A STUDY OF SL-RESOLUTION FOR AUTOMATIC THEOREM-PROVING

A Thesis

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

Au-Yeung Woon Chi, Mary

May 1972

A STUDY OF SL-RESOLUTION FOR AUTOMATIC THEOREM-PROVING

An Abstract of a Thesis

Presented to

the Faculty of the Department of

Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Au-Yeung Woon Chi, Mary

May 1972

ABSTRACT

Automatic theorem-proving by resolution was first proposed by J. A. Robinson in 1965. Since then, quite a number of restricted versions of resolution have been proposed all with the aim of providing more efficient proof procedures.

In this paper, SL-resolution - linear resolution with selection function - recently proposed by Kowalski and Keuhner, is studied. A version of SL-resolution was implemented by means of a LISP program, and its efficiency tested on a number of examples.

In the original paper, a long and tedious proof for the completeness of this inference system was given. A more elegant proof is given here, using the basic technique developed by Anderson and Bledsoe.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

A theorem-proving problem is one which has the form: show that B follows from $A_1, \cdots, A_n$, where $A_1, \cdots, A_n$ and B are statements. $A_1, \cdots, A_n$ may be axioms or postulates of some theory, and that B is a (presumed) theorem of that thecry. This is actually proving the given theorem from the given axioms.

Such problems usually require mathematical skill and ingenuity in reasoning for their solution. However, if we express $A_1, \cdots, A_n$ and B in the symbolism of the predicate calculus, then such theorem-proving problems can be attempted automatically, using the techniques of automatic theorem-proving. Such techniques were originally developed by logicians and recently adapted and further developed for their use by computer scientists.

Besides the obvious application for proving mathematical theorems [8], automatic theorem-proving is also finding use as the deductive component of general problem-solving systems [9], the deductive component of question-answering systems [10], and the deductive component of programs for proving the correctness of computer programs [11].

In 1965, a great step in the development of automatic theorem-proving for first order logic was taken by Robinson [3], with the introduction of the resolution method. Resolutions are

carried out on a given set of clauses and the ones generated by these operations. Robinson's basic result is that if the given set of clauses is unsatisfiable, the resolution will yield the empty clause.

However, this unrestricted resolution method is inefficient because one does not know which chain of resolutions is going to lead to an empty clause in the least number of steps. One has to try out systematically all possible resolutions.

Since then, several restricted resolution principles have been proposed. Among these are hyper-resolution [5], set of support resolution [6], P_1-resolution [5], AM-clash resolution [7], linear resolution, and SL-resolution. All of them were introduced with the aim of yielding more efficient deductive systems.

In this paper, SL-resolution, a highly restricted version of resolution, recently proposed by Kowalski and Kuehner [1], is studied. They believe it to be the best inference system. Some of the theory of efficiency underlying their arguments is as follows.

We can regard a proof procedure as an inference system supplemented by a search strategy. An inference system of a proof procedure is made up of axioms and rules of inference. All restricted versions of resolution mentioned above are examples of inference systems. A search strategy for a given inference system is an algorithm for consecutively generating derivations in the search space of all derivations. Examples of

search strategies are unit preference, fewest components preference and diagonal search [12].

In general, we say that a proof procedure is efficient if a first proof of a theorem is obtained with few superfluous derivations.

The major cause for the inefficiency of present proof procedures is that they generate far too many unnecessary derivations before obtaining a proof. They are either redundant rederivations of the same sentence or irrelevant derivations. In the case of resolution systems, redundancy occurs in a more general form as a derived clause subsuming another. Redundant derivations can, to some extent, be recognized, and hence, discarded as soon as they are produced. But irrelevant derivations usually cannot be identified before the generation of a proof.

As proved by Rabin and Ehrenfaucht (unpublished), every proof procedure generates irrelevant derivations. An efficient proof procedure can only be one that generates the fewest number of irrelevant clauses. All the restricted resolution systems investigated so far succeeded in restricting the generation of redundancies. But in all cases, vast numbers remain.

The elimination of unnecessary derivations, unfortunately, does not necessarily guarantee increased efficiency. This is because shortest proofs obtainable by unrestricted resolution are often eliminated along with unnecessary derivations.

The efficiency of a proof procedure can further be increased if it is supplemented by a search strategy that searches for simplest proofs.

We now summarize the above discussion. The efficiency of a proof procedure can be improved if it has an inference system that allows few redundancies and simplest proofs and a search strategy that searches for simplest proofs. The inference system should be such that it eliminates a vast number of unnecessary derivations and admits proofs that are not much more complex than those admitted by unrestricted resolution, since a refinement often eliminates simplest proofs admitted by unrestricted resolution. Kowalski and Keuhner claim that SL-resolution is a refinement that achieves the above.

According to Kowalski and Keuhner, SL-resolution, in addition to being a good inference system, permits the application of a wide variety of heuristic techniques for the improvement of efficiency. The employment of length of clauses as a heuristic function can be applied as strategy to it as well as to other resolution systems. Unique to SL-resolution is the ability of using intelligent heuristics for the selection of the literals to be resolved upon, of constructing and learning useful lemmas for simplification of proofs and of constructing and solving subgoals for the purpose of more efficient generation of proofs. Because of this, Kowalski and Keuhner claim that SL-resolution is the best inference system for automatic theorem-proving for first-order logic.

Its efficiency was tested by implementing a version of SL-resolution by means of a LISP program. A set of fourteen examples taken from [13] and [3] was tested. Only five of them were proved. The rest used more than 120 blocks of memory without finding a proof. Thus our limited tests do not support the great claims of efficiency made by Kowalski and Keuhner. However, not all of the possible refinements of the method were implemented.

In the original paper, a long and tedious proof was given for the completeness of SL-resolution. A more elegant proof is given here, using the basic technique developed by Anderson and Bledsoe [2].

Chapter II gives an overview of first order predicate calculus, and explains what we mean by resolution in first calculus.

Chapter III describes ground SL-resolution and Chapter IV proves its completeness.

Chapter V is concerned with general SL-resolution.

Chapter VI lifts the completeness proof of Chapter IV to the level of general sets of clauses.

Chapter VII is devoted to the actual computer implementation of the proof procedure, and an evaluation of the results obtained with it.

CHAPTER II

FORMAL PRELIMINARIES

The following symbolism and definitions for predicate calculus are used:

2.1 Constant symbols. The following symbols are constant symbols: a, b, c, ···, etc.

2.2 Variable symbols. The following symbols are variable symbols: x, y, z, ···, etc.

2.3 Function symbols. The following symbols are function symbols: f, g, h, ···, etc.

2.4 The propositional connectives.

$\bar{}$ for not,

$\wedge$ for and,

$\vee$ for or,

$\rightarrow$ for if, ···, then, ···,

$\leftrightarrow$ for if and only if.

2.5 The quantifier symbols.

$\wedge_x$ for for every,

$\vee_x$ for for some.

2.6 Predicate symbols. The following symbols are predicate symbols: P, Q, R, ···, etc.

2.7 Term. A variable or a constant symbol is a term. If f is an n-ary function symbol, and t_1, ···, t_n are n terms, then $f(t_1, \cdots, t_n)$ is a term.

2.8 Atom (or atomic formula). If P is an n-ary predicate symbol, and $t_1, \cdots, t_n$ are n terms, then $P(t_1, \cdots, t_n)$ is an atom.

2.9 Literal. A literal is an atom or the negation of an atom.

2.10 Complement. If A is an atomic formula, then the two literals A and $\bar{A}$ are said to be the complement of each other, and to form, in either order, a complementary pair.

2.11 Well-formed formula. A literal is a w.f.f. If A and B are w.f.f.'s, and v is any variable, then $(\bar{A})$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \leftrightarrow B)$, $V_v(A)$, $\Lambda_v(A)$, are w.f.f.'s.

2.12 Sentence. A sentence is any w.f.f. in which all the variables are quantified.

2.13 Clause. A clause is a finite set of literals. The empty clause is denoted by $\square$. A clause with the n literals $L_1, \cdots, L_n$, stands for the w.f.f., $(L_1 \vee L_2 \vee \cdots \vee L_n)$. All variables in a clause are understood to be universally quantified.

2.14 Ground literal. A literal which contains no variables is called a ground literal.

2.15 Ground clause. A clause, each member of which is a ground literal, is called a ground clause. In particular, $\square$ is a ground clause.

2.16 Interpretation. An interpretation of a collection of sentences of the predicate calculus consists of a set of objects D together with the following associations:

(1) to each constant symbol appearing in any of the sentences, there is associated a particular object in D,

(2) to each n-ary function symbol in any of the sentences is associated an n-ary function from $\underbrace{D\times\cdots\times D}_{n}$ into D,

(3) to each n-ary predicate symbol in the sentence is associated an n-ary function from $\underbrace{D\times\cdots\times D}_{n}$ into {true,false}.

An interpretation of a collection of sentences, then, determines, in a natural way, the truth or falsity (w.r.t. the interpretation) of each of the sentences.

2.17 Unsatisfiability. A set of sentences is said to be unsatisfiable if there is no interpretation in which all of the sentences are true.

2.18 Substitution. A substitution is a list $\sigma = \{t_1/x_1, \cdots, t_n/x_n\}$, where each t_i is a term, and each x_i is a variable, and where no x_i occurs more than once. A substitution σ applied to a clause C gives the clause $C\sigma$ obtained by replacing each occurrence of x_i in C by t_i.

2.19 Unifier. A unifier σ of two literals L_1 and L_2 is a substitution σ such that on applying σ to L_1 and L_2, $L_1\sigma = L_2\sigma$.

2.20 Most general unifier. A unifier σ of two literals L_1 and L_2 is said to be a most general unifier if, given any unifier, σ_1 of L_1 and L_2, there is a substitution σ_2 such that $L_1\sigma\sigma_2 = L_1\sigma_1$, and $L_2\sigma\sigma_2 = L_2\sigma_1$.

It is well-known that in first-order predicate calculus, it is legitimate to assume that each sentence is in prenex form with no existential quantifiers in the prefix, since the existentially quantified variables have been replaced by Skolem functions. Moreover, the matrix of each sentence can be assumed to be a disjunction of formulae each of which is either an atomic formula or the negation of an atomic formula. Therefore, our syntax is set up so that the natural syntactical unit is a finite set of sentences in this special form. The quantifier prefix is omitted from each sentence, since it consists only of universal quantifiers binding each variable in the sentence; furthermore, the matrix of each sentence is regarded simply as the set of its disjuncts, on the grounds that the order and multiplicity of the disjunct in a disjunction are immaterial. In other words, we will assume that we are only dealing with sets of clauses, since any collection of sentences of the first-order predicate calculus can be transformed into such a set.

2.21 Binary factor. If C is a clause containing two literals which are unifiable and λ is the most general unifier of those literals, then $C\lambda$ is a binary factor of C. The procedure is called binary factoring.

For example, the binary factor of $P(x) \vee P(g(a)) \vee Q(x)$, having the most general unifier $\lambda = [g(a)/x]$ is $P(g(a)) \vee Q(g(a))$.

2.22 Factor. A factor of a clause C is any clause obtained from C by repeated applications of binary factoring.

2.23 Resolution. For two unifiable clauses, A and B, having the literals $L \in A$, and $\bar{L}' \in B$, such that σ is a most general unifier of L and $\bar{L}'$, the clause $[(A \setminus L) \cup (B \setminus \bar{L}')]\sigma$ is called a resolvent of A and B. The procedure of obtaining a resolvent of two clauses is called resolution.

For example, if $A = \{\bar{P}(x,y), \bar{Q}(x), T(f(x),y)\}$ and $B = \{P(c,g(w)), \bar{Q}(a)\}$, then $[\{\bar{Q}(x), T(f(x),y)\} \cup \{\bar{Q}(a)\}]\{c/x, g(w)/y\} = \{\bar{Q}(c), T(f(c),g(w)), \bar{Q}(a)\}$ would be a resolvent of A and B.

Note that for ground clauses, no substitutions are possible, and hence, the complementary literals $L, \bar{L}$ must appear in A and B.

CHAPTER III

GROUND SL-RESOLUTION

In this section, we define the notion of SL-resolution for ground clauses.

In SL-resolution, each clause is written in the form of a chain (or ordered list), which is a sequence of literals arranged in some predefined order. Each literal in a chain is assigned a status of either A-literal or B-literal. All literals in the input chain are initially assigned the status of B-literal. Two literals are said to be in the same cell if there are no A-literals between them.

If C is a clause, a chain obtained from C will be denoted by C^*. S^* denotes a set of chains obtained from a set of clauses S.

An SL-derivation of C^* from S^*, for a given support subset S' of S, is a sequence $\underset{\sim}{D}^* = (C_1^*, \cdots, C_n^*)$, where C_1^* is an input chain in S'^*, and $C_n^* = C^*$. For every i, such that, $1 \leq i < n$,

(1) C_{i+1}^* is obtained from C_i^* by either truncation, reduction, or expansion, and

(2) if C_{i+1}^* is not obtained from C_i^* by reduction, then, no two B-literals occurring in distinct cells in C_i^* have the same atom, and no pair of A- and B-literals are complementary.

C^*_{i+1} is obtained from C^*_i by truncation if the leftmost literal is an A-literal and if C^*_{i+1} is C^*_i with this A-literal and all immediately adjacent A-literals deleted. The status of a literal in C^*_{i+1} is the same as its status in C^*_i.

C^*_{i+1} is obtained from C^*_i by reduction if the leftmost literal in C^*_i is a B-literal and

(1) C^*_i is not obtained from C^*_{i-1} by truncation,

(2) C^*_{i+1} is obtained from C^*_i by deleting a B-literal, L, in the leftmost cell of B-literals in C^*_i, and,

(3) (merging) L is identical to a B-literal in some nonleftmost cell of B-literals in C^*_i or
(ancestor resolution) L is complementary to some A-literal of C^*_i.

The literals in C^*_{i+1} have the same status as the corresponding literals in C^*_i.

C^*_{i+1} is obtained from C^*_i by expansion with B^* in S^* if the leftmost literal in C^*_i is a B-literal and the following conditions hold.

(1) The leftmost literal L in C^*_i is complementary to some literal $\bar{L}$ in B^*.

(2) Let C^*_{i0} be obtained from C^* by deleting L, and let B^*_0 be obtained from B^* by deleting $\bar{L}$. Then C^*_{i+1} is the chain $B^*_0 \square L C^*_{i0}$ obtained by concatenating B^*_0, L, and C^*_{i0} in that order. The literals in C^*_{i0} and B_0 have the status of the corresponding literals in C^*_i and B^*.

Figure 1 illustrates a resolution deduction of □ and the corresponding SL-deduction. Input chains are exhibited on the left of the appropriate chain. For notational convenience, A-literals in D* are enclosed within individual square boxes. Chains in D* are written without curly brackets and separating commas.

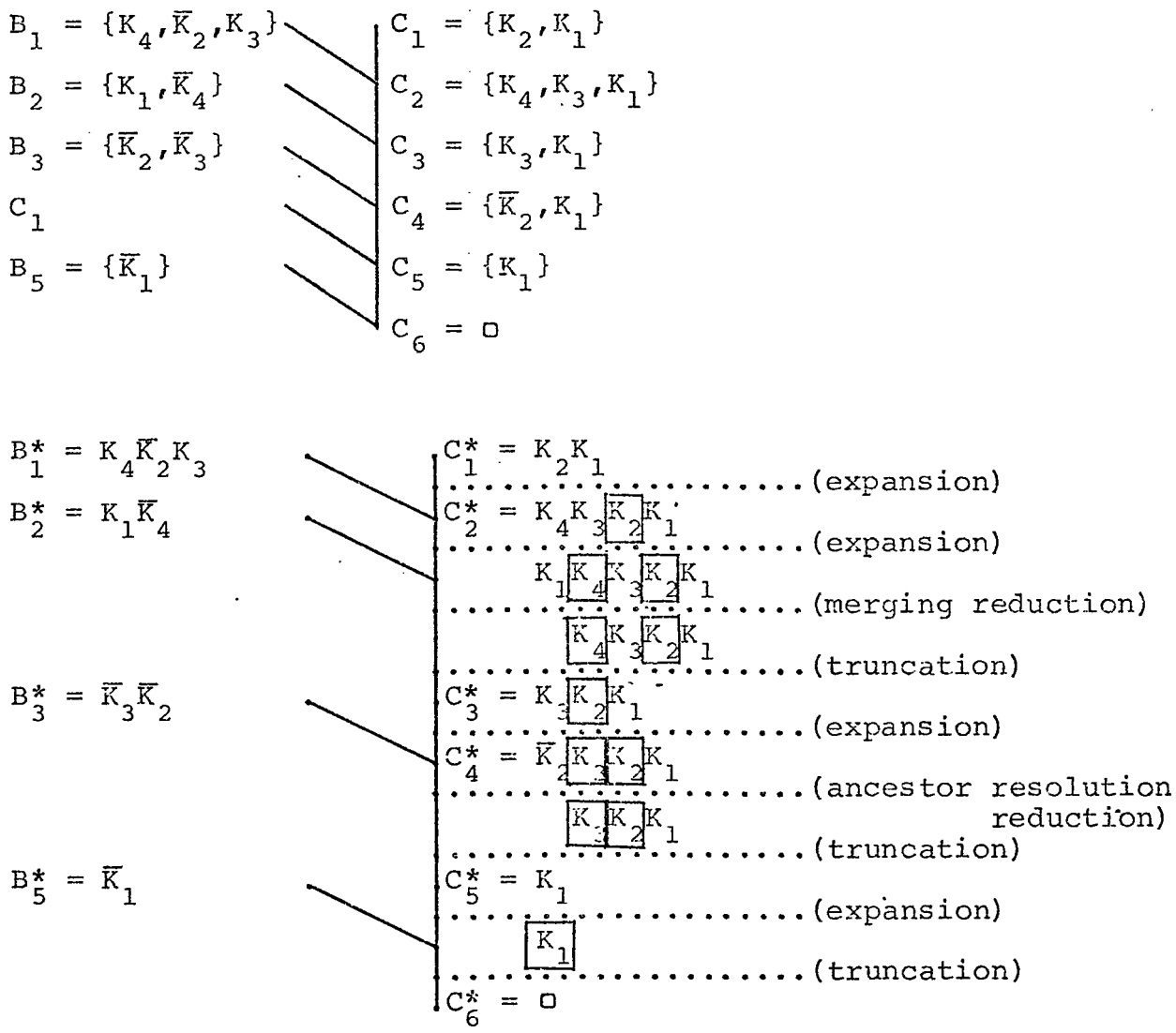


Figure 1.

CHAPTER IV

COMPLETENESS OF GROUND SL-RESOLUTION

In [1], Kowalski and Kuehner give a proof that ground SL-resolution is complete. That proof is quite long and complicated. In this section we will give a somewhat shorter completeness proof using the techniques developed in [2].

The following terminologies are used in the proof of the completeness of ground SL-resolution.

5.1 A ground clause C is said to subsume a ground clause D if every literal in C is also in D.

5.2 If S is a set of (ground) clauses, then "M is a model for S" means that M is a set of literals which does not contain any complementary pair of literals (L and $\bar{L}$) and such that each clause C in S contains at least one literal which is in M. Thus a model is a ground interpretation of a set of clauses, S, which makes each of the clauses in S true, i.e., which satisfies S.

5.3 If S is a set of (ground) clauses, then S is said to be minimally unsatisfiable if S is unsatisfiable and for each clause $C \in S$, $S \setminus \{C\}$ is satisfiable, i.e., the deletion of any clause from S leaves a satisfiable set of clauses.

Before presenting the proof of the main theorem, the results of the following lemmas are needed.

Lemma 1. If S is any finite, unsatisfiable set of ground clauses, then S may be reduced to a subset $S' \subseteq S$, such that, S' is minimally unsatisfiable.

Proof. Assume for contradiction that for every finite unsatisfiable subset S_1 of S, we can always find an unsatisfiable set S_2, such that, $S_2 \subsetneq S_1 \subset S$.

That is, we can construct a sequence of subsets of S, such that $S_n \subsetneq S_{n-1} \subsetneq \cdots \subsetneq S_2 \subsetneq S_1 \subsetneq S$, for some n such that S_n contains only one clause. But there is always an interpretation for a single clause, contradicting our assumption that it is unsatisfiable.

Lemma 2. If S is any minimally unsatisfiable set of ground clauses and L is any literal which occurs in some clause in S, then $\bar{L}$ must occur in some other clause in S.

Proof. Suppose for contradiction that there exists a literal, L, which occurs in a clause C in S but that the literal $\bar{L}$ does not occur in any clause in S. The set S', where $S' = S - \{C\}$ is satisfiable, by the minimal unsatisfiability of S. Let M' be a model for S'. Since the literal $\bar{L}$ does not occur in S', we may assume, without loss of generality, that $\bar{L} \notin M'$. But then $M = M' \cup \{L\}$ is a model of S which satisfies S, contradicting the fact that S is unsatisfiable.

Theorem. If S is any minimally unsatisfiable set of ground clauses and C is any clause in S, then there is an SL-deduction of $\square$ from S^* with top chain C^*.

Proof. For any set S of clauses, let $k(S)$ be the total number of appearances of literals in S minus the number of clauses in S.

The proof is by mathematical induction $k(S)$.

When $k(S) = 0$, S must consist of only two complementary unit clauses, since S is minimally unsatisfiable. $\square$ can be deduced from S^* by expansion and truncation, independent of the choice of either of the two chains in S^* as the top chain.

Assume that it is true that for any minimally unsatisfiable set S, of clauses such that $k(S) < N$, $N > 0$, $\square$ can be deduced from it by SL-deduction with any chain C^* in S^* as top chain.

Now assume that S is a minimally unsatisfiable set of clauses with $k(S) = N$. We have two cases:

Case 1: The top chain C^* is a unit chain.

Case 2: The top chain C^* is not a unit chain.

Proof of Case 1. Let $C = (L)$, and $(S - \{C\})^d$ be the set of clauses obtained from $S-\{C\}$ by deleting all occurences of the literal $\bar{L}$ from them. We claim that $(S-\{C\})^d$ is unsatisfiable. Suppose for contradiction that this is not true. Let M be a model satisfying $(S-\{C\})^d$. Since $\bar{L}$ is not in any clause

in $(S-\{C\})^d$, we may assume without loss of generality, that $\bar{L} \notin M$. But $MU\{L\}$ is a model satisfying S , contradicting that S is unsatisfiable.

For a minimally unsatisfiable subset $(S-\{C\})^{dm}$ of $(S-\{C\})^d$, $k(S-\{C\})^{dm} < N$, since there must exist $\bar{L} \in (S-\{C\})$ by lemma 2. Hence, by induction hypothesis, $\square$ can be deduced from it by SL-deduction. In particular, $\square$ can be deduced by SL-deduction with top chain $(A)^*$ (see figure 2(a)), where A was obtained from a clause $(A \vee \bar{L})$ in S-{C} by deleting the literal $\bar{L}$. For if there is no such clause (A) in $(S-\{C\})^{dm}$, then $(S-\{C\})^{dm}$ will be a subset of S-{C} which is satisfiable.

Now, using $C^* = (L)^*$ as the top chain, we can obtain $(A \vee \boxed{L})^*$, where $\boxed{L}$ means that L is an A-literal. Add $(\bar{L})$ back to all chains from which it was deleted except for the original topmost chain $(A)^*$. Every time that $\bar{L}$ is added back, it is deleted from the resolvent by ancestor resolution reduction, so that we are left with the A-literal L which yields $\square$ by truncation (see figure 2(b)).

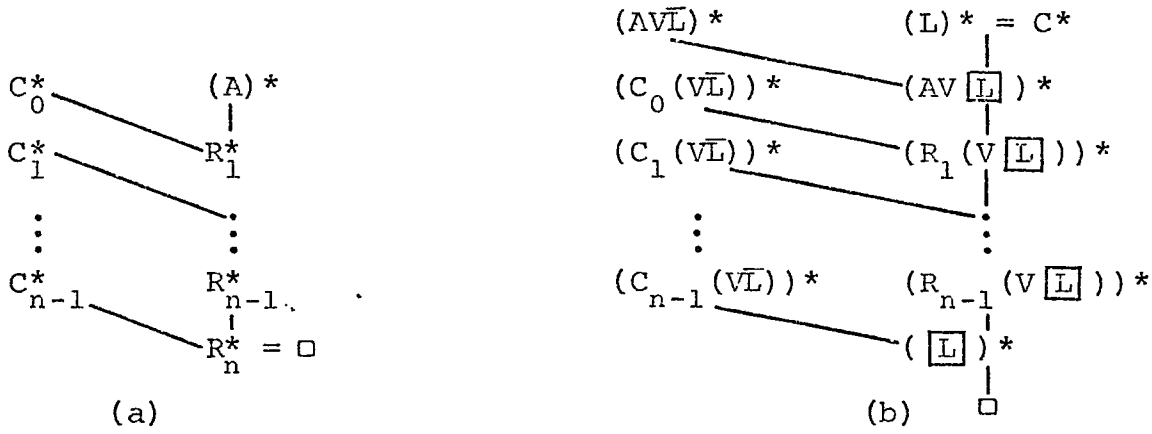


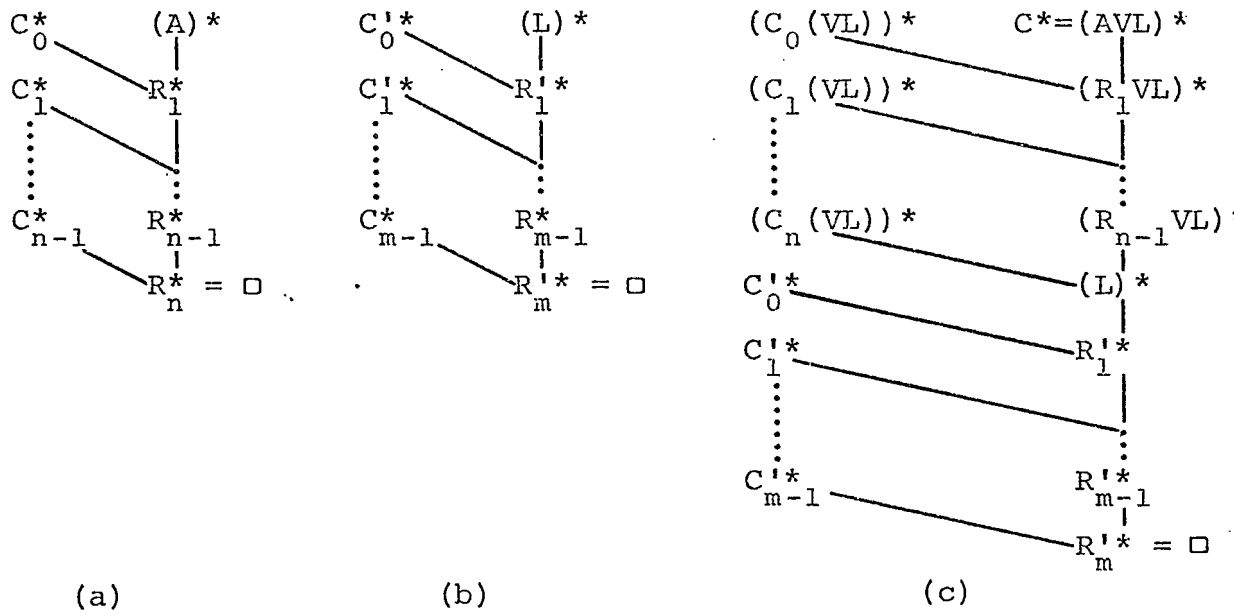
Figure 2. (a) SL-deduction of $\square$ from $(S-\{C\})^{dm*}$ with $(A)^*$ as top chain, obtained by induction hypothesis. (b) SL-deduction of $\square$ from S^* with C^* as top chain, obtained by adding $\bar{L}$ back to clauses from which it was deleted. (The symbol $(V\bar{L})$ means to add the $\bar{L}$ back to those clauses from which it was deleted.)

Proof of Case 2: Since the top chain C^* is not a unit chain, it can be written as $(AVL)^*$, where A is a nonempty disjunction of literals. Since S is minimally unsatisfiable, let M be a model which satisfies $S-\{C\}$. M does not satisfy S and therefore $L \notin M$. Now let S^d be the set of clauses obtained by deleting all occurences of the literal L from clauses in S. Since $L \notin M$ we have that M still satisfies $(S-\{C\})^d$. Therefore when S^d is reduced to a minimally unsatisfiable set of clauses, we can be sure that the clause (A) is in S^{dm} (for otherwise S^{dm} is a subset of $(S-\{C\})^d$ which is satisfiable). The minimally unsatisfiable subset, S^{dm}, of S^d, has $k(S^{dm}) < N$.

Hence, by the induction hypothesis, we can obtain an SL-deduction of □ from it. In particular, we can obtain this deduction with top chain (A)*, as shown in figure 3(a).

Now let $S' = (S-\{C\})\cup\{(L)\}$. S' is still unsatisfiable, since (L) subsumes C. Let S'^m be a minimally unsatisfiable subset of S'. $(L)\in S'^m$, for otherwise, S'^m is a subset of the satisfiable set $S-\{C\}$. Hence, we have a minimally unsatisfiable set of clauses S'^m with $k(S'^m) < N$. By the induction hypothesis, □ can be deduced by SL-deduction with (L)* as top chain as shown in figure 3(b).

When using $C^* = (AVL)^*$ as top chain, add L back to all clauses from which it was deleted. Every time that L is added back, it is deleted from the resulting chain by merging reduction, so that we are left with B-literal L which will yield □ (see figure 3(c)), by the deduction shown in figure 3(b).



(a) (b) (c)

Figure 3. (a) SL-deduction of □ from $S^{dm}*$ with top chain (A)*, obtained by the induction hypothesis. (b) SL-deduction of □ from $S'^{m}*$ with (L)* as top chain, obtained by the induction hypothesis. (c) SL-deduction of □ from S* with top chain C* , obtained by placing the last two deductions one after the other.

CHAPTER V

GENERAL SL-RESOLUTION

For a given set of input clauses S, a set S^* of input chains for S consists of exactly one sequence $(C\theta)^*$ for each factor $C\theta$ of a clause C in S. $(C\theta)^*$ is an ordered list of all literals in $C\theta$. A chain is a sequence of literals each one of which is assigned the status of A- or B-literal. All literals in input chains are B-literals. If there are no A-literals between two B-literals of a chain, then the two B-literals are in the same cell of the chain.

For a given set of clauses S, a support subset S' of S, an SL-derivation of C^* from S'^* is a sequence of chains $\underset{\sim}{D}^* = (C_1^*, \cdots, C_n^*)$, where C_1^* is in S'^*, and $C_n^* = C^*$. For $1 \leqslant i < n$,

(1) C_{i+1}^* is obtained from C_i^* by either truncation, reduction, or expansion, and

(2) if C_{i+1}^* is not obtained from C_i^* by reduction, then no two B-literals occurring in distinct cells of C_i^* have the same atom and no pair of A- and B-literals are complementary literals.

C_{i+1}^* is obtained from C_i^* by truncation if

(i) the leftmost literal in C_i^* is an A-literal, and

(ii) C_{i+1}^* is the longest subsequence of C_i^* whose leftmost literal is a B-literal. The status of a literal in C_{i+1}^* is the same as its status in C_i^*.

C^*_{i+1} is obtained from C^*_i by reduction if the following conditions hold.

(i) The leftmost literal in C^*_i is a B-literal.

(ii) C^*_i is not obtained from C^*_{i+1} by truncation

(iii) The leftmost cell of C^*_i contains a B-literal L and either C^*_i contains a B-literal K which is not in the leftmost cell of C^*_i or C^*_i contains an A-literal $\overline{K}$ and that $\{K,L\}$ is unifiable with most general unifier θ,

(iv) Let C^*_{i0} be obtained by deleting the given occurrence of L in C^*_i. Then $C^*_{i+1} = C^*_{i0}\theta$. A literal in C^*_{i+1} has the same status as the literal in C^*_i from which it descends.

C^*_{i+1} is obtained from C^*_i by expansion with B^* in S'^* if the following conditions hold.

(i) The leftmost literal in C^*_i is a B-literal.

(ii) Let L be the leftmost literal in the leftmost cell of C^*_i. B^* contains a literal K such that $\{L,K\}$ is unifiable with a most general unifier θ.

(iii) Let C^*_{i0} and B^*_0 be obtained by deleting the given occurrence of L in C^*_i and K in B^*. C^*_{i+1} is the chain $B^*_0\theta L\theta C^*_{i0}\theta$ obtained by concatenating $B_0\theta$, $L\theta$ and C^*_{i0} in that order. $L\theta$ is an A-literal in $(B^*_0 L C^*_{i0})\theta$. Every other literal in C^*_{i+1} has the same status as the literal in B^*_0 or C^*_{i0} from which it descends.

CHAPTER VI

COMPLETENESS OF GENERAL SL-RESOLUTION

For the proof of the completeness of general SL-resolution, we make use of the results of Skolem-Herbrand-Gödel Theorem, and the ground completeness theorem that we proved in Chapter IV.

In preparation for the Skolem-Herbrand-Gödel Theorem, we need to define the Herbrand Universe.

6.1 The Herbrand Universe. With any set S of clauses, there is associated a set of ground terms called the Herbrand universe of S, as follows: Let F be the set of all function symbols and constants which occur in S. If F contains any constants, the functional symbol vocabulary of S is F; otherwise, it is the set $\{a\}UF$. The Herbrand universe of S is then the set of all ground terms in which there occur only symbols in the functional vocabulary of S.

The following proof of Skolem-Herbrand-Gödel Theorem is taken from [4].

Skolem-Herbrand-Gödel Theorem. If S is unsatisfiable, then some finite set of ground instances of clauses in S is unsatisfiable.

Proof. Let $A_1, A_2, A_3, \cdots$ be an enumeration of all ground instances over the Herbrand universe of S of atoms in S. Let $C_1, C_2, C_3, \cdots$ be an enumeration of all ground instances over the Herbrand universe of S of clauses in S.

If S is unsatisfiable, then, in particular, no "symbolic" interpretation, with the Herbrand universe of S satisfies S. Therefore no assignment of truth values to the atoms $A_1, A_2, \cdots$ will make every $C_1, C_2, \cdots$ true and therefore every branch in the tree T terminates (see figure 4). But, if each branch of a tree which splits into at most two branches at each node terminates, there can be only finitely many nodes in the tree. Therefore, T has only finitely many nodes. The indices $j_1, j_2, \cdots, j_N$ written below these terminal nodes then provide a finite set $C_{j_1}, C_{j_2}, \cdots, C_{j_N}$ of instances of clauses in S which are unsatisfiable. For if M is the largest index among those of the atoms A_i which actually occur in the instances $C_{j_1}, C_{j_2}, \cdots, C_{j_N}$, then each of the 2^M distinct assignments of truth values to $A_1, A_2, \cdots, A_M$ will falsify at least one instance in the set $C_{j_1}, C_{j_2}, \cdots, C_{j_N}$.

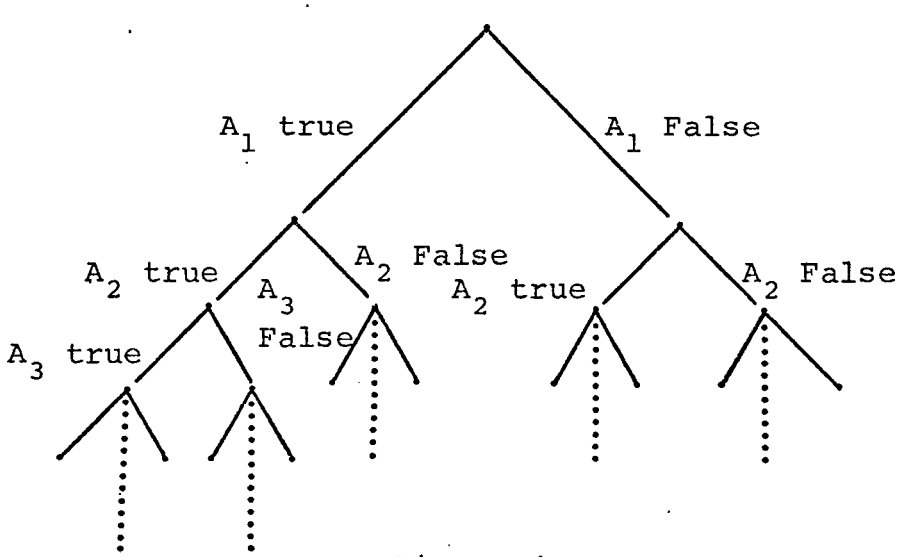


Figure 4

We are now in a position to prove the completeness theorem for general SL-resolution.

Theorem. If S is any minimally unsatisfiable set of general clauses, and C is any clause of S, then there is an SL-deduction of $\square$ from S_f^*, where S_f is the set of all factors of S, with top chain C_f^*, where C_f is either C or some factor of C.

Sketch of proof: By Skolem-Herbrand-Gödel Theorem, there exists some finite subset of ground instances of clauses in S that is unsatisfiable. From this unsatisfiable subset, we take a minimally unsatisfiable subset, S'. Since S' is minimally unsatisfiable, it must be that each clause in S is represented by at least one ground instance in S'. Let C' be the clause in S' that is a ground instance of C. Then, by the ground completeness theorem, we can deduce $\square$ from C'^* by SL-resolution.

We need to show that a corresponding deduction of $\square$ can be constructed for the general clauses which exactly "mimics" the ground level proof by SL-resolution from the set of all factors of S, S_f.

Before we proceed with the discussion, we need to state some facts about most general unifiers. They are as follows:

If there exists any substitution α which unifies two literals L_1, L_2 (i.e., such that $L_1\alpha = L_2\alpha$), then the most general unification algorithm will find a most general unifier

σ which will unify L_1 and L_2 such that $L_1\alpha$ and $L_2\alpha$ are instances of $L_1\sigma$, $L_2\sigma$, i.e., there exists another substitution λ such that $(L_1\sigma)\lambda = L_1\alpha = L_2\alpha = (L_2\sigma)\lambda$.

From this, one can see, for example, that if a clause C' is a ground instance of C , then there exists some factor C_f of C such that C_f and C' have the same number of literals and C' is an instance of C_f . For example, if $C = (P(x)VP(f(y))VQ(y))$ and $C' = (P(f(a))VQ(a))$, note that C has the factor $C_f = (P(f(y))VQ(y))$ with the above properties.

Since C' , the top clause in the ground case, is a ground instance of C , by the above argument, we can use C_f as the general top clause, where C_f is either C or that factor of C which contains the same number of literals as C' .

Each input chain D'^* in the ground proof is an instance of some chain D^* in S^* . By the same reasoning, there exists some D_f^* in S_f^* such that D_f is a factor of D and such that D' is a ground instance of D_f with the same number of literals. We can use D_f^* as the corresponding input chain in the proof for the general case.

When the operation of expansion is applied to the ground top clause and a ground input clause, a corresponding expansion operation takes place for the general top clause, and the corresponding general input chain. Because the ground top clause is a ground instance of the general top clause with the same number of

literals, the ground input clause is a ground instance of the corresponding general input clause or some factor of it having the same number of literals, and because a most general unifier is found by the most general unification algorithm called upon by expansion in the general case, the resulting chain obtained in the ground case is a ground instance of that obtained in the general case with the same number of literals.

Similarly, all subsequent expansion operations in the ground case yield corresponding expansion operations in the general case. The resulting chain in the ground case is a ground instance of that in the general case with the same number of literals.

A reduction operation in the ground case has a corresponding reduction operation in the general case. Because the ground chain being operated upon by reduction is a ground instance of the general chain operated upon by the corresponding reduction with the same number of literals, and reduction in the general case calls upon the most general unifier algorithm, the resulting ground chain is a ground instance of that in the general case with the same number of literals.

A truncation operation in the ground case has a corresponding truncation operation in the general case. Because the ground chain to be operated upon by truncation is a ground instance of the general chain to be operated upon by the

corresponding truncation with the same number of literals, the resulting ground chain is a ground instance of the resulting general chain with the same number of literals.

The following example demonstrates the above discussion.

$$\text{Input set} = \{ \underset{A}{(\bar{S}(x) \vee \bar{S}(e))}, \underset{B}{(S(x) \vee S(y) \vee \bar{P}(x, I(z), e))},$$

$$\underset{C}{(S(x) \vee P(y, w, y) \vee P(e, I(z), e))} \}.$$

with $(\bar{S}(x) \vee \bar{S}(e))$ as top clause.

The ground proof of figure 5(a) will have a corresponding general proof of figure 5(b).

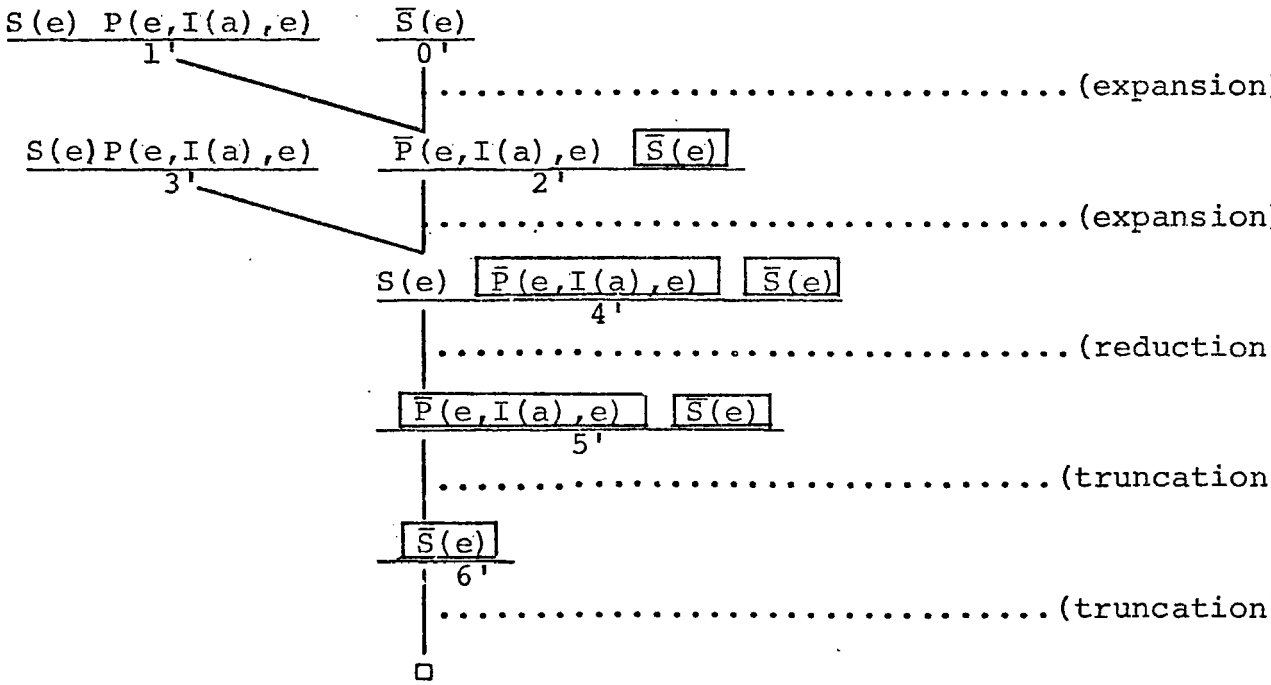


Figure 5(a)

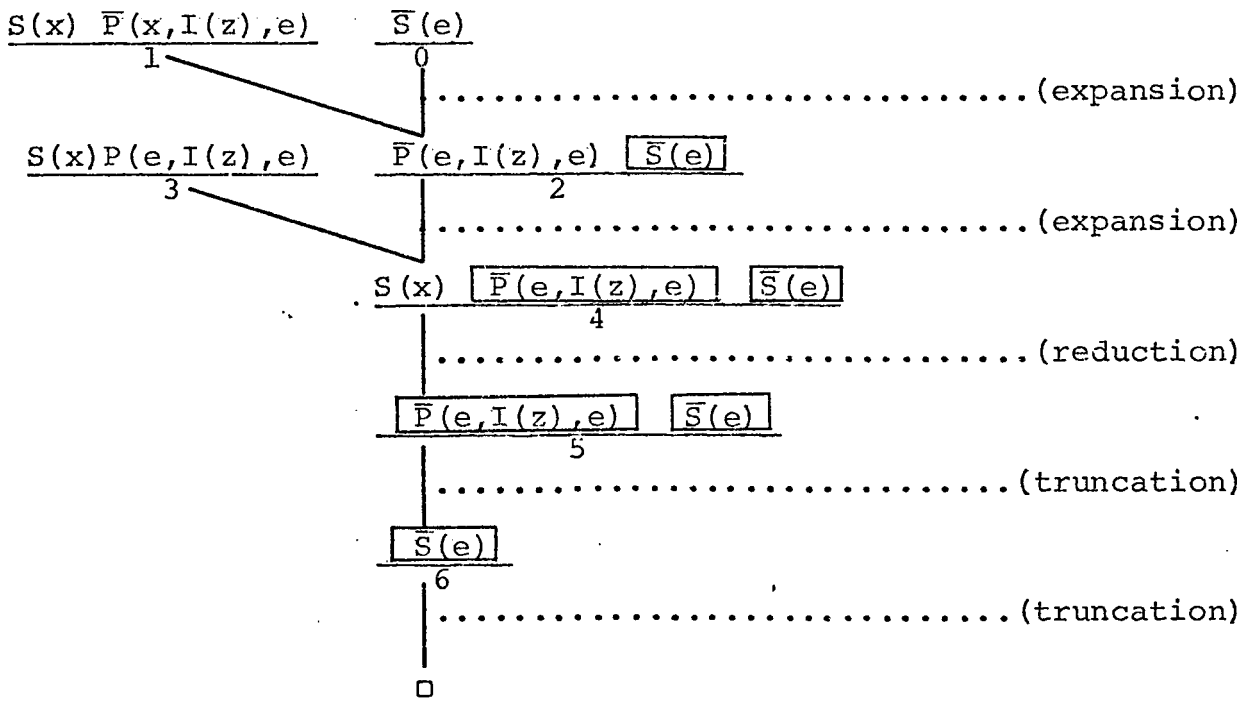


Figure 5(b)

0' is a ground instance of chain 0 with the same number of literals.

0 is a factor of A .

1' is a ground instance of chain 1 with the same number of literals.

1 is a factor of C .

2' is a ground instance of 2 with the same number of literals.

3' is a ground instance of 3 with the same number of literals.

3 is a factor of C .

4' is a ground instance of 4 with the same number of literals.

5' is a ground instance of 5 with the same number of literals.

6' is a ground instance of 6 with the same number of literals.

Theorem. If S is any unsatisfiable set of general clauses, and S' a support set (i.e., $S' \subseteq S$ and $S \setminus S'$ is satisfiable), then there is an SL-deduction of $\square$ from S_f^* with top chain C^*, some chain in $S_f'^*$.

Proof. S can be reduced to a minimally unsatisfiable subset $S^M \subseteq S$. Since $S \setminus S'$ is satisfiable, S^M must contain at least one clause, C, from S'. By the preceding theorem, $\square$ can be deduced from $S_f^{M*} \subseteq S_f^*$ by SL-deduction with top chain C_f^*, some chain in $S_f'^*$.

CHAPTER VII

COMPUTER IMPLEMENTATION AND RESULTS

Instead of giving a first proof directly, the program generates a search space of derivations. This is because the tree that is going to yield the first proof is unknown. Every chain to be expanded upon has to be expanded with every chain in the input set, thus creating the search space of derivations.

The following representations are used in the program.

A variable symbol is represented by an atom.

A constant symbol is represented by a list containing only one atom.

A function is represented by a list the first member of which is an atom representing the function symbol and the rest are terms.

A literal is represented by a list, the first member of which is the literal symbol and the rest are terms. If L is an atom, then L is represented by (NIL L).

A clause is represented by a list of literals.

A chain is represented as a clause with the status of each literal tagged to it. For example, the chain, P(f(x)) $\boxed{Q(y,g(c))}$ R(d), is represented as ((B(P(F X)))(A(Q Y(G(C))))(R(R(D)))).

The function MAIN is defined with two arguments CO and L. CO is the top clause with its length tagged to it. L is the

list of input clauses each tagged with its length. MAIN returns NIL if the set of input clauses is unsatisfiable and may otherwise not terminate.

The algorithm for the program is as follows:

1. Assign to each literal in the top clause a status. (In doing this, the clause is converted into a chain.)
2. The chain is put into the appropriate list in {S(1),S(2),···, S(20)}. For example, a chain of length 5 is put into S(5).
3. Check for the first nonempty list, say, S(K), in this order, S(1),S(2),···,S(20).
4. Take the first chain C(K) in S(K).
5. Check if C(K), using its first literal to be unified upon, is unifiable with the first untried chain CIN in the input list.
6. If not, go to 12.
7. If so, apply expansion to C(K), and CIN.
8. Apply reduction and truncation to the result.
9. If result is NIL, terminate.
10. If not, calculate the complexity of the resulting chain
    Complexity 1 = K + length of CIN-1 .
    Complexity 2 = K + 2 × length of CIN-3 .
11. Append resulting chain to appropriate list of S(1),S(2),···, S(20), according to its complexity.
12. Is CIN the last one in the input list?
13. If not, go to 5.

14. If so, delete C(K) from S(K).

15. Go to 3.

MAIN calls upon the following functions:

MAIN1, the inputs of which are S1, S2, •••, S20, L , where L is the list of input clauses. It returns the value of ERT(S1, S2, •••, S20, U, L1, L2), where U is the length of the first chain, L1, of the first nonempty list of S1, S2, •••, S20 , in that order; L2 is the list of input clauses.

ERT, the inputs of which are S1, S2, •••, S20, U, L1, L2 as described above. For each of the input clauses in L2, it calls on expansion, reduction, and truncation, with L1 as the chain to be expanded upon. The resulting chain, if it exists, is appended to the appropriate list of S1, •••, S20 , according to its complexity. It deletes L1 from its list before it returns the value of MAIN1(S1, S2, •••, S20, L), where L is the list of input clauses.

STRIP, the input of which is a unit clause, and returns the same clause, if it has a positive literal, and the clause with the negation sign of the literal deleted, otherwise.

UNIFY1, the inputs of which are two positive literals, X and Y . It increments the counter UCNT by one and returns the value of UNIFY(X,Y).

UNIFY, the inputs of which are 01 and 02 which are the same as those of UNIFY1. It returns the value T , if 01 and 02 are

unifiable, and the value F , otherwise. It should be noted that UNIFY works for literals of at most three terms.

OCCUR, the inputs of which are X and Y , where X is a variable, and Y is a term. It returns the value T , if X appears in the substituted value of Y (or in Y , if Y is not substituted for), and the value F , otherwise.

RECOVER, the input of which is X , a list. It returns the list with substitutions for the symbols that need to be substituted for, and generates new symbols for those that do not.

RECOVER1, the input of which is X , a list, and returns the value of RECOVER (CAR(X)).

SUBSTITUTE, the inputs of which are L1, and L2, where L1 is a list of chains, and L2 is the empty list, when first called. It returns as its value, the list of chains with appropriate substitutions.

EXPANSION, the inputs of which are L1, L2, L3, where L1 is a chain, L2 is a clause, and L3 is the literal, with its status specified, that is to be expanded upon. It returns the value of COMBINE (L1, L2, (CADR L3), U, NIL), where U is the complementary literal of L3.

COMBINE, the inputs of which are L1, L2, L3, L4 and L5, where L1 is a chain, L2 is a clause, L3 is the literal to be expanded upon, L4 is the complementary literal of L3, and L5 is the empty list, when first called. It checks to see if L1 and L2 can be

expanded, using L3 as the literal to be expanded upon. If not, it returns NIL. If so, it returns the resulting chain.

COMBINE1, the inputs of which are L1, L2, L3, and L4, where L1 is a chain, L2 is a clause, L3 is the literal to be expanded upon, and L4 is the list of all those literals in front of the complementary literal of L3 in L2. It returns the value of the resulting chain of L1, and L2 with L3 as the literal to be expanded upon.

EXPANSIONC, the inputs of which are L1, L2, and L3, where L1 is a chain, L2 is a literal, and L3 is the empty list, when first called. It returns, as its value, the list L3 with L2 deleted.

ADD, the inputs of which are L1 and L2, where L1 is either 'A or 'B , and L2 is a literal.

REDUCTION, the inputs of which are L1, L2, and L3. L1 is a chain. L2 is the list of all literals of L1 except the leftmost cell deleted. L3 is the empty list, when first called. It applies ancester and merging reductions to every literal in the leftmost cell of L1, and returns the resulting chain as its value.

PRELIT, the input of which is L1, a chain, and returns L1 with its leftmost cell deleted.

LASTBS, the inputs of which are L1 and L2, where L1 is a chain and L2 is the empty list, when first called. It returns the leftmost cell of L1 as its value.

NEGATE, the input of which is L , a literal with its status tagged to it. It returns the complementary literal of L with the same status.

TRUNCATION, the input of which is L , a chain. It returns L after applying the operation of truncation to it.

A set of fourteen examples (see appendix), already proved in [13] and [3], was tested twice, each time using a different complexity.

Complexity 1 = (previous complexity of chain)
+ (length of input chain) - 1 .

The second complexity gives more weight to the length of the input clause.

Complexity 2 = (previous complexity of chain)
+ 2 * (length of input chain) - 3 .

The results are shown in the following table.

Examples	Number of ERT Calls		Execution Time in Sec.	
	Complexity 1	Complexity 2	Complexity 1	Complexity 2
1	6	5	13.945	12.989
2	memory exhausted	memory exhausted	29.032	52.170
3	memory exhausted	memory exhausted	40.012	31.175
4	memory exhausted	memory exhausted	51.623	45.063
5	4	4	8.396	11.852
6	memory exhausted	memory exhausted	30.810	36.478
7	11	11	9.890	15.249
8	memory exhausted	memory exhausted	20.724	60.313
9	30	28	28.995	25.328
10	memory exhausted	memory exhausted	39.848	33.452
11	memory exhausted	memory exhausted	42.415	60.643
12	memory exhausted	memory exhausted	33.432	60.990
13	memory exhausted	memory exhausted	30.495	27.361
14	9	5	19.138	14.632

Most of the preceding 14 examples are quite difficult for automatic theorem-proving programs. Some have only been proved by a program which uses an incomplete proof procedure [13]. However, all 14 have been proved by one or more programs. Thus, the results of this particular technique are somewhat disappointing and do not live up to the claims made by Kowalski and Keuhner in this paper (although we did not implement all aspects of their technique). Neither complexity tested is clearly superior to the other. Further experimentation with different complexities would be useful.

BIBLIOGRAPHY

1. Kowalski, R., and Kuehner, D. Linear resolution with selection function. Memo 34, Metamathematics Units, Edinburgh University, Scotland, October 1970.
2. Anderson, R., and Bledsoe, W. W. A linear format for resolution with merging and a new technique for establishing completeness. JACM 17, 3, July 1970, 525-534.
3. Robinson, J. A. A machine-oriented logic based on the resolution principle. JACM 12, 1, January 1965, 23-41.
4. Robinson, J. A. A review of automatic theorem-proving. Proceedings of Symposia in Applied Mathematics, Vol. 19, 1-18.
5. Robinson, J. A. Automatic deduction with hyper-resolution. Int'l. J. of Comp. Math. Vol. 1, 1965, 227-234.
6. Wos, L.; Robinson, G.; and Carson, D. Efficiency and completeness of the set of support strategy in theorem-proving. JACM, 12, 4, October 1965, 536-541.
7. Slagle, J. Automatic theorem-proving with renamable and semantic resolution. JACM 14, 4 October 1967, 687-697.
8. Guard, J., et al. Semi-automated maths. JACM 16, 1, January 1969, 26-48.
9. Green, C. Application of theorem-proving to problem solving. Proc. Int'l. Joint Conference on Artificial Intelligence, Washington, D.C., May 1969.

10. Green, C. Theorem-proving by resolution as a basis for Question-Answering Systems. Machine Intelligence 4, B. Meltzer, and D. Michie (eds.), American Elsevier Publishing Co., Inc., New York, 1969, 183-205.

11. Manna, Z. The correctness of programs. J. of Computer and System Sciences, Vol. 3, May 1969.

12. Nilsson, N. J. Problem-solving methods in artificial intelligence. August 1970, VI-VIII.

13. Chang, C. L. The unit proof and the input proof in theorem-proving. JACM 17, 4 October, 1970, 704-706.

APPENDIX

```
   (CSETQ UCNT 0)
(DEFINE '(
(MAIN (LAMBDA (CO L)
   (PROG (S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S16 S17
   S18 S19 S20)
   (COND
   ((EQUAL (CAR CO) '1) (SETQ S1 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '2) (SETQ S2 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '3) (SETQ S3 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '4) (SETQ S4 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '5) (SETQ S5 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '6) (SETQ S6 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '7) (SETQ S7 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '8) (SETQ S8 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '9) (SETQ S9 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '10) (SETQ S10 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '11) (SETQ S11 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '12) (SETQ S12 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '13) (SETQ S13 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '14) (SETQ S14 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '15) (SETQ S15 (LIST (ADDB (CADR CO) NIL)))
                        (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                         S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
```

```
   ((EQUAL (CAR CO) '16) (SETQ S16 (LIST (ADDB (CADR CO) NIL)))
                         (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                          S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '17) (SETQ S17 (LIST (ADDB (CADR CO) NIL)))
                         (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                          S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '18) (SETQ S18 (LIST (ADDB (CADR CO) NIL)))
                         (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                          S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '19) (SETQ S19 (LIST (ADDB (CADR CO) NIL)))
                         (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                          S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ((EQUAL (CAR CO) '20) (SETQ S20 (LIST (ADDB (CADR CO) NIL)))
                         (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10
                          S11 S12 S13 S14 S15 S16 S17 S18 S19 S20 L)))
   ))))
(MAIN1 (LAMBDA (S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15
                S16 S17 S18 S19 S20 L)
   (COND
   ((NOT (NULL S1)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                         S15 S16 S17 S18 S19 S20 '1 (CAR S1) L))
   ((NOT (NULL S2)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                         S15 S16 S17 S18 S19 S20 '2 (CAR S2) L))
   ((NOT (NULL S3)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                         S15 S16 S17 S18 S19 S20 '3 (CAR S3) L))
   ((NOT (NULL S4)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                         S15 S16 S17 S18 S19 S20 '4 (CAR S4) L))
   ((NOT (NULL S5)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                         S15 S16 S17 S18 S19 S20 '5 (CAR S5) L))
   ((NOT (NULL S6)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                         S15 S16 S17 S18 S19 S20 '6 (CAR S6) L))
   ((NOT (NULL S7)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                         S15 S16 S17 S18 S19 S20 '7 (CAR S7) L))
   ((NOT (NULL S8)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                         S15 S16 S17 S18 S19 S20 '8 (CAR S8) L))
   ((NOT (NULL S9)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                         S15 S16 S17 S18 S19 S20 '9 (CAR S9) L))
   ((NOT (NULL S10)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '10 (CAR S10) L))
   ((NOT (NULL S11)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '11 (CAR S11) L))
   ((NOT (NULL S12)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '12 (CAR S12) L))
   ((NOT (NULL S13)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '13 (CAR S13) L))
   ((NOT (NULL S14)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '14 (CAR S14) L))
   ((NOT (NULL S15)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '15 (CAR S15) L))
   ((NOT (NULL S16)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '16 (CAR S16) L))
```

```
   ((NOT (NULL S17)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '17 (CAR S17) L))
   ((NOT (NULL S18)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '18 (CAR S18) L))
   ((NOT (NULL S19)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '19 (CAR S19) L))
   ((NOT (NULL S20)) (ERT S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14
                          S15 S16 S17 S18 S19 S20 '20 (CAR S20) L))
   )))
(ERT (LAMBDA (S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S16
                 S17 S18 S19 S20 U L1 L2)
   (PROG (E G L V X Y)
   (SETQ L L2)
 LOOP  (COND
   ((NULL L2) (COND
              ((EQUAL U '1) (RETURN (MAIN1 (CDR S1) S2 S3 S4 S5 S6 S7
                                           S8 S9 S10 S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '2) (RETURN (MAIN1 S1 (CDR S2)S3 S4 S5 S6 S7
                                           S8 S9 S10 S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '3) (RETURN (MAIN1 S1 S2 (CDR S3) S4 S5 S6 S7
                                           S8 S9 S10 S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '4) (RETURN (MAIN1 S1 S2 S3 (CDR S4) S5 S6 S7
                                           S8 S9 S10 S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '5) (RETURN (MAIN1 S1 S2 S3 S4 (CDR S5) S6 S7
                                           S8 S9 S10 S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '6) (RETURN (MAIN1 S1 S2 S3 S4 S5 (CDR S6) S7
                                           S8 S9 S10 S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '7) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 (CDR S7)
                                           S8 S9 S10 S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L)))
              ((EQUAL U '8) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 (CDR S8)
                                           S9 S10 S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '9) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8
                                           (CDR S9) S10 S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '10) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8
                                           S9 (CDR S10)S11 S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '11) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8
                                           S9 S10 (CDR S11)S12 S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
              ((EQUAL U '12) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8
                                           S9 S10 S11 (CDR S12) S13 S14
                                           S15 S16 S17 S18 S19 S20 L )))
```

```
            ((EQUAL U '13) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8
                                          S9 S10 S11 S12 (CDR S13) S14
                                          S15 S16 S17 S18 S19 S20 L )))
            ((EQUAL U '14) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8
                                          S9 S10 S11 S12 S13 (CDR S14)
                                          S15 S16 S17 S18 S19 S20 L )))
            ((EQUAL U '15) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9
                                          S10 S11 S12 S13 S14 (CDR S15)
                                          S16 S17 S18 S19 S20 L )))
            ((EQUAL U '16) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9
                                          S10 S11 S12 S13 S14 S15
                                          (CDR S16) S17 S18 S19 S20 L )))
            ((EQUAL U '17) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9
                                          S10 S11 S12 S13 S14 S15
                                          S16 (CDR S17) S18 S19 S20 L )))
            ((EQUAL U '18) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9
                                          S10 S11 S12 S13 S14 S15
                                          S16 S17 (CDR S18) S19 S20 L )))
            ((EQUAL U '19) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9
                                          S10 S11 S12 S13 S14 S15
                                          S16 S17 S18 (CDR S19) S20 L )))
            ((EQUAL U '20) (RETURN (MAIN1 S1 S2 S3 S4 S5 S6 S7 S8 S9
                                          S10 S11 S12 S13 S14 S15
                                          S16 S17 S18 S19 (CDR S20) L )))
)))
(SETQ E (EXPANSION L1 (CADAR L2) (CAR L1)))
(COND
((NULL E) (SETQ L2 (CDR L2)) (GO LOOP))
(T (ERASE '(X Y))  (SETQ G (TRUNCATION (REDUCTION E (PRELIT E)
NIL)))))
(COND
((NULL G) (RETURN NIL))
(T (SETQ V (SUB1 (PLUS (CAAR L2) U)))))
(COND
((EQUAL V '1) (SETQ S1 (APPEND S1 (LIST G))))
((EQUAL V '2) (SETQ S2 (APPEND S2 (LIST G))))
((EQUAL V '3) (SETQ S3 (APPEND S3 (LIST G))))
((EQUAL V '4) (SETQ S4 (APPEND S4 (LIST G))))
((EQUAL V '5) (SETQ S5 (APPEND S5 (LIST G))))
((EQUAL V '6) (SETQ S6 (APPEND S6 (LIST G))))
((EQUAL V '7) (SETQ S7 (APPEND S7 (LIST G))))
((EQUAL V '8) (SETQ S8 (APPEND S8 (LIST G))))
((EQUAL V '9) (SETQ S9 (APPEND S9 (LIST G))))
((EQUAL V '10) (SETQ S10 (APPEND S10 (LIST G))))
((EQUAL V '11) (SETQ S11 (APPEND S11 (LIST G))))
((EQUAL V '12) (SETQ S12 (APPEND S12 (LIST G))))
((EQUAL V '13) (SETQ S13 (APPEND S13 (LIST G))))
((EQUAL V '14) (SETQ S14 (APPEND S14 (LIST G))))
((EQUAL V '15) (SETQ S15 (APPEND S15 (LIST G))))
((EQUAL V '16) (SETQ S16 (APPEND S16 (LIST G))))
((EQUAL V '17) (SETQ S17 (APPEND S17 (LIST G))))
```

```
   ((EQUAL V '18) (SETQ S18 (APPEND S18 (LIST G))))
   ((EQUAL V '19) (SETQ S19 (APPEND S19 (LIST G))))
   ((EQUAL V '20) (SETQ S20 (APPEND S20 (LIST G)))))
   (SETQ L2 (CDR L2))
   (GO LOOP) )))
(STRIP (LAMBDA (L)
   (COND
   ((ATOM (CAR L)) L)
   (T (CONS (CADAR L) (CDR L))))))
(UNIFY1 (LAMBDA (X Y) (PROG ( )
   (CSETQ UCNT (ADD1 UCNT))
   (RETURN (UNIFY X Y)))))
(UNIFY (LAMBDA (O1 O2) (COND
   ((EQ O1 O2) T)
   ((ATOM O1) (COND
    ((EQUAL (GET O1 (QUOTE IS-BOUND)) UCNT) (UNIFY (EVAL O1) O2))
    ((OCCUR O1 O2) NIL)
    (T (DO (PUT O1 (QUOTE IS-BOUND) UCNT) (CSET O1 O2)))))
   ((ATOM O2) (COND
    ((EQUAL (GET O2 (QUOTE IS-BOUND)) UCNT) (UNIFY O1 (EVAL O2)))
    ((OCCUR O2 O1) NIL)
    (T (DO (PUT O2 (QUOTE IS-BOUND) UCNT) (CSET O2 O1)))))
   ((NOT (EQ (CAR O1) (CAR O2))) NIL)
   ((NULL (CDR O1)) T)
   ((NULL (CDDR O1)) (UNIFY (CADR O1) (CADR O2)))
   ((NULL (CDDDR O1)) (AND (UNIFY (CADR O1) (CADR O2))
                           (UNIFY (CADDR O1) (CADDR O2))))
   (T (AND (UNIFY (CADR O1) (CADR O2))
           (UNIFY (CADDR O1) (CADDR O2))
           (UNIFY (CADDDR O1) (CADDDR O2))))))))
(OCCUR (LAMBDA (X Y) (COND
   ((EQ X Y) T)
   ((ATOM Y) (COND
    ((EQUAL (GET Y (QUOTE IS-BOUND)) UCNT) (OCCUR X (EVAL Y)))
    (T NIL)))
   (T (OR (OCCUR X (CAR Y)) (OCCUR X (CDR Y)))))))
(RECOVER (LAMBDA (X) (COND
   ((ATOM X) (COND
    ((EQUAL (GET X (QUOTE IS-BOUND)) UCNT) (RECOVER (EVAL X)))
    ((EQUAL (GET X (QUOTE RECOV)) UCNT) (EVAL X))
    (T (DO (PUT X (QUOTE RECOV) UCNT) (CSET X (GENSYM))))))
   (T (CONS (CAR X) (MAPLIST (CDR X) RECOVER1))))))
(RECOVER1 (LAMBDA (X)
   (RECOVER (CAR X))))
(SUBSTITUTE (LAMBDA (L1 L2)
   (COND
   ((NULL L1) L2)
   (T (SUBSTITUTE (CDR L1) (APPEND L2 (LIST (CONS (CAAR L1) (LIST
   (RECOVER (CADAR L1))))))))))))
```

```
(EXPANSION (LAMBDA (L1 L2 L3)
   (PROG (U)
   (COND
      ((ATOM (CAADR L3)) (SETQ U (CADR (NEGATE L3))))
   (T (SETQ U (CONS (CADAADR L3) (CDADR L3)))))
   (RETURN (COMBINE L1 L2 (CADR L3) U NIL)) ) ))
(EXPANSIONC (LAMBDA (L1 L2 L3)
   (COND
   ((EQUAL (CAR L1) L2) (APPEND L3 (CDR L1)))
   (T (EXPANSIONC (CDR L1) L2 (APPEND L3 (LIST (CAR L1)))))) ) ))
(ADD (LAMBDA (L1 L2)
   (CONS L1 (LIST L2))))
(COMBINE1 (LAMBDA (L1 L2 L3 L4)
        (APPEND (APPEND (ADDB (APPEND L4 (CDR L2)) NIL)
                        (LIST (ADD 'A L3)))
                (EXPANSIONC L1 (ADD 'B L3) NIL)) ))
(COMBINE (LAMBDA (L1 L2 L3 L4 L5)
   (PROG (U X Y)
   (COND
   ((NULL L2) (RETURN NIL))
   ((EQUAL (CAAR L2) (CAR L4)) (SETQ U (UNIFY1 (STRIP (CAR L2)) (STRIP
   L4)))))
   (COND
      ((NULL U) (ERASE '(X Y) ) (RETURN (COMBINE L1 (CDR L2) L3 L4
      (APPEND L5 (LIST (CAR L2))))))
   (T (RETURN (SUBSTITUTE (COMBINE1 L1 L2 L3 L5) NIL))) ) ) ))
(ADDB (LAMBDA (L1 L2)
   (COND
   ((NULL L1 ) L2)
   (T (ADDB (CDR L1) (APPEND L2 (LIST (ADD 'B (CAR L1))))))))
   ))
(REDUCTION (LAMBDA (L1 L2 L3)
    (PROG (U NL1 NL2 NL3 X Y)
    (COND
    ((NULL (LASTBS L1 NIL)) (RETURN (APPEND L3 L2)))
    ((EQ (CAAR L2) 'B) (COND
                       ((EQUAL (CAADAR L1) (CAADAR L2)) (SETQ U (UNIFY1
                       (STRIP (CADAR L1)) (STRIP (CADAR L2))))
                       (T (SETQ U NIL))))
    (T (COND
       ((AND (ATOM (CAADAR L2)) (EQUAL (CAADR (NEGATE (CAR L2)))
       (CAADAR L1)))
        (SETQ U (UNIFY1 (STRIP (CADAR L1)) (STRIP (CADAR L2)))))
       ((AND (ATOM (CAADAR L1)) (EQUAL (CAADR (NEGATE (CAR L1)))
        (CAADAR L2)))
        (SETQ U (UNIFY1 (STRIP (CADAR L1)) (STRIP (CADAR L2)))))))
    (COND
    ((NULL U) (COND
              ((NULL (CDR L2)) (ERASE '(X Y)) (RETURN (REDUCTION (CDR L1
               (PRELIT L1) (APPEND L3 (LIST (CAR L1))))))
              (T (ERASE '(X Y) ) (RETURN (REDUCTION L1 (CDR L2) L3)))))
    (T (SETQ NL1 (SUBSTITUTE (CDR L1) NIL))
```

```
        (SETQ NL2 (SUBSTITUTE (PRELIT L1) NIL))
        (SETQ NL3 (SUBSTITUTE L3 NIL))
       (ERASE '(X Y))
       (RETURN (REDUCTION NL1 NL2 NL3)))) )))
(PRELIT (LAMBDA (L)
   (COND
   ((EQ 'B (CAAR L)) (PRELIT (CDR L)))
   (T L))))
(LASTBS (LAMBDA (L1 L2)
   (COND
   ((EQ (QUOTE A) (CAAR L1)) L2)
   (T (LASTBS (CDR L1) (APPEND L2 (LIST (CAR L1))))) ) ))
(NEGATE (LAMBDA (L)
   (CONS (CAR L) (LIST (CONS (ADD NIL (CAADR L))(CDADR L))) )
   ))
(TRUNCATION (LAMBDA (L)
   (COND
   ((NULL L) NIL)
   ((EQ 'B (CAAR L)) L)
   (T (TRUNCATION (CDR L))) ) ))
))
```

```
1. (MAIN '(1 (((NIL P) (K X4) X4 (K X4))))
         '((1 (((NIL P) (K X4) X4 (K X4))))
           (1 ((P (G X1 Y1) X1 Y1)))
           (1 ((P X2 (H X2 Y2) Y2)))
           (4 (((NIL P) X3 Y3 U3) ((NIL P) Y3 Z3 V3) ((NIL P) X3 V3 W3)
                (P U3 Z3 W3)))))
2. (MAIN '(1 (((NIL P) (B) (A) (C))))
         '((1 (((NIL P) (B) (A) (C))))
           (1 ((P X1 (E) X1)))
           (1 ((P (E) X2 X2)))
           (4 (((NIL P) X3 Y3 U3) ((NIL P) Y3 Z3 V3) ((NIL P) U3 Z3 W3)
                (P X3 V3 W3)))
           (4 (((NIL P) X4 Y4 U4) ((NIL P) Y4 Z4 V4) ((NIL P) X4 V4 W4)
                (P U4 Z4 W4)))
           (1 ((P X5 X5 (E))))
           (1 ((P (A) (B) (C))))))
3. (MAIN '(1 (((NIL P) (A) (E) (A))))
         '((1 (((NIL P) (A) (E) (A))))
           (1 ((P (I X1) X1 (E))))
           (1 ((P (E) X2 X2)))
           (4 (((NIL P) X3 Y3 U3) ((NIL P) Y3 Z3 V3)
                ((NIL P) U3 Z3 W3) (P X3 V3 W3)))
           (4 (((NIL P) X4 Y4 U4) ((NIL P) Y4 Z4 V4)
                ((NIL P) X4 V4 W4) (P U4 Z4 W4)))))
4. (MAIN '(1 (((NIL P) (A) X1 (E))))
         '((1 (((NIL P) (A) X1 (E))))
           (1 ((P (I X2) X2 (E))))
           (1 ((P (E) X5 X5)))
           (4 (((NIL P) X3 Y3 U3) ((NIL P) Y3 Z3 V3)
                ((NIL P) U3 Z3 W3) (P X3 V3 W3)))
           (4 (((NIL P) X4 Y4 U4) ((NIL P) Y4 Z4 V4)
                ((NIL P) X4 V4 W4) (P U4 Z4 W4)))))
5. (MAIN '(1 (((NIL S) (E))))
         '((1 (((NIL S) (E))))
           (1 ((P (E) X1 X1)))
           (1 ((P X2 (E) X2)))
           (1 ((P X3 (I X3) (E))))
           (1 ((P (I X4) X4 (E))))
           (1 ((S (A))))
           (4 (((NIL S) X5) ((NIL S) Y5) ((NIL P) X5 (I Y5) Z5) (S Z5)))
           (4 (((NIL P) X6 Y6 U6) ((NIL P) Y6 Z6 V6) ((NIL P) X6 V6 W6)
                (P U6 Z6 W6)))
           (4 (((NIL P) X7 Y7 U7) ((NIL P) Y7 Z7 V7) ((NIL P) U7 Z7 W7)
                (P X7 V7 W7)))))
```

```
6. (MAIN '(1 (((NIL S) (I (B)))))
         '((1 (((NIL S) (I (B)))))
           (1 ((P (E) X1 X1)))
           (1 ((P X2 (E) X2)))
           (1 ((P X3 (I X3) (E))))
           (1 ((P (I X4) X4 (E))))
           (1 ((S (B))))
           (4 (((NIL S) X5) ((NIL S) Y5) ((NIL P) X5 (I Y5) Z5) (S Z5)))
           (4 (((NIL P) X6 Y6 U6) ((NIL P) Y6 Z6 V6) ((NIL P) X6 V6 W6)
               (P U6 Z6 W6)))
           (4 (((NIL P) X7 Y7 U7) ((NIL P) Y7 Z7 V7) ((NIL P) U7 Z7 W7)
               (P X7 V7 W7))) ))
 7. (MAIN '(1 (((NIL D) (A) (B))))
         '((1 (((NIL D) (A) (B))))
           (1 ((P (A))))
           (1 ((M (A) (S (C)) (S (B)))))
           (1 ((M X1 X1 (S X1))))
           (2 (((NIL M) X2 Y2 Z2) (M Y2 X2 Z2)))
           (2 (((NIL M) X3 Y3 Z3) (D X3 Z3)))
           (5 (((NIL P) X4) ((NIL M) Y4 Z4 U4)
              ((NIL D) X4 U4) (D X4 Y4) (D X4 Z4)))))
 8. (MAIN '(3 (((NIL L) (I) X8) ((NIL L) X8 (A)) (D (F X8) X8)))
         '((3 (((NIL L) (I) X8) ((NIL L) X8 (A) ) (D (F X8) X8)))
           (1 ((D X1 X1)))
           (3 (((NIL D) X2 Y2) ((NIL D) Y2 Z2) (D X2 Z2)))
           (2 ((P X3) (D (G X3) X3)))
           (2 ((P X4) (L (I) (G X4))))
           (2 ((P X5) (L (G X5) X5)))
           (1 ((L (I) (A))))
           (2 (((NIL P) X6) ((NIL D) X6 (A))))
           (3 (((NIL L) (I) X7) ((NIL L) X7 (A)) (P (F X7))))))
 9. (MAIN '(3 (((NIL P) X8) ((NIL L) (A) X8) (L (F (A)) X8)))
         '((3 (((NIL P) X8) ((NIL L) (A) X8) (L (F (A)) X8)))
           (1 ((L X1 (F X1))))
           (1 (((NIL L) X2 X2)))
           (2 (((NIL L) X3 Y3) ((NIL L) Y3 X3)))
           (2 (((NIL D) X4 (F Y4)) (L Y4 X4)))
           (2 ((P X5) (D (H X5) X5)))
           (2 ((P X6) (P (H X6))))
           (2 ((P X7) (L (H X7) X7))) ))
```

```
10.(MAIN '(1 (((NIL R) (F (F (A) (B)) (C)) (F (A) (F (B) (C))))))
         '((1 (((NIL R) (F (F (A) (B)) (C)) (F (A) (F (B) (C))))))
           (1 ((R (F X1 Y1) (F Y1 X1))))
           (1 ((R (F X2 (F Y2 Z2)) (F (F X2 Y2) Z2))))
           (1 ((R (G (F X3 Y3) Y3) X3)))
           (1 ((R X4 (G (F X4 Y4) Y4))))
           (1 ((R (F (G X5 Y5) Z5) (G (F X5 Z5) Y5))))
           (1 ((R (G (F X6 Y6) Z6) (F (G X6 Z6) Y6))))
           (3 (((NIL R) X7 Y7) ((NIL R) Y7 Z7) (R X7 Z7)))
           (1 ((R X8 X8)))
           (3 (((NIL R) X9 Y9) ((NIL R) U9 (F X9 V9)) (R U9 (F Y9 V9))))
           (3 (((NIL R) X10 Y10) ((NIL R) U10 (G X10 V10)) (R U10 (G Y10 V10))))
           (3 (((NIL R) X11 Y11) ((NIL R) U11 (G V11 X11)) (R U11 (G V11 Y11))))
11.(MAIN '(1 (((NIL R) (F (G (A) (B)) (C)) (F (A) (G (C) (B))))))
         '((1 (((NIL R) (F (G (A) (B)) (C)) (F (A) (G (C) (B))))))
           (1 ((R (F X1 Y1) (F Y1 X1))))
           (1 ((R (F X2 (F Y2 Z2)) (F (F X2 Y2) Z2))))
           (1 ((R (G (F X3 Y3) Y3) X3)))
           (1 ((R X4 (G (F X4 Y4) Y4))))
           (1 ((R (F (G X5 Y5) Z5) (G (F X5 Z5) Y5))))
           (1 ((R (G (F X6 Y6) Z6) (F (G X6 Z6) Y6))))
           (3 (((NIL R) X7 Y7) ((NIL R) Y7 Z7) (R X7 Z7)))
           (1 ((R X8 X8)))
           (3 (((NIL R) X9 Y9) ((NIL R) U9 (F X9 V9)) (R U9 (F Y9 V9))))
           (3 (((NIL R) X10 Y10) ((NIL R) U10 (G X10 V10)) (R U10 (G Y10 V10))))
           (3 (((NIL R) X11 Y11) ((NIL R) U11 (G V11 X11)) (R U11 (G V11 Y11))))
12.(MAIN '(1 (((NIL R) (F (A) (G (B) (C))) (F (G (A) (C)) (B)))))
         '((1 (((NIL R) (F (A) (G (B) (C))) (F (G (A) (C)) (B)))))
           (1 ((R (F X1 Y1) (F Y1 X1))))
           (1 ((R (F X2 (F Y2 Z2)) (F (F X2 Y2) Z2))))
           (1 ((R (G (F X3 Y3) Y3) X3)))
           (1 ((R X4 (G (F X4 Y4) Y4))))
           (1 ((R (F (G X5 Y5) Z5) (G (F X5 Z5) Y5))))
           (1 ((R (G (F X6 Y6) Z6) (F (G X6 Z6) Y6))))
           (3 (((NIL R) X7 Y7) ((NIL R) Y7 Z7) (R X7 Z7)))
           (1 ((R X8 X8)))
           (3 (((NIL R) X9 Y9) ((NIL R) U9 (F X9 V9)) (R U9 (F Y9 V9))))
           (3 (((NIL R) X10 Y10) ((NIL R) U10 (G X10 V10)) (R U10 (G Y10 V10))))
           (3 (((NIL R) X11 Y11) ((NIL R) U11 (G V11 X11)) (R U11 (G V11 Y11))))
13.(MAIN '(1 (((NIL R) (G (F (A) (B)) (C)) (F (A) (G (B) (C))))))
         '((1 (((NIL R) (G (F (A) (B)) (C)) (F (A) (G (B) (C))))))
           (1 ((R (F X1 Y1) (F Y1 X1))))
           (1 ((R (F X2 (F Y2 Z2)) (F (F X2 Y2) Z2))))
           (1 ((R (G (F X3 Y3) Y3) X3)))
           (1 ((R X4 (G (F X4 Y4) Y4))))
           (1 ((R (F (G X5 Y5) Z5) (G (F X5 Z5) Y5))))
           (1 ((R (G (F X6 Y6) Z6) (F (G X6 Z6) Y6))))
           (3 (((NIL R) X7 Y7) ((NIL R) Y7 Z7) (R X7 Z7)))
           (1 ((R X8 X8)))
           (3 (((NIL R) X9 Y9) ((NIL R) U9 (F X9 V9)) (R U9 (F Y9 V9))))
           (3 (((NIL R) X10 Y10) ((NIL R) U10 (G X10 V10)) (R U10 (G Y10 V10))))
           (3 (((NIL R) X11 Y11) ((NIL R) U11 (G V11 X11)) (R U11 (G V11 Y11))))
```

```
14. (MAIN '(1 (((NIL Q) (K X1) X1 (K X1))))
          '((1 (((NIL Q) (K X1) X1 (K X1))))
            (4 (((NIL Q) X2 Y2 U2) ((NIL Q) Y2 Z2 V2)
                ((NIL Q) X2 V2 W2) (Q U2 Z2 W2)))
            (4 (((NIL Q) X3 Y3 U3) ((NIL Q) Y3 Z3 V3)
               ((NIL Q) U3 Z3 W3) (Q X3 V3 W3)))
            (1 ((Q (G X4 Y4) X4 Y4)))
            (1 ((Q X5 (H X5 Y5) Y5)))
            (1 ((Q X6 Y6 (F X6 Y6))))))
```