(324)

A STUDY OF RESOLUTION STRATEGIES IN FIRST ORDER PREDICATE CALCULUS

OINAM IMOCHA SINGH

Submitted in partial fulfilment of the requirement for the degree of Master of Philosophy

at the
School of Computer & Systems Sciences
Jawaharlal Nehru University
New Delhi - 110 067

July 1981.

CERTIFICATE

SCHOOL OF COMPUTER & SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110 067.

This research work presented in this dissertation entitled "A Study of Resolution Strategies in First Order Predicate Calculus" has been carried out at the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi - 110 067.

This work is original and has not been submitted in part or full for any other degree or diploma of any University.

O. Imocha
OINAM IMOCHA SINGH
STUDENT

N. P. Mukherjee
PROF. N.P. MUKHERJEE
DEAN

R. Sadananda
DR. R. SADANANDA
SUPERVISOR

ACKNOWLEDGEMENTS

I owe a great debt to Dr. R. Sadananda for developing the ideas of Automatic Theorem Proving in Artificial Intellegence. This dissertation itself was done in Jawaharlal Nehru University under his valuable guidance and supervision.

I also thank the Dean of the School of Computer and Systems Sciences, for giving all facilities of producing this dissertation.

I also thank the staff members and my colleagues of the School and specially to my classmates S.K. Rao and B. Rajeshwar for their important suggessions and encouragements during my years in Delhi.

My special thanks are also due to Ramesh Chander for his hardship in typing the thesis.

Finally I express my acknowledgement to my family members for their great care and long patience during my years in the University.

TABLE OF CONTENTS

CHAPTER - I

INTRODUCTION

Significant research effort on [illegible]orem Proving has surfaced in 1930 with the work of Herbrand [21]. And in late 1950s Newell, Simon, and Shaw produced some programmes on Whitehead and Russel's "Principia Mathematica" [63,45,46], consequently serious research attention was drawn towards this area. These works include those of Glertner, Haye, Newell, Simon, Shaw, Davis and Putnam, Gilmore [11,12,17, 27]. But perhaps the most remarkable achievement of it was seen during the mid 60s with the coming of Resolution Principle developed by J.A. Robinson [53].

Resolution Principle has added an additional dimension to the area of mechanical theorem proving. In this, one can directly visualize various practical applications of the theorem proving activity. The resolution procedure has become a powerful model for both proof finding and consequence finding in various branches of mathematics, and also in application areas where problems can be converted into one of proving an assertion. In practical implementation of resolution principle in mechanical theorem proving, one has to solve difficult problems in terms of combinatorial explosions demanding prohibitively large memory space and search efforts in digital computers. It means that theorems can be proved, or problems posed as theorems in some domain can be solved from the given premises or axioms using resolution principle.

Here attempts are being made to study various strategies available. And these strategies can be developed in resolution process under different problem situations. The discussion also includes a study of how the resolution inferences work differently on different problems. And for economic purposes (i.e. manual computation, easy understanding and economic computer time) the preference of the strategies for particular problems are also being made.

Further, since the resolution principle is defined within the domain of the first order predicate calculus, the scope and limitations inherent in the predicate calculii, in gereral, are also discussed.

CHAPTER - 2

RESOLUTION PRINCIPLE AND ITS BACKGROUND

This chapter provides a brief account of the resolution principle in automatic theorem proving and its fundamental requirements.

Mathematical logic some times called symbolic logic is the basic foundation of resolution principle and is the logic treated by mathematical methods [25A]. Logic can also be considered as a branch of knowledge concerning the reasoning process. In effect its primary concern is to find what follows from what. In any orderly arrangement of the content of mathematics, one could see the exhibition of logical connections. Similarly, logic is used in organising scientific knowledge, and as a tool of reasoning and argumentation in daily life. Some of the most frequently used connectives are $\sim$(not), $\vee$(or), $\wedge$(and), $\rightarrow$ (implies), $\leftrightarrow$ (if and only if) though a subset of this is sufficient from the logical point of view. The logical meanings of these connectives are well known.

Any declarative sentence which is represented by symbols using or without using the connectives can have the values either true (T) or false (F) but not both is termed as proposition and its logic is known as propositional logic. Given the truth values of the component sentences, one can find the validity of an arbitrary sentence in the propositional calculus by constructing a truth table.

But the propositional calculus is limited in its expressive power on account of non-availability of variables and quantifiers. Therefore, it often becomes difficult or some times even impossible to express many reality or mathematical prblems with the propositional calculus. However, the first order predicate calculus allows variables and quantifiers in it. In order to describe the first order predicate calculus we define the following concepts.

Quantifiers: The meaning of quantifier is well known. There are two quantifiers in first order logic -- The universal quantifier ($\forall$) and the existential quantifier ($\exists$), the scope of each is to be described by the paranthesis and are within the domain considered.

Predicate letter: A predicate letter of n arguments is an n-ary or n-place predicate letter.

Term: A term is either a constant (usually denoted by a, b, c, d, ..., numericals), a variable (usually denoted by s, t, u, v, x, y, ...) or a function (f, g, etc.).

Atomic formula: It is obtained by applying a predicate letter to terms.

Well formed formula(w.f.f.): A w.f.f. is recursively defined as follows:

1. A term is a w.f.f.

2. If G is a w.f.f., then $\sim$ G is a w.f.f.

3. If G and H are w.f.f.s, then any formula derived by applying the connectives on G and H is a w.f.f.

4. All w.f.f.s are generated by applying only these rules.

Free variables: Any variable in a w.f.f. is said to be a free variable if it is not bounded by any quantifier.

Closed w.f.f.: Any w.f.f. containing no free variable is called a closed w.f.f.

There are two aspects of logic -- the syntactic notion and the semantic notion. A w.f.f. is a syntactic or linguistic entity and is completely specified by a set of grammar rules. One can interprete a w.f.f. as a meaning or semantic notion.

The syntax of predicate calculus system involves

(a) The specification of an alphabet of symbols, and

(b) The definitions of various useful expressions that can be constructed from these symbols whereas the semantics of a statement is described by its interpretation.

Interpretation: It consists of

(1) a non-empty set of objects called the domain,

(2) an assignment of an object in the domain to each constant,

(3) an assignment of an n-ary function letter, and

(4) an assignment of an n-ary relation on the domain to each n-ary predicate letter.

If a w.f.f. G evaluates T under the interpretation I we say I satisfies G or I is a model of G.

Satisfiable: A w.f.f. G is said to be satisfiable if and only if there exists an interpretation I satisfying G.

Tautology: A tautology is a w.f.f. which is satisfied by all interpretations.

Unsatisfiable: A w.f.f. is said to be unsatisfiable if it gives the value false (F) for all possible interpretations.

Logical validity: A w.f.f. G is said to be logically valid if and only if it is satisfied by all possible interpretations.

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

Conjunctive and disjunctive formulae: A formula of the form $F_1 \wedge F_2 \wedge \ldots \wedge F_n$ is said to be conjunctive normal form if all F_i's are literals. And if it is in the form $F_1 \vee F_2 \vee \ldots \vee F_n$ it is of disjunctive form. In the later case also F_i's have the same meaning.

Logical consequence: A w.f.f. Q is a logical consequence of a set of axioms (premises) B if and only if every model of B is a model of Q.

Prenex normal reduction: A given set of w.f.f.s may be reduced to prenex normal form which is also sometimes referred to as the Skolem standard form. The following example will be able to explain its reduction procedure. The given w.f.f. is

$$(\forall x)\ (P(x) \rightarrow ((\forall y)\ (P(y) \rightarrow P(f(x,y))) \wedge \sim (\forall y)\ (Q(x,y) \rightarrow P(y)))) \qquad \ldots (*)$$

Step 1: Eliminating the implecation sign using the equivalence of $A \rightarrow B$ and $\sim A \vee B$ our example takes the form

$$(\forall x)\ (\sim P(x)\ \vee((\forall y)\ (\sim P(y)\ \vee P(f(x,y)))$$
$$\wedge \sim (\forall y)\ (\sim Q(x,y)\ \vee P(y))) \qquad \ldots (1)$$

Step 2: The scope of each negation sign is reduced till the scope reduces its application to a single predicate letter with the replace of

$\sim(A \wedge B)$ by $\sim A \vee \sim B$

$\sim(A \vee B)$ by $\sim A \wedge \sim B$

$\sim(\sim A)$ by A

$\sim(\forall x)\ A$ by $(\exists x)\ (\sim A)$

$\sim(\exists x)\ A$ by $(\forall x)\ (\sim A)$.

Hence our example reduces to

$$(\forall x)\ (\sim P(x)\ \vee\ ((\forall y)\ (\sim P(y)\ \vee P(f(x,y)))$$
$$\wedge\ (\exists y)\ (\sim(\sim Q(x,y)\ \vee P(y))))) \qquad \ldots (2)$$

and then to

$$(\forall x)\ (\sim P(x)\ \vee((\forall y)\ (\sim P(y)\ \vee P(f(x,y)))$$
$$\wedge\ (\exists y))\ (Q(x,y) \wedge \sim P(y)))) \qquad \ldots (3)$$

Step 3: To standardize variables: Within the scope of any quantifier symbol a variable bounded by that symbol is a dummy variable. It can be uniformly replaced by any other variable throughout the scope of the quantifier without changing the truth value of the w.f.f. Standardizing variables within a w.f.f. means to rename the dummy variables, to ensure that each quantifier has its own unique dummy variable. Thus, instead of writing $(\forall x)\ (P(x) \rightarrow (\exists x)\ Q(x))$, we write $(\forall x)\ (P(x) \longrightarrow (\exists y)\ Q(y))$. Hence our example reduces to

$$(\forall x)\ (\sim P(x) \vee ((\forall y)\ (\sim P(y) \vee P(f(x,y))) \wedge (\exists w)\ (Q(x,w) \wedge \sim P(w)))) \qquad \ldots (4)$$

Step 4: Here we eliminate existential quantifiers by replacing the variables within its corresponding bound by some constant or constants or functions of the bounded universally quantified variables in it. Hence our example is reduced to

$$(\forall x)\ (\sim P(x) \vee ((\forall y)\ (\sim P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \wedge \sim P(g(x))))) \qquad \ldots (5)$$

where $g(x)$ is called the Skolem function in it for it is the new function we derive in removing the existential quantifiers.

Step 5: Here the reduced form is converted into premex form by keeping all the quantifiers separately below. Hence the expression (5) is reduced to

$$(\forall x)\ (\forall y)\ (\sim P(x) \vee ((\sim P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \wedge \sim P(g(x))))) \qquad \ldots (6)$$

The quantifier part $(\forall x)\ (\forall y)$ is known as prefix and the rest part is known as matrix of expression (*).

Step 6: Here the matrix is put in conjunctive normal form using the rule -- to replace $A \vee (B \wedge C)$ by $(A \vee B) \wedge (A \vee C)$. Thus (6) reduces to

$$(\forall x)\ (\forall y)\ ((\sim P(x) \vee \sim P(y) \vee P(f(x,y))) \wedge (\sim P(x) \vee Q(x,g(x))) \wedge (\sim P(x) \vee \sim P(g(x)))) \qquad \ldots (7)$$

Step 7: Here we simply leave the universal quantifiers.

Step 8: Here we use commas in place of $\wedge$ signs and each expression separated by commas will be treated as an individual clause. Hence (*) finally reduced to

$\sim P(x) \vee \sim P(y) \vee P(f(x,y))$, $\sim P(x) \vee Q(x,g(x))$,
$\sim P(x) \vee \sim P(g(x))$

which is the standard form.

Clause: As in the above, is a finite disjunction of literals; when no literal is there we call the clause to be an empty clause and is denoted by .

Herbrand universe: Herbrand universe of a set of clauses is the set of interpretations over an unsatisfiable set of clauses.

Let Ho be the set of constants appearing in an unsatisfiable set of clauses. If no constants appears in S, H_0 is taken as the constant's singleton set $\{ a \}$. For i=0, 1,2, ..., let H_{i+1} be the union of H_i and the set of all terms of the form $f^n(t_1, \ldots, t_n)$ for all n-place functions f^n occurring in S, where t_j, j=1, ..., n are members of H_i. Then each H_i is called the i-level constant set of S and H_∞ or $\lim_{i \to \infty} H_i$ is called the Herbrand universe of S.

Example: $S = \{ P(a), \sim P(x) \vee P(f(x)) \}$ is an unsatisfiable set of clauses where

$$H_0 = \{ a \}$$
$$H_1 = \{ a, f(a) \}$$
$$H_2 = \{ a, f(a), f(f(a)) \}$$

and so on.

Herbrand base: The set of ground atoms of the form $P^n(t_1, \ldots, t_n)$ of all n-place predicates P^n occurring in the set S of clauses, where F_i's are elements of the Herbrand universe of S, is called the Herbrand base, or the atom set of S.

Ground instance: A ground instance of a clause C of a set S of clauses is a clause obtained by replacing variables in C by members of the Herbrand universe of S.

H-interpretation: Let S be the set of clauses; H, the Herbrand universe of S; and I an interpretation of S over H. I is said to be an H-interpretation of S if

(1) I maps all constants in S to themselves.

(2) if f is an n-place function symbol and $(h_1, \ldots, h_n)$ an element of H^n then $f(h_1, \ldots, h_n)$ is in H.

Let $A = \{A_1, A_2, \ldots, A_n, \ldots\}$ be the atom set of S, then an H-interpretation I is conveniently represented by $I = \{m_1, m_2, \ldots, m_n, \ldots\}$ in which m_j is either A_j or $\sim A_j$ for $j = 1, 2, \ldots$. The meaning of this convention is that if m_j is A_j, then A_j is assigned T, otherwise F.

Semantic tree: Let A be the atom set of the Set S of clauses. Then the semantic tree for S is a downward tree T, where each link is attached with a finite set of atoms or negations of atoms from A in such a way that:

(1) For each node N, there are only finitely many immediate links $L_1, \ldots, L_n$ from N. Let Q_i be the conjunction of all literals in the set attached to L_i, $i = 1, \ldots, n$. Then $Q_1 \vee \ldots \vee Q_n$ is a valid propositional formula.

(2) For each node N, let I(N) be the union of all sets attached to the links of the branch of T down to and including N. Then I(N) does not contain any complementary pair.

Complete Semantic tree: A semantic tree is said to be complete if and only if for every tip node N of the semantic tree, i.e. a node that has no links sprouting from it, I(N) contains either A_i or $\sim A_i$ for $i=1,2, \ldots$.

Failure node: A node N is a failure node if I(N) falsifies some ground instance of a clause in S, but I(N') does not falsify any ground instance of a clause in S for every ancester node N' of N.

Closed semantic tree: A semantic tree T is said to be closed if and only if every branch of T terminates at a failure node.

Inference node: A node N of a closed semantic tree is called an inference node if all the immediate descendant nodes of N are failure nodes.

To test whether a set S of clauses is unsatisfiable, we need to consider interpretations over the Herbrand universe of S. If S is false under all interpretations over the Herbrand universe of S, then we can conclude that S is unsatisfiable. Since there are usually many, possibly an infinite number of these interpretations, it is done in the systematic way of using a semantic tree. Here two versions of Herbrand's theorem are giving which are connected with it, i.e., with the notions of semantic tree. These versions are of very important in developing the resolution pro- perties.

Herbrand's theorem:

Version I: A set S of clauses is unsatisfiable if and only if corresponding to every complete semantic tree of S, there is a finite closed semantic tree.

Version II: A set S of clauses is unsatisfiable if and only if there is a finite unsatisfiable set S' of ground instances of the clauses of S.

Substitution: A substitution is a finite set of the form $\sigma = \{ a_1/x_1, \ldots, a_n/x_n \}$, where every term a_i is to be substituted for every variable x_i. If L is a formula $L\sigma$ denotes the formula resulting from performing the substitution of σ on L.

Unification: Two formulas L_1 and L_2 are said to unify if there exists a substitution σ such that $L_1\sigma$ is equal to $L_2\sigma$ and if L' is $L_1\sigma$, then L' is said to be an instance of L_1.

Most general unifier (m.g.u.): The substitution σ is said to be m.g.u. of two formulas L_1 and L_2 if $L_1\sigma = L_2\sigma$ and, for any other unifier λ of L_1 and L_2, $L_1\lambda = L_2\lambda$ is an instance of $L_1\sigma = L_2\sigma$. If two formulas unify, there exist a m.g.u. of the two formulas [12].

Using the set notation to represent clauses, the resolution rule of inference is:

Given two clauses $\{ L_1, \alpha \}$ and $\{ \sim L_2, \beta \}$, where α and β are disjunctions of literals and L_1 and L_2 are atomic formulas, and if L_1 and L_2 have the m.g.u. σ infer by resolvent $\{\alpha, \beta\}\sigma$. Here L_1 is any atomic formula and $\sim L_2$ is the negation of an atomic formula consisting of the same predicate symbol of L_1, but in general with different arguments.

Factor of a clause: Given a clause $C = \{L_1 \vee L_2 \vee \beta\}$, where L_1 and L_2 are literals and β is a disjunction of literals, if L_1 and L_2 have the m.g.u. then infer the factor $C' = (L_1\sigma \vee \beta\sigma)$. For example in $C = P(x) \vee P(f(y)) \vee \sim Q(x)$, the first and second literals have the m.g.u. $\sigma = \{f(y)/x\}$. Hence $C\sigma = P(f(y)) \vee \sim Q(f(y))$ is a factor of C.

Binary resolvent: Let C_1 and C_2 be two clauses with no variables in common. Let L_1 and L_2 be two literals in C_1 and C_2 respectively. If L_1 and $\sim L_2$ have a m.g.u. σ then the clause $(C_1\sigma - L_2\sigma) \cup (C_2\sigma - L_1\sigma)$ is called a binary resolvent of C_1 and C_2.

<u>Resolution Principle</u>:

Robinson's resolution proof procedures are refutation procedures; i.e., instead of proving a formula valid, it proves the negation of the formula to be an inconsistant system of clauses. The essential idea of resolution principle is to check whether a set S of clauses contains or could reduce from the clauses of S the empty clause $\square$.

If S contains the empty clause, then S is unsatisfiable [12]. If S does not contain the empty caluse directly we have to check up whether □ can be derived from S. By Herbrand's theorem version I, checking for the presence of □ is equivalent to counting the numbers of nodes of a closed semantic tree for S. Thus, S is unsatisfiable if and only if there is a finite closed semantic tree T for S. Clearly S contains □ if and only if T consists of only one node -- the root node. If S does not contain □ and is unsatisfiable set, T contains more than one node and we can reduce the number of nodes in T to one, and □ can be forced to appear. This is what resolution principle does. Thus resolution is an inference used to generate new clauses from S so that some nodes of the original T could be reduced to failure nodes and to derive □ at the end.

The resolution procedure is the most efficient one amongst other techniques for theorem proving, and is better than the Herbrand's method. In Herbrand's procedure it is a very long process to check up the inconsistency of a set of clauses taking enourmous H-interpretations from Herbrand's universe of the set, which is again based on ground instances only.

The resolution principle is sound, effective and complete [62]. The soundness of resolution principle means that a clause logically implies each of its factors and that two clauses together, logically imply each of their resolvents. If every theorem produced by an inference is valid, then the inference is called a sound inference. The effectiveness of the resolution principle means that one can write a computer program, which, in a finite number of steps, will find the factors of any clause and the resolvents of any two clauses. And, if a search organization, which we have been referring as inference or proof procedure, permits a theorem to be established, whenever the conclusion logically follows from the premises and the asserted rules, then the proof search organization system is called complete [36].

Here statements of two theorems on the completeness of resolution principle are given. These will be of great importance in the next chapter where we discuss the refinements of resolution principle.

Theorem(completeness of resolution): A set S of clauses is unsatisfiable if and only if there is a deduction of the empty clause $\square$.

Theorem (on consequence finding): If a clause C is a consequence of a finite non-empty set of clauses, then a clause T can be found in a finite number of applications of the resolution principle such that C is an immediate consequence of T alone [8].

CHAPTER - 3

REFINEMENTS OF RESOLUTION PRINCIPLE

In proving a theorem by resolution principle many inferences could be used, some of which may not be effective for the particular application or may produce clauses which could be of no use in that proof finding or consequence finding of the problem. It infers irrelevent and redundant resolvents or rather, it may lengthen the computer time in proving the theorem. To restrict and overcome these difficulties or inefficiencies many refinements or simplifiers of resolution procedure have been developed. For an effective application of resolution principle in proving a theorem in computer these refinements are discussed in this chapter.

Semantic Resolution

In semantic resolution a given set S of clauses is divided into two sets S_1 and S_2 such that clauses within the same set can not be resolved upon. For a clear idea, the explanation is given with the help of an example from propositional logic. Let S be the unsatisfiable set of clauses $\{\sim P \vee \sim Q \vee R, P \vee R, Q \vee R, \sim R\}$. Here $C_1 = \sim P \vee \sim Q \vee R$, $C_2 = P \vee R$, $C_3 = Q \vee R$, $C_4 = \sim R$. Now take the interpretation $I = \langle \sim P, \sim Q, \sim R \rangle$. C_2 and C_3 are falsified by I, whereas C_1 and C_4 are satisfied by it. Since S is unsatisfiable, no interpretation can satisfy or falsify all the clauses. Therefore, every interpretation partitions S into two non-empty sets. In this case the sets are $S_1 = \langle C_2, C_3 \rangle$ and $S_2 = \rangle C_1, C_4 \langle$. To block the resolutions, we use ordering of predicate symbols in it, as $P \rangle Q \rangle R$. Here $\rangle$ is convenned for ordering only. When we resolve one clause from S_1 with another from S_2, the resolved literal in the clause from S_1 should contain largest (i.e. the first in the ordering $\rangle$) symbol in that clause. With these two restrictions (splitting of S and ordering of predicates in S) we reduce many possible resolutions. Using these we can generate $C_5 = \sim Q \vee R$ from C_1 and C_2, and $C_6 = \sim P \vee R$ from C_1 and C_3. Now C_5 and C_6 are I-satisfiable. Hence we put in S_2. Resolving C_5 and C_2 we get $C_7 = R$, and resolving C_6 and C_3 we get the same clause $C_7 = R$. Since it is false in I, we keep it in S_1. Now resolvent of C_7 with C_4 gives $\square$. Here, two deductions of R are seen as deduced from C_1, C_2 and C_3 just by changing the order. To generate this R directly from C_1, C_2 and C_3 without going through the intermediate clauses C_5 or C_6, is the notion of <u>clash</u>. Here the set $\langle C_1, C_2, C_3 \rangle$ is called clash. Thus, combining the concepts of interpretational split of the set of clauses, ordering of the predicates and clash we can restrict many resolutions.

Semantic clash: Let I be an interpretation and P be an ordering of predicate symbols. Then the set of clauses $\{E_1, E_2, \ldots, E_q, N\}$, $q \geqslant 1$ is called a semantic clash with respect to P and I (or in short P I - clash), if and the only if E_i's and N satisfy the following conditions.

1. $E_1, \ldots, E_q$ are false in I.

2. Let $R_1 = N$, For each $i = 1, \ldots, q$, there is a resolvent R_{i+1} of R_i and E_i.

3. The literal in E_i, which is resolved upon, contains the largest predicate symbol in E_i, $i = 1, \ldots, q$.

4. R_{q+1} is false in I.

R_{q+1} is called a P I -resolvent of the P I clash $\{E_1, \ldots, E_q, N\}$. We also call these E_i's as electrons and N as neucleus with these properties.

Example: Let $E_1 = \sim Q(z) \vee \sim Q(a)$,

$E_2 = R(b) \vee S(c)$,

$N = Q(x) \vee Q(a) \vee \sim R(y) \vee \sim R(b) \vee S(c)$.

Let $I = \{Q(a), Q(b), Q(c), \sim R(a), \sim R(b), \sim R(c), \sim S(a), \sim S(b), \sim S(c)\}$. Let P be an ordering of predicate symbols in which $Q > R > S$. Let $R_1 = N$. There is a resolvent of R_1 and E, namely $R_2 = (\sim R(y) \vee \sim R(b) \vee S(c))$. Then there is a resolvent of R_2 and E_2, namely $R_3 = S(c)$ which is false in I. Since $\{E_1, E_2, N\}$ satisfies all the four conditions, it is a P I-clash. The P I -resolvent of this clash is S(c).

P I -deduction: Let I be an interpretation for the set S of clauses, and P be an ordering of predicate symbols in S. A deduction from S is called a P I - deduction if and only if each clause in the deduction is either a clause in S, or a P I -resolvent.

Example: Let $S = \{A_1 \vee A_2,\ A_1 \vee \sim A_2,\ A_1 \vee A_2,\ \sim A_1 \vee \sim A_2\}$, $I = \{A_1, \sim A_2\}$ and ordering P as $A_1 < A_2$. then the deduction in Fig.3.1 is a P I -deduction of the empty clause $\square$ from S, and P I -resolvents in it are $\sim A_1$, A_2 and $\square$.

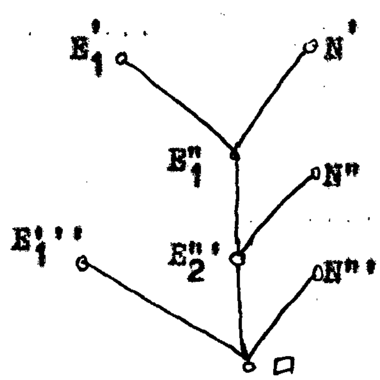


Fig.3.1

$E_1' = \sim A_1 \vee A_2$ $\quad N' = A_1 \vee \sim A_2$

$E_1'' = \sim A_1$ $\quad N'' = A_1 \vee A_2$

$E_1''' = \sim A_1$ $\quad N''' = A_1 \vee \sim A_2$

$E_2''' = A_2$

Theorem (completeness of P I -resolution): If P is an ordering of predicate symbols in a finite and unsatisfiable set S of clauses, and if I is an interpretation of S, then there is a P I -deduction of the empty clause from S.

Hyper resolution and set of support strategy: special cases of semantic resolution:

Hyper resolution [54].

If we use an interpretation I in which every literal is the negation of an atom, every electron and every P I -resolvent must contain only atoms. Similarly, if every literal in I is an atom, then every electron and every P I -resolvent must contain only negated atoms. Thus simplifies in choosing the clauses to be resolved upon. This method is called Hyper resolution.

A <u>positive clause</u> is the clause which contains nor negation sign. A negative clause is the one where all of its literals contain negation sign. If it is neither negative nor positive then it is termed as mixed clause.

A <u>positive hyper resolution</u> is a special case of P I -resolution, in which every literal in the interpretation I contains the negative signs only. And if it contains no negatiion sign, then it is <u>negative hyper</u>

resolution.

Set-of-support strategy [70]

A subset T of a set S of clauses is called a set of support of S if S-T is satisfiable. A set-of-support resolution is a resolution of two clauses that are not both from S-T. A set-of-support deduction is a deduction in which every deduction is a set-of-support resolution.

Set-of-support strategy consists designating the conclusion of the projected theorem, and a small number of relevent axioms, as - having the support property, i.e., lying in the set of support for the theorem. Thereafter, only those pairs of clauses, which contain at leat one member with support, are considered for resolution, and every clause is automatically attributed to the support property. If the theorem consists of the axioms A_1, A_2, ..., A_n and the conclusion B, then in resolution procedure we need to prove $A_1 \wedge \ldots \wedge A_n \wedge \sim B$ to be unsatisfiable. Usually $A_1 \wedge \ldots \wedge A_n$ is satisfiable.

The set-of-support strategy is complete. It is employed only in proof finding. And it is aimed at avoiding the deduction of consequences which are redundant to the particular conclusion desired.

O

Semantic resolution using ordered clauses:

There are cases in which using ordering of predicate symbols only may not allow to single out uniquely a literal in an electron. Hence the inference may generate more than one semantic resolvent from a semantic clash. For example take

$$E = Q(a) \vee Q(b) \vee Q(c) \vee Q(d)$$
$$N = \sim Q(x).$$

where I is an interpretation with negative in every literal and P may be any ordering of the predicate symbols. Here even if $\langle E, N \rangle$ is a P I -clash, all literals in E contain the same predicate symbol Q, so we can not single out the literal on which the P I deduction is to be applied by the above rule. In order to remedy this situation, 'ordered clauses' is introduced [52,61,66] .

An ordered clause is a sequence of distinct literals and not a set of literals. According to our concept in a clause, if the literal L_2 follows another literal L_1 in the specified sequence, then we take $L_2 > L_1$. Thus, the last literal in an ordered clause will be considered to be the largest literal in the clause. An ordered clause is also written as a disjunction of literals. Here we define few more relevent concepts.

Ordered factor: If two or more literals with the same sign of an ordered clause C have a m.g.u., σ, then the ordered clause — obtained from the sequence $C\sigma$ by deleting any literal that is identical to a smaller literal in the sequence is called an ordered factor of C.

Ordered binary resolvent: A clause C is said to be an ordered binary resolvent of the clause C_1 against the clause C_2 if the following conditions are satisfied

(1) C_1 and C_2 are ordered clauses with no variables in common.

(2) if L_1 and L_2 are two literals in C_1 and C_2 such that L_1 and $\sim L_2$ have a m.g.u. σ, and

(3) if the ordered clause C is obtained by concatenating the sequences $C_1\sigma$ and $C_2\sigma$ by removing $L_1\sigma$ and $L_2\sigma$ and deleting any literal that is identical to a smaller literal in the remaining sequence.

<u>An ordered resolvent</u> of an ordered clause C_1 against an ordered clause C_2 is one of the following ordered binary resolvents:

1) an ordered binary resolvent of C_1 against C_2 ;

2) an ordered binary resolvent of C_1 against an ordered factor of C_2 ;

3) an ordered binary resolvent of an ordered factor of C_1 against C_2;

4) an ordered binary resolvent of an ordered factor of C_1 against an ordered factor of C_2.

Ordered resolution is an inference rule that generates ordered resolvents from a set of ordered clauses. It is also complete.

Ordered semantic clash: Let I be an interpretation. A finite sequence of ordered clauses, $(E_1, \ldots, E_q, N)$, $q \geqslant 1$ is called an ordered semantic clash with respect to I (or O I -clash) if and only if $\{E_1, E_2, \ldots, E_q\}$, referred as ordered electrons and N, referred as Ordered nucleus, satisfy the following conditions.

1) $E_1, \ldots, E_q$ are false in I.

2) Let $R_q = N$. For each $i = q, q-1, \ldots, 1$ there is an ordered resolvent R_{i-1} of E_i against R_i.

3) The literal in E_i that is resolved upon is the

last literal in E_i, $i=1, \ldots, q$; the literal in R_i that is resolved upon is the largest literal that has an instance which is true in I.

4) R_0 is false in I.

R_0 is called an O I -resolvent of O I -clash ($E_1, \ldots, E_q$, N).

O I -deduction (ordered semantic deduction): Let I be an interpretation for a set S of ordered clauses. A deduction from S is called an O I -deduction if and only if each ordered clause in the deduction is either an ordered clause in So or an O I -resolvent.

Slagle and Norton experimented with O I -resolution proving many theorems quite effectively [66]. But O I -resolution is not complete. The following is a counter example of it due to Anderson [1].

Example: Let $S = \{C_1, C_2, C_3, C_4, C_5, C_6\}$, where $C_1 = P \vee Q$, $C_2 = Q \vee R$, $C_3 = R \vee W$, $C_4 = \sim R \vee \sim P$, $C_5 = \sim W \vee \sim Q$, $C_6 = \sim Q \vee \sim R$. And C_i's are ordered clauses. The interpretation I is given as $I = \{\sim P, \sim Q, \sim R, \sim W$. Thus the clauses C_1, C_2, C_3 can be taken as ordered electrons, E_i's and clauses C_4, C_5, C_6 as ordered nucleus N. Now we get the resolvents

C_7 = R V P from O I -clash $\{C_3, C_1, C_5\}$

C_8 = P V Q from O I -clash $\{C_1, C_2, C_6\}$

from the clauses C_{1-8} we can obtain the O I -resolvent

C_9 = Q V R from O I -clash $\{C_2, C_7, C_4\}$.

We know that C_8 and C_9 are in S. Therefore, from clauses C_{1-9} we can not produce any new O I -resolvent. That is the empty clause can not be produced by O I -resolution even though S is unsatisfiable. Hence, O I -resolution is not complete.

Lock Resolution [4]

This refinement was introduced by Boyer in 1971 [4]. It is similar to the concept of ordered clauses of semantic resolution. Given a set S of clauses, ordering of literals of the clauses of S using indices is the idea of lock resolution. It involves arbitrary indexing each occurrence of a literal in S with an integer ; different occurrences of the same literal in S may be indexed differently. Resolution is then permitted only on literals of lowest index in each clause. The literals in resolvents inherit their indices from their parent clauses. If a literal in a resolvent has more than one possible inherited indieces, the lowest index is assigned to this literal.

Consider $S = \{C_1, C_2\}$ where $C_1 = {}_1P \vee {}_2Q$ and $C_2 = {}_3{\sim}P \vee {}_4Q$. The integer beneath a literal is the index associated with that literal. Since the index of ${}_1P$ is lower than that of ${}_2Q$, ${}_1P$ is permitted to be resolved upon. Similarly, since the index of ${}_3{\sim}P$ is lower than that of ${}_4Q$, ${}_3{\sim}P$ is permitted to be resolved upon. Thus, resolving C_1 and C_2 upon ${}_1P$ and ${}_3{\sim}P$, we obtain $C_3 = {}_2Q \vee {}_4Q$. Now ${}_2Q$ and ${}_4Q$ mean the same literal anQ and $2 < 4$, so Q is indexed 2 in C_3. Hence $C_3 = C_4 = {}_2Q$. C_4 is called lock resolvent of C_1 and C_2. If the literals of C_2 are indexed differently as $C_2^* = {}_4{\sim}P \vee {}_3Q$, then the literals in C_2^* that is permitted to be resolved upon is ${}_3Q$. However ${}_1P$ and ${}_3Q$ can not be resolved. Therefore, there is no lock resolvent of C_1 and C_2^*. Below, we give explanations on few relevent concepts.

Lock factor: Let C be a clause that has every one of its literals indexed with integers. If two or more literals with the sign of C have a m.g.u. σ, then the clause obtained from $C\sigma$ by deleting any literal that is identical to a literal of lower index is called a lock factor of C.

Example: Let $C = {}_2P(x) \vee {}_8Q(a) \vee {}_{11}P(a) \vee {}_5Q(x)$. Here we know that ${}_2P(x)$ and ${}_{11}P(a)$ has a m.g.u. $\sigma = \{a/x\}$. Thus $C \sigma = {}_2P(a) \vee {}_8Q(a) \vee {}_{11}P(a) \vee {}_5Q(a)$

$= {}_2P(a) \vee {}_5Q(a)$ (by lock resolution)

which is a lock factor of C.

The operation of keeping only the literal with the lowest index and deleting the other identical literals is called merging low for identical literals.

Binary lock resolvent: Let C_1 and C_2 be two clauses with no variable in common and with every literal in them indexed. Let L_1 and L_2 be two literals of 'lowest' index in C_1 and C_2 respectively. If L_1 and $\sim L_2$ have a m.g.u. σ, and if C is the clause obtained from $(C_1\sigma \vee C_2\sigma)$ by removing $L_1\sigma$ and $L_2\sigma$ and by merging low for any identical literals in the remaining clause, then C is called a binary lock resolvent of C_1 and C_2. The literals L_1 and L_2 are called the literals resolved upon.

Lock resolvent: Let C_1 and C_2 be two clauses with every literal in them indexed. A lock resolvent of C_1 and C_2 is one of the following binary lock resolvents:

1) a binary lock resolvent of C_1 and C_2;

2) a binary lock resolvent of C_1 and a lock factor of C_2;

3) a binary lock resolvent of a lock factor of C_1 and C_2;

4) a binary lock resolvent of a lock factor of C_1 and a lock factor of C_2.

Lock deduction: Let S be a set of clauses, where every literal in S is indexed with an integer. A deduction from S is called a lock deduction if and only if every clause in the deduction is either a clause in s S or a lock resolvent.

Lock resolution is effectively efficient. Below is an example using lock resolution. This example when solved by ordinary resolution i.e. by level saturation method 35 resolution steps is to go through.

Example: Let $S = \{C_1, C_2, C_3, C_4\}$ where

$$C_1 = {}_1P \vee {}_2Q$$
$$C_2 = {}_3P \vee {}_4{\sim}Q$$
$$C_3 = {}_6{\sim}P \vee {}_5Q$$
$$C_4 = {}_8{\sim}P \vee {}_7{\sim}Q\ .$$

S is an unsatisfiable set of clauses. Now, from clauses C_{1-4}, there is only one lock resolvent as $C_5 = {}_6{\sim}P$ from C_3 and C_4. From clauses C_{1-5}, there are two lock resolvents as $C_6 = {}_2Q$ from C_1 and C_5 and $C_7 = {}_4{\sim}Q$ from C_2 and C_5. Finally resolving C_6 and C_7 we obtain $C_8 = \square$. This lock deduction is shown in Fig. 3.2.

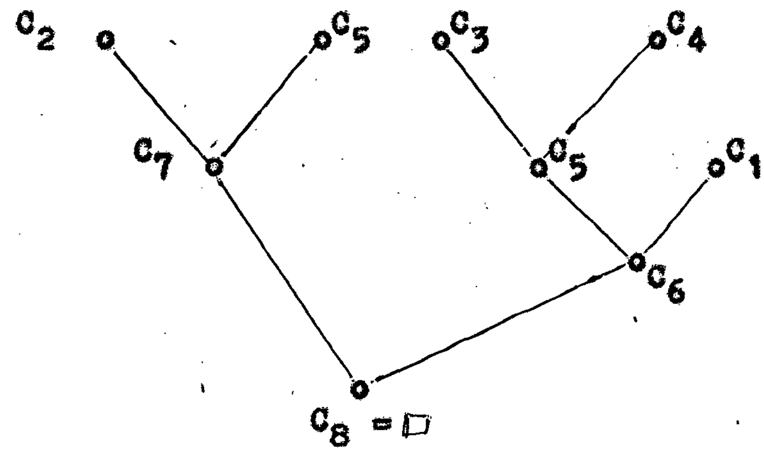


Fig. 3.2.

Lock resolution thus saves much computer time in resolution steps. It is also complete i.e. for any unsatisfiable set S of clauses there exists a lock deduction for the empty clause .

Linear Resolution

Linear resolution starts with a clause resolving against another clause to obtain a resolvent which again resolves with yet another clause and thus applying the chain until we get the empty clause $\square$. It is complete and also can be conveniently applied to the heuristic methods. Thus, given a set S of clauses and a clause C_0 in S, a 'linear deduction' of the clause C_n from S with top clause C_0 is a deduction of the form given in Fig. 3.3 where

1) for $i = 0, 1, \ldots, n-1$, C_{i+1} is a resolvent of C_i and B_i (C_i's are called centre clauses where B_i's are called side clauses); and

2) each B_i is either in S or is a C_j for some j, such that $j < i$

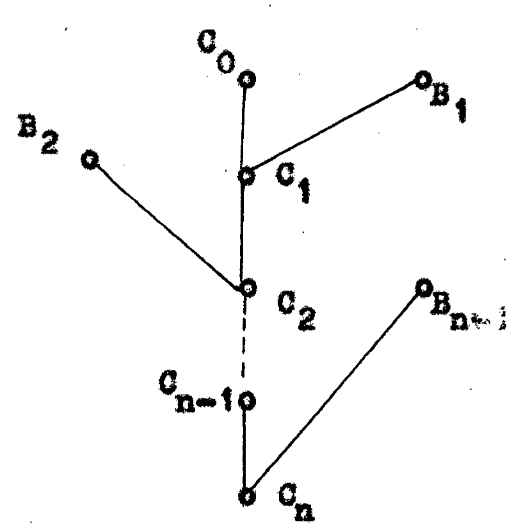


Fig. 3.3

Here C_{i-1} is known as 'near parent' of C_i and C_j, $j < i-1$ is as 'far parent' of C_i. Now we discuss some of the linear resolution procedures.

The unit proof and the input proof [9,6] :

The unit preference strategy or unit resolution essentially orders the clause to be resolved by their length i.e. by the number of literals they contain. A clause consisting of only one literal is termed as unit clause. Contradictions become apparent only when two unit clauses resolve together to produce the empty clause. Therefore, we can think of discovering a contradiction at the end of the process by working first with shortest clauses. This strategy says to fiproduce the shortest resolvent possible first in which at least one of the parent clauses is a unit clause. If no such resolutions are possible, the shortest possible resolvent or factor is produced. Suppose one resolves a unit clause U1 with the first literal in a clause, and then resolves a unit U2 with the descendant of the second literal in the clause. Slagle proved that one will obtain the same resolvent if instead one first resolves U2 with the second literal in the clause and then U1 with the descendant of the first literal in the clause [61]. Hence, in this case, unit resolution restricts resolution to only one of the two possible orders in which the resolvent could be obtained.

In general, if there are q units to be resolved with a clause, unit resolution restricts resolution to one of the $q!$ possible orders in which the resolvent could be obtained. Unit resolution is a special case of semantic resolution.

An input resolution is a resolution in which one of the two parent clauses is an input clause. An input deduction is a deduction in which every resolution is an input resolution. Fig. 3.4 is an input refutation from
$S = \{\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(x,v,w) \vee P(u,z,w),$
$P(g(x,y),x,y),\ P(x,h(x,y),y),\ \sim P(k(x),x,\ k(x))\}.$

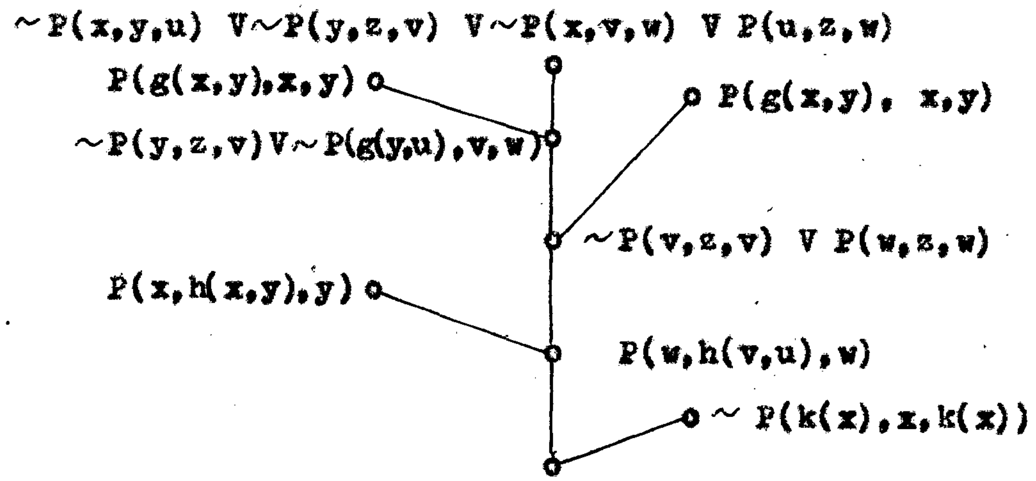


Fig. 3.4

Now we concentrate on the effective application of linear resolution using ordered clauses and its deduction and tree searching.

Ordered linear resolution:

In linear resolution the concept of ordered clauses will not destroy its completeness. We use a concept 'information of resolved literals' in some linear resolution where when a resolvent is obtained, literals resolved upon are deleted. The algorithm that employs both the concepts of ordered clauses and the information of resolved literals is called O L -deduction (ordered linear deduction) [8]. If we take $S = \{P \vee Q, P \vee \sim Q, \sim P \vee Q, \sim P \vee \sim Q\}$ and $C_1 = P \vee Q$, $C_2 = \sim Q \vee R$ there is an ordered resolvent namely $P \vee R$ of C_1 against C_2 with Q and $\sim Q$ being the literals resolved upon. Since Q and $\sim Q$ are complementary, we record only one of them. Suppose we record Q, the last literal of C_1. Then the ordered resolvent can be represented by $P \vee \boxed{Q} \vee R$, where the framed literal is the literal resolved upon. If the framed literal is not followed by any unframed literal, we shall delete this framed literal.

An ordered clause C is a reducible ordered clause if and only if the last literal of C is unifiable with the negation of the framed literal of C.

Whenever a reducible ordered clause is generated we do not have to search the memory for a deduced centre clause to resolve it with. Instead, we may simply delete the last literal from this ordered clause. For example, if $\boxed{P} \vee \boxed{Q} \vee \sim P$ is generated, we simply delete $\sim P$, then $\boxed{Q}$, then $\boxed{P}$ as they are not followed by a non-framed literal then we get $\square$. This kind of operation is called reduction of reducible ordered clause.

The reduced ordered clause of a reducible ordered clause C is the ordered clause obtained from $C\sigma$ by deleting $L\sigma$ and every subsequent framed literal not followed by an unframed literal, where, L is the last literal of C and unifiable with the negation of some framed literal with a m.g.u. σ .

Once the reduction of reducible clauses is incorporated into O L -deduction, we do not have to store intermediate clauses any more. This important aspect of O L -deduction makes it very suitable for computer implementation [8]. The detection of a reducible ordered clause already narrows down the choice of resolutions and reduction mechanism effectively reduces it to only one resolution. O L -deduction is essentially similar to Loveland's model elimination [31,36], which we define in later section. In an ordered clause if the occurrence of an unframed literal is more than one, then we keep it only the leftmost ω one and delete the other identical literals. This process is called <u>left merging</u>. For example, from P V Q V $\boxed{R}$ V S V Q V P we obtain P V Q V $\boxed{R}$ V S. Now we define ordered factor of C as, the factor of C after merging left in the C σ for the m.g.u. of C for which factor we are concerned and by deleting e every framed literal not followed by an unframed literal. Now few relevent concepts are defined below with explanations whereever necessary.

S-resolvent: An S-resolvent or subsumed resolvent C of near parent C_1 and far parent C_2 is a factor of resolvent of C_1 and C_2 such that C subsumes an instance of C_1.

S-linear deduction: An S-linear deduction of a clause C from a set S of a strictly linear deduction of C from the set S_f of all factors of S such that each clause in the deduction not a member of S_f or a factor of the preceeding clause either is a resolvent with far parent from S_f or is an S-resolvent; also no tautologies are permitted.

<u>Ordered binary resolvent</u> C of an ordered clause C_1 against another ordered clause C_2 is obtained from a clause C^* by removing every framed literal not followed by any unframed literal in C^*, where C^* is such that — C_1, C_2 have no variables in common and L_1, L_2 are two unframed literals in C_1 and C_2 respectively. And θ is m.g.u. of L_1 and $\sim L_2$. Then C^* is the ordered clause obtained by concatening the sequence $C_1\theta$ and $C_2\theta$ framing $L_1\theta$, removing $L_2\theta$, and merging left for any identical unframed literals in the remaining sequence.

O L -deduction: Let S be a set of ordered clauses with C_0 as the top clause, then an O L -deduction of C_n from S is a deduction of the form shown in Fig. 3.5 which satisfies

1) for $i = 0,1,2, \ldots, n-1$, C_{i+1} is an ordered resolvent of C_i, a centre ordered clause against B_i, a side

ordered clause.

2) each B_i is either in S or C_j, $j < i$, or an instance of C_j, $j < i$ if and only if C_i is a reducible ordered clause. In this case C_{i+1} is the reduced ordered clause of C_i.

3) no tautology is in the deduction.

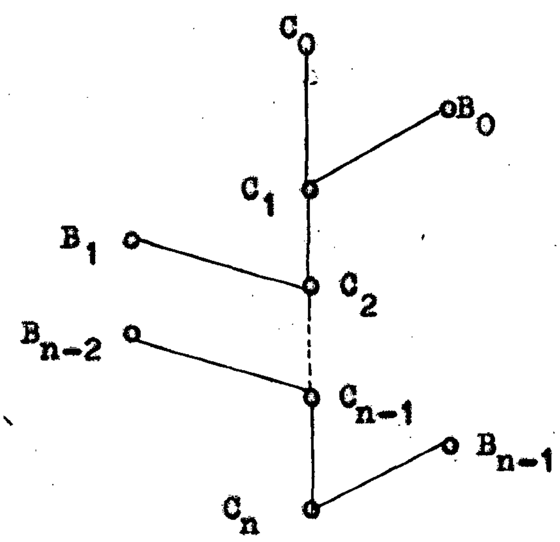


Fig. 3.5

Linear Deduction and Tree Searching:

Linear deduction and tree searching techniques developed by Luckham [37] and Slagle [62] are applied in O L -deduction. Now the algorithm of the tree searching method — 'Breadth - First Method' - is defined. Suppose S is a set of ordered clauses to be proved unsatisfiable.

C_0 is an ordered clause in S to be the top clause. Then the Breadth First Method algorithm follows as :

Step 1. CLIST = (C_0)

Step 2. If CLIST = $\square$, Stop, otherwise, continue.

Step 3. Let C be the first ordered clause in CLIST. Select C from CLIST.

Step 4. Find all the ordered clauses in S that can be side clauses of C. If no such clause exist goto step 2; otherwise, resolve C with all these side clauses. Let $R_1, \ldots, R_m$ denote these ordered resolvents. Let R_i^* be the reduced ordered clause of R_i if R_i is reducible. If R_i is not reducible, let $R_i^* = R_i$.

Step 5. If some R_q^* is $\square$, $1 \leqslant q \leqslant m$, stop, otherwise, continue.

Step 6. Put $R_1^*, \ldots, R_m^*$ (in an arbitrary order) at the end of CLIST and goto step 2.

Below is an example using this method in O L - reduction

Example: Let $S = \{ P \vee Q, \sim P \vee Q, P \vee \sim Q, \sim P \vee \sim Q \}$ and $C_0 = P \vee Q$. Then the O L -refutation of S is given in Fig. 3.6.

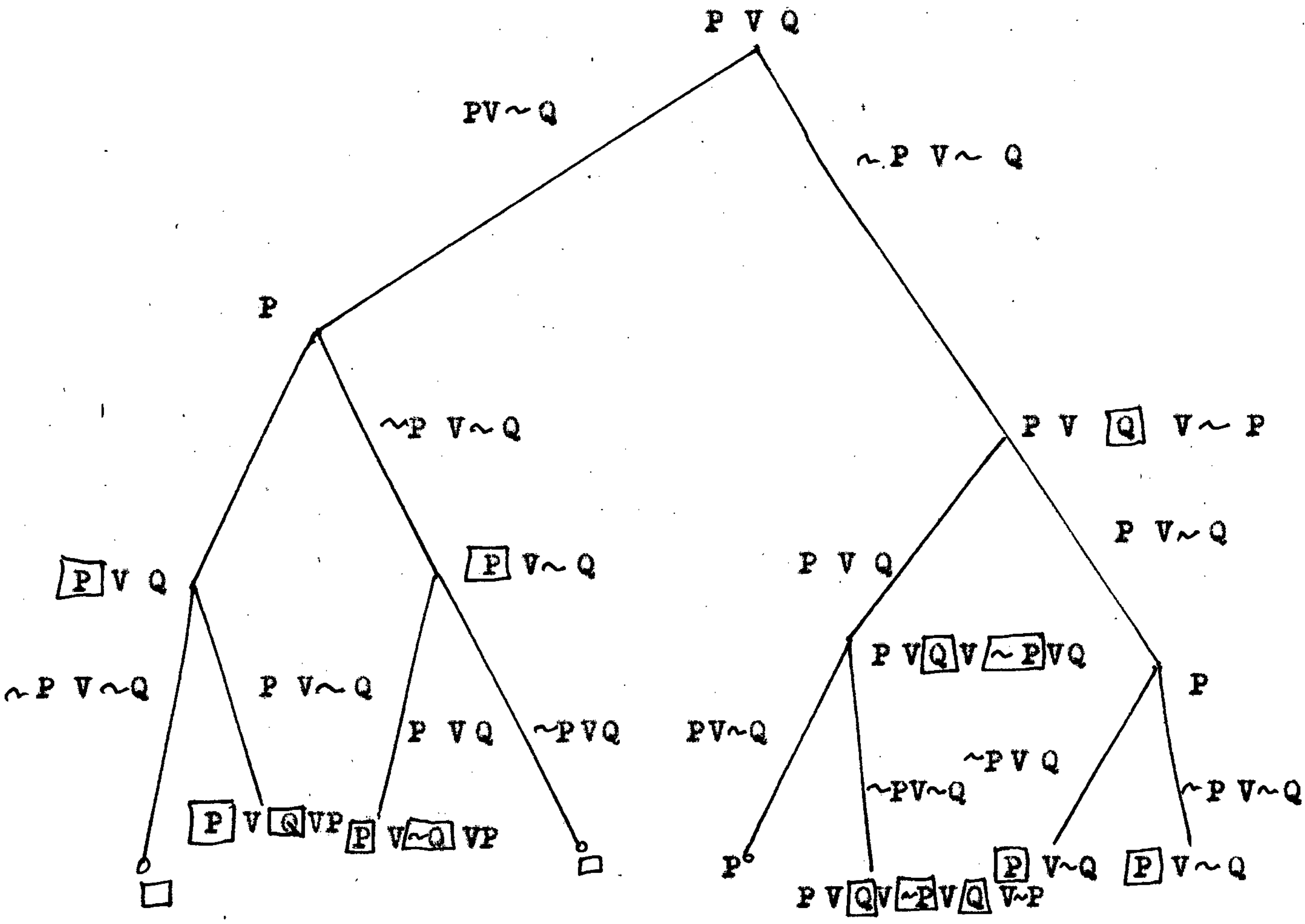


Fig. 3.6a

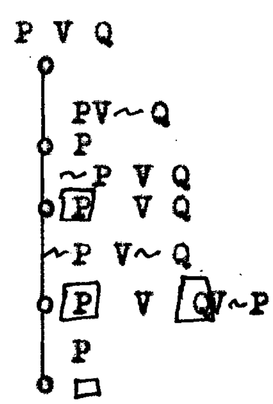


Fig. 3.6b

The breadth-first strategy is described for the reasons that (1) simple to understand, (2) is basic strategy used in Lee's consequence finding program and (3) it was used for theoretical purposes, for example in Robinson's proof of completeness of the resolution principle for proof finding it is used [53].

The Depth First Method:

It is another way to search a tree, unlike the breadth-first method it expands nodes from left to right and not from top to bottom. The depth first method is almost the same as the breadth first method except that in step 5 where in this case we put $R_1^*, \ldots, R_m^*$ at the beginning of the CLIST, instead of putting at the end of CLIST.

In an O L -deduction with top ordered clause C_0, the depth of C_0 is 0. If the depth of a certain ordered clause C is k and R is an ordered resolvent of C and some side clause, then the depth of R is (k+1). Applying the depth-first method to the same example given in breadth-first method we generate the tree of Fig. 3.7 which is smaller than the one in Fig. 3.6a.

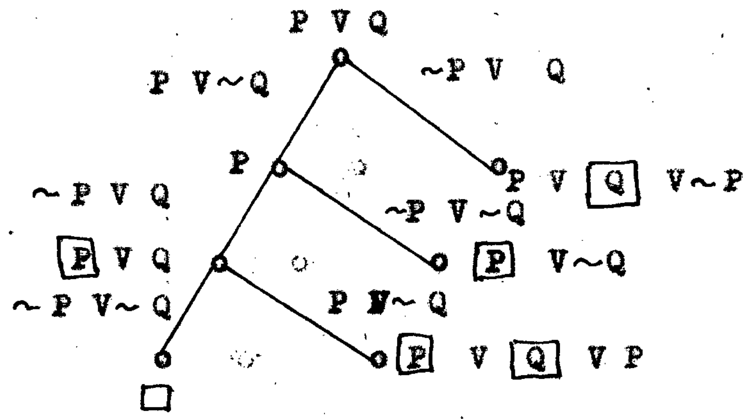


Fig. 3.7

Deletion Strategy:

Now we discuss the heuristic tree searching method of deletion strategy. In order to check the unlimited application of resolution causing irrelevent and redundant clauses to be generated the level-saturation method is used. In any unsatisfiable set of clauses S, for level saturation we generate the sequence S^0, S^1, S^2, ..., where $S^0 = S$,

$$S^n = \text{resolvents of } C_1 \text{ and } C_2 \;\Big|\; C_1 \in \bigcup_{i=0}^{n-1} S^i \text{ and } C_2 \in S^{n-1}, \quad n = 1, 2, 3, \ldots .$$

To program this method on computer, one has to list clauses of $\bigcup_{i=0}^{n-1} S^i$ in order, then to compute resolvents by comparing every clause $C_1 \in \bigcup_{i=0}^{n-1} S^i$ with a clause $C_2 \in S^{n-1}$ that is listed after C_1. When a resolvent is computed is appended to the end of the list so far generated.

A clause C is said to subsume a clause D if and only if there is a substitution σ such that $C\sigma \rightarrow D$. D is called subsumed clause.

For example in $C = P(x)$ and $D = P(a) \vee Q(a)$ if $\sigma = \{a/x\}$, $C\sigma = P(a)$. Since $C\sigma \subseteq D$, C subsumes D.

Here D is identical to C, or D is an instance of C. Then D is subsumed by C. The deletion strategy is the deletion of any tautology and any subsumed clause whenever possible. It is generalization of David and Putnam's tautology rule [12]. The completeness of it depends on how tautologies and subsumed clauses are deleted [25]. It is complete only when it follows the rule to delete the newly generated clause if it is neither a tautology nor subsumed by any clause in the list as derived in the first part of this strategy.

Subsumption Algorithm: Let C and D be two clauses and $\theta = \{a_1/x_1, \ldots, a_n/x_n\}$, where x_i's are variables occurring in D, and a_i's are new distinct constants not occurring in C or D. Suppose $D = L_1 \vee L_2 \vee \ldots \vee L_m$. Then the algorithm is

Step 1. Assume $W = \sim D\theta = \{\sim L_1\theta, \ldots, \sim L_m\theta\}$

Step 2. Set $k = 0$ and $U^0 = \{C\}$

Step 3. If U^k contains $\Box$, terminate; C subsumes D. Otherwise, let $U^{k+1} = \{$resolvents of C_1 and C_2 such that $C_1 \in U^k$ and $C_2 \in W\}$.

Step 4. If U^{k+1} is empty terminate, C does not subsume D. Otherwise, set $k = k+1$ and do step 3.

Here each clause in U^{k+1} is smaller by one literal than the clause in U^k. Therefore, the sequence U^0, U^1, U^2, ..., must eventually contains a set that contains $\Box$.

Model Elimination

The model elimination was introduced by Loveland and much improvement of it was given by him [15,31,32,36] to avoid the difficulties of testing for S-resolvents in some of the linear resolution methods. The model elimination procedure is not formally expressible in the resolution formate, but is a variant of resolution. A primary difference is that the basic entities are not ordered clauses but are literal lists containing two types of literals.

We recall the definition of interpretation by specifying the truth values of the atoms in the Herbrand base. We call such a specification a model. Thus if the Herbrand base consist of the atoms $\{P(a,b), P(a,a), P(b,b), P(b,a), Q(a), Q(b)\}$, then one possible model would be $\{\sim P(a,b), P(a,a), P(b,b), \sim P(b,a), \sim Q(a), Q(b)\}$. In general, a model (possibly infinite) is a set of literals constructed from the Herbrand base in such a way that each atom in the Herbrand base is appeared either in the negated or non-negated form but not in both in the model. A given model M may not satisfy a clause C. For example, $\{\sim P(a,b), P(a,a), P(b,b), \sim P(b,a), \sim Q(a), Q(b)\}$ does not satisfy $\sim Q(x) \vee P(y,x)$, since the substitution $\{(b/x), (a/y)\}$ creates a ground instance having the value F. This model does satisfy the clause $\{\sim Q(x) \vee P(y,y)\}$, since no substitution creates a ground instance having the value F.

A model can often be more compactly specified by listing a set of literals all of whose ground instances over the Herbrand universe have the value T. Thus take the set of clauses

$$S = \begin{cases} \sim Q(x) \vee P(a,f(x)), \\ Q(x) \vee P(f(x),a), \\ \sim P(x,f(x)) \vee R(y), \\ \sim R(a) \end{cases}$$

which can be specified by the set $\{Q(x), \sim P(x,f(x)), P(f(x),a), R(x)\}$. In terms of atoms in the Herbrand base this model is the infinite set

$$M = \begin{cases} R(a),\ R(f(a)),\ R(f(f(a))),\ \ldots, \\ Q(a),\ Q(f(a)),\ Q(f(f(a))),\ \ldots, \\ \sim P(a,f(a)),\ \sim P(a,f(f(a))),\ \ldots, \\ P(f(a),a),\ P(f(f(a)),a),\ \ldots . \end{cases}$$

Here we notice that the first and the last clauses are not satisfied by this model, whereas the second and third are satisfied.

The model strategy is based on the principle that -- for an unsatisfiable set S of clauses and any model M over the Herbrand base of S, a refutation graph for S exists having the property that each node in the graph is either a clause in S or has as one of its immediate ancestors a clause that is not satisfied by M.

The criterion that must be satisfied by a pair of aclauses (A,B) in order that they be resolved relative to the model strategy is that at least one of (A,B) must not be satisfied by the model.

In Fig. 3.8, a refutation graph for the set of unsatisfiable clauses S = {Q(x) V P(a), ~Q(x) V P(x), ~Q(x) V ~P(x), Q(x) V ~P(x)} is given. Each resolution in the graph satisfies the model criterion with {Q(x), P(x)} used to define a model. Those clauses not satisfied by the model are enclosed in boxes. The extent to which the model strategy reduces the number of needed resolutions depends on M, any model M not satisfying any of the clauses in S is a bad choice.

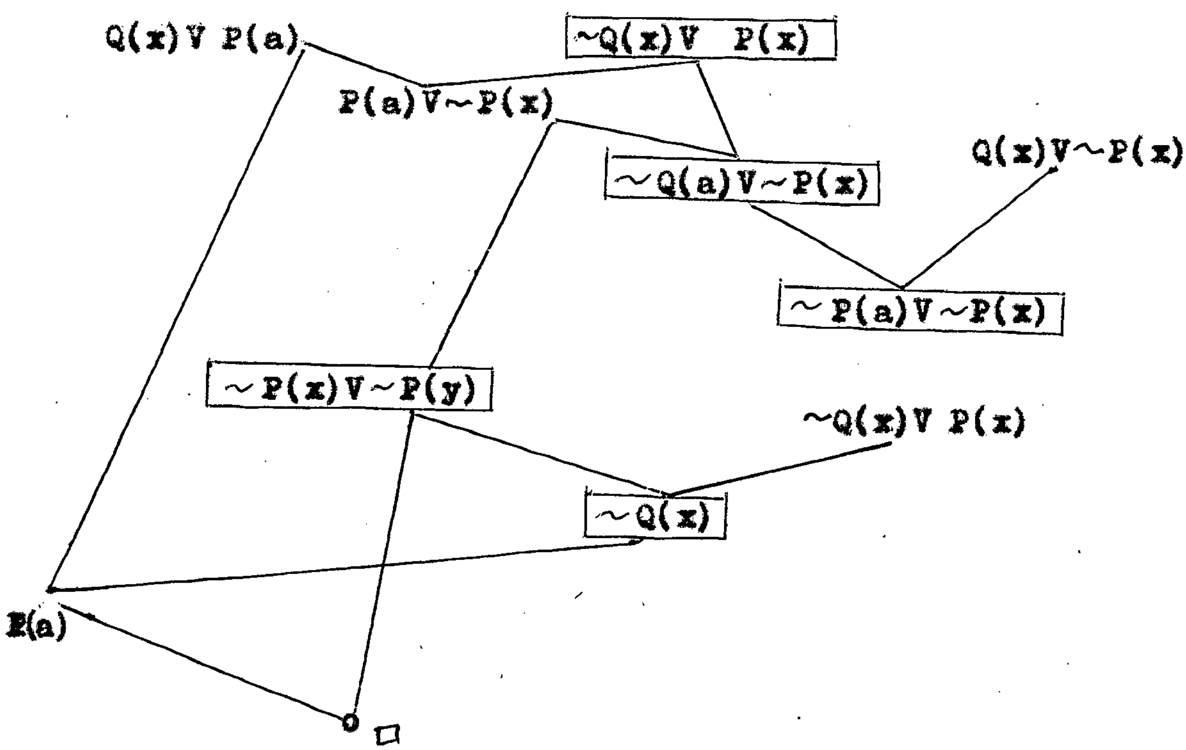


Fig.3.8

CHAPTER - 4

APPLICATIONS OF RESOLUTION PRINCIPLE

There are many application areas of resolution principle. It is applied in areas like question-answering deductive system, program analysis, state-space problems, concept formation, etc., And in what is common to all of these is the formation of these problems into the framework of theorem proving problems and applying resolution principle to establish assertions. The discussions on these are following now.

In previous chapters it has discussed how resolution principle is used in proving theorems either a proof finding or a consequence finding. A proof finding program attempts to find a proof for a certain theorem where certain premises or axioms are given. And in a consequence finding program given axioms then tries to deduce consequences from the axioms and to select 'interesting' consequences. Both types of programs use resolution principle. The resolution principle does provide a machine oriented algorithm. It is a straight forward procedure which combines several well known syllogisms described in the earlier parts of chapter 2 in the area covering the mathematical logic. The resolution procedure is mechanisable in the sense that when once a problem is formulated in terms of predicate calculus, it is possible to develop an appropriate programme for it.

Towards Deductive Question Answering:

To use computer for getting answers of questions from given facts of premises is best to be on a time-sharing system, with interaction between the computer and the human user. For applying theorem proving to question answering it is obvious that given a question to answer, a question answering system should not conduct an exhaustive search among all the facts in order to answer the question. Also in applying theorem proving to question answering, we often need to translate English sentences into formulas of the first-order logic which is by no means trivial. This area is also related to the general representation problem.

Green and Raphel [19] made a break through in this field by pointing out that mechanical theorem proving technique of resolution principle can be applied to design question-answering and problem solving systems. Their concept is that the set of facts necessary for question-answering or problem solving can be viewed as axioms of a theorem, and the question or the problem can be viewed as the conclusion of the theorem.

Towards Program Analysis:

The resolution principle is also used in program analysis. Here the aim of using it is to understand the I/O relationship of the program. Floyd and Manna described about the I/O relation of programs in 1960s [16,38,39]. Considering the logical formulas describing the execution of a program as axioms the logical consequences can be done. Chang and Lee in [7] also worked in this area. In particular, there is one clause called the 'halting' clause that will be mechanically deduced using the resolution principle if and only if the program terminates. Using this concept one checks whether a program has the I/O relation.

Towards State-Space Problems:

A state space problem denoted by (S,F,G) consists of a description of a set S of possible starting states, a set F of operators that convert one state into another, and a set G of goal states i.e. the output set after applying the elements of F on the elements of S. Here how resolution-based theorem provers can be used as a GPS for state-space problems is briefly discussed.

The terminology for state space problems includes (1) 'states' which is represented by S, S', S'', etc. (2) 'objects' denoted by O, O', O'', etc., (3) 'relations' denoted by fluent symbols (underlined) and are defined between objects and properties of states and actions; and (4) 'operators'. Here the first-order predicate calculus is applied in state space problems. The following is an example of it using resolution principle in solving a state space problem in first-order predicate calculus.

Example: 'A monkey is in a room where a bunch of bananas is hanging from the ceiling, too high to reach. In the corner of the room is a box, which is not under the bananas. How can the monkey get the bananas ?'

The solution to the monkey's problem is to move the box under the bananas and climb onto the box, from which the banana will be reached. The objects used in the state space description of this problem are <u>monkey</u>, <u>box</u>, bananas, place 1, place 2, place 3. The operators used are goto, move, climb, and reach for, each of which will be a situational-fluent function. The relations used in the description are under, on, at, and has-bananas, each of which will be a situational fluent predicate. The adjoining table-Table 4-1 gives the first-order predicate formulation that correspond to a description of this state space, using this objects, operators and relations. The monkey's problems is represented by the formula

$$(\forall s)\ (\text{has-bananas}\ (s)) \qquad \ldots\ (4.1)^*$$

which is to be proved using the formulas in table 4-1 by resolution principle. Fig. 4.1 shows the refutation tree of it after adding the negation of the solution given in the expression (4.1)* giving a deduction of and hence showing the unsatisfiability of the clauses in Table 4-1, with negation of (4.1)*. In Table 4-1, we take 'mon' for monkey and 'ban' for bananas. Table is in next pages.

Table 4-1A. The Monkey-Banana Problem Aximatized

A1. $\forall p \forall p' \forall s$(at(box,p,s)$\rightarrow$ at(box,p,goto(p',s)))

A2. $\forall p \forall p' \forall s$(at(ban,p,s)$\rightarrow$at(ban,p,goto(p',s)))

A3. $\forall p \forall s$(at(mon,p,goto(p,s)))

A4. $\forall p \forall p' \forall s$((at(box,p,s)$\wedge$at(mon,p,s))$\rightarrow$(at(box,p',move (mon,box,p,p',s))))

A5. $\forall p \forall p' \forall P'' \forall s$(at(ban,p,s)$\rightarrow$ at(ban,p,move(mon,box,p',P",s)))

A6. $\forall p \forall p' \forall s$(at(mon,,p,s)$\rightarrow$at(mon,p',move(mon,box,p,p',s)))

A7. $\forall s$(under(box,ban,s)$\rightarrow$ under(box,ban,climb(mon,box,s)))

A8. $\forall p \forall s$(at(mon,p,s)$\wedge$at(box,p,s)$\rightarrow$on(mon,box,climb(mon, box,s)))

A9. $\forall s$(under(box,ban,s)$\wedge$on(mon,box,s))$\rightarrow$has-bananas(reach for, mon,ban,s)))

A10. $\forall s$(at(box,p3,s)$\wedge$at(ban,p3,s))$\rightarrow$under(box,ban,s))

A11. at(box,p2,s_0)$\wedge$at(ban,p3,s_0)

Table 4-1B: Clause form of Table 4-1A

A1. $\sim$at(box,p,s)Vat(box,p,goto(p',s))

A2. $\sim$at(ban,q,s')V at(ban,q,goto(q',s'))

A3. at(mon,r,goto(r,r'))

A4. $\sim$at(box,u,v)V$\sim$ at(mon,u,v)V at(box,u',move(mon,box,u, u',v))

A5. $\sim$at(ban,t,t")V at(ban,t,move(mon,box,t',t"',t"))

A6. $\sim$at(mon,v',v"')V at(mon,v",move(mon,box,v',v",v"'))

A7. $\sim$under(box,ban,w)V under(box,ban,climb(mon,box,w))

A8. $\sim$at(mon,w',w")V $\sim$at(box,w',w")V on(mon,box,climb(mon,
box,w"))

A9. $\sim$under(box,ban,x)V $\sim$on(mon,box,x)V has-bananas(reach
for(mon,ban,x))

A10. $\sim$at(box,p3,y)V $\sim$at(ban,p3,y)V under(box,ban,y)

A11. at(box,p2,0)

A12. at(ban,p23,0)

A13. Negated conjencture (NC): $\sim$has-bananas(z).

Table 4.1C: Consequences of Fig.4.1

C1. at(box,p2,goto(p',s_0))

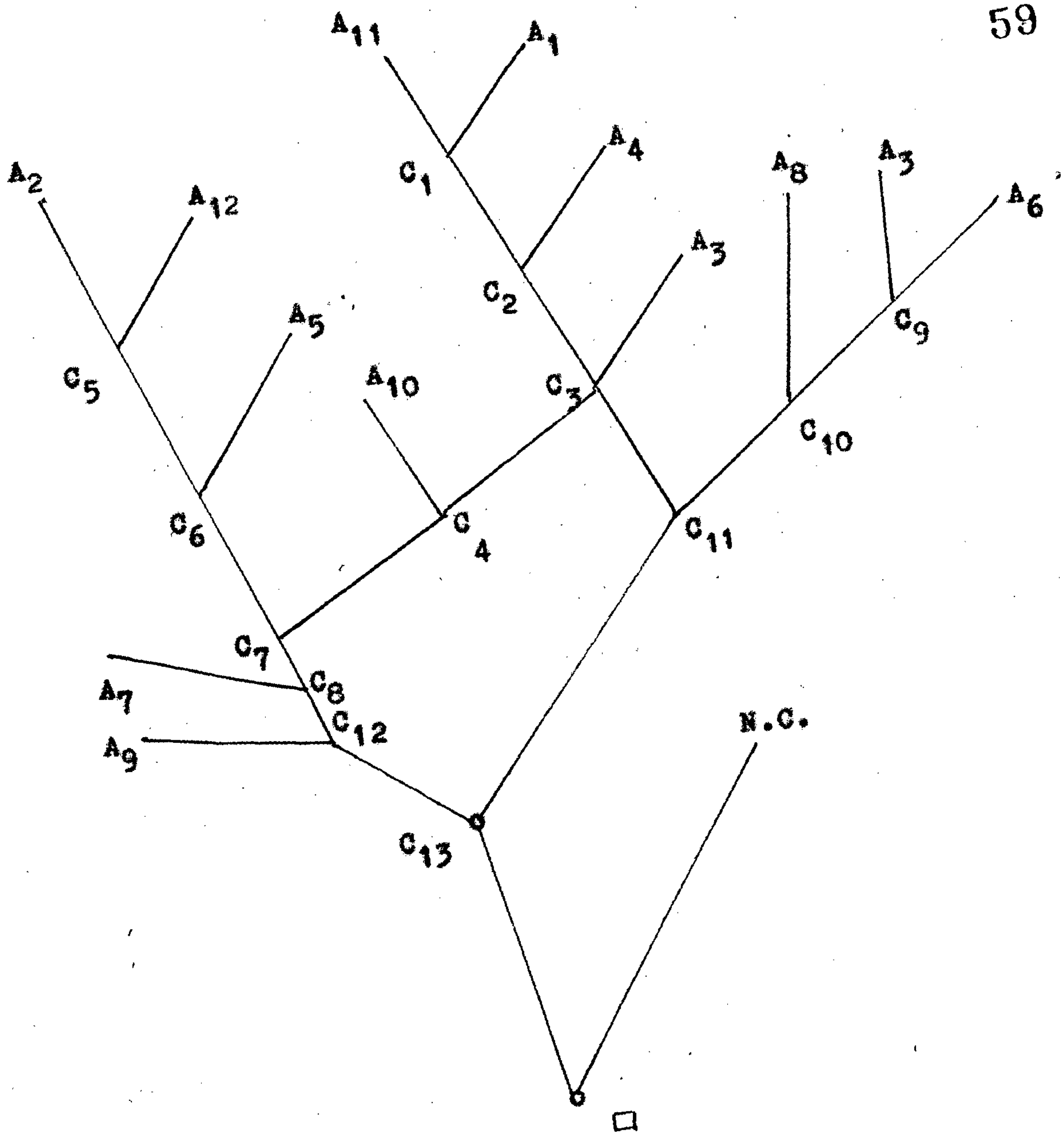
C2. $\sim$at(mon,p2,goto(p',s_0))V at(box,u',move(mon,box,p2,
u',goto(p',s_0)))

C3. at(box,u',move(mon,box,p2,u',goto(p2,s_0)))

C4. $\sim$at(ban,p3,move(mon,box,p2,p3,goto(p2,s_0)))V under
(box,ban,move(mon,box,p2,p3,goto(p2,s_0)))

C5. at(ban, p3,goto(q',s_0))

C6. at(ban,p3,move(mon,box,t',t"',goto(q',s_0)))



A11
A1
A4
A8
A3
A6
A2
A12
C1
A3
C2
A5
C9
C5
A10
C3
C10
C6
C4
C11
C7
A7
C8
C12
A9
N.C.
C13

Fig. 4.1

C7. under(box,ban,move(mon,box,p2,p3,goto(p2,s_0)))

C8. under(box,ban,climb(mon,box,move(mon,box,p2,p3,goto
(p2,s_0))))

C9. at(mon,v",move(mon,box,r,v",goto(r,r')))

C10. at(box,v",move(mon,box,r,v",goto(r,r')))V on(mon,box,
climb(mon,box,move(mon,box,r,r",goto(r,r')))).

C11. on(mon,box,climb(mon,box,move(mon,box,p2,u',goto
(p2,s_0))))

C12. on(mon,box,climb(mon,box,move(mon,box,p2,p3,goto
(p2,s_0))))V has-bananas(reachfor(mon,ban,climb(man,
box,move(mon,box,p2,p3,goto (p2,s_0))))

C13. has-bananas(reachfor(non,ban,climb(mon,box,move(mon,
box,p2,p3goto(p2,s_0)))))

Even though the application area of resolution principle is quite wide, like areas on first theorems on group theory, geometry theorem proving etc., we are briefing it with the explanation of one more application area only — that is on concept formation.

2.7.3 Towards Concept Formation:

The concept formation is a system that develops concepts and infers on pictorial data 58 . The general concept about the recognition is a visual scene. Bruner argues this point effectively on the basis of experimental evidence[5]. Looking into this account Sadananda and Mahabala developed the idea of conceptference in [58] using resolution principle.

Conceptference works in two phases: (1) initialization of concepts necessary to describe the scene, and (2) to operate on these concepts in the domain of a first order theory.

Concept Formation: In this phase the necessary concepts or functions to describe the scene is formalized. They used LISP compiler to define these.

Operation Phase: Here the entire scene is expressed in terms of the well formed formulas of the first order predicate calculus. The names of the concepts formed in first phase are predicate letters constituting the w.f.fs generated in this phase.

The system of conceptference generates binary relations between every pair of neighbouring pictures expressed in the form of w.f.f.s. Two pictures are taken to be in a neighbourhood if the distance between them is less than a pre-assigned positive number d. For calculation of this distance Euclidian distances between the labeled co-ordinates of the first picture to that of the second picture are considered and the last is chosen.

'Adhoc' interpretation for the predicate letters representing binary relations such as LEFT, INSIDE, ABOVE are assigned and no attempt is made to obtain general interpretations.

The inference mechanism of conceptference utilizes the resolution principle after converting the set of w.f.f.s, including the negation of the assertion expressed as a w.f.f., into the clause form. The deductive strategy takes into account the nature of the assertion to be established.

The clause representing the negation of the assertion is rearranged such that an order of priority of importance of the predicate letter appears, and this is followed by the priority of the arguments. However, if no importance of priority is found, an alphabetical order is chosen. The clauses unifiable with higher order literals of the clause representing the negation of the assertion are chosen, and all such resolvents are obtained in one stage. In the case of clauses involving variables and ground instances the latter are assigned priority in resolution over the former. The clauses not unifiable at any stage are discarded, saving space and effort. The next level is reached for all corresponding resolvents, and the process is carried out unitl an empty clause □ is reached or certain prespecified steps are carried out.

If a question on the existence of an object responds after generating the description of the scene in terms of w.f.f.s and after attempting to resolve to a clause with no literals, the system would terminate with a YES or NO or DO NOT KNOW statement if □ is not generated within a reasonable number of steps. Besides attempting an existence problems, conceptference can handle an identification problem. The Fig.4.2 describes briefly how identification could be performed using relationship of different pictures.

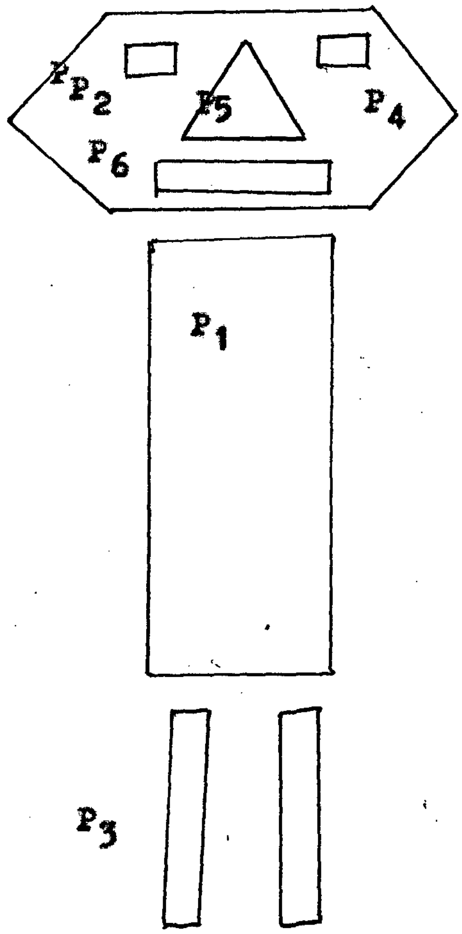


Fig. 4.2

The descriptiv scene which is generated in terms of a set of w.f.f.s defined on the domain of pictures of triangles, squares, rectangles and polygons to identify the picture. Some of the w.f.f.s are RECTANGLE(P_1), SQUARE(P_2), RECTANGLE(P_3), HEXAGON(P_4), TRIANGLE(P_5), RECTANGLE(P_6).

CHAPTER - 5

CHOOSING AN ECONOMIC RESOLUTION STRATEGY

Here examples in automatic theorem proving are dealt in order to choose an economic resolution strategy for different types of problems. The contex of economy here is the usual economy. The number of steps in an example, the time consumption of the problem by computer, the easeness in handling the problem, etc. all are factors of the economy here.

Example 5.1

To show that the alternate interior angles formed by a diagonal of a trapezoid are equal.

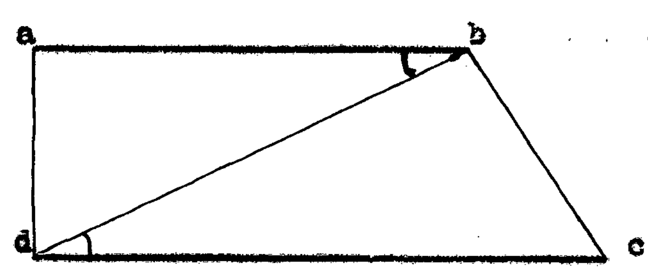


Fig. 5.1

The axiomatization of this problem is as follows:

$T(x,y,u,v)$ means that $x\ y\ u\ v$ is a trapezoid with upper-left vertex x, upper-right vertex y, lower-right vertex u, lower-left vertex v;

$P(x,y,u,v)$ means that the line segment $x\ y$ is parallel to the line segment $u\ v$;

$E(x,y,z,u,v,w)$ means that the angle $x\ y\ z$ is equal to the angle $u\ v\ w$.

Then we have the following axioms:

A1: $(\forall x)(\forall Y)(\forall u)(\forall v)\ \big(T(x,y,i,v) \rightarrow P(x,y,u,v)\big)$
defining the trapezoid.

A2: $(\forall x)(\forall y)(\forall u)(\forall v) \rightarrow P(x,y,u,v) \rightarrow E(x,y,v,u,v,y)$
states that the alternate interior angles of parallel lines are equal.

A3: $T(a,b,c,d)$ from Fig. 5.1.

From these axioms we conclude that $E(a,b,d,c,d,b)$ is true, that is,

$A1 \wedge A2 \wedge A3 \rightarrow E(a,b,d,c,d,b)$ is a valid formula. Since we have to prove it by resolution principle we negate the conclusion for getting an unsatisfiable set. Thus the unsatisfiable set of clauses of this problem is

$S = C_1, C_2, C_3, C_4$ where

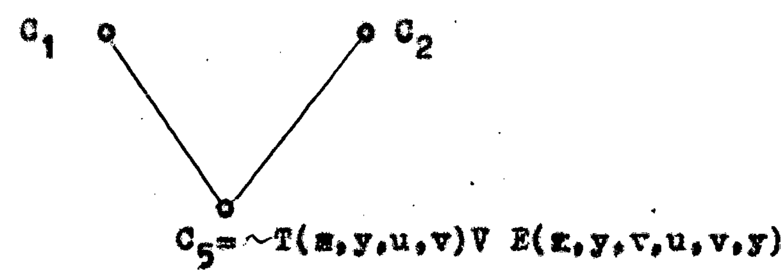
$C_1 = \sim T(x,y,u,v) \vee P(x,y,u,v)$

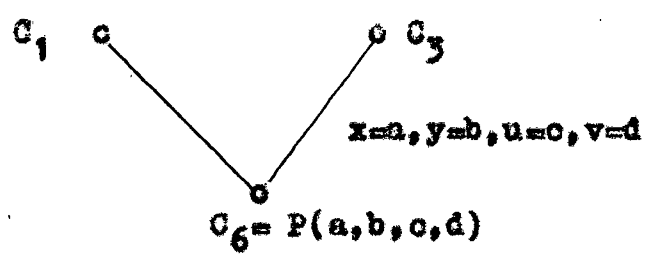
$C_2 = \sim P(x,y,y,v) \vee E(x,y,v,u,v,y)$

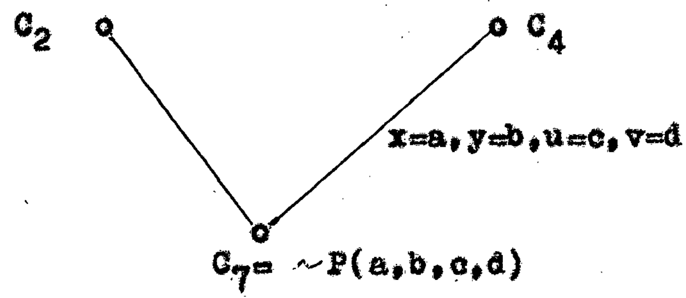
$C_3 = T(a,b,c,d)$

$C_4 = \sim E(a,b,d,c,d,b)$

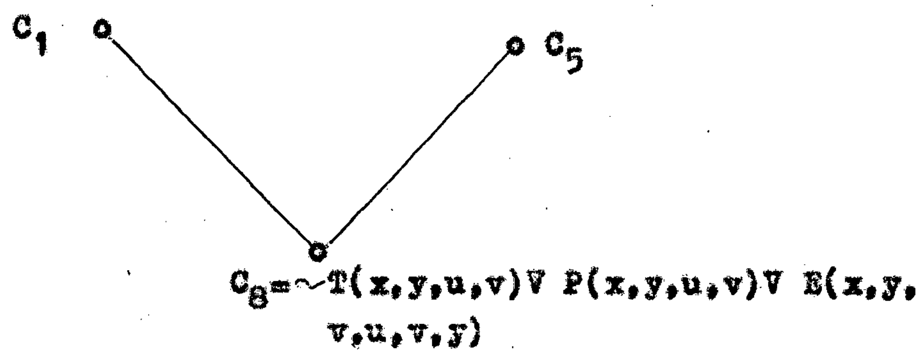
To begin, resolution process is now called out between these clauses and resolvents of them in terms of their levels which we call level saturation method. $S = \{C_1, C_2, C_3, C_4\}$ will be taken as S^0 meaning at zero level. Now the resolutions in the first level are as below:

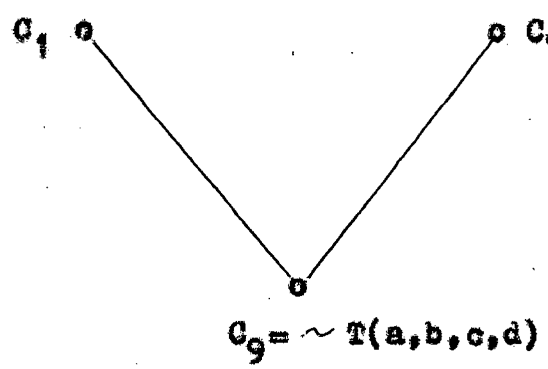


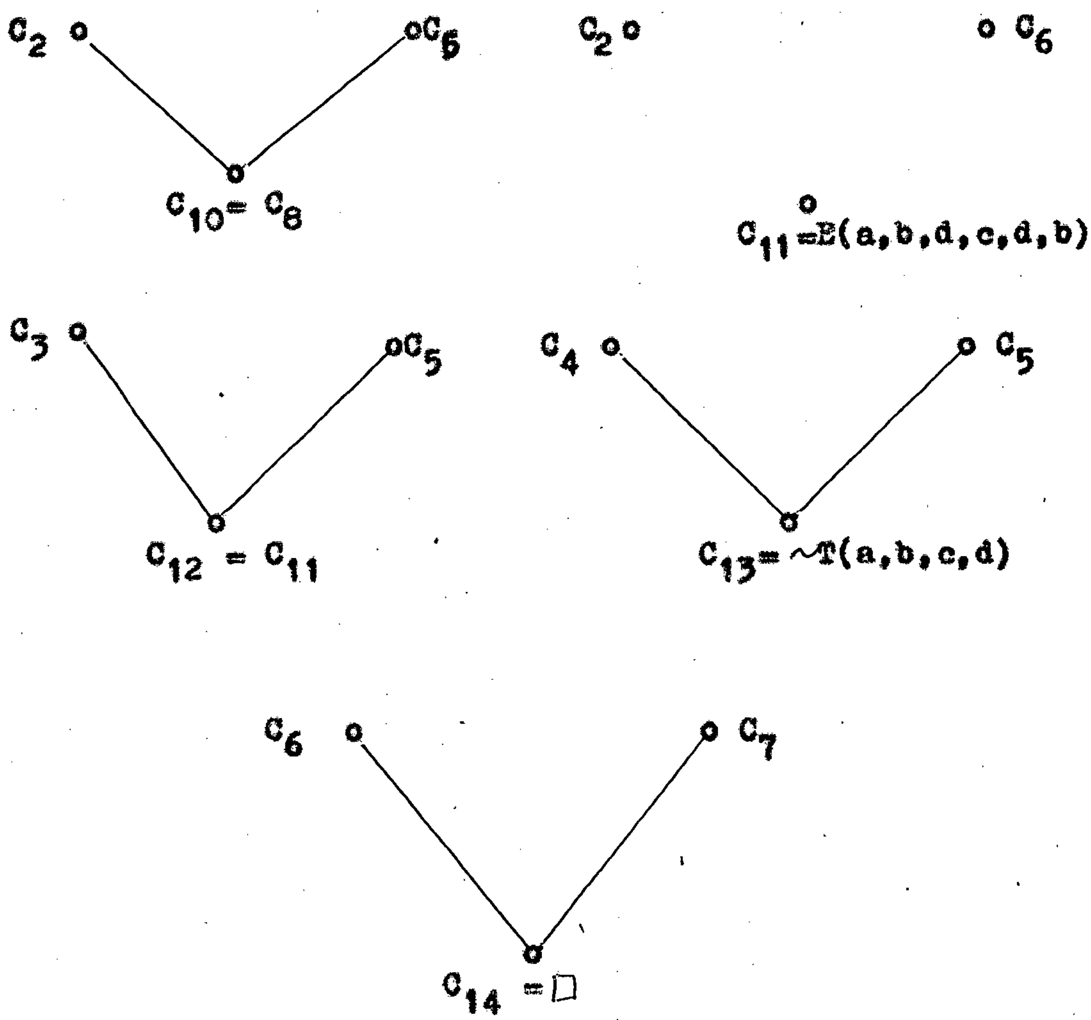




The set of clauses $S^1 = \{C_5, C_6, C_7\}$ is in the first level S^1. So we proceed for the second level.







$S^2 = \{C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13}, C_{14}\}$ is the set of clauses in the second level. Here we have derived the empty clause in C_{14}. Hence we have solved our problem by inconsistency method of resolution principle with 14-4 =10 resolution steps.

Now the some problem is solved by linear resolution. The solution is given in Fig. 5.2.

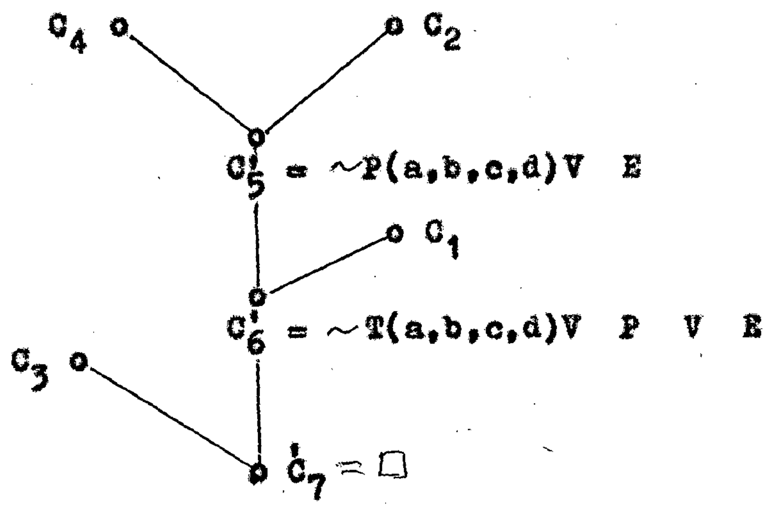
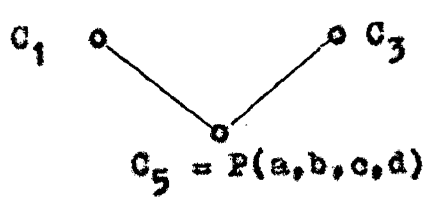


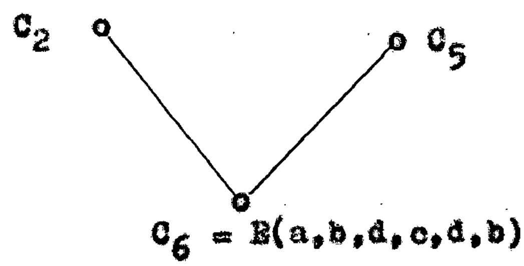
Fig. 5.2

Here in linear method it takes only 3 steps of resolution. Hence by linear resolution it saves 10-3=7 units of computer time of resolution steps.

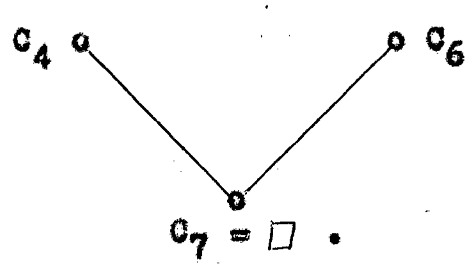
Again, we solve the problem by semantic resolution. The interpretation is taken as $I = \{\sim T, \sim P, \sim E\}$. Hence $S_1 = \{C_1, C_2, C_4,\}$, $S_2 = \{C_3\}$ are the non-empty sets of S partitioned by I, and ordering is taken as $T > P > E$. Thus the resolution steps due to P I -resolution strategy are given as follows:



Now C_5 enters S_2. Therefore $S_1' = \{C_1, C_2, C_4\}$ and $S_2' = \{C_3, C_5\}$. Now C_1 does not resolve with C_5 as $T > P$, instead C_2 resolve with C_5.



Therefore $S_1 = S_1' = S_2'' = \{C_1, C_2, C_4\}$, and $S_2'' = \{C_3, C_5, C_6\}$
Now C_2 does not resolve with C_6 as $P > E$ instead, C_4 resolves with C_6.



Since we get the empty clause here the process halts.
Here the number of resolution step is 3.

In semantic resolution we need to order the predicates or literals of the sets of clauses which are derived from the problem and also we need to give an interpretation for the partition of the set S whereas in linear resolution we need to check whether there is unit clause in S or in resolvents of S which we prefer to resolve first with some other clause possibly of multiliteral. This is the case of unit resolution in linear resolution. And also, in linear resolution if we use the concept of order of the literals, we have to destroy the concept of completeness. Now let us see, how the same problem is solved by lock resolution.

In the case of lock resolution we index the literals in all the clauses differently and those literals whose indices are lower in the same clause is allowed to be resolved upon.

$$C_1 = {}_1{\sim}T(x,y,u,v) \vee {}_2P(x,y,u,v)$$
$$C_2 = {}_3{\sim}P(x,y,u,v) \vee {}_4E(x,y,v,u,v,y)$$
$$C_3 = {}_5T(a,b,c,d)$$
$$C_4 = {}_6{\sim}E(a,b,d,c,d,b)$$

Here we can not resolve ${}_2P$ of C_1 with any other clause in S for, the index of ${}_2P$ is not lowest in C_1. And we can resolve ${}_1{\sim}T$ of C_1 with any other clause, in this case with C_3. Likewise we can not resolve upon ${}_4E$ with ${}_6{\sim}E$. The refutation graph of the problem solved by lock resolution is thus giving below:

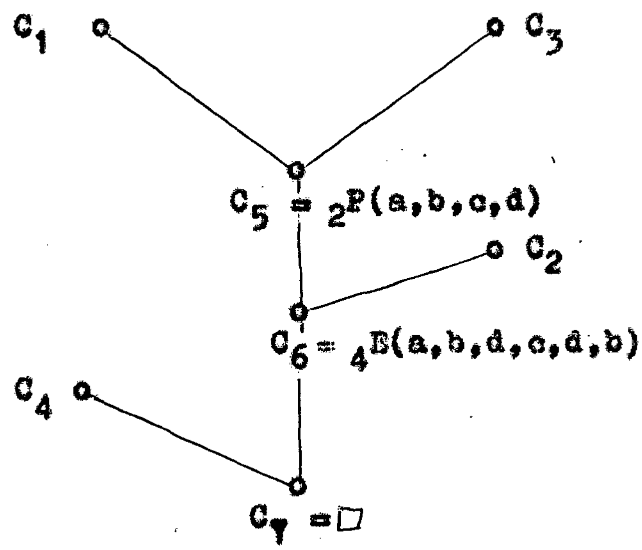


Fig. 5.3

Here also the number of resolution steps is only 3. Hence if we assume that the fime taken by computer in resolving two clause for getting a third clause is as unit time and neglecting the scanning time by computer of clauses then in the refined resolutions in the above problem time saved by each refined strategy is 7 units of time.

Example 5.24.

To prove by resolution principle that (1) finger is a part of man, with the conditions (2) finger is a part of hand, (3) hand is a part of arm, (4) arm is a part of man and (5) x is a part of v and v is a part of y implies that x is a part of y, Here the axiomatization is done by $P(s,t)$ meaning that s is a part of t. Hence the axioms are

1) part $(x,v) \wedge$ part $(v,y) \longrightarrow$ part (x,y)
2) part (finger, hand)
3) part (hand, arm)
4) part (arm, man)
5) part (finger, man) (conclusion)

For using automatic theorem proving we reduce (1) - (4) in the following clauses and negate (5) for the inconsistent proof.

C_1: $\sim$ part$(x,v) \vee \sim$ part$(v,y) \vee$ part(x,y)
C_2: part(finger,hand)
C_3: part(hand,arm)
C_4: part(arm,man)
C_5: $\sim$ part(finger, man).

$S = \{C_1, C_2, C_3, C_4, C_5\}$ is an unsatisfiable set of clauses. Now we solve the problem by using the level saturation method of resolution principle. Here we use the rotation $r(C_i, C_j)$: $C_k(\varphi)$: L meaning that C_i resolves with C_j and gets the resolvent C_k = L by using the substitution φ.

$r(C_2, C_1)$: C_6(x=finger, v=hand): ~ part(hand, y) V part(finger, y).

$r(C_3, C_1)$: C_7(x=hand, v=arm): ~ part(arm, y) V part(hand, y).

$r(C_4, C_1)$: C_8(x=arm, v=man): ~ part(man, y) V part(arm, y).

$r(C_5, C_1)$: C_9(x=finger, y=man): ~ part(finger, v) V ~ part(v, man).

$r(C_2, C_1)$: C_{10}(v=finger, y=hand): ~ part(x, finger) V part(x, hand).

$r(C_3, C_1)$: C_{11}(v=hand, y=arm): ~ part(x, hand) V part(x, arm).

$r(C_4, C_1)$: C_{12}(v=arm, y=man): ~ part(x, arm) V part(x, man).

$S = S^0$ is the level zero clause set and $S^1 = C_{6-12}$ is the first level clause set. Now we find S^2.

$r(C_1, C_6)$: C_{13}(x=hand): ~ part(hand, v) V ~ part(v, y) V part(finger, y).

$r(C_1, C_6)$: C_{14}(x=finger, v=y): ~ part(y, y) V part(finger, y)

$r(C_1, C_6)$: C_{15}(v=finger): ~ part(x, finger) V part(x, y) V ~ part(hand, y).

$r(C_3, C_6)$: C_{16}(y=arm): part(finger, arm).

$r(C_5, C_6)$: C_{17}(y=man): ~ part(hand, man).

$r(C_1, C_7)$: C_{18}(x=arm): $\sim$ part(arm, v) V $\sim$ part(x, y) V
part(hand, y).

$r(C_1, C_7)$: C_{19}(v=hand): $\sim$ part(x, hand) V $\sim$ part(arm, y) V
part(x, y).

$r(C_1, C_7)$: C_{20}(x=arm, v=hand): $\sim$ part(arm, hand).

$r(C_4, C_7)$: C_{21}(y=man): part(hand, man).

$r(C_6, C_7)$: C_{22}: part(finger, y) V $\sim$ part(arm, y).

$r(C_1, C_8)$: C_{23}(x=man): $\sim$ part(man, v) V $\sim$ part(v, y) V
part(arm, y).

$r(C_1, C_8)$: C_{24}(v=arm): $\sim$ part(x, arm) V $\sim$ part(man, y) V
part(x, y).

$r(C_1, C_8)$: C_{25}(x=man, v=arm): $\sim$ part(man, arm).

$r(C_1, C_8)$: C_{26}(x=arm, v=y): $\sim$ part(y, y) V part(arm, y) V
$\sim$ part(man, y).

$r(C_7, C_8)$: C_{27}: $\sim$ part(man, y) V part(hand, y).

$r(C_1, C_9)$: C_{28}(x=finger, v=y): $\sim$ part(finger, y) V
$\sim$ part(y, y) V part(y, man).

$r(C_1, C_9)$: C_{29}(v=x, y=man): $\sim$ part(x, x) V $\sim$ part(x, man) V
$\sim$ part(finger, x).

$r(C_2, C_9)$: $C_{30}(v{=}hand)$: $\sim part(hand, man)$.

$r(C_4, C_9)$: $C_{31}(v{=}arm)$: $\sim part(finger, arm)$.

$r(C_6, C_9)$: $C_{32}(y{=}v)$: $\sim part(hand, v) \vee \sim part(v, man)$

$r(C_6, C_9)$: $C_{33}(v{=}finger, y{=}man)$: $\sim part(hand, man) \vee \sim part(finger, finger)$.

$r(C_7, C_9)$: $C_{34}(v{=}hand, y{=}man)$: $\sim part(arm, man) \vee \sim part(finger, hand)$.

$r(C_8, C_9)$: $C_{35}(v{=}arm, y{=}man)$: $\sim part(man, man) \vee \sim part(finger, arm)$.

$r(C_1, C_{10})$: $C_{36}(y{=}finger)$: $\sim part(x, v) \vee \sim part(v, finger) \vee part(x, hand)$.

$r(C_1, C_{10})$: $C_{37}(v{=}hand)$: $\sim part(hand, y) \vee part(x, y) \vee \sim part(x, finger)$.

$r(C_1, C_{10})$: $C_{38}(x{=}v, y{=}hand)$: $\sim part(v, v) \vee part(v, y) \vee \sim part(v, finger)$.

$r(C_6, C_{10})$: $C_{39}(x{=}hand, y{=}hand)$: $part(finger, hand) \vee \sim part(hand, finger)$.

$r(C_6, C_{10})$: $C_{40}(x{=}finger, y{=}finger)$: $\sim part(hand, finger) \vee part(finger, hand)$.

$r(C_7, C_{10})$: $(C_{41}(x{=}arm, y{=}hand)$: $part(hand, hand) \vee \sim part(arm, finger)$.

$r(C_7, C_{10})$: C_{42}(x=hand, y=finger): $\sim$ part(arm, finger) $\vee$ part(hand, hand).

$r(C_8, C_{10})$: C_{43}(x=man, y=hand): part(arm, hand) $\vee$ $\sim$ part(man, finger).

$r(C_8, C_{10})$: C_{44}(x=arm, y=finger): $\sim$ part(man, finger) $\vee$ part(arm,hand).

$r(C_9, C_{10})$: C_{45}(x=finger, v=hand): $\sim$ part(hand, man) $\vee$ $\sim$ part(finger, finger).

$r(C_1, C_{11})$: C_{46}(y=hand): $\sim$ part(x, v) $\vee$ $\sim$ part(v, hand) $\vee$ part(x, arm).

$r(C_1, C_{11})$: C_{47}(v=arm): $\sim$ part(arm, y) $\vee$ part(x, arm) $\vee$ $\sim$ part(x, hand).

$r(C_1, C_{11})$: C_{48}(x=v, y=arm): $\sim$ part(v, v) $\vee$ part(v, arm) $\vee$ $\sim$ part(v, hand).

$r(C_2, C_{11})$: C_{49}(x=finger): part(finger, arm).

$r(C_6, C_{11})$: C_{50}(x=finger, y=hand): $\sim$ part(hand, hand) $\vee$ part(finger, arm).

$r(C_6, C_{11})$: C_{51}(x=hand, y=arm): part(finger, arm) $\vee$ $\sim$ part(hand, hand).

$r(C_7, C_{11})$: C_{52}(x=y=hand): $\sim$ part(arm, hand) $\vee$ part(hand,arm).

$r(C_7, C_{11})$: $C_{53}(x=arm=y)$: part(hand, arm) V $\sim$ part(arm, hand).

$r(C_8, C_{11})$: $C_{54}(x=arm, y=hand)$: $\sim$ part(man, hand) V part(arm, arm).

$r(C_8, C_{11})$: $C_{55}(x=man, y=arm)$: part(arm, man) V $\sim$part(man, hand).

$r(C_9, C_{11})$: $C_{56}(x=finger, v=arm)$: $\sim$ part(arm, man) V $\sim$ part(finger, hand).

$r(C_{10}, C_{11})$: C_{57}: $\sim$ part(x, finger) V part(x, arm).

$r(C_1, C_{12})$: $C_{58}(y=arm)$: $\sim$ part(x, v) V $\sim$ part(v, arm) V part(x, man).

$r(C_1, C_{12})$: $C_{59}(v=man)$: $\sim$ part(man, y) V part(x, y) V $\sim$ part(x, arm).

$r(C_1, C_{12})$: $C_{60}(x=v, y=man)$: $\sim$ part(v, v) V part(v, y) V $\sim$ part(v, arm).

$r(C_3, C_{12})$: $C_{61}(x=hand)$: part(hand, man).

$r(C_5, C_{12})$: $C_{62}(x=finger)$: $\sim$ part(finger, arm).

$r(C_6, C_{12})$: $C_{63}(x=finger, y=arm)$: $\sim$ part(hand, arm) V part(finger, man).

$r(C_6, C_{12})$: C_{64}(x=hand, y=man): part(finger, man)V
~part(hand, hand).

$r(C_7, C_{12})$: C_{65}(x=hand, y=arm): ~ part(arm, y)V
part(hand, man).

$r(C_7, C_{12})$: C_{66}(x=arm, y=man): part(hand, man)V
~part(arm, arm).

$r(C_8, C_{12})$: C_{67}(x=y=arm): ~ part(man, arm)V part(arm,man).

$r(C_8, C_{12})$: C_{68}(x=y=man): part(arm, man)V~part(man, arm).

$r(C_9, C_{12})$: C_{69}(x=finger, v=man): ~ part(man, man)V
~part(finger, finger).

$r(C_{11}, C_{12})$: C_{70}: ~ part(x, hand)V part(x, man).

Here C_{14-70} are in S^2 level. Now in the process of getting S^3, C_{13} can resolve for 21 different resolvents and C_{14} can resolve for 17 different resolvents. But still we would not be able to get the empty clause $\square$. But uptil that time we already have to cover 70+21+17=108 resolution steps. The empty clause will be getting only when the turn for C_{21} resolving with C_{17} in S^3 level.comes. This way the number of resolution steps before getting the empty clause must be around 300. Thus even if the level saturation method is complete it consumes too much computer time in solving problems of such type. Now let us see how this problem is solved by different refined strategies in much shorter computer time.

By linear resolution with unit preference strategy:

$$S = \begin{cases} C_1 : \sim \text{part}(x, v) \vee \sim \text{part}(v, y) \vee \text{part}(x, y) \\ C_2 : \quad \text{part}(\text{finger}, \text{hand}) \\ C_3 : \quad \text{part}(\text{hand}, \text{arm}) \\ C_4 : \quad \text{part}(\text{arm}, \text{man}) \\ C_5 : \sim \text{part}(\text{finger}, \text{man}) \end{cases}$$

$r(C_2, C_1)$: C_6(x=finger, v=hand): $\sim$ part(hand, y) $\vee$ part(finger, y).

$r(C_2, C_1)$: C_7(v=finger, y=hand): $\sim$ part(x, finger) $\vee$ part(x, hand).

$r(C_3, C_1)$: C_8(v=arm, x=hand): $\sim$ part(arm, y) $\vee$ part(hand, y).

$r(C_6, C_3)$: C_9(y=arm): part(finger, arm)

$r(C_8, C_4)$: $C_{10}(y=man)$: part(hand, man).

$r(C_6, C_5)$: $C_{11}(y=man)$: $\sim$ part(hand, man).

$r(C_{11}, C_{10})$: C_{12}: $\square$.

The refutation tree of this solution is giving in Fig. 5.4 where we combine two linear trees.

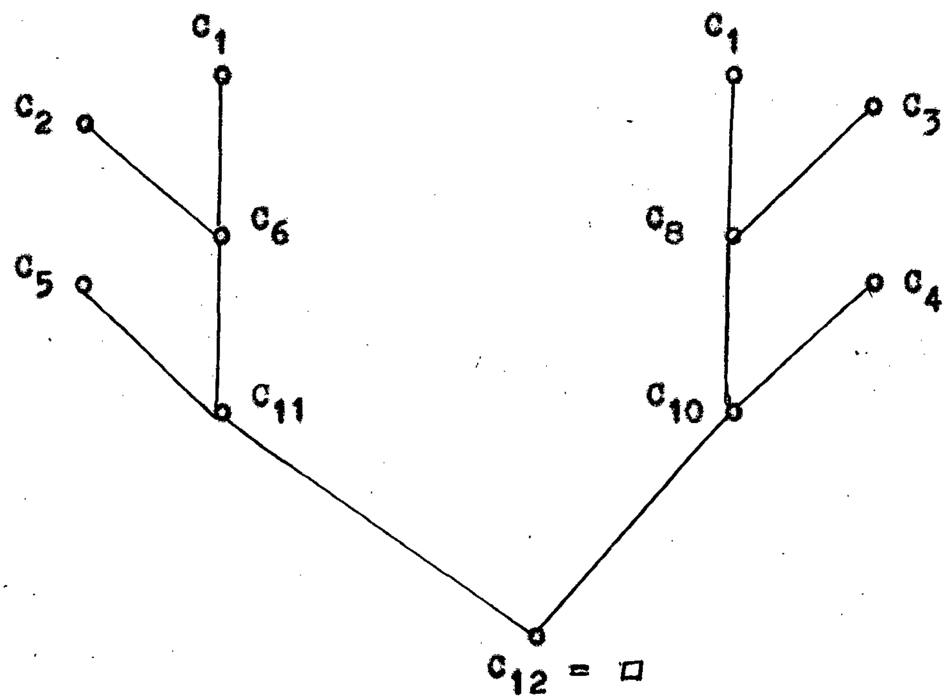


Fig. 5.4

Hence in this case we could see the amount of computer units of time saved by using this refined strategy.

Again we solve this problem by set-of-support strategy (semantic resolution):

Here C_{1-5} are as above.

$r(C_1, C_5)$: $C_6(x{=}finger, y{=}man)$: $\sim part(finger, v) \vee$
$\sim part(v, man)$.

$r(C_2, C_6)$: $C_7(v{=}hand)$: $\sim part(hand, man)$.

$r(C_4, C_6)$: $C_8(v{=}arm)$: $\sim part(finger, arm)$.

$r(C_1, C_7)$: $C_9(x{=}hand, y{=}man)$: $\sim part(hand, v) \vee$
$\sim part(v, man)$.

$r(C_3, C_9)$: $C_{10}(v{=}arm)$: $\sim part(arm, man)$.

$r(C_4, C_{10})$: C_{11} : $\square$.

Here the set of support clause is C_5 only. In this case we save one more computer unit time than the one solved by unit resolution.

By lock resolution: We make the order of the literals in S by prefix in it as below:

C_1 : ${}_1\sim part(x, v) \vee {}_2\sim part(v, y) \vee {}_3 part(x, y)$.

C_2 : ${}_4 part(finger, hand)$.

C_3 : ${}_5$part(hand, arm).

C_4 : ${}_6$part(arm, man).

C_5 : ${}_7\sim$part(finger, man).

$r(C_2, C_1)$: C_6(x=finger, v=hand): ${}_8\sim$part(hand, y) V ${}_9$part(finger, y).

$r(C_3, C_1)$: C_7(x=hand, v=arm): ${}_{10}\sim$part(arm, y) V ${}_{11}$part (hand, y).

$r(C_4, C_1)$: C_8(x=arm, v=man): ${}_{12}\sim$part(man, y) V ${}_{13}$part (arm, y).

$r(C_6, C_3)$: C_9(y=arm): ${}_{14}$part(finger, arm).

$r(C_7, C_4)$: C_{10}(y=man): ${}_{15}$part(hand, man).

$r(C_9, C_1)$: C_{11}(x=finger, v=arm): ${}_{16}\sim$part(arm, y) V ${}_{17}$part(finger, y).

$r(C_{10}, C_1)$: C_{12}(x=hand, v=man): ${}_{18}\sim$part(man, y) V ${}_{19}$part(hand, y).

$r(C_{11}, C_4)$: C_{13}(y=man): ${}_{20}$part(finger, man)

$r(C_{13}, C_1)$: C_{14}(x=finger, v=man): ${}_{21}\sim$part(man, y) V ${}_{22}$part(finger, y).

$r(C_{13}, C_5)$: C_{15} : $\square$.

Here also it takes much shorter computer time than the one in level saturation method but longer than by other resolution strategies given above for the example.

From these examples we conclude that if the given problem could get the first order predicate formulae i.e., the clauses with more unit literals we prefer to use linear resolution and semantic resolution strategies to the solution by lock resolution. And in case, if the formula reduced problem contains more multi-literal clauses, we prefer lock resolution to the rest of the resolution strategies. Again, since the semantic resolution (as in the case of O L -resolution) is not complete we may get problems where it can not be applied even if it contains unit literals. An example of it was given in chapter 3. In such cases the priority is higher in linear strategy and lock resolution strategy.

CHAPTER - 6

INEFFICIENCIES AND LIMITATIONS OF RESOLUTION PRINCIPLE

In previous chapters many resolution strategies have been mentioned, each of which has its own merits and demerits. For a specific theorem or problem some strategy may work well while others may perform poorly, as is explained in the last chapter by giving examples.

The application of resolution theorem provers in real world problem starts only in 1969 due to Hewitt by developing the programming language PLANNER, which permits the statement and execution of plans in a theorem proving format [22, 23]. The extent to which resolution theorem provers can be used for solving real world problem depends on several factors, including how well predicate calculus can be used to describe real world situations and actions, and how efficiently theorem provers can be used to find solutions to problems that are given predicate calculus formulizations. Any mathematical theory can be expressed as a system of predicate calculus formulas. Thus predicate calculus offers a metaphysically adequate mathematical frame work for the description of the real world, if any such frame work can be constructed at all. Thus in resolution methods, the question is about the <u>epistomological adequacy</u> i.e., how it can represent every day aspects of the real world, and about its <u>heuristic adequacy</u> i.e., how it can be used to express information that is helpful in solving problems. The answer to the first adequacy is satisfactory even if many-valued logics are more desirable than predicate calculus [18, 42]. However, any embodiment in a predicate calculus machine would require a set of axioms to define the functions and predicates that were associated with each of the aspects like ambiguities, inaccuracies, probabilities, multiple interpretations, etc., which are really in a real world environment. Thus even if no completely satisfactory many-valued logic has yet been developed we can conclude that the resolution theorem proving is imperfect or inefficient in the real-world problems. Hewitt's PLANNER is a powerful language for solving a theorem proving problem [22]. Hewitt's work in [22,23] is concerned with the heuristic adequacy of predicate calculus. He showed that it is possible, not only to use predicate calculus formulas as statements of facts, but also to use them as recommendations for how to proceed in solving problems.

Mc Carthy and Hayes in [42] discuss the inefficiency of theorom provers in "frame problem" of problems, The frame problem arises from the fact that, in a state-space problem, an application of an operator to a state will usually affect some relations between objects in the state and not affect others. In predicate calculus formalization for such a problem, there must generally be axioms for each operator to express both the relations that exists and are not changed by the application of that operator. For example in the Monkey-Banana problem of chapter 4 we had to state and use the fact that the application of the operator <u>climb</u> would not affect the position of the <u>box</u>. Various techniques for overcoming the frame problem have been investigated by Hewitt, Fikes and Nilsson in [22,23,14]. In [14], Fikes et.al. presents a GPS-like program that controls the application of a theorem proving program to various sets S_i of clauses, each set S_i representing a given state in a state space. Each operator has associated with it a collection of 'delete' and 'add' instructions that identify the relations changed by the application of that operator. The program performs a heuristic search in state space untill it finds a sequence of operators that will produce a set S_g of clauses containing the desired goal relations. It performs tasks in real-world environment.

Another limitation of resolution procedure is that all resolution based theorem provers are designed to be general and complete programs for proving and disproving theorems within mathematical theories, the primary ascent in their development has been a concentration on their completeness and soundness, i.e. on proving their applicability to any mathematical system and increasing their efficiency as much as possible without relinquishing that applicability. But in the system using PLANNER it provides frame work in which it is possible to write very sophisticated programs for special purpose types of theorem proving. There are many types of information processing and problem solving that involve logical deduction, or theorem proving, without requiring full completeness or generality.

In previous chapters we have been discussing about the proof finding and consequence finding résolution based theorem provers, but we have not crossed of any idea about decision procedure for first-order-logic. There is no guarantee that a proof procedure will converge to a proof in a finite number of steps when attempting to prove a non-theorem. As a practical matter, however, this lack of decision procedure does not limit the applicability of logic as much as it may at first appear. Because of the time and space constraints on practical computation, the heuristic power of a proof procedure i.e., its ability to prove useful theorems efficiently is more important than its theoretical limitations. A decision procedure that requires enormous amounts of time or intermediate storage is indistinguishable, in practice, from a proof procedure that never terminates from some w.f.f.s. In fact, for reasons that include both theoretical and practical limitations, no theorem prover can be really complete. Eventhough a theorem may be logically implied by a set of axioms, we can not gurantee that the theorem pröver will eventually develop a proof for it, because of —

1) the limitations of space and time which aff-

ects the computational ability of any machine and,

2) the undecidability of the predicate calculus.

Again the resolution procedure for automatic theorem proving fails in solving problems in mathematics which are hard, hard is in the sense that difficult or fail to express in predicate form like problems on infinite set, such as problems on set theory, group theory, ring theory, fields, etc. To avoid this somewhat, W.W. Bledsoe in 1977 in [3] used a method termed as 'complete set of reductions'. His idea was based on the work [29] of Lankford. He cited an example for it as is giving below.

$$(\forall A)(\forall B)(\text{Subsets}(A \cap B) = \text{Subsets}(A) \cap \text{Subsets}(B))$$

where A, B are sets, possibly infinite. This problem was proved by his method of non-resolution theorem prover [29]. Not only Bledsoe's work, there are many-refined works on mathematical theories on sets where only resolution principle can not work effectively. Following example due to chang and Lee [8] is an example of such problem.

Example 6.1.

Let (G, O)be a group in which $x \mathrm{O} x = e$, the identity element of G for any x in G; then G is a commutative group.

Let A_1, A_2, A_3, A_4 are the axioms of G to be a group and B the conclusion of this problem. Then

A_1 : $x, y \in G \Rightarrow x \mathrm{O} y \in G$

A_2 : $x, y, z \in G \Rightarrow x \mathrm{O}(y \mathrm{O} z) = (x \mathrm{O} y) \mathrm{O} z$

A_3 : $x \mathrm{O} e = e \mathrm{O} x = x \quad \forall x \in G$

A_4 : $\forall x \in G$ there exist x^{-1} such that $x \mathrm{O} x^{-1} = x^{-1} \mathrm{O} x = e$.

B : $(x \mathrm{O} x = e \; \forall x \in G) \Rightarrow u \mathrm{O} v = v \mathrm{O} u \; \forall u, v \in G$.

Here they use the convention $= (x, y)$, $\sim = (x, y)$ and $\mathrm{O}(x, y)$ for $x = y$, $x \neq y$ and $(x \mathrm{O} y)$ respectively. Then the axioms are reduced respectively to

A_1' : $(\forall x)(\forall y)(\exists z)\ (=(\mathrm{O}(x, y), z))$

A_2' : $(\forall x)(\forall y)(\forall z)\ (\mathrm{O}(x, \mathrm{O}(y, z)) = \mathrm{O}(\mathrm{O}(x, y), z))$

A_3' : $(\forall x)((=(\mathrm{O}(x, e), x)) \wedge (=(\mathrm{O}(e, x), x)))$

A_4' : $(\forall x)((=(\mathrm{O}(x, i(x)), e)) \wedge (=(\mathrm{O}(i(x), x), e)))$

B' : $(\forall x)(=(\mathrm{O}(x, x), e)) \rightarrow ((\forall u)(\forall v)(=(\mathrm{O}(u, v),$
$\mathrm{O}(v, u))))$.

Now negating B' and transforming $A'_1 \wedge A'_2 \wedge A'_3 \wedge A'_4 \wedge \sim B'$ into the prenex normal form the set S consisting of the following clauses is obtained, where f, a, b, are skolem functions.

1) $x \, 0 \, y = f(x)$

2) $x \, 0 (y \, 0 \, z) = (x \, 0 \, y) \, 0 \, z$

3) $x \, 0 \, e = x$

4) $e \, 0 \, x = x$

5) $x \, 0 \, i(x)^{-1} = e$

6) $i(x)^{-1} \, 0 \, x = e$

7) $x \, 0 \, x = e$

8) $a \, 0 \, b \neq b \, 0 \, a.$

This set S of the clausess can not be proved unsatisfiable by using only resolution principle even if we know that the statement of the problem is true mathematically. It is because of the reason that we have not defined the properties of equality in S. Hence for proving it, Chang and Lee define 10 more axioms for defining the equality as defined by reflexibility, symmetry, transitivity and substitutivity of equality. After that using the overall 18 axioms and applying resolution principle they prove the statement of the problem. This problem is too clumsy in resolution

Principle for it generates an unmanageable number of resolvents therein. Avoiding this difficulty they (Chang and Lee) proved the problem statement solution using the concept of 'Equality' [8].

These points given in this Chapter about the limitations and inefficiencies of resolution strategies have lead serious research effort to go into the fundamentals of theorem proving. Active efforts by Bledsoe, Nelvins, Chang and Lee [3,6,7,9] on to this area try to remedy the weakness of theorem proving. The exact mathematical relation connecting the number of parent clauses and the number of resolution steps for different resolution strategies in solving a problem will be a very interesting one. However, this relation has been developed here emperically with the help of few examples. For certain class of problems one can overcome the limitations imposed by resolution process by adopting specific non-resolution strategies [3,8]. It also will be interesting and important to identify these classes of problems. Inspite of these, there are aspects in resolution theorem proving. It is also possible to think of applications in the domain of real-world, such as in the well known 'analogy problems', 'picture identification problems' and in data base areas.

REFERENCES

1. Anderson, R.(1971): 'Completeness of the Locking Resolution for Paramodulation', Dept. of Computer Science, University of Houston, Houston, Texas.

2. Anderson, R. and Bledsoe, W.W.(1970): 'A Linear format for Resolution with Merging and a New Technique for Establishing Completeness; JACM, Vol.17, No.3, pp. 525-534.

3. Bledsoe, W.W.(1977): 'Non-Resolution Theorem Proving; JAI, Vol.9, pp. 1-35.

4. Boyer, R.S.(1971): 'Locking, a Restriction of Resolution; (Mimeographed Ph.D, thesis), University of Texas, Austin.

5. Bruner, J.S. et.al.(1956): 'A Study of Thinking', New York.

6. Chang, C.L.(1970a): 'The Unit Proof and the Input Proof in Theorem Proving; JACM, Vol.17,No.4, pp.698-707.

7. Chang, C.L. and Lee, R.C.T.(1971): 'Program Analysis and Theorem Proving; Div. of Computer Research and Tech., National Institute of Health, Bethesda.

8. Chang, C.L. and Lee, R.C.T.(1973): 'Symbolic Logic and Mechanical Theorem Proving.'

9. Chang, C.L. and Slagle, J.R.(1971): 'Completeness of Linear Resolution for Theories with Equality; JACM, Vol.18, No.1, pp.

10. Darlington(1968b): 'Automatic Theorem Proving with Equality Substitutions and Mathematical Inductions;' Machine Intelligence, Vol.3.

11. Davis, M.(1963): 'Eliminating the irrelevent from mechanical proofs', Proc. 15th Symp, in Applied Mathematics, A.M.S., Providence, R I, pp.15-30.

12. Davis, M. and Putnam, H.(1960): 'A computing procedure for quantification Theory', J.A.C.M. 7, No.3.

13. Feigenbaum, E.(1968): 'Artificial Intelligence Themes in the second decade', Proc., Int. Fed. of Information Processing Society.

14. Fikes and Nilsson(1971): 'STRIPS: A new Approach to the Application of Theorem proving to Problem Solving', J.A.I., Vol.2, PP. 189-208.

15. Fleising, Loveland, Smiley and Yarmash(1974): 'An Implementation Proof Procedure', J.A.C.M. 21.

16. Floyd, R.W.(1967): 'Assigning Meaning to Programs', Proc. Symp. Appl. Math., A.M.S. Vol.19.

17. Gilmore, P.C.(1960): AA Proof Method for Quantification Theory, its justification and realization', IBM J. Res. Development, pp. 28-35.

18. Green, C.C,(1969a): 'Application of Theorem Proving to Problem Solving', I.J. CAI pp.219-239.

19. Green, C.C.(1973): 'The Application of Theorem Proving to Question-Answering System', Ph.D. Thesis Management Information Services: Lib. of Congress, USA.

20. Henschen, L. and Wos, L.(1974): 'Unit refutations and Horn sets', JACM.

21. Herbrand, J.(1930): 'Researches Surla Theores dela demonstration', Trav. Soc. Sci., letters Varzovie, classe III Sci, Math. Phys., No.33, 1930.

22. Hewitt, C(1969): 'PLANNER: A language for Proving Theorems in Robots', IJCAI, pp. 295-301.

23. Hewitt, C.(1971): 'Procedural Embedding of Knowledge in PLANNER', IJCAI, pp. 167-182.

24. Hunf, E.B.(1975): 'Artificial Intellegence'.

25A. Kleene, S.C.(1967): 'Mathematical Logic', John Wiley and sons Inc., New York.

25. Kowalski, R.(1970b): 'Studies in the Completeness and Efficiency of Theorem proving by Resolution', Ph.D. Thesis, University of Edinburg at Edinburg, Scotland.

26. Kowalski, R.(1970c): 'Search Strategies for Theorem Proving', Machine Intellegence, Vol.5, New York, pp. 181-201.

27. Kowalski, R. and Haye, P.(1969): 'Semantic Trees in automatic Theorem Proving', Machine Intellegence Vol.4.

28. Kowalski and Kuehner(1970): 'Linear Resolution with Selection function', Mathematics Unit, Edinburgh University, Scotland.

29. Lankford, D.S.(1975): 'Complete sets of reductions for Computational Logic', University of Texas, at Austin, Math. Dept. Memo A TP-21.

30. Lee, R.C.T.(1967): 'A Complete Theorem and a Computer Program for Finding Theorems Derivable from given axioms' Ph.D. Thesis, Dept. of Electrical Engineering, and Computer Science, University of California, Berkley.

31. Loveland, D.W.(1968): 'Mechanical Theorem Proving by Model elimination', JACM 15.

32. Loveland, D.W.(1969): 'A Simplified Format for the Model Elimination Procedure', JACM 16.

33. Loveland, D.W.(1970a): 'A Linear Format for Resolution', Proc. IRIA Symp. Automatic Demonstration, Springer - Verlang, New York, pp. 147-162.

34. Loveland, D.W.(1970b): 'Some Linear Herbrand Proof Procedures: An Analysis', Dept. of Computer Science, Carnegie, Mellon University, Pittsburg, Pennylvania.

35. Loveland, D.W.(1972): 'A unifying view of Some Linear Herbrand Procedures', JACM 19 No.2, pp. 366-384.

36. Loveland, D.W.(1978): 'Automatic Theorem Proving: A Logical Basis', Duke University, Durham, North California.

37. Luckham, D.(1970): 'Refinements in Resolution Theory', Proc. IRIA, Symp. Automatic Demonstration, Versailes, France, 1968, Springer-Verbag, New York, pp. 163-190.

38. Manna, Z.(1969a): 'The Correctness of Programs', J.Comp. System Science, Vol.3, No.2, pp. 119-127.

39. Manna, Z.(1969b): 'Properties of Programs and the first Order Predicate Calculus', JACM 16, pp. 244-255.

40. Mc Carthy, J.(1962): 'LISP 1.5 Programmer Manual', The MIT Press, Cambridge, Massachusetts.

41. Mc Carthy,J.(1963a): 'Situations, Actions, and Casual Laws', A.I. Memo 2, Stanford: Stanford A.I. Project.

42. Mc Carthy and Hayes(1968): 'Some Phylosophical Problems from the standpoint of A.I.', AIM-13, Stanford: Stanford A.I. Project.

43. Meltzer, B.(1966): 'Theorem Proving for Computers: Some results on Resolution and Renaming', Comp. J.8, pp. 341-343.

44. Morris(1969): 'E-resolution: Extension of Resolution to Include the Equality', Proc. Int. J. CAJ, Washington D.C.

45. Newell, A. and Shaw, J.C.(1956): 'The logic Theory Machine', IRE Trans, Inf. Theory IT-2, No.3, pp. 61-79.

46. Newell, A., Shaw, J.C. and Simon, H.(1957): 'Emperical Explorations of the Logic Theory Machine', Proc., West Joint Comput. Conf. 15, pp. 218-239.

47. Nilsson, N.J.(1971): 'Problem Solving Methods in A.I.'

48. Norton, L.M.(1971): 'Experiments with heuristic Theorem Proving for Predicate Calculus with Equality', AI. 2, No.3/4, pp. 261-284.

49. Pastre, D.(1978): 'Automatic Theorem Proving in Set Theory', University of Paris VII France. J.AI. 10, pp. 1-27.

50. Philip C. Jackson (1974): 'Introduction to A.I.', New York, Petrocelli/Charter.

51, Reiter, R.(1971): 'Two Results on Ordering for Resolution with Merging and Linear Format', JACM 18, No.4,pp. 636 - 646.

52. Reiter, R.(1973): 'A Semantically Guided Deductive System for Automatic Theorem Proving', Proc. of 3rd Joint Conf. A.I., pp. 41-46.

53. Robinson, J.A.(1965a): 'A machine oriented Logic Based on the Resolution Principle', JACM 1965, 1/12, pp.23-41.

54. Robinson, J.A.(1965b): 'Automatic Deduction with Hyper-Resolution', ~~Int.~~ J. Comp. Math.1, pp. 227-234.

55. Robinson, J.A.(1967): 'A Review of Automatic Theorem Proving', Proc. Symp., Appl. Math, Vol. 19, AMS, Providence RI.

56. Robinson, J.A.(1968): 'The Generalized Resolution Principle', Machine Intellegence 3, Michie and B. Meltzer, Eds, (Edinburgh, University Press, Edinburgh, Scotland).

57. Robinson and Wos(1969): 'Paramodulation and Theorem Proving in first Order Theories with Equality', Machine Intellegence, Vol.4

58. Sadananda, R. and Mahabala, H.N.(1978): 'Conceptference: A System that develops concepts and Infers on Pictorial Data', Proc. IEEE, Vol. SMC-8, No.3, March 1978.

59. Sibert(1969): 'A Machine Oriented Logic Incorporating the Equality Relation', Machine Intellegence Vol.4.

60. Simon, H. (1971): 'The Theory of Problem Solving', Infn. Processing 1971 Amsterdam;

61. Slagle, J.R.(1967): 'Automatic Theorem Proving with renamable and Semantic Resolution(, JACM 14, No.4 pp. 687-697.

62. Slagle, J.R.(1971): 'Artificial Intellegence: A Heuristic Approach',

63. Slagle, J.R.(1972): 'Automatic Theorem Proving with built-in Theories Including Equality, Partial Ordering and Sets'. JACM 19, No.1 pp. 120-135.

64. Slagle, J.R., Chang, C.L., and Lee, R.C.T(1969): 'Completeness Theorems for Semantic Resolution in Consequence Finding', Proc. Ist Int. Jt. Conf. A.I. pp. 281-285.

65. Slagle and Farrell(1971): 'Experiements in automatic Learning for a Multipurpose Heuristic Program', Comun. ACM 14, No.2, pp. 91-99.

66. Slagle, J.R. and Norton, L.(1971): 'Experiments with an Automated Theorem Prover Having Partial Ordering Rules', Div. of Comput. Res. and Tech. Instt. of Health, Bethesda, Maryland.

67. Waldinger and Lee(1969): 'PROW: A step Towards Automatic Program Writing', I.J.C.A.I. pp 241-252.

68. Whitehead, A.N. and Russel, B.(1925-27): 'Principia Mathematica,'2nd Edn, ISBN, Cambridge University Press.

69. Wos, L., Carson, D., and Robinson, G.A.(1964): 'The Unit Preference Strategy in Theorem Proving', Proc. A.F.I.P.S. 1964, Full Joint Comp. Conf. 26, pp. 616-621.

70. Wos, L., Robinson, G.A., and Carson, D.F.(1965a): 'Efficiency and Completeness of the Set of Support Strategy in Theorem Proving', JACM 12 No.4, pp. 536-541.

71. Yates, R., Raphael, B., and Hart, T.(1970): 'Resolution Graphs', J.A.I.1 No. 4, pp. 257-290.