13. Level subgoal reordering

As was shown before, the hierarchical deduction procedure produces resolvents (H-resolvents) by resolving upon the first literals of the goal clauses. Different selection of the first literal of a goal clause usually results in different resolvents and so different search trees. Thus the search process of the prover can be partially controlled by suitable selection of the first literal of each goal clause.

Recall that in the definition of H-resolvent (H0-resolvent) and in the procedure H1-DEDUC (H0-DEDUC), we did not claim any particular order for the literals inherited from the rule clause except that these literals must be all on the left side of the resolvent. Recall also that we have proved that the procedure H1-DEDUC (H0-DEDUC) is complete. This means that reordering the literals of an H-resolvent inherited from the rule clause will not cause incompleteness to the procedure.

According to the above considerations, and since the literals of a resolvent inherited from rule clauses must have a largest index in the resolvent, we will let our reordering subroutine select a literal among only those literals of the goal clause which have the largest index in the clause to be the first literal. So our reordering strategy is called *level subgoal reordering*. It works as following:

For a goal clause G to be reordered, the reordering routine first collects all literals of the clause which have the largest index in the clause. If there is only one of them, (it must be the left-most literal of G), then the the order of the goal G will not be changed. Otherwise each literal L being considered is assigned a value, E(L), according to the information stored in the TWINSYMS field of the node being constructed and the mass of each non-variable symbol occurring in L, by the following formula:

$$E(L) = \sum_{k \in L} W(k,n) * MASS(k),$$

where W(k,n) is a weight which is 1.0 if k occurs in the TWINSYMS field of the node n, otherwise is 0.4.

As was mentioned before, the fewer the occurrences of a twin symbol, (so the bigger its mass), the more important is the match of it for success in a proof. By the above formula, a subgoal containing a bigger mass symbol will obtain a higher priority for being selected as the first subgoal.

This formula uses a heavier weight, 1.0, for the mass of an unmatched occurrence of a twin symbol. Because, according to our twin match requirement, the first match of each occurrence of a twin symbol is a necessary condition to obtain a proof.

By this formula, a literal having a complex structure is preferable to be selected as the first subgoal, because it usually contains more non-variable symbols. This literal, if it is used as the first literal of the goal clause, usually results in a smaller branching factor

The final decision of which literal is to be first, is made by the formula by balancing the above considerations.

Our experiments have shown that the level subgoal reordering strategy makes an important addition to the efficiency of our theorem prover. Besides it makes the prover concentrate on the important things and leads to a smaller branching factor, it is also beneficial for the evaluation routine (to be discussed next) to select the next promising goal. As will be shown next, the evaluation of a resolvent is based mainly on the feature of matches of twin symbols with which the resolvent is obtained. By resolving the goal clause upon its important subgoal, we can usually obtain a great difference between the matches of twin symbols of the different resolvents to be evaluated.

14. Evaluation of the resolvent

There are mainly two tasks of the evaluation subroutine. One is to reject some resolvents, another is to assign a priority to each accepted resolvent.

A resolvent will be rejected by the evaluation routine in the following cases:

1. Local depth limit. The number of framed literals of it is greater than a predefined local depth limit.

2. Unit subsumed. It is subsumed by a unit clause inputted or produced in the deduction.

3. Nest depth limit. For each function symbol, there is a nest depth limit assigned at the beginning of the deduction by the procedure itself or by the user. The nest depth of a function symbol in a literal is counted as the number of times that the function symbol occurs in the substructure of itself (for example, f(x) is counted as 1, f(f(x)) is counted as 2). The nest depth limit for each functional symbol can be predefined by the user, otherwise the default value will be used by procedure automatically. The default nest depth limit of uninterpreted function symbols is 1 (nesting by itself is forbidden), the default nest depth limit of the interpreted function symbols, such as the inverse function "i" for group theory and arithmetic operators, is infinite. A resolvent will be rejected if the nest depth of a function symbol contained in it is greater than the nest depth limit of the function symbol.

The value of a clause C, denoted by VALUE(C), is a real number. Each input clause will be assigned by the main program a negative value. The value of a resolvent produced during deduction is given by the evaluation routine.

A large positive value of a resolvent corresponds to a higher priority that it will be used as a goal clause; large negative value of a resolvent corresponds to a lower priority. The value of a resolvent is also related to the probability that it is or is not redundant. The lower the value of a resolvent, the higher the possibility that it is redundant. If the value of a resolvent is lower than the predefined threshold, then it will be discarded.

Let H be the resolvent of a goal clause G against a rule clause R upon the first literal L_g of G. Then VALUE(H) is the weighted sum of a series of evaluation functions, which looks like the following:

$$\begin{aligned}\text{VALUE(H)} = \; & W_1 * \text{PARENT-EVAL(G,R)} \\ & + W_2 * \text{DEPTH-EVAL}(H_R,H_G) \\ & + W_3 * \text{NEWTWIN-EVAL}(TS_n,n,G) \\ & + W_4 * \text{MATCH-EVAL}(TS_m,n) \\ & + W_5 * \text{SKIN-COST}(TS_G) \\ & + W_6 * \text{SKOUT-COST}(TS_R) \\ & + W_7 * \text{NEST-EVAL(H)} \\ & + W_8 * \text{UNIFY-EVAL(H)}\end{aligned}$$

where

W_i: a positive real number as a weight parameter, i = 1,...,8;

H_G: the subclause of H consisting of literals inherited from the goal clause G;

H_R: the subclause of H consisting of literals inherited from the rule clause R;

TS_n: the set of occurrences of non-variables symbols of H inherited from the rule clause R;

TS_m: the set of occurrences of non-variable symbols which were matched in obtaining H;

TS_G: the set of occurrences of non-variable symbols of H which were carried into H_G by the variables in G during the resolution;

TS_R: the set of occurrences of non-variable symbols of H which were carried into H_R by the variables in R during the resolution.

In the following, we give an explanation for each of the above evaluation functions:

1. PARENT-EVAL. By this function, a resolvent inherits a portion of the values from its goal parent and rule parent.

2. DEPTH-EVAL. This function returns a value which depends on the general features of the process of the hierarchical deduction:

A successful deduction usually appears as follows: Staring from the first literal of the goal clause, the procedure proceeds with one or more steps of expansions, followed by one or more steps of reductions to obtain a reduction of the original goal clause, then this process repeats again. So, at the first one or more steps (The steps are counted by the the number of framed literals) of a deduction, the function, DEPTH-EVAL, will contribute a bigger value to a longer resolvent (expansion) than to a shorter resolvent (reduction).

For the correct path, the numbers of twin symbols in TWINSYMS fields of nodes should decrease as the length of the deduction increases. When most of the twin symbols have been matched once in a deduction path, then the proof should be about finished, so a resolvent consisting of many literals seems to be unpromising, and such a resolvent should be assigned a lower value.

Combining these considerations, the function is defined as follows:

If H_R is empty (H is a reduction), then

$$\begin{aligned} &\text{DEPTH-EVAL}(H_R, H_G) \\ =\ &\text{MINIMUM}\{0, W_d * (FL_h\# - W_s * \text{SIZE})\} \\ &- H_G\# * \{1 + W_p * [\text{NTS}(1) - \text{NTS}(n)] / \text{SIZE}\}, \end{aligned}$$

If H_R is not empty (H is a expansion), then

$$\begin{aligned} &\text{DEPTH-EVAL}(H_R,H_G) \\ =\ &\text{MAXIMUM}\{0, W_e * H_R\# * (W_s * \text{SIZE} - \text{FL}\#))\} \\ &- (H_R\# + H_G\#) * \{1 + W_p * [\text{NTS}(1) - \text{NTS}(n)] / \text{SIZE}\}, \end{aligned}$$

where

W_d: a parameter; the same with W_s,W_e,W_p;

H_R#: the number of literals of H_R;

H_G#: the number of literals of H_G;

FL_h#: the number of framed literals of H;

SIZE: The number of literals in the set S;

NTS(n): the number of non-variable symbols in the TWINSYMS field of node n being produced in the deduction;

NTS(1): the number of non-variable symbols in the TWINSYMS field of the first node.

3. NEWTWIN-EVAL. This function obtains a value by evaluating the non-variable symbols carried into resolvents from the rule clause R.

$$\text{NEWTWIN-EVAL}(TS_n,G,n) = \sum_{k \in TS_n} f(k,n,G) * \text{MASS}(k)$$

where f(k,n,G) is a weight function. If an unmatched occurrence of a symbol k in TS_n does not occur in the goal clause G, then f(k,n,G) is a lager positive number; if this symbol occurs in G, then f(k,n,G) is a smaller positive positive. For the matched occurrence k in TS_n, f(k,n,G) is a small negative number.

4. MATCHED-EVAL. This function will contribute a positive value according to the masses of all matched non-variable symbols.

$$\text{MATCHED-EVAL}(TS_m,n) = \sum_{k \in TS_m} m(k,n) * \text{MASS}(k).$$

For each symbol k in TS_m, m(k,n) is a larger positive number if k is a twin symbol and k is in the TWINSYMS field of node n, otherwise if is a smaller positive number.

5. SKIN-COST. Because there is usually a possibility that the variables in the rest of goal clause have obtained wrong substitutions in the resulting resolvent, and a literal of a goal clause will usually be harder to unify when its variables are substituted by some terms containing non-variable symbols, we therefore assign a cost to the resolvent, for each such non-variable symbol substitution.

$$\text{SKIN-COST}(TS_G,n) = \sum_{k \in TS_G} g(k,n),$$

where g(k,n) is a negative number: it is more negative if k is not in the TWINSYMS field of node n.

6. SKOUT-COST. This function obtains a (negative) value from the non-variable symbols which were carried into H_R by the variables in R in the resolution according to the same consideration as the above. But this kind of wrong substitution is usually less harmful than that in the above case: an unprovable subgoal in the left side of a goal clause is easier to detect than an unprovable subgoal in the right side of the goal clause. So for the same arguments, the value obtained by this function will be less negative than the value obtained by SKIN-COST.

$$\text{SKOUT-COST}(TS_R,n) = \sum_{k \in TS_R} r(k,n).$$

7. NEST-EVAL(H). It evaluates the terms of H according to a global variable ALLTERMS, which is the set consisting of all terms in the original set S. The value is obtained by means of two supplementary evaluation functions:

TERM-MATCH(TM1,TM2). It obtains a value by evaluating the unification of two terms. Each match of a twin symbol in the unification contributes a positive number according to the mass of the symbol. The substitution of a variable by a non-variable term contributes a negative value to the evaluation.

TERM-EVAL(TM) = MAXIMUM {TERM-MATCH(TM,TMi)}.
TMi ∈ ALLTERMS

The final value of NEST-EVAL(H) is:

NEST-EVAL(H) = MINIMUM {TERM-EVAL(TM)}
TM ∈ H

8. UNIFY-EVAL(H). This function evaluates literals of H. We have a global variable UNITS which consists of all unit clauses in the set S and a global variable LITERALS consisting of all literals of the non unit rule clauses in the set S. The evaluation process is similar to the NEST-EVAL described above. It uses two supplementary functions:

LITERAL-MATCH(L1,L2). It returns a value by evaluating the matches of non-variable symbols in L1 and L2 and the substitution for this unification.

LITERAL-EVAL(L) = MAXIMUM {LITERAL-MATCH(L,Li)}
Li ∈ UNITS ∪ LITERALS
UNIFY-EVAL(H) = MINIMUM {LITERAL-EVAL(L)}
L ∈ H

But the minimum operator in the last function, UNIFY-EVAL, may be replaced by a maximum operator if, for each literal L in H, LITERAL-EVAL(L), is greater than a predefined threshold.

It should be noticed that the evaluation functions NEST-EVAL and UNIFY-EVAL are most expensive to compute than the other evaluation functions. In order to decrease the expense, some indexing methods can be used. We can also use hash technique to record the results of NEST-EVAL and UNIFY-EVAL for the new terms and new literals produced in the deduction. Then these results can be used for directly retrieving values for terms or literals whose values (NEST-EVAL or UNIFY-EVAL) are already recorded.

Obviously, to balance so many functions and parameters is very difficult. As a first step, we require only that the evaluation routine can roughly do well: for some explicit case, it can make a correct value assignment; for some sophisticated case, it may make a mistake; in general, it should not assign a lowest value to a useful resolvent, or a highest value to a redundant one.

15. H2-DEDUC and implementation results

We now present an implemented hierarchical deduction procedure, H2-DEDUC, which is H1-DEDUC augmented by "correct reduction", "learning algorithm", "locking", "level subgoal reordering" and "resolvent evaluation" described in the last several sections. Procedure H2-DEDUC uses best first search strategy.

In H2-DEDUC, the GOALS field is deleted from each node. Instead a global variable, *goallist*, is used to store the resolvents, (goal clauses) produced by the procedure.

Before a proof is started, the goallist contains only the initial goal clauses obtained from input data. The procedure proceeds by taking the first clause in the goallist to resolve next. The newly produced and accepted resolvents will be inserted into the goallist on their priority in descending order by comparison with all other clauses in the goallist.

Each node of H2-DEDUC consists of four fields, POINTER, RULES, FAILED and TWINSYMS. In the following definition of H2-DEDUC, only three fields of a node, consisting of a triple, (POINTER RULES FAILED), are presented. The TWINSYMS field of a node is assigned or modified similarly to that

described in section 12. Note, instead of copying the rule clauses and the failed goals into the descendant node, a more efficient storage and allocation method is used in the following definition of the procedure H2-DEDUC.

PROCEDURE H2-DEDUC(goallist)

```
Step 1 If goallist is empty then exit with "FAIL"
       G := FIRST(goallist)
       goallist := REST(goallist)
       If G = NIL then exit with "BOX"
       I := the index of the first literal of G

Step 2 If I = 0 then go step 3
       If G is subsumed by a clause in the FAILED field of the node I
         then  return H2-DEDUC(goallist)
       I := POINTER-GET(I)
       Go step 2

Step 3  nextnode := GETNODE()
        G := REORDER(G)
        L := FIRST(G)
        I := the index of L
        If G is obtained by reduction then
           Store G in the FAILED field of node I[17]
        GS := NIL

Step 4 If I = 0 then go Step 5
       RS := RULES-GET(I)[18]
       GS := GS ∪ {H: H is an H-resolvent of G against C in level nextnode,
                          C ∈ RS,
                          the literal of C resolved upon is not locked}
       If there is a correct reduction H obtained then
          push H into goallist
          return H2-DEDUC(goallist)
       I := POINTER-GET(I)
       GO Step 4

Step 5 Insert all clauses in GS into goallist by their priority in descending order.
       I := index of L
       nextnode <= (I GS NIL)
       Return H2-DEDUC(goallist)
```

The priority of a goal clause (resolvent) in H2-DEDUC is the sum of two values, one is called the "clause value" obtained by the evaluation procedure similar to that of H1-DEDUC (with some slight changes of the parameters). Another is called the "path value" of the clause which is obtained by the following formulas:

1. The path value of input clause = 0;

2. The path value of H = E + the path value of G + W_m * MATCHED-EVAL(TS_m,n),

where

---

[17] Different from H1-DEDUC, only the goal clauses obtained by reduction may be recorded into the FAILED field of some related node in H2-DEDUC.

[18] In H2-DEDUC, the definition of the tail index of a clause is modified, so that, the index of a rule clause which is less than the node name I and which is not 0 (the lock) is a tail index of the clause. This modification can lead to more efficient implementation of the constraint on common tails without losing completeness.

H: the H-resolvent of a goal clause G;

E: an negative number as the cost of producing a resolvent by H2-DEDUC;

W_m: weight, a positive number;

TS_m: the set of matched non-variable symbols in obtaining H;

n: the current node.

Recall that the MATCHED-EVAL(TS_m,n) is a function used by the evaluation routine which will return a non-negative value depending on the masses of the matched symbols in obtaining the resolvent H and the content of the TWINSYMS field of the current node n.

Adding this value to compensate for the path expense is based on the twin match requirement: the twin match is an important criterion for the potential of a deduction path. By adjusting the parameter W_m, we can make the searching process of the procedure more likes breadth first (for example, let W_m be near to 0) or more likes depth first (for example, let w_m be a larger positive number).

H2-DEDUC is a complete procedure for the first order logic (certainly, with H-factors included).

This procedure was used to prove automatically a number of theorems. Some implementation results are presented in TABLE I. The prover was run on the Research 2060, University of Texas at Austin; it was programmed in UCILSP with a part of the code compiled. In the examples presented below, all literals with the index 0 are locked.

Example E.1. A logical theorem (Example in [5] and [10]).

$$\exists x \exists y \forall z (([F(xy) \Rightarrow (F(xz) \Leftrightarrow G(yz)] \wedge [F(xy) \Leftrightarrow (F(zz) \Rightarrow G(zz))]) \Rightarrow (G(xy) \Leftrightarrow G(zz)))$$

$_1G(yk(xy))$ $_1\tilde{}F(xk(xy))$ $_1\tilde{}F(xy)$
$_1F(xk(xy))$ $_1\tilde{}G(yk(xy))$ $_1\tilde{}F(xy)$
$_1G(k(xy)k(xy))$ $_1\tilde{}F(k(xy)k(xy))$ $_1\tilde{}F(xy)$
$_1F(xy)$ $_1F(k(xy)k(xy))$
$_1F(xy)$ $_1\tilde{}G(k(xy)k(xy))$
$_1G(k(xy)k(xy))$ $_1G(xy)$
$_1\tilde{}G(k(xy)k(xy))$ $_1\tilde{}G(xy)$ goal

This is an example to which the full locking according to a setting M = {G(xy),F(xy)} causes failure, though the setting seems to be reliable (In fact, for this setting M, the unsatisfiable set of ground clauses obtained from the above set contains more than one ground clauses false in M). But, this theorem is easier to prove by hierarchical deduction without locking.

Example E.2. The same as Example 10.2 in section 10.

Besides the locking described in section 10, the evaluation of resolvents is also important to obtain the efficient proof.

Example E.3 If xox = e for all x in group G, where o is a binary operator and e is the identity in G, then G is commutative.(A typical example in literature [6,10,14]).

$_1P(xyf(xy))$

$_1$P(xex)
$_1$P(exx)
$_1$P(xi(x)e)
$_1$P(i(x)xe)
$_1$P(xx(e))
$_1$P(abc)
$_0\tilde{}$P(xyu) $_0\tilde{}$P(yzv) $_0\tilde{}$P(uzw) $_1$P(xvw)
$_0\tilde{}$P(xyu) $_0\tilde{}$P(yzv) $_0\tilde{}$P(zvw) $_1$P(uzw)
$_1\tilde{}$P(bac) goal

For Horn set, the power of the hierarchical deduction can not be increased by the constraints on common tails, neither by the use of the partial set of support. This theorem is proved mainly by the heuristics on twin symbols. The interesting point is that the procedure "found" a proof where the inverse axiom was useful. Though it is well known, for proving this theorem, both the closure axiom and the inverse axiom can be excluded.

Example E.4. The same as Example 10.1 (A simplified version of AM8). H2-DEDUC proved also this theorem, without the using of locking, in the similar efficiency as the proof shown in TABLE I, because the evaluation function can make the correct selection of the goal clause for almost each midstep.

Example E.5 Theorem AM8.

$$
\begin{aligned}
&[\ a\le l\le b\\
&\wedge\ \forall t(a\le t\le l \Rightarrow F(l)\le F(t))\\
&\wedge\ \forall x(a\le x\le b \wedge \forall t[a\le t\le x \Rightarrow F(x)\le F(t)] \Rightarrow x\le l)]\\
&\wedge\ \forall w\exists g[\ (a\le w\le b \Rightarrow a\le g\le b \wedge\ F(g)\le F(w))\\
&\qquad\qquad \wedge\forall x([a\le w\le b \wedge a\le x\le b \wedge F(x)\le F(w)] \Rightarrow g\le x)]\\
&\Rightarrow \exists u[a\le u\le b \wedge \forall t(a\le t\le b \Rightarrow F(u)\le F(t)]
\end{aligned}
$$

The set of clauses obtained from the above formula and the inequality axioms consists of the following clauses (The predicate P stands for $\le$, the same for the example E.6 to give next):

R1: $_1$P(xx)
R2: $_1$P(xy) $_0$P(yx)
R3: $_1$P(xz) $_0\tilde{}$P(xy) $_0\tilde{}$P(yz)
R4: $_1$P(f(x)f(y)) $_0\tilde{}$P(xy) $_0\tilde{}$P(yx)
R5: $_1$P(al)
R6: $_1$P(lb)
R7: $_1$P(f(l)f(x)) $_0\tilde{}$P(ax) $_0\tilde{}$P(xl)
R8: $_1$P(xl) $_0\tilde{}$P(ax) $_0\tilde{}$P(xb) $_1$P(aq(x))
R9: $_1$P(xl) $_0\tilde{}$P(ax) $_0\tilde{}$P(xb) $_0\tilde{}$P(f(x)f(q(x)))
R10: $_1$P(xl) $_0\tilde{}$P(ax) $_0\tilde{}$P(xb) $_1$P(q(x)x)
R11: $_1$P(ah(x)) $_0\tilde{}$P(ax) $_0\tilde{}$P(xb)
R12: $_1$P(h(x)b) $_0\tilde{}$P(ax) $_0\tilde{}$P(xb)
R13: $_1$P(f(h(x))f(x)) $_0\tilde{}$P(ax) $_0\tilde{}$P(xb)
R14: $_1$P(h(x)y) $_0\tilde{}$P(ax) $_0\tilde{}$P(xb) $_0\tilde{}$P(ay) $_0\tilde{}$P(yb) $_0\tilde{}$P(f(y)f(x))
R15: $_0\tilde{}$P(ax) $_0\tilde{}$P(xb) $_1$P(ak(x))
R16: $_0\tilde{}$P(ax) $_0\tilde{}$P(xb) $_1$P(k(x)b)
R17: $_1\tilde{}$P(f(x)f(k(x))) $_1\tilde{}$P(ax) $_1\tilde{}$P(xb) goal

This theorem is more difficult to prove automatically than AM8'. For example, in the first round of the deduction, there are four resolvents produced. Though only the resolvent obtained by resolving the goal

R17 against R3 can lead to a proof, the prover "thinks" the resolvent C1 of R17 against R4,

$$C1 = {}_2\sim P(xk(x))\ {}_2\sim P(k(x)x)\ {}_1\sim P(ax)\ {}_1\sim P(xb),$$

is more promising, because the literal $\sim P(xk(x))$ can match with a literal of R15, and the literal $\sim P(k(x)x)$ can match with a literal of R16 (This mistake seems to be natural even for human resolution prover). In fact, there were many pitfalls along the correct path, especially if the negative literals of R3, R4 and the negative literals $\sim P(ax)$, $\sim P(xb)$ that occur in many input clauses were not locked. So, the use of the partial set of support strategy (with setting $\{P(xy)\}$) is essential for H2-DEDUC to efficiently prove this theorem.

Example E.6. Theorem IMV.

```
[  a≤b
 ∧ F(a)≤0
 ∧ 0≤F(b)
 ∧ ∀x(a≤x≤b ⇒ ∃y[(x≤y ⇒ F(x)≤0) ∧ ∀z[(y<z≤x ∧ F(z)≤0) ⇒ F(x)≤0]])
 ∧ ∀x(a≤x≤b ⇒ ∃y[(y≤x ⇒ 0≤F(x)) ∧ ∀z[(x≤y<z ∧ 0≤F(z)) ⇒ 0≤F(x)]])
 ∧ ∀x[(x≤b ∧ F(x)≤0) ⇒ x≤1]
 ∧∀x[∀y[(y≤b∧F(y)≤0) ⇒ y≤x] ⇒ 1≤x]]
⇒ ∃u(F(u)≤0 ∧ 0≤F(u))
```

The set of clauses obtained from the above formula and the inequality axioms used consists of the following clauses:

```
1P(xx)
1P(xz)0~P(xy)0~P(yz)
1P(xy) 0P(yx)
1~P(xq(yx)) 0P(xy)                                   interpolation axiom
1~P(q(xy)x) 0P(yx)                                   interpolation axiom
1P(ab)
1P(f(a)0)
1P(0f(a))
1P(f(x)0) 1~P(xh(x)) 0~P(ax) 0~P(xb)
1P(f(x)0) 0~P(yx) 1~P(f(y)0) 1P(yh(x)) 0~P(ax) 0~P(xb)
1P(0f(x)) 1~P(k(x)x) 0~P(ax) 0~P(xb)
1P(0f(x)) 0~P(xy) 1~P(0f(y)) 1P(k(x)x) 0~P(ax) 0~P(xb)
1P(x1) 0~P(xb) 1~P(f(x)0)
1P(g(x)b) 1P(1x)
1P(f(g(x))0) 1P(1x)
0~P(g(x)x) 1P(1x)
1~P(f(x)0) 1~P(0f(x))                                goal
```

H2-DEDUC proved this theorem also by using the full locking according to a setting $\{P(xy)\}$. But, because the free literals of the clauses obtained from interpolation axioms were not locked under this setting, the proof was obtained a little slower. By this example, we show that, in some case, a locking by guessing a model of the hypothesis of the theorem to be proved can result in an efficient proof.

TABLE I. Some results of the procedure H2-DEDUC

Example	E.1	E.2	E.3	E.4	E.5	E.6
Local depth limit	7	8	4	8	8	8
Nest depth limit[19]	(k.4)	(m.2)	(i.2)	-	-	-
Unit subsumed	11	16	0	15	1	21
Local subsumed	48	8	1	10	76	50
Tail unmergeable	43	0	0	2	27	13
Correct reduction	5	1	0	0	19	8
Learning algorithm	5	0	1	0	0	9
Accepted clauses	64	48	210	37	205	144
Useful clauses	23	17	19	14	27	38
CPU seconds	33.7	8.1	44.7	10.1	60.8	32.4

16. Summary

We would not expect that there is a general deduction method powerful in proving all theorems. In fact, for a different axiom system, it is essential that a different representation and different technique in deduction be used. Our intention in studying the hierarchical deduction is mainly to provide some ideas about the basic deduction schema, techniques useful in restricting searching space, and heuristics depending on the syntactic features of the theorems, which may be useful in designing some of powerful theorem provers that use also domain dependent knowledge.

I. Appendix. Proofs of the Theorem 1 and Theorem 2

We will first prove theorems for a modified version of the procedure H0-DEDUC. It is called H-DEDUC which is different from H0-DEDUC only in the following:

1. The framed literals described in section 4 are included (recorded) in the H0-resolvents produced by H0-DEDUC.
2. The name of the next produced node will be the index I of the first literal of the goal clause increased by an integer which is the value of a function RANDOM(). We suppose that the function RANDOM, when it is called, asks us to obtain a value which may be any positive integer according to our wishes.

Following is the definition of H-DEDUC.

PROCEDURE H-DEDUC(node)

```
Step 1. If node = 0 then exit with "FAIL"
```

[19] The nest depth limit is 1 for each of the function symbols not indicated in the slots

```
        GS := GS(node)[20]
        If GS = NIL then return H-DEDUC(POINTER-GET(node))

Step 2. G := FIRST(GS)
        If G = NIL then exit with "box"
        I := the index of L
        GOALS-MAKE(node, REST(GS))
        S : = GS(I)∪RS(I)

Step 3  nextnode := I + RANDOM()
        NEWGS := {H: H is a H0-resolvent of G against C in level nextnode, C∈S}
        NEWRS := S - S^L[21] - S^~L]

Step 4 nextnode <= (I G NEWRS NEWGS)
       Return H-DEDUC(nextnode)
```

Notice the following:

1. Though the framed literal is recorded in the H0-resolvent, but the local subsumption test is not used in H-DEDUC.
2. The terminology "common tail mergeable" introduced in section 5 will be used in the following proofs, though it is not a restriction for the H0-resolvent.
3. The literals in an H0-resolvent (H-resolvent) inherited from the rule clauses can be in any order but all of them must be on the left side of the resolvent.
4. We do not claim any particular order to the resolvents obtained from a goal clause (contained in NEWGS) by H-DEDUC.

We will proceed in our proofs by the following steps:

Step 1: Investigate properties of the nodes produced by H-DEDUC.

Step 2: Define a measure function for the procedure H-DEDUC.

Step 3: Prove the finite termination property of H-DEDUC.

Step 4: Prove that H-DEDUC is a decision procedure for propositional logic.

Step 5: Prove H0-DEDUC is a decision procedure for propositional logic.

Step 6: Prove that H1-DEDUC is a complete procedure for the first order logic.

Definition: FLS(n). Let C be the parent[22] of a node n. If C is not NIL, then FLS(n) is the set of framed literals of C increased by the first literal of C. If C is NIL (it is the case for the first node), then FLS(n) is NIL.

Definition: A clause C is called a *legal clause against a node n* iff no literal in C shares the same atom with a framed literal in FLS(n).

---

[20] Let n be a node. We define RS(n) = RULES-GET(n), GS(n) = GOALS-GET(n), SS(n) = GS(n)∪RS(n)

[21] Recall S^L = {C: C∈S ∧ L∈C}

[22] Recall that the parent of a node is a goal clause from which the node is produced.

Definition: TG(n). Let C be the parent of node n. If C is not NIL, then TG(n) is REST(C). If C is NIL, then TG(n) is NIL.

Definition: Suffix clause. If n is a node and G is a clause in GS(n). Then the subclause obtained from G by deleting each literal of G, which is indexed by n, is called the *suffix clause* of G against node n.

Definition: Regular node. A node n is called a *regular node* iff it satisfies the following requirements:

1. The pointer[23] of the node, is an integer less than n;
2. The largest index of the literals in SS(n) is not greater than n;
3. Every clause in SS(n) is a properly indexed clause;
4. The suffix clause of each clause in GS(n) is identical (including the order and indices of their literals) to TG(n).
5. All clauses in SS(n) are common tail mergeable;
6. Every clause in SS(n) is a simple ground clause[24];
7. The set of framed literals of each clause in GS(n) is equal to the set FLS(n);
8. All clauses in SS(n) are legal clauses against node n;

Theorem I.1. If n is a regular node and the first goal clause in GS(n) is G,

$$G = {}_{n}L1\ {}_{n}L2...{}_{n}Lh\ {}_{i1}T1...{}_{im}Tm,$$

where $h \geq 1$, $m \geq 0$, then the next node produced by calling H-DEDUC(n) must be a regular node.

Proof: By the definition of H-DEDUC, the next node produced by H-DEDUC(n) is n+j, $j > 0$. The first three requirements for node n+j being a regular node are obviously satisfied by the node n+j. We only prove that node n+j satisfies the last five requirements.

Because each clause in GS(n+j) is an H0-resolvent of G against some clause C in RS(n)∪REST(GS(n)) upon the literal L1 in level n+j, the resolvent must be in one of the following two forms:

F1: ${}_{n+j}P1...{}_{n+j}Pr\ {}_{n}L2...{}_{n}Lh\ {}_{i1}T1...{}_{im}Tm$;

F2: ${}_{n}L2...{}_{n}Lh\ {}_{i1}T1...{}_{im}Tm$,

where P1...Pr ($r \geq 1$), are the literals inherited from a rule clause C and retained by merging right.

Obviously, clause F2 is the suffix clause of each clause in GS(n+j). Because G is the parent of node n+j, so REST(G) =[25] TG(n+j). Then F2 = TG(n+j). So, node n+j satisfies the fourth requirement.

Obviously, all clauses in GS(n+j) are common tail mergeable. Note the clauses in RS(n+j) are some clauses in SS(n), but the common tail of a clause in GS(n+j) against any clause C in SS(n) must be the same as that of G against the clause C, because n is not a tail index of any clause in SS(n). So the clauses

---

[23] The content of the POINTER field of the node.

[24] Recall that the simple ground clause is a ground clause such that it is not a tautology and it contains no repetition of a literal.

[25] If C1 and C2 are two clauses, then C1 = C2 means that C1 is identical to C2 (including the order and indices of their literals).

in GS(n+j) must be common tail mergeable with all clauses in RS(n+j). Further more, the clauses in RS(n+j) must be common tail mergeable because RS(n+j) is a subset of SS(n). Therefore all clauses in SS(n+j) are common tail mergeable. Thus the fifth requirement is satisfied by node n+j.

Because of the use of the restricted merging,[26] , and since there is originally no repetition of literals in the suffix clause of a clause G' in GS(n+j) , there must be no repetition of literals in the whole clause G'. Also note tautology is rejected from the H0-resolvent. So all clauses must be simple ground clauses. Thus, node n+j satisfies the sixth requirement.

According to the definition of the framed literals in section 4, the set of framed literals of each clause in GS(n+j) must be the set of framed literals of the goal clause G increased by the first literal of G, which, by definition, is FLS(n+j). So, the seventh requirement is justified.

Let C be any clause in RS(n+j). Because, by definition of H-DEDUC, C must be a clause in SS(n) - SS(n)^L1 - SS(n)^~L1, C must not contain L1 or ~L1. But, by hypothesis, C is a legal clause against node n, so no literal of C shares an atom with a literal in FLS(n). Note that FLS(n+j) contains only one more literal L1 than FLS(n), but we have shown that C does not contain literal L1 or ~L1, so no literal of C shares an atom with a literal in FLS(n+j). Then C is a legal clause against node n+j.

Because each clause G' in GS(n+j) is an H0-resolvent of two simple ground clauses upon the literal L1 and ~L1, no literal L1 or ~L1 is included in the clause G'. But all literals of G' were inherited from the clauses in SS(n) which did not share an atom with FLS(n). Then all literals of G' must not share an atom with any literal in FLS(n+j). So all clauses in GS(n+j) must be legal clauses against node n+j.

Q.E.D.

Definition: Let C be a clause, and n be a node. Then the *active part* of C against node n, ACT(C,n), is defined by following equation:

ACT(C,n) = C - TG(n).

where the operator "-" means that ACT(C,n) is the subclause obtained from C by deleting every literal of C which is identical to a literal in TG(n) (not count indices). ACT(C,n) has the same set of framed literals as that of C.

Definition: Let S be a set of clauses, and n be a node. Then ACTS(S,n) is a set of clauses such that:

ACTS(S,n) = {ACT(C,n): C∈S}.

Lemma I.1. If n is a regular node, G is the first goal clause in node n, n is the index of the first literal of G, and n+j is the next node produced by H-DEDUC(n), then, according to Theorem I.1, the following are obviously true:

1. TG(n) is identical to a part of REST(G);
2. TG(n+j) = REST(G);
3. TG(n) is identical to a part of TG(n+j);
4. ACT(C,n+j) = C - TG(n+j) = (C - TG(n)) - TG(n+j) = ACT(C,n) - TG(n+j).

Definition: Let n be a regular node, then SIZE(n) is the total number of (non framed) literals in the set ACTS(SS(n),n).

---

[26] Recall that in restricted merging, only the literal inherited from the rule clause will be merged if this literal is identical to a literal inherited from the goal clause. See section 3.

Theorem I.2. Let n be a regular node, SIZE(n) = K. Suppose G is the first goal clause of the node n, L is the first literal of G and the index of L is n. Let n+j be the next node produced by H-DEDUC(n). Then if K > 0, then SIZE(n+j) < K.

Proof: By the definition of H-DEDUC, the set S of rule clauses used for resolving with the goal clause G is

$$S = REST(GS(n)) \cup RS(n).$$

We divide S into three subsets: S^L, S^~L and S - S^L - S^~L.

Because there is no repetition of literals in a clause, then, for each clause C in S^~L, there is at most one H0-resolvent of G against C. Suppose there is an H0-resolvent H of G against C. Obviously, ACT(H,n+j) contains only literals inherited from C. Because any literal of C identical to a literal in REST(G) will be merged in H, and by Lemma I.1, TG(n) is identical to a part of REST(G). So all literals in ACT(H,n+j) must be among the literals in C - TG(n) which is ACT(C,n). Note ~L must be a literal in ACT(C,n) but is not included in ACT(H,n+j). Therefore the number of literals in ACT(H,n+j) must be less than that in ACT(C,n). Because all clauses in S^~L will not be included in node n+j, and each clause H in GS(n+j) corresponds to one and only one clause C, we conclude that the number of literals in ACTS(GS(n+j),n+j) is less than that in the set ACTS(S^~L, n).

Obviously, each clause in RS(n+j) corresponds to one and only one clause in the set, S'= S - S^L - S^~L. According to Lemma I.1, ACT(C,n+j) = ACT(C,n) - TG(n+j). Then the number of literals in ACTS(S',n+j) must be less than or equal to that in the set ACTS(S',n).

Any clause in S^L and in S^~L will not be included in the node n+j.

Thus, we conclude that SIZE(n+j) < K.

Q.E.D.

Theorem I.3. Starting from a regular node n, the procedure H-DEDUC(n) either obtains a current goal which is identical to TG(n) or backtracks to the pointer of node n in finitely many steps, and, during the process, all nodes produced by the procedure are regular nodes whose names are greater than n.

Proof: If GS(n) is empty, then, at the next step, the procedure H-DEDUC backtracks to the pointer of node n, so the theorem is trivially true. Now suppose GS(n) is not empty. We prove the theorem by induction on the size of the node n.

Case 1. SIZE (n) = 0. Then for any clause G in GS(n), ACT(G,n) is NIL (empty). Note ACT(G,n) = G - TG(n), and because G does not contain a repetition of a literal, then G = TG(n). Because we suppose GS(n) is not empty, the current goal of H-DEDUC must be identical to TG(n). So the theorem is true.

Case 2. Suppose the theorem is true for the case SIZE(n) < K and K > 0. We prove the theorem for the case SIZE(n) = K. If the first clause G in GS(n) is identical to TG(n), then we have obtained a proof. Now we suppose that G is not identical to TG(n).

According to the definition of a regular node, the suffix clause of G must be identical to TG(n). Without losing generality, we suppose TG(n) is the following clause:

$$TG(n) = {}_{i1}T1\ {}_{i2}T2 \ldots {}_{im}T_m, \text{ where } m \geq 0.$$

Then G must have the following form:

$$G = {}_{n}L1\ {}_{n}L2\ldots{}_{n}Lh\ {}_{i1}T1\ {}_{i2}T2\ldots{}_{im}Tm, \text{ where } h \geq 1.$$

According to Theorem I.1, the second node produced by H-DEDUC is a regular node with the name n+j, where j is a positive integer. For the node n+j, TG(n+j) = REST(G), i.e.,

$$TG(n+j) = {}_{n}L2\ldots{}_{n}Lh\ {}_{i1}T1\ldots{}_{im}Tm.$$

Now the procedure returns to a call H-DEDUC(n+j). It has been proved by Theorem I.2 that SIZE(n+j) < K. By the induction hypothesis, the theorem should be true for node n+j. This means that each node produced by H-DEDUC(n+j) during the process described by the theorem must be a regular node with a name greater than n+j. According to the conclusion of this theorem, there are only two cases:

Case 2.1. The procedure backtracks to the pointer of node n+j in finitely many steps. Because the pointer of node n+j is n, the procedure will call H-DEDUC (n) again. Note that the goal clause G has been deleted from the node n and ACT(G,n) contains at least one literal L1, so this time, SIZE(n) < K. By the induction hypothesis, the theorem is true.

Case 2.2 A current goal G' of H-DEDUC identical to TG(n+j) is obtained, i.e.,

$$G' = {}_{n}L2\ldots{}_{n}Lh\ {}_{i1}T1\ldots{}_{im}Tm.$$

Again, there are two cases:

Case 2.2.1 h = 1. Then TG(n+j) = TG(n). This is the case that we want to prove.

Case 2.2.2 h > 1. By the definition, the procedure will retrieve rule clauses from node n according to the index n of the first literal of G'. But this round of deduction and the following must be the same as the call of H-DEDUC(n), where the node n has been modified such that only the goal clause G is replaced by the current goal clause G'. To distinguish, we denote this modified node n by node n' (note that numerically, n = n').

We justify that node n' is also a regular node. Obvious, G' is common tail mergeable with all other clauses in SS(n'). So we only need to prove that G' is a legal clause against node n'. To do this, we need only to prove that the set of framed literals of G' is the same as that of G. Note the set of framed literals of an H0-resolvent is the set of framed literals of its goal clause increased by the first literals of this clause. By the induction hypotheses, we have proved that all nodes before the current goal G' is obtained have names greater than n+j. Note that the name of a node produced by H-DEDUC is the index of the first literal of the goal clause, from which the node is produced, increased by some positive integer, which is assigned by the random function RANDOM. So each newly increased framed literal of the goal G' must have an index greater than or equal to n. Recall that we have a stipulation in the definition of the framed literals (see section 4), that any framed literal of a current goal clause whose index is greater than or equal to the index of the first literal of this clause is deleted. So when G' becomes the current goal clause, only the framed literals that are same as G are retained. The other requirements for a regular node are obviously satisfied by the node n'.

Because G' is identical to REST(G), and since all other conditions of node n' are similar to that of node n, then for node n', SIZE(n') < K. By hypothesis, the theorem is true.

All the cases have been proved.

Q.E.D.

Notice that the primary node defined in the section 3, node 1, is a special regular node, such that TG(1) = NIL and the pointer of it is 0. Then, we have the following corollary:

Corollary I.1. Let node 1 be a primary node. Then H-DEDUC(1) must exit with "box" (when a current goal is NIL) or with "FAIL" (when current node is 0) in finitely many steps.

Lemma I.2 to Lemma I.8, to be proved next, are all under the following hypotheses: n is a regular node, G is the first goal of node n, L is the first literal of G, the index of L is n, and n+j is the next node produced by H-DEDUC(n).

Lemma I.2. ACTS((S1∪S2), n) = ACTS(S1,n)∪ACTS(S2,n)

Proof: Trivial.

Lemma I.3. ACTS((S - S^L),n) = ACTS(S,n) - ACTS(S^L,n)

Proof: Because G is a simple ground clause, and by Lemma I.1, TG(n) is a part of REST(G), so TG(n) must not contain literal L. So any clause in ACTS(S^L,n) contains a literal L.

Suppose C' is a clause in ACTS((S - S^L),n). Then there must be a clause C, C∈(S - S^L), such that C - TG(n) = C'. Because there is no clause in S - S^L containing literal L, C' must not contain the literal L. But we have shown that any clause in ACTS(S^L,n) contains the literal L, then C' must not be a member of ACTS(S^L,n). So C' is a member of ACTS(S,n) - ACTS(S^L,n).

Suppose C' is a clause in ACTS(S,n) - ACTS(S^L,n). Then there must be a clause C, C∈S, such that C - TG(n) = C'. But C can not contain L. Otherwise, suppose C ∈S^L. Then C'∈ACTS(S^L,n), and then C' would not be in ACTS(S,n) - ACTS(S^L,n). So we conclude C∈(S - S^L). Therefore C'∈ACTS((S-S^L),n).

Q.E.D.

Lemma I.4. ACTS((S - S^~L),n) = ACTS(S,n) - ACTS(S^~L,n)

Proof: Same as the proof for Lemma I.3.

Lemma I.5. If M is a model of the set ACTS(S,n+j), then M must be a model of the set ACTS(S,n).

Proof: Let C be any clause in S. According to Lemma I.1, each literal contained in the clause ACT(C,n+j) must be a literal in the clause ACT(C,n). So a model M of the set ACTS(S,n+j) must be a model of ACTS(S,n).

Q.E.D.

Lemma I.6. ACTS(S,n+j) = ACTS(ACTS(S,n),n+j)

Proof: Let C' be any clause in ACTS(S,n+j). Then there must be a clause C, C∈S, such that C' = ACT(C,n+j). But

C' = ACT(C,n+j)
= ACT(C,n) - TG(n+j) /* Lemma I.1 */
= ACT(ACT(C,n),n+j). / * definition */

so C'∈ACTS(ACTS(S,n),n+j).

Let C' be a clause in ACTS(ACTS(S,n),n+j). Then there is a clause C", C"∈ACTS(S,n), such that C' = ACT(C",n+j), i.e. C' = C" - TG(n+j). Because C"∈ACTS(S,n), there must be a clause C, C∈S, such that C" = C - TG(n). So C' = (C - TG(n)) - TG(n+j). According to Lemma I.1, C' = C - TG(n+j) = ACT(C,n+j). So C'∈ACTS(S,n+j).

Q.E.D.

Lemma I.7. Let C be a clause in SS(n)`~L. If C does not contain a literal complementary to a literal in REST(G), then there must be one and only one H0-resolvent HR(C,G,L,n) of G against C upon L in level n+j, and one and only one H0-resolvent HR(ACT(C,n),ACT(G,n),L,n) of ACT(G,n) against ACT(C,n) upon L in level n+j, such that ACT(HR(C,G,L,n),n) = HR(ACT(C,n),ACT(G,n),L,n).

Proof: Because G and L are all simple ground clauses, and, according to the hypothesis, C will not contain a literal complementary to a literal in REST(G), then there must be one and only one H0-resolvent of G against C, HR(C,G,L,n). We have shown in the proof of Lemma I.2 that TG(n) does not contain literal L or ~L, so ACT(G,n) contains literal L and ACT(C,n) contains literal ~L. Then there is one and only one H0-resolvent of ACT(G,n) against ACT(C,n), HR(ACT(C,n),ACT(G,n),L,n).

Let H = HR(C,G,L,n), H' = HR(ACT(C,n),ACT(G,n),L,n). We prove H' = ACT(H,n), i.e.. H' = H - TG(n).

Case 1: Let P be a literal of H'. According to the index of P, there are only two cases:

Case 1.1: The index of P in H' is n. Then P must be a literal inherited from ACT(G,n). Because ACT(G,n) = G - TG(n), TG(n) must not contain literal P. Then G contains a literal $_{n}$P, so H must contain a literal $_{n}$P. So H - TG(n) must contain a literal $_{n}$P.

Case 1.2: The index of P in H' is n+j. Then P must be a literal inherited from ACT(C,n) and there must be no literal the same as P in ACT(G,n). Because ACT(C,n) = C - TG(n), TG(n) does not contain literal P. Then P will not be merged by any literals in REST(G) because all literals in REST(G) must be identical to a literal in REST(ACT(G,n)) or a literal in TG(n). So H must contain a literal $_{n+j}$P. We have shown that TG(n) does not contain P, so H - TG(n) (ACT(H,n) contain a literal $_{n+j}$P.

Case 2: Let P be a literal of H - TG(n). According to the index of P, there are only two cases:

Case 2.1: P has an index n. Then there must be a literal P with an index n in G. So there is a literal with an index n in ACT(G,n). Then H' contains a literal $_{n}$P.

Case 2.2: P has an index n+j. Then TG(n) must not contain a literal P. Because the literal P of H must be inherited from C, and ACT(C,n) = C - TG(n), ACT(C,n) must contain a literal P. Note REST(G) must not contain a literal P, because otherwise the P of H inherited from C will be merged in H. Then REST(ACT(G,n)) does not contain a literal P. So the H-resolvent of ACT(G,n) against ACT(C,n) must contain a literal $_{n+j}$P.

Thus we have proved that ACT(H,n) and H' have the same set of literals (including indices). We suppose that the literals in ACT(H,n) and in H' are ordered similarly (we can do so, because that we did not claim any particular order for the literals inherited from the rule clause in the definition of H0-resolvent).

Q.E.D.

Lemma I.8. Let S be a set of clauses obtained from SS(n) by deleting any clause containing a literal complementary to a literal in REST(G). Then

$$ACTS(HRS(S,G,L,n),n) = HRS(ACTS(S,n),ACT(G,n),L,n),$$

where HRS(S,G,L,n) = {HR(C,G,L,n): C∈S}.

Proof: Trivial according to Lemma I.7.

Theorem I.4. If n is a regular node, G is the first goal clause of node n, L is the first literal of G, the index of L is n and n+j is the first node produced by H-DEDUC(n), and if ACTS(SS(n),n) is unsatisfiable, then ACTS(SS(n+j),n+j) is unsatisfiable.

Proof: Suppose the hypothesis of the theorem is true, but ACTS(SS(n+j),n+j) is satisfiable. Let M be a model of the set ACTS(SS(n+j),n+j). We are to obtain contradiction by constructing a model for the set ACTS(S(n),n).

Because no clause in the set ACTS(SS(n+j),n+j) contains the literal L or $\sim$L or a literal in TG(n+j), we can suppose that M does not contain $\sim$L and any literaL identical to a literal in TG(n+j).

First we notice that

ACTS(SS(n+j),n+j)
= ACTS((GS(n+j)$\cup$RS(n+j)),n+j) /* definition */
= ACTS((GS(n+j),n+j)$\cup$ACTS(RS(n+j)),n+j). /* Lemma I.2 */

Then M must be a model of the set ACTS(RS(n+j),n+j). According to Lemma I.5, M must be a model of the set ACTS(RS(n+j),n).

Because

ACTS(RS(n+j),n)
= ACTS((SS(n) - SS(n)^L - SS(n)^$\sim$L),n) /* definition */
= ACTS(SS(n),n) - ACTS(SS(n)^L,n) - ACTS(SS(n)^$\sim$L,n), /* Lemma I.3 */

M must be a model of the set:

ACTS(SS(n),n) - ACTS(SS(n)^L,n) - ACTS(SS(n)^$\sim$L,n).

We divide the set SS(n)^$\sim$L into two sets: Let the clauses in SS(n)^$\sim$L, which contain no literal complementary to a literal in REST(G), form a set S1, and the rest clauses of SS(n)^$\sim$L form a set S2. Note that any resolvent of G against a clause in S2 upon the literal L must be rejected from the H0-resolvent of G in the deduction, because it is a tautology. Then any clause in GS(n+j) must be an H0-resolvent of G against a rule clause which is a member of the set S1. But, for the set S1, we have the following:

ACTS(GS(n+j),n+j)
= ACTS(HRS(S1,G,L,n),n+j)
= ACTS(ACTS(HRS(S1,G,L,n),n),n+j) /* Lemma I.6 */
= ACTS(HRS(ACTS(S1,n), ACT(G,n), L, n), n+j) /* Lemma I.8 */

Because M is a model of the set ACTS(GS(n+j),n+j), then, according to the above equation, M is a model of the set:

ACTS(HRS(ACTS(S1,n), ACT(G,n), L, n), n+j).

Then for any clause H in the set HRS(ACTS(S1,n),ACT(G,n),L,n), ACT(H,n+j) must be true in M.

We have shown in the proof of Lemma I.7 and Lemma I.8 that, for each clause C in S1, there must be one and only one H0-resolvent H of ACT(G,n) against ACT(C,n), and all literals of H with the index n+j must be inherited from the literals of ACT(C,n). But ACT(H,n+j) can only contain the literals whose index is n+j. So each literal in ACT(H,n+j) must be contained in ACT(C,n). We have proved that ACT(H,n+j) is true in M, so ACT(C,n) is also true in M. Therefore M must be a model of the set ACTS(S1,n).

Let M' be the set consisting of all members of M and the literal L as well as all literals which are

complementary to some literal in REST(G). Note that REST(G) = TG(n+j). Recall that M does not contain ~L or any literal identical to some literal in TG(n+j). So M' is consistent.

Because, for each clause C in S2, ACT(C,n) must contain at least one literal complementary to a literal in REST(G), then M' must be a model of ACTS(S2,n).

Because each clause in ACT(SS(n)˙L,n) must contain a literal L, then M' must be a model of the set ACTS(SS(n)˙L,n).

Thus M' is a model of the whole set ACTS(SS(n),n). Contradiction.

Q.E.D.

Theorem I.5. If n is a regular node and ACTS(SS(n),n) is unsatisfiable, then if H-DEDUC(n) backtracks to the pointer of the node n before a current goal G', G' = TG(n), is obtained, then ACTS(RS(n),n) must be unsatisfiable.

Proof: Suppose the hypothesis of the theorem is true, we prove ACTS(RS(n),n) is unsatisfiable.

If GS(n) is empty, then ACTS(SS(n),n) = ACTS(RS(n),n). So ACTS(RS(n),n) is unsatisfiable.

Now suppose GS(n) is not empty. We do the proof by induction on the size of node n. Let TG(n) be the clause, TG(n) = $_{i1}$T1...$_{im}$Tm, m$\geq$0, and let G be the first goal clause in GS(n).

If SIZE(n) = 0, we have shown in the proof of Theorem I.3, that G must be identical to TG(n). This means that the hypothesis of the theorem is false, so the theorem is true.

Induction hypothesis: Suppose the theorem is true for the case SIZE(n) < K, K > 0. Now we prove the theorem for the case that SIZE(n) = K.

If G is identical to TG(n), then the theorem is true because the hypothesis of the theorem is false. Now we suppose that G is not identical to TG(n). According to the definition of regular node, G must be in the following form:

$$G = {}_nL1\ {}_nL2 \ldots {}_nLh\ {}_{i1}T1 \ldots {}_{im}Tm,$$

where h > 0.

Then the next node produced by H-DEDUC(n) is node n+j. According to Theorem I.1, node n+j is a regular node. At next step, the procedure calls H-DEDUC(n+j). According to Theorem I.3, in a finitely many steps, H-DEDUC(n+j) must attain one of the following states:

State 1. A current goal G' = TG(n+j) is obtained. Because TG(n+j) = REST(G), there are only two cases:

Case 1. TG(n+j) = $_{i1}$T1...$_{im}$Tm. This means that the current goal of the procedure is identical to TG(n). So the theorem is true because the hypothesis of the theorem is false.

Case 2. G' = $_n$L2...$_n$Lh $_{i1}$T1...$_{im}$Tm, h > 1. Then the procedure will retrieve node n to obtain the rule clauses according to the index n of the first literal of G', and the next node produced should be n+j', j' > 0. So, the following deduction of the procedure is the same as the call H-DEDUC(n'), where the node n' is the node n modified by just replacing the goal G with the goal G'. As was verified in the proof of Theorem I.3, node n' is a regular node. Obviously ACTS(SS(n'),n') is unsatisfiable. Note SIZE(n') < K. According to the induction hypothesis, the theorem is true.

State 2. The procedure has backtracked to the pointer n of node n+j. By the definition of H-DEDUC, H-DEDUC(n) is called again in the next step. But this time the node n is modified: the original goal clause G has been deleted from GS(n). To distinguish, we denote this modified node n by n'(numerically n = n'). Now we prove that ACTS(SS(n'),n') is unsatisfiable.

Suppose ACTS(SS(n'),n') is satisfiable. Then it has a model M. Because ACTS(SS(n),n) = ACTS(SS(n'),n')∪{ACT(G,n')}, and since ACTS(S(n),n) is unsatisfiable and ACT(G,n') contains all of the literals L2,...,Lh, then M must not contain any of L2,...,Lh.

Because each clause C in RS(n+j) must be a clause in SS(n'), then according to the following equations:

$$\begin{aligned} & ACT(C,n+j) \\ &= C - TG(n+j) \\ &= C - {}_{n}L2\ldots{}_{n}Lh\ {}_{i1}T1\ldots{}_{im}Tm \\ &= C - {}_{i1}T1\ldots{}_{im}Tm - {}_{n}L2\ldots{}_{n}Lh \\ &= ACT(C,n') - {}_{n}L2\ldots{}_{n}Lh, \end{aligned}$$

and noticing that M does not contain any of L2,...,Lh, ACT(C,n+j) must be true in M. So RS(n+j) is satisfiable.

But, according to Theorem I.4, SS(n+j) must be unsatisfiable, and, according to Theorem I.2, SIZE(n+j) < K. Then, by the induction hypothesis, RS(n+j) should be unsatisfiable. Contradiction.

Finally, we conclude that ACTS(SS(n'),n') is unsatisfiable. Obvious, node n' is a regular node and SIZE(n') < K. By the induction hypothesis, the theorem is true for H-DEDUC(n'). So the theorem is true for H-DEDUC(n).

Q.E.D.

Corollary I.2. Procedure H-DEDUC is a decision procedure for propositional logic. Stated precisely, if S is a set of simple ground clauses and n is a primary node where RS(n) is empty and GS(n) = S, then H-DEDUC(n) must exit in a finitely many steps; if it exit with "box", then S is unsatisfiable, if it exits with "FAIL", then S is satisfiable.

Proof: This follows from Corollary I.1 and Theorem I.5. Since all new clauses produced in the procedure are obtained by resolution,and resolution itself is sound, the corollary must be true.

Q.E.D.

Now we can prove Theorem 1 of section 3. The theorem is stated precisely as following:

Theorem 1. If S is a set of simple ground clauses, node 1 is a primary node where RULES-GET(1) = NIL and GOALS-GET(1) = S, then H0-DEDUC(1) will exit in finitely many steps. If S is unsatisfiable, then the procedure will exit with "box", otherwise it will exit with "FAIL".

Proof: Note we allow RANDOM() to return any positive integer that we want. If we suppose that RANDOM() always returns 1, then we obtain the procedure H0-DEDUC.

Q.E.D.

Next, we lift the completeness theorem of H-DEDUC to be the completeness theorem of H1-DEDUC. First, we introduce two lemmas.

Lemma I.9. If node 1 is a primary node inputted to H-DEDUC, then we can use H-resolvents instead of

H0-resolvents in H-DEDUC and preserve the validity of all lemmas and theorems proved above for H-DEDUC.

Proof: Because there are more constraints applied to the H-resolvent than to the H0-resolvent, then the lemma is proved if we can justify that each H0-resolvent produced by H-DEDUC is a legal H-resolvent. This can be done by verifying the properties of regular nodes.

Because all nodes produced by H-DEDUC(1) are regular nodes, all H0-resolvents produced by H-DEDUC(1) are legal clauses. But, any H0-resolvent will be accepted by the local subsumption test if the resolvent is a legal clause. So the local subsumption test, if it is used in H-DEDUC, will not discard any H0-resolvent produced by H-DEDUC(1).

Note also that all clauses stored in a regular node are common tail mergeable. So, the restriction on common tail mergeable, if it is used in H-DEDUC, must not discard any H0-resolvent.

Furthermore, because non of the clauses in the regular nodes is a tautology, then for any goal clause used in H-DEDUC(1), the first literal must not be complementary to any other literal of the clause. Because the goal clause must be common tail mergeable with its rule clause, the first literal can not be complementary to a literal of the rule clause whose index is among the common tail indices of these two clause. This means that the constraint on common tail indices of the H-resolvent, if it is used in H-DEDUC, will not discard any H0-resolvent.

Q.E.D.

Due to Lemma I.9, we can use H-resolvents in H-DEDUC in the place of the H0-resolvents in the following discussion.

Corollary I.3 The level subgoal reordering (see definition in the section 13), if used in reordering the literals of the goal clause encountered[27] in H-DEDUC, will not cause incompleteness for the procedure H-DEDUC.

Proof: Because we did not claim any particular order of the literals of an H-resolvent inherited from the rule clauses during the resolution except that they are all on the left side of the resolvent, then we can suppose the resulting order of these literals of the goal clause obtained by level subgoal reordering was arranged before, at the time that this goal clause was obtained in the previous deduction, or at the time that this goal clause was inputted if it is an input goal clause.

Q.E.D.

Corollary I.4 For each recursive call of H-DEDUC(n) where n is a node, the clauses in GS(n) can be reordered without causing incompleteness to H-DEDUC.

Proof: Because we did not claim any particular order of the H-resolvents obtained in a recursive call of the procedure, then, for any call of H-DEDUC(n) where n is a node produced before, we can suppose the resulting order of clauses in GS(n), if they are reordered, was arranged before, at the time that these clauses (H-resolvents) were produced or at the time that these clauses (if they were input clauses in the first node) were inputted.

Q.E.D.

---

[27] For each recursive call of H-DEDUC(n) or H1-DEDUC(n) where n is a node, the first goal clause in GS(n) will be the goal clause encountered (in next step)

Definition: IO-instance. Clause C' is called an *IO-instance* of a clause C if C' and C are both properly indexed clauses and there is a substitution θ, such that C' is identical to Cθ up to the order and indices of their literals and $FL_{c'}$ is equal to $FL_c\theta$ where $FL_{c'}$ and FL_c are the sets of framed literals of C' and C respectively.

Definition: A set S1 of clauses is said *subsumed by a set* S2 of clauses iff each clause in S1 is an IO-instance of a clause in S2.

Definition: Node subsumption. Let n' be a node produced by the procedure H-DEDUC, n be a node produced by the procedure H1-DEDUC. Then node n' is said to be *subsumed by the node* n iff

1. numerically, n' = n;
2. SS(n') is subsumed by RS(n);
3. GS(n') is subsumed by GS(n);

Definition: A goal clause of a hierarchical deduction is called a *wrong goal clause* iff the procedure will backtrack in finitely many steps to the pointer of such a node whose parent is this goal clause (or say, the node is produced from this goal clause). Otherwise the goal clause is called a *correct goal.*

Lemma I.10 For the procedure H-DEDUC, a wrong goal clause can be deleted without affecting the final result of the procedure.

Proof: Trivial.

Lemma I.11. Let S be a set of simple ground clauses, and n be a primary node where RS(1) = NIL and GS(1) = S. If we set a local depth limit, that is greater than the total number of the different atoms in S, for the deduction of H-DEDUC, then this local depth limit will not cause incompleteness.

Proof: Recall the local depth limit is defined to be the maximum number of framed literals of a legal H-resolvent (previously H0-resolvent). Suppose the number of different atoms in S is N. Because each clause C produced in H-DEDUC is a legal and simple ground clause, there must be no two literals in C or in the framed literals of C which share the same atom. So the procedure H-DEDUC(1) can not produce (accept) an H-resolvent which has more than N framed literals. So the local depth limit with a value greater than N will not affect the procedure H-DEDUC(1).

Q.E.D.

Lemma I.12. Let S be a set of clauses, node 1 be the first node where RS(1) = S and GS(1) = S. Let n be any node produced by H1-DEDUC(1). Then for any finite local depth limit, H1-DEDUC(n) must backtrack to the pointer of n if "box" is not produced in finitely many steps.

Proof: Suppose the hypothesis of the lemma is true, we first prove the following proposition.

> Under any finite local depth limit, if there is an infinite deduction path, $\Delta = G_0, G_1, \ldots, G_i, \ldots$, from any clause G_0, where G_0 is a resolvent produced by H1-DEDUC(1) or an input goal clause, such that G_{i+1} is a legal H-resolvent of G_i produced by H1-DEDUC, then there must be a clause G' in the Δ such that G' is a reduction of G_0.

We prove the proposition by induction on the difference of the local depth limit N from the number of framed literals of the starting clause G, N - $FL_G\#$.

Suppose

$$G_0 = {}_pL1\ {}_pL2...{}_pLh\ {}_{q1}T1...{}_{qm}Tm,$$

where $p > q1 \geq ... \geq qm$, $h > 0$, $m \geq 0$.

1. $N - FL_{G_0}\# = 0$. In this case, only the reduction of G_0 can be a legal H-resolvent of G_0. Then the immediate descendant G_1 of G_0 in Δ must be a reduction of G_0. So the proposition is true.

2. Suppose the proposition is true for $N - FL_{G_0}\# < K$, $K > 0$. Now we prove the proposition for the case $N - FL_{G_0}\# = K$.

If G_1 is a reduction of G_0 then we have proved the proposition. Now suppose G_1 is an expansion of G_0. Suppose

$$G_1 = {}_rP1...{}_rPs\ {}_pL2\theta...{}_pLh\theta\ {}_{q1}T1\theta...{}_{qm}Tm\theta.$$

where $r > p > q1 \geq ... \geq qm$, $s \geq 1$, and θ is the substitution for obtaining G_1 from G_0.

Note the framed literal of G_1 is $FL_{G_0}\theta \cup \{{}_pL1\theta\}$, so $N - FL_{G_1}\# < K$. Then by the induction hypothesis, starting from G_1, there is a clause G_k in Δ which is a reduction of G_1.

If $s = 1$, then G_k must also be a reduction of G_0. Thus we have proved the proposition.

Suppose $s > 1$, then G_k must have the same number of framed literals as that of G_1, because the index r of the first literal of G_k is the same as that of G1 (This point was verified in the proof of Theorem I.3). Now, starting from G_k in Δ, we can repeat the above justification to find a reduction of G_k in Δ. But, there are only a finite number of literals of G_1 whose index is r, so the above justification can repeat at most finitely many times until a reduction G' of G_1 in Δ, which is also a reduction of G_0, is obtained.

$$G' = {}_pL2\lambda...{}_pLh\lambda\ {}_{q1}T1\lambda...{}_{qm}Tm\lambda.$$

where λ is a composition of the substitutions used in obtained G' from G.

Thus we have proved the proposition.

According to the proposition just proved, we can repeatedly find a reduction G' of G in Δ, then find a reduction of G" of G' in Δ, and so on. But, there are only a finite number of literals in the goal clause G_0, so there must be a clause in Δ, which is "box", and from which, there can not be any legal H-resolvent producible by H1-DEDUC. This Contradicts the hypothesis that Δ has an infinite length.

Thus, the proposition can be true only in the case, such that the hypothesis is false, i.e., under any finite local depth limit, H1-DEDUC can not produce an infinite sequence of descendants of a goal clause.

Noticing further that from each goal clause, H1-DEDUC can produce only a finite number of resolvents, so we finally conclude that from any node n produced in H1-DEDUC(1), there is only a finite searching space. If "box" is not produced in finitely many steps, then procedure must backtrack to the pointer of the node n.

Q.E.D.

Lifting Lemma. Let C and G be two properly indexed clauses with no variable in common. Let C' and G' be two simple ground clauses, which are the IO-instances of clause C and G respectively. Then if there is an H-resolvent H' of G' against C' in level n, where n is any integer greater than every index of G' and C', then there must be an H-resolvent H of G against C in the level n, such that H' is an IO-instance of H or an IO-instance of an H-factor of H.

We will not present the formal proof for this lemma, because the proof is essentially the same as the typical proof of the lifting lemma for the ordered resolvents of the ordered clauses, and augmented by the justification concerning the sets of framed literals of the resolvents. The reader may refer to [11].

Now we prove Theorem 2 of section 7, i.e., H1-DEDUC is a complete procedure for first order logic. This theorem is stated precisely as following:

Theorem 2. If S is a unsatisfiable set of clauses (each clause has empty set of framed literals), S_f is the set consisting of all clauses of S and all factors of the clauses of S, and if node 1 is the first node where RS(1) = S_f, GS(1) = S_f, then there is a local depth limit N, such that H1-DEDUC(1) exits with "box" in finitely many of steps.

Proof: According to Herbrand's theorem [6], we can obtain a minimally unsatisfiable set S' of clauses such that each member of S' is a ground IO-instance of a clause in S_f. Because S_f includes all factors of the clauses in S, we can suppose, without losing generality, that each clause in S' is a simple ground clause. We suppose that the local depth limit used in H1-DEDUC is larger than the number of different atoms in S'.

First we construct a primary node 1' such that RS(1') = NIL and GS(1') = S'. Then we initialize both calls of the procedures, H-DEDUC(1') and H1-DEDUC(1). We will use n' to denote a node produced by H-DEDUC (numerically n' = n).

We first prove the following proposition:

If each wrong goal encountered in the deduction of H-DEDUC(1') is deleted, then we can obtain a deduction path of H-DEDUC(1') and a deduction path of H1-DEDUC(1) such that:

1. For each recursive call of H-DEDUC(m'), there is a recursive call of H1-DEDUC(m), such that node m' is subsumed by node m;
2. For each node n' produced by H-DEDUC(1') before H-DEDUC(m') is called, there is a node n produced by H1-DEDUC(1) before H1-DEDUC(m) is called, such that SS(n') is subsumed by RS(n).

We prove the proposition by induction on the number of times of recursive calls of H-DEDUC.

1. H-DEDUC is called only 1 time, i.e. H-DEDUC(1'). Because, by condition, node 1' is subsumed by node 1, and since no other node is produced before, then the above proposition is trivially true.

2. We suppose that the proposition is true until the K'th call of H-DEDUC where K > 0. Now H-DEDUC(n') and H1-DEDUC(n) are currently called respectively.

We show that the proposition is true for the K+1'th call of H-DEDUC. If as was supposed each wrong goal clause is deleted, then each node produced by H-DEDUC, until now, is obtained from a correct goal. Since, according to lemma I.10, deleting the wrong goals from H-DEDUC will not cause incompleteness to H-DEDUC, there must be at least one correct goal clause in GS(n'). Note that, by the induction hypothesis, node n' is subsumed by the node n. To prove the proposition, we need only to consider the following cases:

Case 1. There is a clause G' in GS(n') which is an IO-instance of the first goal clause G in GS(n), and G' is a correct goal.

According to Corollary I.4, without losing completeness, we can change the order of the clauses in GS(n')

to make G' be the first in GS(n'). So, without losing generality, we can suppose G' is the first goal clause G' in GS(n').

Let H1-DEDUC(n) go ahead for one step, until the next call H1-DEDUC(m), where m is the node name assigned by function GETNODE()(see definition of H1-DEDUC). Let G1 = REORDER(G). Now we reorder G' to be G1', so that G1' is an IO-instance of G1. Note that, according to Corollary I.3, the level subgoal reordering for the goal clause G1' will not cause incompleteness to H-DEDUC. Then let H-DEDUC(n') go though this round of deduction, during which let RANDOM() = m'- I', where I' is the index of the first literal of G1. Then H-DEDUC(n') is going to call H-DEDUC(m').

Because node n' is a regular node, I'$\leq$n'. By the definition of GETNODE, m = n+1 (m' = n'+1'). So m' - I' must be a positive integer. This conforms to the definition of the function RANDOM.

We need to prove that node m' is subsumed by node m. Because G1' is an IO-instance of G1, the indices of the first literals of G1' and G1 must be identical numerically. Suppose they are I' and I respectively. By the induction hypothesis, the set SS(I') must be subsumed by the set RS(I) (Recall that the rule set used in H-DEDUC is SS(I') = GS(I')$\cup$RS(I') and the rule set used in H1-DEDUC is RS(I)).

Then for any clause C' in SS(I'), there must be a clause C in RS(I), such that C' is an IO-instance of C.

Suppose there is an H-resolvent H' of G1' against C'. Then, because H-factors are included in H1-DEDUC, and according to the lifting lemma, there must be an H-resolvent H of G1 against C such that H' is an IO-instance of H or H' is an IO-instance of an H-factor of H.

Therefore GS(m') must be subsumed by GS(m).

Because RS(m') is a subset of SS(I') and RS(m) = GS(m)$\cup$SS(I), SS(m') must be subsumed by RS(m).

Thus we can conclude that node m' is subsumed by node m.

Because only the node n' and the node n are modified in the above step and since SS(n) is not affected by the modification of node n, and G' is deleted from SS(n'), SS(n') must still be subsumed by RS(n).

Thus, the proposition is true for case 1.

Case 2. There is no correct goal clause in GS(n') that is an IO-instance of the first goal clause of the node n. Let H-DEDUC(n') pause, and let H1-DEDUC(n) go ahead. According to Lemma I.12, by means of the local depth limit, the procedure will backtrack to the node n again if the "box" is not produced in finitely many steps.

If "box" is obtained by H1-DEDUC during the above process, then we have proved the proposition. Otherwise we can suppose that H1-DEDUC(n) has backtracked to node n. Note the original first goal clause has been deleted from the GOALS field of node n. But node n' must still be subsumed by this modified node n. Now we can repeat the discussion from case 1.

Case 3. The first goal clause G' in node n' is an IO-instance of the first goal clause in node n, but it is a wrong goal. According to Lemma I.10, we can simply delete G' from the GOALS field of node n' without losing completeness. Obviously, the node n' with G' deleted is subsumed by the node n. Now we can repeat the discussion from case 1.

All the cases have been proved.

In fact, we may not know whether a goal clause in the node n' discussed above is a correct goal or a wrong goal before "box" is finally obtained by H-DEDUC(1'). But, we can construct the above proofs in following way:

1. Record the whole history of the deductions in constructing the proofs of H-DEDUC(1') and H1-DEDUC(1).

2. Each goal clause encountered in H-DEDUC is first treated as a correct goal in the execution of the above proof process.

3. If a backtracking happens in the deduction of H-DEDUC, or if a node produced (or modified later) by H-DEDUC is not subsumed by the corresponding node produced (or modified later) by H1-DEDUC, then the parent of the node, which has the empty GOALS field, must be a wrong goal. Remember this wrong goal. Then according to the recorded history, redo the proof process of H-DEDUC(1') and H1-DEDUC(1) until this wrong goal is encountered. But, this time, we delete the wrong goal, and then continue the construction of the proofs.

Because H-DEDUC is a decision procedure for propositional logic, there can be only a finite number of possibilities of whether or not the goal clauses encountered in H-DEDUC are wrong. Also, note that, according to Lemma I.11, the local depth limit used in H1-DEDUC is large enough for deducing "box" by H-DEDUC(1'). Then, after a finite number of repetitions of the above process, we must obtain a straightforward proof of H-DEDUC(1') and, accordingly, a deduction of "box" by H1-DEDUC(1) is obtained.

Q. E. D.

Acknowledgments

This project was done under the principle guidance of my supervisor, Professor W. W. Bledsoe. He contributed substantially to this paper by his mathematical thinking, valuable suggestion, phrasing and correction of the manuscripts. I am deeply indebted to Prof. W.W. Bledsoe, also for his inspiration and encouragement.

I would like to thank Prof. R.B. Anderson and D.L. Simon, who read this manuscript and made a number of corrections. Thanks S. Q. Chou and L. M. Hines for helpful talks and for help in my use of SCRIBE.

References

1. Anderson, R., and Bledsoe, W. W. A linear format for resolution with merging and a new technique for establishing completeness. J. ACM 17 (July 1970), 525-534.

2. Bledsoe, W. W. Non-resolution theorem proving. Artificial Intelligence 9 (1977), 1-35.

3. Bledsoe, W. W. and Hines, L. M. Variable elimination and chaining in a resolution-based prover for inequalities. Technique report, ATP-56A, University of Texas at Austin. 5th Conference on Automated Deduction.

4. Boyer, R. S., and Moore, J S. A computational logic. Academic Press, 1979.

5. Boyer, R. S. Locking: a restriction of resolution. Ph.D. Thesis. University of Texas at Austin.

6. Chang, C., and Lee, R. C. Symbol logic and mechanical theorem proving. Academic Press, 1973.

7. Kowalski, R., and Kuehner D. Linear resolution with selection function. Artificial Intelligence 2 (1971), 227-260.

8. Loveland, D. W. Mechanical theorem-proving by model elimination. J. ACM 15, (April 1968), 236-251.

9. Loveland, D. W. A simplified format for the model-elimination theorem-proving procedure. J. ACM 16, (July 1969), 349-363

10. Fleising, S., Loveland D., Smiley III A. K., and Yarmush D. L. An implementation of the model elimination proof procedure. J. ACM 21, (January 1974), 124-139.

11. Loveland, D. W. Automated theorem proving; a logical basis. North Holland, 1978.

12. McCharen, J. D., Overbeek, R. A. and Wos, L. A. Problems and experiments for and with automated theorem-proving programs. IEEE trans. on compt. C-25, NO.8, (August 1976).

13. Nilsson, N. J. Principles of artificial intelligence. Tioga, Palo Alto, Calif., 1980.

14. Overbeek, R. A. A new class of automatic theorem proving algorithms. J. ACM 21 (1974) 191-200.

15. Tyson, W. M. A priority-ordered agenda theorem prover, Ph.D. dissertation, University of Texas at Austin (1981).

16. Wos, L. T., Carson, D. F. and Robison, G. A. Efficiency and completeness of the set of support strategy in theorem-proving. J. ACM 12, (1965), 687-697.