMENTS FOR AUTOMATED THEOREM-PROVING PROGRAMS

R. OVERBEEK, J. MCCHAREN and L. WOS*

Department of Mathematics, Computer Science, Northern Illinois University, Dekalb, Illinois 60115, U.S.A.

Communicated by E. Y. Rodin

(*Received April* 1975; *and in revised form August* 1975)

Abstract—In this paper some enhancements for automated theorem-proving programs, techniques which can be combined with the various well known inference rules which are employed in such programs, are given. Proofs of some of the more difficult problems previously sought by use of automated theorem proving were obtained by a program utilizing these enhancements. Among those proven in less than 3 min were:

1. $x^3 = e$ implies $((x, y), y) = e$.
2. Boolean rings are commutative.
3. Subgroups of index 2 are normal (without case analysis).

The most powerful enhancement is that of ordering generated clauses with a heuristic which assigns an integral value to each term that occurs in any clause. The heuristic is designed to favor those clauses containing less "complex" terms. By changing input parameter values the values assigned to terms can be varied to any desired extent which in turn can sharply affect the proof search. Another enhancement is that of dynamic adjunction of demodulators including as a subcase the previous approach which has demodulators occurring only as input clauses. Methods are given for rapidly selecting the possible set of demodulators for use with a generated clause and for controlling the adjunction to reflect potential importance of the demodulator.

1. INTRODUCTION

Many of the algorithms found in the literature on automated theorem proving are of interest and often useful, but are seldom sufficient in themselves since a straightforward implementation of such algorithms often yields a program incapable of finding proofs for moderately difficult theorems. In order to find such proofs, it is generally necessary to add various enhancements, which may initially seem *ad hoc*, to the algorithm. In this paper several such enhancements, usable in conjunction with the various well known approaches, will be given in detail. The techniques discussed herein were discovered in the process of implementing a number of the better known algorithms. A program utilizing these enhancements and the inference rules (occasionally employed simultaneously) of hyper-resolution and UR-resolution, the latter discussed in Section 5, has found proofs for a number of theorems which were previously considered quite difficult from the viewpoint of automated theorem proving. Among the theorems proven with this program are the following:

(1) Let G be a group of exponent 3; i.e. for all x, $x^3 = e$, the identity. For any pair x and y, let $h(x, y)$ be the commutator of x and y, $h(x, y) = xyx^{-1}y^{-1}$. Then for all x and y $h(h(x, y), y) = e$.

(2) If R is a Boolean ring ($x^2 = x$ for all x), then R is commutative. This proof was obtained without the lemma, usually included as an input clause, that R has characteristic 2 ($x + x = 0$ for all x).

(3) Subgroups of index 2 are normal. This proof was obtained without the usual case analysis approach; i.e. was obtained without breaking the problem into two subproblems.

(4) Suppose that a Boolean algebra is defined as a set B together with two binary operations (+) and (.) such that

P1 The operations (+) and (.) are commutative.

P2 There exist in B distinct identity elements 0 and 1 relative to the operations (+) and (.) respectively.

P3 Each operation is distributive over the other.

P4 For every a in B there exists an element a' in B such that

$$a + a' = 1 \quad \text{and} \quad a \,.\, a' = 0$$

*Argonne National Laboratory, Argonne, Illinois, U.S.A.

Suppose further that the following identities have been proven:

For all a, b in B

$$a + a = a \qquad a \,.\, a = a$$

$$a + 1 = 1 \qquad a \,.\, 0 = 0$$

$$a + (a \,.\, b) = a \qquad a \,.\, (a + b) = a$$

THEOREM. *The operator* (+) *is associative.*

(5) Using the axioms, given by Tarski[3] for plane geometry, one can show that the relation "betweeness" is symmetric.

Since the number of lemmas which are included in the formulation of problems such as the above critically determine their relative difficulty, the clauses which were used as input to the programs are included in Appendix A.

A careful re-evaluation of the features in the program and the performance of the program on the previous set of problems led the authors to the following conclusions:

(a) With respect to clause generation, clause retention and most importantly the time required to obtain a proof, this program compares favorably with other reported implementations.

(b) The program would not have successfully obtained proofs of most of the problems mentioned above had not the techniques presented in the following sections been included.

(c) Either the exact methods described below or variants of these methods could be profitably employed in conjunction with most strategies which have been proposed to date.

(d) By considering these techniques, others may see generalizations or improvements which could significantly affect the development of automated theorem provers.

2. COMPLEXITY OF TERMS

A variety of authors have utilized some measure of the "complexity" of terms and restricted the use of terms exceeding some specified degree of complexity. Examples of these measures, which have been tried with varying amounts of success, are as follows:

(1) Complexity of a term is equal to the number, excluding parentheses and commas, of symbols therein.

(2) Integers can be associated with each symbol. In this case the complexity of a term is determined by summing the integers associated with the symbols that occur in the term. For example, suppose that the following associations were used:

a	1
b	3
g	1
f	2
any variable	1

With measure 2 the complexity of $f(g(a), g(x))$ would be one less than that of $g(f(a, b))$, while with measure 1 $g(f(a, b))$ would have less complexity.

(3) The complexity of a term can be measured by the maximum depth of nesting of the function symbols that occur in the term. In this case $f(g(a), g(x))$ is of the same complexity as $g(f(a, b))$, but $f(f(g(x), y), z)$ would be more complex than $f(f(a, b), f(x, d))$.

(4) The complexity of terms is measured indirectly by counting the number of symbols excluding parentheses and commas occurring in a literal.

The motivation for employing some measure of term complexity is rather obvious. If the set of allowable terms is not restricted in some manner, the sets of generated clauses and retained clauses may become so large that the clause space gets saturated and/or the time required to complete the proof search becomes prohibitive. Since the degree of complexity of all terms occurring in a given refutation is bounded, the corresponding restriction of the allowable term sets can result in a sharp reduction in the number of extraneous clauses, while still permitting the

proof search to be successful. In most cases that have been studied, some such restriction is absolutely necessary to obtain the desired proof. On the other hand the choice of an overly restrictive bound may result in the derivation of a proof substantially longer than necessary or may even cause the proof search to terminate unsuccessfully.

The effectiveness of a measure of complexity is determined by how well it can be used to circumscribe the terms required in a proof. The more difficult it is to describe the syntax of "interesting" or "potentially valuable" terms, the more difficult it will be to define the appropriate measure of complexity of terms. A restriction based on the appropriate choice of the four given measures works well for most of the theorems cited in the literature. The mathematical difficulty of most such problems could honestly be classed only as trivial. As theorem provers are applied to increasingly demanding problems, these measures will have to be generalized into more appropriate measures. For example, in certain ring theory problems it may be important to have the proof search concentrate heavily on terms which simultaneously contain the function symbols for the operators of addition and multiplication rather than on those terms which contain only one of those two function symbols. Since, in order to achieve this emphasis, the context in which a symbol occurs must be taken into account, none of the previous four measures of complexity would suffice. The following definitions are used to describe the measure implemented in the program under discussion.

Definition. Any term which is neither a constant nor a variable will be referred to as a *composite term.*

Definition. Let $t = f(t_1, t_2, t_3, \ldots, t_n)$ be a composite term. Then f is the *major function symbol of* t.

Definition. If t is the composite term $f(t_1, t_2, t_3, \ldots, t_n)$, the i^{th} *argument of* t is t_i.

Definition. A *c-pattern* can be formed by the following rules:

(1) A constant symbol is a c-pattern.

(2) A function symbol of degree $n > 0$ followed by n "arguments" inclosed by parentheses and separated by commas is a c-pattern. The "arguments" in this case must be either c-patterns or integers.

(3) Any c-pattern can always be constructed by repeated application of the previous two rules.

Definition. A *c-pair* is an ordered pair. The first element of the pair must be c-pattern and the second element must be an integer.

Definition. A *c-set* is an ordered set of c-pairs.

Instead of giving the precise algorithms for obtaining the complexity of a term, let us consider some specific examples. In each of these examples let S be a c-set composed of the following c-pairs.

1 $(a, 1)$
2 $(b, 2)$
3 $(g(1), 1)$
4 $(f(g(0), 0), 99)$
5 $(f(1, 1), 1)$

Further, suppose that each variable is to be assigned a complexity of 1.

Example 1. To determine the complexity of the constant b, simply go down the list of c-pairs until b can be matched. In this case the second element of the c-pair, 2, is the complexity of b.

Example 2. To determine the complexity of $g(f(a, b))$, the c-pair $(g(1), 1)$ is used. The c-pattern $g(1)$ "matches" $g(f(a, b))$. The integer occurring as an argument of the c-pattern means that the complexity of the corresponding subterm should be multiplied by that coefficient. Thus, $g(1)$ indicates that the complexity of $f(a, b)$ should be multiplied by 1. The final result is obtained by adding the second element of the c-pair to the complexity of $f(a, b)$. The complexity of $f(a, b)$ is equal to

$$(1 \times \text{complexity of } a) + (1 \times \text{complexity of } b) + 1$$

by the 5th c-pair. Thus, the complexity of $f(a, b)$ is 4, and the complexity of $g(f(a, b))$ is 5.

For the next example, remember that a c-set is an ordered set. Thus, if two c-patterns apply, the earlier of the two is used.

Example 3. The complexity of $f(g(x), a)$ is 99, since the first c-pattern which matches is the 4th pattern in the list. No subterms need be evaluated, since all of the coefficients are 0. The increment given by the second element of the c-pair is 99, which is then the complexity of the term.

The actual c-sets used by the program to obtain proofs to the problems in Appendix A are given in Appendix B.

3. THE BASIC ALGORITHM

Algorithm 1, which follows, is the underlying algorithm of strategy and flow for the program under discussion. For this algorithm let $h(C)$ be a function which assigns to any clause C a non-negative integer.

Algorithm 1. This algorithm takes as input an ordered set of clauses $S = \{C_1, C_2, C_3, \ldots, C_n\}$ and attempts to establish the unsatisfiability of S by utilizing some inference rule I (such as hyper-resolution, unit-resolution, etc.). The following variables are used:

LIST 1. This variable is an ordered list of clauses. The list is ordered on a field in each clause which will be referred to as the KEY of the clause. Whether clauses with equal values are entered on a first-in first-out or first-in last-out basis is determined by an input parameter.

LIST 2. This variable is an ordered list of clauses. This list has the property that all clauses which can be inferred by the inference rule I from clauses in the list have already been generated.

G LIST. This variable is also an ordered list of clauses. It roughly corresponds to the set of clauses which have been generated, but not yet added to LIST 1.

D LIST 1. This variable is a list of clauses which are to be deleted because they were subsumed by clauses which were added at a later point in time.

D LIST 2. This variable is a list of clauses which are in either LIST 1 or LIST 2, but can be further demodulated.

DEMOD-SET. This variable represents the set of demodulators. The structure of this variable will be discussed in the next section.

Originally all lists are empty, the DEMOD-SET contains no demodulators.

Step 1. Add $C_1, C_2, \ldots, C_n$ to G LIST. Set the KEY field in each clause to 0. Go to Step 5.

Step 2. If LIST 1 is empty, stop (no proof has been derived). Else, move the first clause in LIST 1 to the end of LIST 2.

Step 3. Generate the set of clauses which can be inferred from the clauses in LIST 2 using the inference rule I subject to the restriction that one of the parent clauses must be the clause just moved to LIST 2. As each such clause is generated perform the following actions:

(a) Demodulate the clause using the demodulators in DEMOD-SET. This process is described in the next section.

(b) If the clause violates some restriction such as too many literals, the occurrence of an excessively complex term, etc. terminate processing of the clause.

(c) If the clause is subsumed by a clause in LIST 1, LIST 2, or G LIST, terminate processing of the clause.

(d) If the clause is the null clause, stop. If the clause is a unit clause and can be unified with a unit clause of the opposite sign in LIST 1, LIST 2, or G LIST, stop (a proof has been completed).

(e) If any clause in LIST 1, LIST 2, or G LIST is subsumed by this clause, note the subsumed clause in D LIST.

(f) Add the clause to G LIST.

(g) Set the KEY field in the clause to the value which results from applying h to the clause.

Step 4. Delete any clause noted in D LIST from the list containing the clause. Reset D LIST to the null list.

Step 5. For each positive unit equality clause in G LIST that should be used as a demodulator, perform the following actions:

(a) Add the corresponding demodulator to DEMOD-SET.

(b) For any clause in LIST 1, LIST 2, or G LIST which can be demodulated by the added demodulator note the clause in D LIST 2.

Step 6. Add the clauses in G LIST to LIST 1. For each clause use the KEY field to determine where in LIST 1 the clause should be inserted. Reset G LIST to the null list.

Step 7. For each clause noted in D LIST 2 perform the following actions:

(a) Create the demodulant of the clause. Set the KEY field in the demodulant to the value in the KEY field of the demodulated clause.

(b) Add the demodulant to G LIST.

(c) Delete the demodulated clause from the list containing it.

Step 8. Reset D LIST 2 to the null list. If G LIST is empty, go to Step 2. Else, go to Step 5.

The function h, from which the value of the KEY field of each clause is obtained, strongly affects the performance of the program. For the results cited in this paper h assigned to each clause C

(the number of literals in C minus 1) $\times K +$ (the maximum of the complexities of the composite terms of C).

K is an input parameter and generally is set to 1. The purpose of K is to take into account in the function h the number of literals of C for each clause C.

For a more detailed discussion of how the set of clauses generated in Step 3 can be determined efficiently when hyper-resolution is used as the inference rule the reader is referred to[2]. The algorithms and data structures presented in that paper have been generalized in a natural way and are part of the present implementation under discussion. Programs have been written implementing positive hyper-resolution, negative hyper-resolution, P1-resolution, unit resolution and UR-resolution.

4. DEMODULATION

The advantages of demodulation have been adequately covered elsewhere[4]. There are several aspects, however, of this implementation which are worth commenting on:

(a) Given a clause C and a set W of demodulators, there may be several (final) distinct demodulants of C. As currently implemented, only one of these demodulants will be generated. Such an approach, although common, is questionable; in the future the authors intend to implement the more general approach. The reader should note that Algorithm 1 must be modified if all demodulants of a given clause are to be generated.

(b) In most implementations the set of demodulators is given as input and is not altered during the execution of the program. This is not true of this implementation. With more difficult problems, a significant advantage can be gained by adding demodulators as appropriate positive unit equality clauses are deduced. For example, one criterion, now present in the program, for adjoining a clause of the demodulator list is: adjoin the equality clause C if and only if the difference in the term complexity occurring in the two arguments of the literal of C is greater than or equal to the value of some input parameter. By setting this input parameter to a very large positive integer and by inputting some possibly empty set of demodulators one can have the present program perform with respect to demodulation as the previous implementations cited in the literature have. If the input parameter is set to 0, it is possible for a term t to be a demodulant of itself. For example, if $F(x, y) -> F(y, x)$ is a demodulator,

$$F(g(a), b) -> F(b, g(a)) -> F(g(a), b)$$

To prevent this phenomenon the following restrictions on demodulation are used:

(1) For t to be replaced by t', the complexity of t must be at least as great as that of t'.

(2) If t and t' are of equal complexity, an arbitrary but fixed ordering of terms will be used to determine whether or not t can be replaced.

(c) Because the number of demodulators in the set W can increase to a relatively large number, a means of rapidly selecting a subset of W that could be used to demodulate a term t was implemented. The process can be described as follows:

(1) Let $\alpha => \beta$ be a new demodulator, where $\alpha = f(t_1, t_2, t_3, \ldots t_n)$. Associate with α the following ordered $n+1$-tuple:

$$(f, a_1, a_2 \ldots a_n)$$

where a_i is the leftmost symbol in t_i, unless t_i is a variable. In this case let a_i be V^*. As an example, with $f(x, g(x)) = >e$ the 3-tuple (f, V^*, g) would be associated. The $n+1$-tuple associated with a term will be called the d-key of the term.

(2) For each demodulator $\alpha = >\beta$, add the d-key of α to a "demodulation" tree. Thus, for the set of demodulators

	demodulator	d-key
d_0	$f(x, e) = >x$	(f, V^*, e)
d_1	$f(e, x) = >X$	(f, e, V^*)
d_2	$f(x, g(x)) = >e$	(f, V^*, g)
d_3	$f(g(x), x) = >e$	(f, g, V^*)
d_4	$f(f(x, y), g(y)) = >x$	(f, f, g)
d_5	$g(g(x)) = >x$	(g, g)
d_6	$f(x, g(f(y, x))) = >y$	(f, V^*, g)

the tree, would be created.

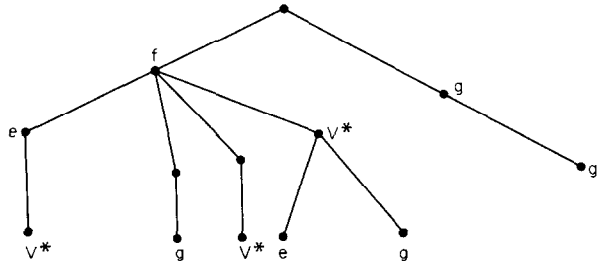

Fig. 1.

The set of demodulators represented by a given branch is attached to the appropriate leaf, and the resulting tree is converted to a binary tree. The result for the previous example would be

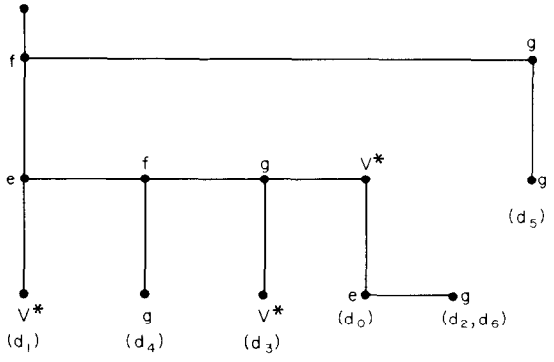

Fig. 2.

(3) Given any term t with its associated d-key, it is a simple matter to determine the branches of the binary tree that represent potential demodulators. For example, if the term $t = f(f(x, a), g(a))$ were to be demodulated, the d-key associated with t would be (f, f, g). The branches of the tree that represent potential demodulators are (f, f, g) and (f, V^*, g), and the resulting set of demodulators is (d_4, d_2, d_6).

The use of such a "screening" technique to reduce the number of required unification attempts can dramatically reduce the time required to demodulate a term or clause.

5. UR-RESOLUTION

When hyper-resolution is used as an inference rule on a problem which formulates as a Horn set, only positive unit clauses are generated. The fact that only unit clauses are generated appears beneficial. However, in those problems in which the clauses contradicting the theorem contain a

negative unit clause, the negative unit clause cannot be utilized until the last step of the proof. This inability to exploit the denial of the theorem until the last step of the proof is a serious drawback of hyper-resolution. In order to retain the positive effect of generating only unit clauses and to allow the use of negative unit clauses, an inference rule yielding unit resolvents (similar to one discussed by Chang[1]), Unit-Resulting (UR) resolution, was defined and implemented.

Definition. Suppose that the ordered set of (possibly non-unique) clauses

$$\{N, E_1, E_2, \ldots E_{i-1}, E_{i+1}, \ldots Eq\}$$

satisfy the following conditions:

(i) No two clauses in the set contain occurrences of the same variable.
(ii) Each of the clauses E_k, $1 \leq k \leq q$, $k \neq i$ is a unit clause.
(iii) The clause N contains q literals, which may be ordered and referred to as $\langle L_1, L_2, L_3, \ldots Lq \rangle$.
(iv) There exists a most general substitution α such that α unifies the literal in E_k with L_k (except that the signs must be opposite), $1 \leq k \leq q$, $k \neq i$.

Then the unit clause

$$\{Li\alpha\}$$

is called the UR-resolvent of the ordered set of clauses. It is convenient to represent the UR-resolvent as

$$\langle N, E_1, E_2, \ldots E_{i-1}, \mathrm{I}, E_{i+1}, \ldots Eq \rangle,$$

since it is then clear which literal in N is matched.

Example. Consider the clauses

$$N = \{-\mathrm{P(V0, V1, V2)}, -\mathrm{P(V1, V3, V4)}, -\mathrm{P(V0, V4, V5)}, \mathrm{P(V2, V3, V5)}\}$$
$$A = \{\mathrm{P(V0, V0, E)}\} \quad B = \{\mathrm{P(E, V0, V0)}\}$$

which arise typically from the axioms of an associative multiplicative system with an identity. The clause {P(E, E, E)} is a UR-resolvent represented an $\langle N, A, A, B, \mathrm{I} \rangle$.

UR-resolution has been successfully implemented. The results of this implementation are given in a later section. It is only proper to note that authors in the past have considered inference procedures in which unit clauses are inferred by a sequence of unit resolutions. It should be stressed that in the implementation of UR-resolution it was not implemented as a sequence of unit resolutions. Rather, the program proceeds by extending a substitution until all of the pertinent literals are unified. Only then is the substitution applied to a literal, creating the UR-resolvent. It is essentially the same distinction that exists between viewing hyper-resolution as a positive inference rule and viewing it as producing a resolvent by a sequence of P1 resolutions.

6. SCHEMATA

In many instances it will be the case that UR-resolution will produce a clause that is already in the clause space, a clause which demodulates to a clause in the clause space, or having some other property which would preclude it from being added to the clause space. It is desirable to classify those clashes which give rise to resolvents which should not be added to the clause space. For example suppose the clause space contains the following clauses:

$$W = \{-\mathrm{P(V0, V1, V2)}, -\mathrm{P(V1, V3, V4)}, -\mathrm{P(V0, V4, V5)}, \mathrm{P(V2, V3, V5)}\}$$
$$A = \{\mathrm{P(V0, V0, E)}\} \quad B = \{\mathrm{P(V0, E, V0)}\} \quad C = \{\mathrm{P(E, V0, V0)}\}$$

Then the UR-resolvent $\langle N, B, C, A, \mathrm{I} \rangle$ is A, and the UR-resolvent $\langle N, B, C, B, \mathrm{I} \rangle$ is B. In fact if X is any unit clause in the clause space, the UR-resolvent $\langle N, B, C, X, \mathrm{I} \rangle$ is X. Thus, there is no need to

form the resolvents of the form $\langle N, B, C, X, \mathrm{I}\rangle$. In order to describe such a family of clashes the following concepts have proven useful.

Definition. A schema is an ordered set

$$S = \{A_0, A_1, A_2, \ldots A_{i-1}, \mathrm{I}, A_{i+1}, \ldots A_n\}$$

where

(i) A_0 is a clause containing n literals, and
(ii) A_k, $1 \leq k \leq n$, $k \neq i$ is either a clause or a variable.
Thus if A and B are clauses, each of the following ordered sets is a schema.

(a) $\{A, B, \mathrm{I}\}$

(b) $\{A, \mathrm{I}, X\}$

(c) $\{A, X, \mathrm{I}, X, B\}$

(d) $\{A, X, \mathrm{I}, Y, B\}$

Here the symbols X and Y are used to denote distinct variables. The concept of a schema facilitates the description of families of UR-resolvents as described below.

Definition. A schema $S = \langle A_0, A_1, \ldots, A_q\rangle$ is said to match the UR-resolvent $\langle N, E_1, \ldots E_{i-1}, \mathrm{I}, E_{i+1}, \ldots Eq\rangle$ providing.

(i) $A_0 = N$ and $n = q$
(ii) there is an assignment of values to the variables of S such that for each A_k $1 \leq k \leq q$, $k \neq i$, $A_k = E_k$ i.e. if A_k is a clause, $A_k = E_k$; and if A_k is a variable the value of the variable A_k is E_k.

Example. Consider the clauses:

$$N = \{-\mathrm{P(V0, V1, V2)}, -\mathrm{P(V1, V3, V4)}, -\mathrm{P(V0, V4, V5)}, \mathrm{P(V2, V3, V5)}\}$$

$$A = \{\mathrm{P(V0, V0, E)}\} \quad B = \{\mathrm{P(V0, E, V0)}\}$$

The clause $\{\mathrm{P(E, E, E)}\}$ obtained by clashing A against the first and second literals of N and B against the third literal of N is a resolvent represented by $\langle N, A, A, B, \mathrm{I}\rangle$. This UR-resolvent matches each of the schemata

(a) $\{N, A, A, B, \mathrm{I}\}$

(b) $\{N, X, X, B, \mathrm{I}\}$

(c) $\{N, X, X, Y, \mathrm{I}\}$

(d) $\{N, X, Y, B, \mathrm{I}\}$

but not the schema

(e) $\{N, X, B, X, \mathrm{I}\}$

The theorem proving program has the capability of accepting a set of schema as input. In this case only UR-resolvents which match no schema are formed. By a judicious selection of the set of input schema a significant reduction in the number of generated clauses can be obtained for some problems.

7. THE IMPLEMENTATION OF UR-RESOLUTION

UR-resolution has been successfully implemented. Generally it seems that for this type of theorem, the schemata may be used to reduce the number of generated clauses by 5 to 20%. As

there are no known necessary and sufficient conditions for selecting schemata as input to the theorem-prover, a representative set of the schemata used for theorem 4 are discussed here.

(A) Consider the following clauses:

$$C7 = \{-\text{PROD}(V0, V1, V2), -\text{PROD}(V0, V3, V4), -\text{SUM}(V1, V3, V5), -\text{SUM}(V2, V4, V6), \text{PROD}(V0, V5, V6)\}$$

$$C31 = \{\text{PROD}(V0, 0, 0)\} \qquad C4 = \{\text{SUM}(V0, 0, V0)\}$$

If C4 is used to remove the third and fourth literals of C7, and C31 used to remove the second literal the resulting clause is

$$\{-\text{PROD}(V0, V1, V2), \text{PROD}(V0, V1, V2)\}.$$

As this is a tautology it suggests the two schemata {C7, I, C31, C4, C4, X} and {C7, X, C31, C4, C4, I} to be used as input to the theorem prover. Indeed if A is any unit clause such that ⟨C7, A, C31, C4, C4, I⟩ or ⟨C7, I, C4, C4, *A*⟩ is a UR-resolvent, the resolvent is *A*.

(B) Consider the clause

$$C8 = \{-\text{SUM}(V0, V1, V2), -\text{SUM}(V0, V3, V4), -\text{PROD}(V1, V3, V5), -\text{SUM}(V0, V5, V6), \text{PROD}(V2, V4, V6)\}$$

$$C29 = \{\text{PROD}(V0, V0, V0)\}.$$

If C29 is used to remove the third literal, the resulting clause is

$$\{-\text{SUM}(V0, V1, V2), -\text{SUM}(V0, V1, V6), -\text{SUM}(V0, V1, V6), \text{PROD}(V2, V4, V6)\}.$$

It is easily seen that any clause *A* that matches against the first three literals may be used to form the UR-resolvent ⟨C8; A, A, C29, *A*, I⟩. The resolvent if $\{P(t, t, t)\}$ where t is the third argument of *A*. But this is subsumed by the axiom C29 = {P(V0, V0, V0)}, so the schema {C8, X, X, C29, X, I} should be used to prevent such clashes.

(C) Consider the clause

$$C6 = \{-\text{PROD}(V0, V1, V2), -\text{PROD}(V0, V3, V4), -\text{SUM}(V1, V3, V5), -\text{PROD}(V0, V5, V6), \text{SUM}(V2, V4, V6)\}.$$

and the derived clauses

$$C40 = \{\text{PROD}(V0, V1, F2(V1, V0)\}$$

$$C42 = \{\text{PROD}(1, V0, V0)\}$$

If C40 is used to remove the first and second literals, and C42 used to remove the fourth literal the clause

$$\{-\text{SUM}(V1, V3, V5), \text{SUM}(F1(V1, 1), F2(V3, 1), V5)\}$$

is obtained. If *A* = SUM(t1, t3, t5) is used to clash against the first literal of this clause, the clause SUM(F2(t1, 1), F2(t3, 1), t5) is obtained. Since this clause demodulates to *A*, the schema ⟨C6; C40, C40, X, C42, I⟩ is used to prevent such clashes.

(D) Finally if clause C42 is used to remove the first and second literals of C7, the clause

$$\{-\text{SUM}(V1, V3, V5), -\text{SUM}(V1, V3, V6), \text{PROD}(1, V5, V6)\}$$

is obtained. If *A* and *B* are any clauses such that ⟨C7, C42, C42, A, B, I⟩ is a UR-resolvent, the

resolvent is of the form PROD(1, $t5$, $t6$), where $t5$ is the third term of A, and $t6$ is the third term of B. However, this resolvent will be obtained from the clauses

$$C12 = \{-SUM(V0, V1, V2), -SUM(V0, V1, V3), EQUAL(V2, V3)\}$$

and

$$C20 = \{-EQUAL(V0, V1), -PROD(V2, V3, V0), PROD(V2, V3, V1)\}$$

as follows. The UR-resolvent $\langle C12, A, B, I\rangle$ is $C = \{EQUAL\ (t5, t6)\}$ then the clash

$$\langle C20, C, C40, 1\rangle$$

is $\{P(1, t5, t6)\}$. Since there is no reason to generate the clause twice, the schema {C7, C42, C42, X, Y, 1} was added to prevent one derivation of this clause.

In summary the schemata presented above are used to curtail the generation of UR-resolvents that (i) are already in the clause space, or (ii) demodulate to a clause in the clause space or (iii) may be derived by another series of inferences. In both theorems each schema was added for one of these reasons.

REFERENCES

1. C. L. Chang, *The unit proof and the input proof in theorem proving. JACM*, **17**, 689–707.
2. R. A. Overbeek, *An implementation of hyper-resolution, Comp. Maths. with Appls.*, **1**, 201–214 (1975).
3. A. Tarski, What is Elementary Geometry?, *Symposium on the Axiomatic Method.*
4. L. Wos, G. A. Robinson, F. Carson and L. Shalla, *The Concept of Demodulation in Theorem-proving, JACM*, **14**(4), 698–709 (1967).

APPENDIX A

```
     ---- THE FOLLOWING ARE GROUP AXIOMS ----

     .... THE OPERATION * IS CLOSED ....

CL   P(X,Y,F(X,Y))

     .... E IS A LEFT AND RIGHT IDENTITY ....

CL   P(E,X,X)
CL   P(X,E,X)

     .... G(X) IS A LEFT AND RIGHT INVERSE ....

CL   P(G(X),X,E)
CL   P(X,G(X),E)

     .... THE OPERATION * IS ASSOCIATIVE ....

CL -P(X,Y,V0) -P(Y,Z,V) -P(V0,Z,W)  P(X,V,W)
CL -P(X,Y,V0) -P(Y,Z,V) -P(X,V,W)   P(V0,Z,W)

     ..... THE OPERATION * IS WELL DEFINED ....

CL -P(X,Y,Z) -P(X,Y,W)  EQUAL(Z,W)

     ---- EQUALITY AXIOMS ----

CL  EQUAL(X,X)
CL -EQUAL(X,Y) -EQUAL(Y,Z)  EQUAL(X,Z)
CL -EQUAL(X,Y)  EQUAL(Y,X)

     ---- EQUALITY SUBSTITUTION AXIOMS ----

CL -EQUAL(X,Y)  EQUAL(G(X),G(Y))
CL -EQUAL(X,Y)  EQUAL(F(X,W),F(Y,W))
CL -EQUAL(X,Y)  EQUAL(F(W,X),F(W,Y))
CL -EQUAL(X,Y) -P(X,W,Z)  P(Y,W,Z)
CL -EQUAL(X,Y) -P(W,X,Z)  P(W,Y,Z)
CL -EQUAL(X,Y) -P(W,Z,X)  P(W,Z,Y)
```

```
THEOREM 1

     ----- SPECIAL HYPOTHESIS ----

     .... X CUBED EQUALS THE IDENTITY E ....

CL -P(X,X,Y)  P(X,Y,E)
CL -P(X,X,Y)  P(Y,X,E)

     ---- THE NEGATION OF THE THEOREM ----

     .... ((X,Y),Y)=E ....

CL  P(A,B,C)
CL  P(C,G(A),D)
CL  P(D,G(B),H)
CL  P(H,B,J)
CL  P(J,G(H),K)
CL -P(K,G(B),E)

THEOREM 2

     ---- THE FOLLOWING ARE RING AXIOMS ----

     .... E IS AN ADDITIVE IDENTITY ....

CL  S(E,X,X)
CL  S(X,E,X)

     .... THE SUM IS CLOSED ....

CL  S(X,Y,J(X,Y))

     .... PRODUCT IS CLOSED ....

CL  P(X,Y,F(X,Y))

     .... G(X) IS THE LEFT AND RIGHT ADDITIVE INVERSE ....

CL  S(G(X),X,E)
CL  S(X,G(X),E)

     .... ADDITION IS ASSOCIATIVE ....

CL -S(X,Y,V0) -S(Y,Z,V) -S(X,V,W)   S(V0,Z,W)
CL -S(X,Y,V0) -S(Y,Z,V) -S(V0,Z,W)  S(X,V,W)

     .... MULTIPLICATION IS ASSOCIATIVE ....

CL -P(X,Y,V0) -P(Y,Z,V) -P(X,V,W)   P(V0,Z,W)
CL -P(X,Y,V0) -P(Y,Z,V) -P(V0,Z,W)  P(X,V,W)

     .... ADDITION IS COMMUTATIVE ....

CL -S(X,Y,Z)  S(Y,X,Z)

     .... MULTIPLICATION DISTRIBUTES OVER ADDITION ....

CL -P(V0,V1,V2) -P(V0,V3,V4) -S(V1,V3,V5)
   -P(V0,V5,V6)  S(V2,V4,V6)
CL -P(V0,V1,V2) -P(V0,V3,V4) -S(V1,V3,V5)
   -S(V2,V4,V6)  P(V0,V5,V6)
CL -P(V0,V1,V2) -P(V3,V1,V4) -S(V0,V3,V5)
   -P(V5,V1,V6)  S(V2,V4,V6)
CL -P(V0,V1,V2) -P(V3,V1,V4) -S(V0,V3,V5)
   -S(V2,V4,V6)  P(V5,V1,V6)

     ---- EQUALITY AXIOMS ----

CL  EQUAL(X,X)
CL -EQUAL(X,Y)  EQUAL(Y,X)
CL -EQUAL(X,Y) -EQUAL(Y,Z)  EQUAL(X,Z)

     ---- EQUALITY SUBSTITUTION AXIOMS ----

CL -EQUAL(X,Y) -S(X,W,Z)  S(Y,W,Z)
CL -EQUAL(X,Y) -S(W,X,Z)  S(W,Y,Z)
CL -EQUAL(X,Y) -S(W,Z,X)  S(W,Z,Y)
CL -EQUAL(X,Y) -P(X,W,Z)  P(Y,W,Z)
```

```
CL -EQUAL(X,Y) -P(W,X,Z)  P(W,Y,Z)
CL -EQUAL(X,Y) -P(W,Z,X)  P(W,Z,Y)
CL -EQUAL(X,Y)  EQUAL(G(X),G(Y))
CL -EQUAL(X,Y)  EQUAL(F(X,W),F(Y,W))
CL -EQUAL(X,Y)  EQUAL(F(W,X),F(W,Y))
CL -EQUAL(X,Y)  EQUAL(J(X,W),J(Y,W))
CL -EQUAL(X,Y)  EQUAL(J(W,X),J(W,Y))

    .... SUM AND PRODUCT ARE WELL DEFINED ....

CL -S(X,Y,W) -S(X,Y,Z)  EQUAL(W,Z)
CL -P(X,Y,W) -P(X,Y,Z)  EQUAL(W,Z)

    ---- LEMMAS PROVED ----

    .... CANCELLATION LAWS ....

CL -S(X,Y,Z) -S(X,W,Z)  EQUAL(Y,W)
CL -S(X,Y,Z) -S(W,Y,Z)  EQUAL(X,W)

    ---- THE NEGATION OF THE THEOREM ----

    .... X**2 = X  =>  COMMUTATIVITY ....

CL  P(X,X,X)
CL  P(A,B,C)
CL -P(B,A,C)

THEOREM 3

    ---- LEMMAS PROVED ----

    .... ESTABLISHED IDENTITIES ....

CL  EQUAL(F(X,E),X)
CL  EQUAL(F(E,X),X)
CL  EQUAL(F(G(X),X),E)
CL  EQUAL(F(X,G(X)),E)
CL  EQUAL(G(G(X)),X)
CL  EQUAL(G(E),E)

    ---- THE NEGATION OF THE THEOREM ----

    .... THE FOLLOWING CLAUSES DENY THAT ....
    .... SUBGROUPS OF INDEX 2 ARE NORMAL ....

CL  0(E)
CL -0(X)  0(G(X))
CL -0(X) -0(Y) -P(X,Y,Z)  0(Z)
CL -EQUAL(X,Y) -0(X)  0(Y)
CL -EQUAL(X,Y)  EQUAL(I(W,X),I(W,Y))
CL -EQUAL(X,Y)  EQUAL(I(X,W),I(Y,W))
CL  0(X)  0(Y)  0(I(X,Y))
CL  0(X)  0(Y)  P(X,I(X,Y),Y)
CL  0(B)
CL  P(B,G(A),C)
CL  P(A,C,D)
CL -0(D)

THEOREM 4

    ---- A BOOLEAN ALGEBRA IS DEFINED TO BE A SET (B) ----
    ---- TOGETHER WITH TWO BINARY OPERATIONS (+), (.) ----
    ---- SATISFYING THE FOLLOWING AXIOMS ----

    .... DEFINITION OF THE SUM AND PRODUCTS ....
    .... PREFIXES AND FUNCTIONS ....

CL  SUM(X,Y,F1(X,Y))
CL  PROD(X,Y,F2(X,Y))

    .... P1) ....
    .... THE OPERATIONS (+) AND (.) ARE COMMUTATIVE ....

CL -SUM(X,Y,V3)  SUM(Y,X,V3)
CL -PROD(X,Y,V3)  PROD(Y,X,V3)

    .... P2) ....
    .... THERE EXISTS IN (B) IDENTITY ELEMENTS (C1,C2) ....
```

```
     .... RELATIVE TO THE OPERATIONS (+) AND (.) ....
     .... RESPECTIVELY ....

CL   SUM(X,C1,X)
CL   SUM(C1,X,X)
CL   PROD(X,C2,X)
CL   PROD(C2,X,X)

     .... P3) ....
     .... EACH OPERATION DISTRIBUTES OVER THE OTHER ....
     .... X.(Y+Z) = (X,Y) +(X.Z)  (Y+Z).X = (Y.X) + (Z.X) ....

CL -PROD(X,Y,V1) -PROD(X,Z,V2) -SUM(Y,Z,V3) -PROD(X,V3,V4)
     SUM(V1,V2,V4)
CL -PROD(X,Y,V1) -PROD(X,Z,V2) -SUM(Y,Z,V3) -SUM(V1,V2,V4)
     PROD(X,V3,V4)
CL -PROD(Y,X,V1) -PROD(Z,X,V2) -SUM(Y,Z,V3) -PROD(V3,X,V4)
     SUM(V1,V2,V4)
CL -PROD(Y,X,V1) -PROD(Z,X,V2) -SUM(Y,Z,V3) -SUM(V1,V2,V4)
     PROD(V3,X,V4)

     .... X+(Y.Z) = (X+Y).(X+Z)  (Y.Z)+X = (Y+X).(Y+Z) ....

CL -SUM(X,Y,V1) -SUM(X,Z,V2) -PROD(Y,Z,V3) -SUM(X,V3,V4)
     PROD(V1,V2,V4)
CL -SUM(X,Y,V1) -SUM(X,Z,V2) -PROD(Y,Z,V3) -PROD(V1,V2,V4)
     SUM(X,V3,V4)
CL -SUM(Y,X,V1) -SUM(Z,X,V2) -PROD(Y,Z,V3) -SUM(V3,X,V4)
     PROD(V1,V2,V4)
CL -SUM(Y,X,V1) -SUM(Z,X,V2) -PROD(Y,Z,V3) -PROD(V1,V2,V4)
     SUM(V3,X,V4)

     .... P4) ....
     .... FOR EVERY ELEMENT (X) IN A BOOLEAN ALGEBRA ....
     .... THERE EXISTS -(X) SUCH THAT: ....
     .... X + -(X) = C2 AND -(X) . X = C1 ....
     .... X . -(X) = C1 AND -(X) + X = C1 ....

CL   SUM(X,F3(X),C2)
CL   SUM(F3(X),X,C2)
CL   PROD(X,F3(X),C1)
CL   PROD(F3(X),X,C1)

     .... THE OPERATIONS (+) AND (.) ARE WELL DEFINED ....

CL -SUM(X,Y,V3) -SUM(X,Y,V4)  EQUAL(V3,V4)
CL -PROD(X,Y,V3) -PROD(X,Y,V4)  EQUAL(V3,V4)

     ---- EQUALITY AXIOMS ----

CL   EQUAL(X,X)
CL -EQUAL(X,Y)  EQUAL(Y,X)
CL -EQUAL(X,Y) -EQUAL(Y,V1)  EQUAL(X,V1)

     ---- EQUALITY SUBSTITUTION AXIOMS ----

CL -EQUAL(X,Y) -SUM(X,V1,V2)  SUM(Y,V1,V2)
CL -EQUAL(X,Y) -SUM(V1,X,V2)  SUM(V1,Y,V2)
CL -EQUAL(X,Y) -SUM(V1,V2,X)  SUM(V1,V2,Y)
CL -EQUAL(X,Y) -PROD(X,V1,V2)  PROD(Y,V1,V2)
CL -EQUAL(X,Y) -PROD(V1,X,V2)  PROD(V1,Y,V2)
CL -EQUAL(X,Y) -PROD(V1,V2,X)  PROD(V1,V2,Y)


CL -EQUAL(X,Y)  EQUAL(F1(X,V1),F1(Y,V1))
CL -EQUAL(X,Y)  EQUAL(F1(V1,X),F1(V1,Y))
CL -EQUAL(X,Y)  EQUAL(F2(X,V1),F2(Y,V1))
CL -EQUAL(X,Y)  EQUAL(F2(V1,X),F2(V1,Y))
CL -EQUAL(X,Y)  EQUAL(F3(X),F3(Y))

     ---- LEMMAS PROVED ----

     .... THE THEOREM PROVER HAS PREVIOUSLY GIVEN PROOFS ....
     .... FOR THE FOLLOWING IDENTITIES ....

     .... FOR ELEMENTS X,Y IN A BOOLEAN ALGEBRA (B): ....
     .... X + X = X          AND          X . X = X ....

CL   SUM(X,X,X)
```

```
CL  PROD(X,X,X)

    .... X + C2 = C1          AND          X . C1 = C1 ....

CL  SUM(X,C2,C2)
CL  PROD(X,C1,C1)

    .... X + (X.Y) = X        AND          X . (X+Y) = X ....

CL  SUM(X,F2(X,Y)X)
CL  PROD(X,F1(X,Y),X)

    ---- THE NEGATION OF THE THEOREM ----

    .... X + (Y+Z) = (X+Y) + Z ....

CL  SUM(C4,C5,C7)
CL  SUM(C3,C7,C8)
CL  SUM(C3,C4,C9)
CL  SUM(C9,C5,C10)
CL -EQUAL(C8,C10)

THEOREM 5
    THIS PROBLEM DEALS WITH TARSKI'S AXIOMATIZATION OF ELEMENTARY
    GEOMETRY.  THE PROBLEM IS TO SHOW THAT IF THE POINT A LIES
    BETWEEN POINTS B AND C, THEN A MUST LIE BETWEEN C AND B
    THE MEANINGS OF THE FUNCTIONS AND PREDICATES ARE AS FOLLOWS:

         C1, C2, C3 ARE INTRODUCED AS SKOLEM FUNCTIONS IN THE CONVERTED
           FORM OF THE "LOWER DIMENSION" (A11).
         F1 IS A SKOLEM FUNCTION INTRODUCED IN THE CONVERTED FORM OF
           "PASCH'S AXIOM" (A7).
         F2 AND F3 ARE SKOLEM FUNCTIONS INTRODUCED INTO THE CONVERTED
           FORM OF "EUCLID'S AXIOM" (A8).
         F4 IS A SKOLEM FUNCTION INTRODUCED IN THE CONVERTED FORM OF
           THE "AXIOM OF SEGMENT CONSTRUCTION" (A10).
         F5 IS A SKOLEM FUNCTION INTRODUCED INTO THE CONVERTED FORM OF
           THE WEAKENED "ELEMENTARY CONTINUITY AXIOMS" (A13').


         A, B, AND C ARE THE POINTS MENTIONED IN THE STATEMENT OF THE
           PROBLEM.


         B(X,Y,Z) IS TRUE IFF Y LIES BETWEEN X AND Z.
         L(X1,Y1,X2,Y2) IS TRUE IFF X1 IS THE SAME DISTANCE FROM Y1
           THAT X2 IS FROM Y2.

         NOTE THAT THE WEAKENED FORM OF AXIOM A13 (A13') IS USED.

    .... A1 ....

CL - B(X,Y,X)  EQUAL(X,Y)

    .... A2 ....

CL -B(X,Y,V) -B(Y,Z,V)  B(X,Y,Z)

    .... A3 ....

CL -B(X,Y,Z) -B(X,Y,V)  EQUAL(X,Y)  B(X,Z,V)  B(X,V,Z)

    .... A4 ....

CL  L(X,Y,Y,X)

    .... A5 ....

CL -L(X,Y,Z,Z)  EQUAL(X,Y)

    .... A6 ....

CL -L(X,Y,Z,V) -L(X,Y,V2,W)  L(Z,V,V2,W)

    .... A7 ....

CL -B(X,W,V) -B(Y,V,Z)  B(X,F1(W,X,Y,Z,V),Y)
CL -B(X,W,V) -B(Y,V,Z)  B(Z,W,F1(W,X,Y,Z,V))

    .... A8 ....
```

```
CL -B(X,V,W) -B(Y,V,Z)  EQUAL(X,V)  B(X,Z,F2(W,X,Y,Z,V))
CL -B(X,V,W) -B(Y,V,Z)  EQUAL(X,V)  B(X,Y,F3(W,X,Y,Z,V))
CL -B(X,V,W) -B(Y,V,Z)  EQUAL(X,V)  B(F2(W,X,Y,Z,V),W,F3(W,X,Y,Z,V))

    .... A9 ....

CL -L(X,Y,X1,Y1) -L(Y,Z,Y1,Z1) -L(X,V,X1,V1) -L(Y,V,Y1,V1)
   -B(X,Y,Z) -B(X1,Y1,Z1)  EQUAL(X,Y)  L(Z,V,Z1,V1)

    .... A10 ....

CL  B(X,Y,F4(X,Y,W,V))
CL  L(Y,F4(X,Y,W,V),W,V)

    .... A11 ....

CL -B(C1,C2,C3)
CL -B(C2,C3,C1)
CL -B(C3,C1,C2)

    .... A12 ....

CL -L(X,W,X,V) -L(Y,W,Y,V) -L(Z,W,Z,V)  EQUAL(W,V)  B(X,Y,Z)
    B(Y,Z,X)  B(Z,X,Y)

    .... A13 ....'

CL -L(V,X,V,X1) -L(V,Z,V,Z1) -B(V,X,Z) -B(X,Y,Z)
    L(V,Y,V,F5(X,Y,Z,X1,Z1,V))
CL -L(V,X,V,X1) -L(V,Z,V,Z1) -B(V,X,Z) -B(X,Y,Z)
    B(X1,F5(X,Y,Z,X1,Z1,V),Z1)

    .... EQUALITY AXIOMS ....

CL  EQUAL(X,X)
CL -EQUAL(X,Y)  EQUAL(Y,X)
CL -EQUAL(X,Y) -EQUAL(Y,Z)  EQUAL(X,Z)


    ---- EQUALITY SUBSTITUTION AXIOMS ----

CL -EQUAL(X,Y) -B(X,W,Z)  B(Y,W,Z)
CL -EQUAL(X,Y) -B(W,X,Z)  B(W,Y,Z)
CL -EQUAL(X,Y) -B(W,Z,X)  B(W,Z,Y)
CL -EQUAL(X,Y) -L(X,V,W,Z)  L(Y,V,W,Z)
CL -EQUAL(X,Y) -L(V,X,W,Z)  L(V,Y,W,Z)
CL -EQUAL(X,Y) -L(V,W,X,Z)  L(V,W,Y,Z)
CL -EQUAL(X,Y) -L(V,W,Z,X)  L(V,W,Y,Z)
CL -EQUAL(X,Y)  EQUAL(F1(X,V1,V2,V3,V4),F1(Y,V1,V2,V3,V4))
CL -EQUAL(X,Y)  EQUAL(F1(V1,X,V2,V3,V4),F1(V1,Y,V2,V3,V4))
CL -EQUAL(X,Y)  EQUAL(F1(V1,V2,X,V3,V4),F1(V1,V2,Y,V3,V4))
CL -EQUAL(X,Y)  EQUAL(F1(V1,V2,V3,X,V4),F1(V1,V2,V3,Y,V4))
CL -EQUAL(X,Y)  EQUAL(F1(V1,V2,V3,V4,X),F1(V1,V2,V3,V4,Y))
CL -EQUAL(X,Y)  EQUAL(F2(X,V1,V2,V3,V4),F2(Y,V1,V2,V3,V4))
CL -EQUAL(X,Y)  EQUAL(F2(V1,X,V2,V3,V4),F2(V1,Y,V2,V3,V4))
CL -EQUAL(X,Y)  EQUAL(F2(V1,V2,X,V3,V4),F2(V1,V2,Y,V3,V4))
CL -EQUAL(X,Y)  EQUAL(F2(V1,V2,V3,X,V4),F2(V1,V2,V3,Y,V4))
CL -EQUAL(X,Y)  EQUAL(F2(V1,V2,V3,V4,X),F2(V1,V2,V3,V4,Y))
CL -EQUAL(X,Y)  EQUAL(F3(X,V1,V2,V3,V4),F3(Y,V1,V2,V3,V4))
CL -EQUAL(X,Y)  EQUAL(F3(V1,X,V2,V3,V4),F3(V1,Y,V2,V3,V4))
CL -EQUAL(X,Y)  EQUAL(F3(V1,V2,X,V3,V4),F3(V1,V2,Y,V3,V4))
CL -EQUAL(X,Y)  EQUAL(F3(V1,V2,V3,X,V4),F3(V1,V2,V3,Y,V4))
CL -EQUAL(X,Y)  EQUAL(F3(V1,V2,V3,V4,X),F3(V1,V2,V3,V4,Y))
CL -EQUAL(X,Y)  EQUAL(F4(X,V1,V2,V3),F4(Y,V1,V2,V3))
CL -EQUAL(X,Y)  EQUAL(F4(V1,X,V2,V3),F4(V1,Y,V2,V3))
CL -EQUAL(X,Y)  EQUAL(F4(V1,V2,X,V3),F4(V1,V2,Y,V3))
CL -EQUAL(X,Y)  EQUAL(F4(V1,V2,V3,X),F4(V1,V2,V3,Y))
CL -EQUAL(X,Y)  EQUAL(F5(X,V1,V2,V3,V4,V5),F5(Y,V1,V2,V3,V4,V5))
CL -EQUAL(X,Y)  EQUAL(F5(V1,X,V2,V3,V4,V5),F5(V1,Y,V2,V3,V4,V5))
CL -EQUAL(X,Y)  EQUAL(F5(V1,V2,X,V3,V4,V5),F5(V1,V2,Y,V3,V4,V5))
CL -EQUAL(X,Y)  EQUAL(F5(V1,V2,V3,X,V4,V5),F5(V1,V2,V3,Y,V4,V5))
CL -EQUAL(X,Y)  EQUAL(F5(V1,V2,V3,V4,X,V5),F5(V1,V2,V3,V4,Y,V5))
CL -EQUAL(X,Y)  EQUAL(F5(V1,V2,V3,V4,V5,X),F5(V1,V2,V3,V4,V5,Y))

    ---- THE NEGATION OF THE THOERM ----

CL  B(A,B,C)
CL -B(C,B,A)
```

Appendix B

Theorem 1.
CPU time - 1 min. 9 sec.
C-sets

1. (F(1,1),1)
2. (G(D,1)
3. (K,22)
4. (J,11)
5. (H,9)
6. (E,1)
7. (D,6)
8. (C,3)
9. (B,1)
10. (A,1)

Theorem 2.
CPU time - 3 min. 14 sec.
C-sets

1. (F(0,F(0,0)),99)
2. (F(F(0,0),0),99)
3. (G(1),1)
4. (F(1,1),1)
5. (J(1,1),1)
6. (C,3)
7. (A,1)
8. (E,1)

Theorem 3.
CPU time - 1 min. 45 sec.
C-sets

1. (A,1)
2. (B,1)
3. (C,1)
4. (D,1)
5. (E,1)
6. (G(1),1)
7. (F(1,1),1)
8. (I(1,1),1)

Theorem 4.
CPU time - 6 min. 26 sec.
C-sets

1. (F2(1,1),1)
2. (F1(1,1),1)
3. (F3(1,1),1)
4. (C11,1)
5. (C10,5)
6. (C9,3)
7. (C8,5)
8. (C7,3)
9. (C6,1)
10. (C5,1)
11. (C4,1)
12. (C3,1)
13. (C2,1)
14. (C1,1)

Theorem 5.
CPU time - 0 min. 38 sec.
C-sets

1. (F5(1,1,1,1,1,1),1)
2. (F4(1,1,1,1),1)
3. (F3(1,1,1,1,1),1)
4. (F2(1,1,1,1,1),1)
5. (F1(1,1,1,1,1),1)
6. (C,1)
7. (B,1)
8. (A,1)
9. (C3,1)
10. (C2,1)
11. (C1,1)