PROOF STRATEGIES FOR NON-CLAUSAL THEOREM PROVING

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements for the
award of the Degree of
MASTER OF PHILOSOPHY

SHANTARAM VASIKARLA

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110067
1983

CERTIFICATE

This dissertation entitled 'Proof Strategies for Non-Clausal Theorem Proving' embodies work carried out at the School of Computer and Systems Sciences, Jawaharlal Nehru University, New Delhi - 110067.

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

V. Shantaram

(V. SHANTARAM)
Student

R. Sadananda

(DR. R. SADANANDA)
Acting Dean
School of Computer and
Systems Sciences
Jawaharlal Nehru University
New Delhi - 110067.

R. Sadananda.

(DR. R. SADANANDA)
Supervisor

ACKNOWLEDGEMENTS

I express my deep sense of gratitude to my Supervisor Dr. R. Sadananda, for his excellent guidance, constant encouragement and inspiration, constructive criticism and critical comments.

I would also like to express my sincere thanks to Prof. N.P. Mukherjee, Dean, School of Computer and Systems Sciences, Jawaharlal Nehru University for helping me at various stages.

I am very grateful to Mr. R. Rajeshwar, Senior Research Fellow, who contributed a great deal of constructive advice on my writing and research.

I thank all the staff members, colleagues and friends for their help and encouragement throughout my stay here.

My special thanks are due to Mr. S.K. Sapra, for typing this dissertation with utmost care and patience.

The Financial aid from Jawaharlal Nehru University, in the form of Junior Research Fellowship, is thankfully acknowledged.

CONTENTS

CHAPTER - I

Introduction

Artificial Intelligence is the study of intelligence using the ideas and methods of computation. Unfortunately, a precise definition of intelligence is not yet defined. Many Human mental activities such as writing computer programs, doing mathematics, speaking and understanding ability of language, commonsense reasoning and even an automobile driving are said to demand some 'intelligence'. It appears to be an amalgam of so many information processing and information representation abilities.

'Human Intelligence' as per the "Webster's New Collegiate Dictionary" (1956) is

1. "The power of meeting any situation, especially a novel situation, successfully by proper behaviour adjustments.
2. The ability to apprehend interrelationships of presented facts in such a way as to guide action toward a desired goal."

These definition can well be applied to the behaviour of a machine as to that of a human. Intelligence is a multipurpose and involves the ability to learn.

Psychology, Philosophy, Linguistics and many related disciplines deal in various perspectives and methodologies for studying intelligence. For the most

part, however, the theories proposed in these fields are too incomplete and too vaguely stated. It is just impossible to realise Intelligence in computational terms in which we are interested. Even though the traditional studies of Intelligence gives us valuable ideas, relationships and constraints, something more is needed.

Artificial Intelligence offers a new perspective and a new methodology. It's central goal is to make computers more useful and to understand principles that make intelligence possible. It is obvious that intelligent computers are extremely useful. The most important point is that Artificial Intelligence aims to understand intelligence using the ideas and method of computation, thus offering totally new and different basis for theory formation.

The field is very young and it first emerged in the mid 1930's. The most important branch in the area of Artificial Intelligence is theorem proving based on logic. We can say that it all started with the advances made in mathematical logic in 1930's and 1940's, which had been under rapid development since the end of the 19th century and new concepts about computation came into being. The logical systems of Frege, Whitehead and Russell and others showed that some aspects of reasoning could be formalised in a relatively simple frame work. Mathematical logic

has remained as an active area of investigation in Artificial Intelligence as a part because logic deductive systems have been successfully implemented on computers. The mathematical formation of logical reasoning shaped people's conception of relation between computation and Intelligence even before there were computers. The ideas of the nature of computation of Turing [3] and Church and of many other Scientists, provided the link between the notion of formalisation of reasoning and the computing machines to be invented. And during that time only they thought of "Symbol processing" and Turing and Church could foresee that numbers were an inessential aspect of computation. Turing, who has been called as the father of Artificial Intelligence, not only invented a simple, universal, and non numerical model of computation, but also argued for the possibility that computational mechanisms could behave in a way that would be perceived as intelligent. After the development of computing machines guided by Babbage, Turing, Von Neumann, people began to write programs to solve puzzles, play chess, and translate text from one language to other - the first Artificial Intelligence programs.

In theorem proving, one is given a set of statements (premises) and a statement whose truth is unknown (generally known as the theorem to be proved). By applying a sequence of allowable operations such as the inference rules of logic

expand the true statements to include the theorem. The desire of finding a general decision procedure to prove theorems clearly dates back to Leibnitz (1646-1716), again revived by Peano around the century after and by Hilbert's School in 1920's. Though, Herbrand proved a very important theorem in 1930 and proposed a mechanical method to prove theorems, unfortunately his method was very difficult to apply as it was extremely time consuming to carryout by hand. After the invention of digital computer, this theorem proving became realistic.

Newell et al first produced the logic theorist (LT), a program designed to prove theorems only in the area of sentential calculus. The LT successfully proved 38 of 52 theorems of chapter-2 of Whitehead and Russell's Principia Mathematica. However, they could implement only problems in propositional logic and did not prove the theorems in first order logic. They suggested instead following a "heuristic" approach, a term they took from polya (1954, 1957) who believed that most mathematical proofs are achieved by guessing the nature of a solution, then proving that the guess is incorrect. In writing the Logic Theorist, Newell, Shaw and Simon encountered problems that forced them to develop an important new programming tool, "List Processing" (LISP). It is well suited for symbolic manupulation.

Later in 1960, Herbrand procedure was implemented on computer by Gilmore [9], followed shortly by a more efficient procedure proposed by Davis and Putnum [4].

A major breakthrough in mechanical theorem proving was made by J.A. Robinson in 1969 [19]. He developed a single inference rule, "the Resolution Principle", which was shown to be highly efficient and very easily implemented on computers. Since then, many improvements of the resolution principle have been made. Mechanical theorem proving has been applied to many areas such as program analysis, program synthesis, deductive question-answering systems, Problem solving systems, deductive database systems and robot technology. The number of applications are rapidly increasing.

CHAPTER - II

2.1 "What is a problem" ?

Though it looks very odd it is necessary to know 'what is a problem', we all know what a problem is but, to have a clear idea of what a problem is, we are difining it here. The most important characteristics of a problem is that everyone - the problem poser and the problem solver - must have common understanding of the meanings of the symbols used to represent a problem. For example let us take :

```
  2960
+ 3467
------
  ?
```

This looks perfectly like a arithmatic problem and we all know what they represent. At the same time we do not know the radix of the number system used i.e., whether it is represented radix 8, 9 or 10. They could be even the complements of respective radixes. So each and every detail of it is very important and should be clearly stated along with any problem. Suppose the computer is solver, then the programmer who is the problem poser must be sure to express his thoughts in a Language the computer understands.

2.2 Representation

Once the problem is clearly stated including all the relevant background knowledge, what we need in problem solving process is, to choose a representation. Here we

shall consider some of the representations people use for their convenience in solving problems (example : cryptarithmetic problem representation discussed in next section).

A representation is meant another data domain in which there exists an analogous well-defined problem that we shall attempt to solve, instead of attempting to solve the originally described problem.

It is rather strange to consider a problem in different domain when we are given a problem in a particular domain. Why should we forget it and try to solve a different problem i.e. the problem in different domain. The answer is that almost every problem has certain features that make it very difficult to attack it directly. By selecting an appropriate representation, however, the difficult features can be avoided. Moreover, if the representation has been well designed, its essential features correspond closely enough to those of the real problem that a solution to the well-defined problem can easily be translated into the solution of real problem. Such examples can be found frequently in Integral calculus (substitution problems).

There is a spectrum of representation alternatives and representations vary in power considerably it is said that very limited work is done on representation. Amarel (1968) has written a classic paper on the subject; it takes the reader through a series of progressive by better representations for the missionaries-and-cannibals problem. Minsky's theory of frames, Winston's learning theory (based on Minsky's theory of frames), Martin's theory of representation

dealing with natural language issues, particularly those having to do with concept hierarchies and Fahlman's Study on networks of concepts are some of the important works on representations [24, 23].

An interesting puzzle that emphesizes the importance of good representation is mentioned by McCarthy (1964). The monkey-banana problem discussed by Amarel (1966) is a standard example problem of commonsense reasoning. The use of state-description schemas has been mentioned as a powerful representation technique. A paper by Newell (1965) examines some possible approaches and their limitations toward making progress on representation problem.

2.3 Search

Search is an important part of most problem solving processes. Since, in order to make computers more smarter, we must program them to master efficient search principles. We must first understand those principles ourselves.

The first requirement for effective search is that the collection of places in which we are prepared to look-called the search Space - actually contains a solution. For many problems we are not even sure that a solution is possible. The simple fact that a solution exist in a given search space can be of tremendous help in solving a problem. The need to limit the search space arises as the search space to most of the problems is enormous. Some of the problems might take even one's life-time to solve. (example : British museum algorithm).

Searching should be performed systematically i.e. a problem-solving searching should not consist of blindly considering elements of the search space until it trips over the solution, rather it should be directed toward the solution with the examination of each element helping to suggest which other elements be examined next. For that purpose we represent the solution in the form of tree structure.

Let us take the following crypterithmetic problem [17] and try to represent in tree structure.

```
  D  O  U  V
+ Ø  C  F  G
-----------
Ø C  U  O  T
```

The problem is to find different integers for each of the letters C, D, F, G, Ø, T, U and V that will make the sum correct. If we select eight of the ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 and assign them in any order to the eight letters in the problem. There are 1,814,400 possible ways of making these assignments, which might take even one's life time to solve it. Let us find out a better way so that we can solve it fast. For the convenience we represent in the following way

	C_4	C_3	C_2	C_1	
		D	G	U	V
+		Ø	C	F	G
	Ø	C	U	G	T
	5	4	3	2	1

where C_1, C_2, C_3, C_4 represent the number 'carried' from the sum of the column to the next column.

We shall begin our analysis with column 5. From column 5 we can tell that Ø = 1, it cannot be zero as the leading zero is normally not written, at the same time it cannot be more than 1, because the most that two four digit numbers can add up to is less than 20,000 (9999+9999 = 19,998). Since Ø in column 5 is simply C_4 added to nothing else, we know that C_4 = 1 and that therefore column 4 must generate a carry.

Turning to column 4, we do not know whether C_3 is zero or one we shall consider the two alternatives, each at a time. First we assume that C_3 = 0. Since C_3 = 0 and C_4 = 1 the column 4 sum must be D + Ø = C + 10, and since we already determined that Ø = 1, we have D = C + 9. Since D cannot be more than 9, we have C = 0 and D = 9. We can now go on to the next column, where C_2 might be zero or one. The column sum is = C_2 + G + C = U. For C_2 = 0 and we know

C = 0, this means G = U. Since both letters can not be equal to same number. We are trying other combination (C_3 = 0, C_2 = 1) and C_3 = 1. Proceeding further we get $\emptyset$ = 1, C = 0, D = 9, F = 8, 1 + G = U, V + G = T + 10 = 12 (because T cannot be 0 or 1) we still have to select assignments for V, G, U and T from the remaining numbers 2, 3, 4, 5, 6 and 7 that satisfy the following conditions :

$$G+1 = U \quad \text{and} \quad T+10 = V+G = 12$$

we get them as U = 6, V = 7 T = 2 G = 5 as we know more about G (it is in two equations), we begin to grow the last node on the tree with all possible assignments to E. The resultant impossible requirement on F or $\emptyset$ again quickly narrow the search and a unique solution can be found. The complete search tree is shown in fig.2.1.

Typical search procedures [22, 13] include "breadth first", which tries to look for solutions paths in all directions at once; "depth first", which tries to follow each path it discovers as far as possible before considering any others; and "progressive deepening, which is a simple compromise between breadth first and depth first. A more effective compromise strategy, called A*, is essentially a cost first strategy; at every point it extends which ever known path is least costly at that point, and thus is guaranteed always to find the minimum cost solution path.

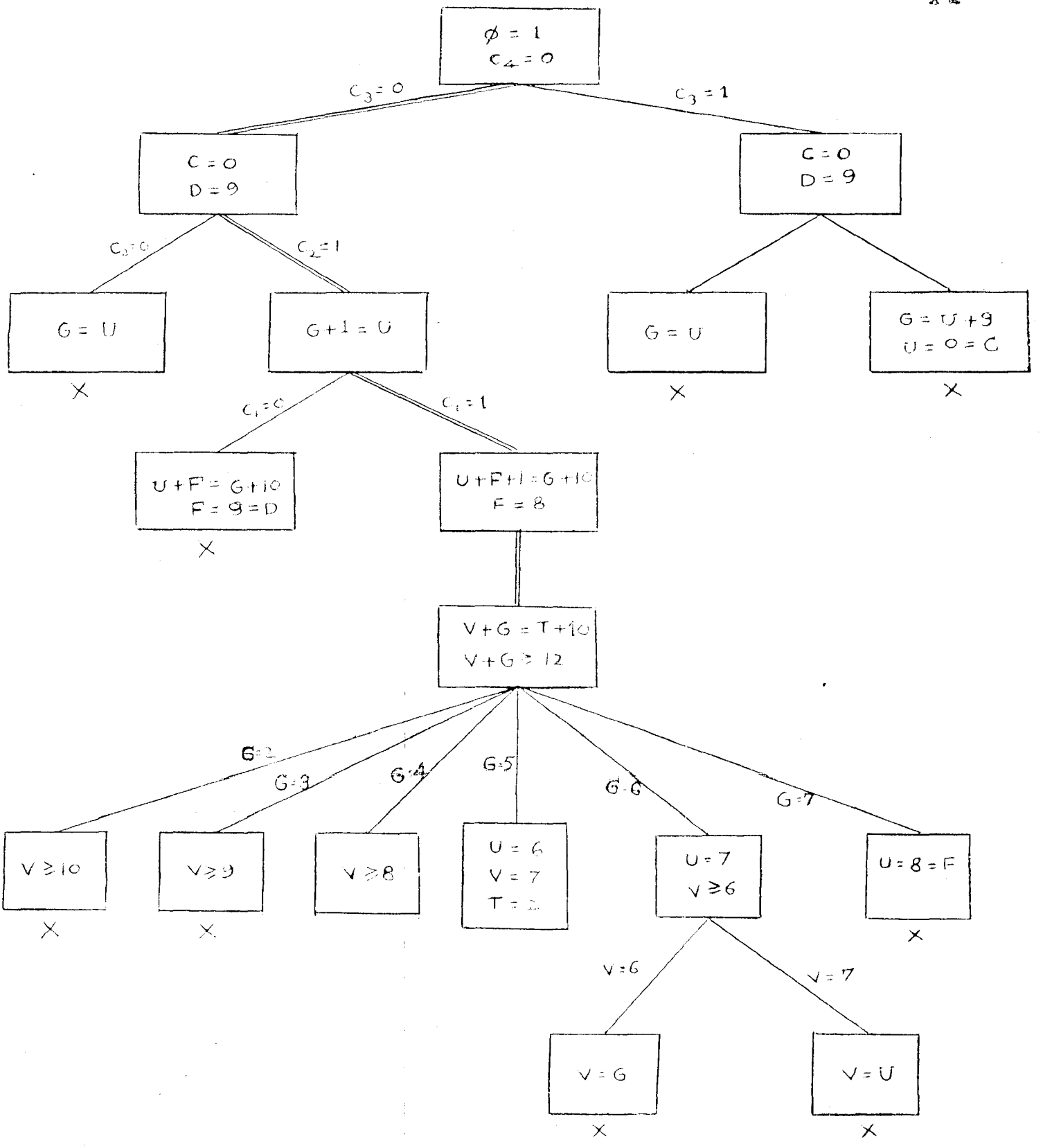


Fig.2.1 Complete search tree for cryptarithmatic problem

Sometimes, we have useful knowledge about a problem that is not captured in its tree or graph representation. It may still be possible to use this extra knowledge in order to help the search procedure.

2.4 Deduction & Propositional ('0' order predicate) Calculus

'Deduction' means obtaining solutions to problems by using some systematic reasoning procedure. The propositional calculus [6] is also called as zero predicate calculus. In propositional calculus the method for proving theorem is known as the method of truth tables. This is a semantic method, in which we examine all possible combinations of interpretations for the propositional variables. It is inefficient and laboureous though it can be implemented on computer. About 1960 Hao wang at Harvard University developed a syntactic method that is about as efficient as any general method for propositional calculus can be. It produces exactly the same results as truth tables, usually requires much less computational effort and easy to program. It is frequently used now as an exercise in the use of symbol-manipulation programming methods.

Natural languages are not suitable for the systematic study of the propositional logic as they are not precise and clear. Hence, we need formal languages. The study of formal languages is the study of syntax and semantics.

2.4.1 Syntax

So specify a syntax we must specify the alphabet of the symbols to be used in the language and how these symbols are to be put together into legitimate expressions in the language. An important class of expressions of the predicate calculus are called the well-formed formulas(wffs).

The alphabet forms with the following set of symbols.

1. An infinite set of propositional letters or atoms. These are normally represented by capital letters (ex. P, Q, R . . .).
2. Punctuation marks : , ( )
3. A finite set of logical connectors $\{\sim, \vee, \wedge, \Rightarrow, \Leftrightarrow\}$
4. A set of rules to determine valid strings

From these symbols we construct various expressions. The classes of interesting expressions can be defined recursively as follows :

1. All atoms are Wffs
2. If P is a wff, then $\sim P$ is wff
3. If P and Q are wffs, there $P \vee Q$, $P \wedge Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$ are also wffs.

Typical truth table for wffs $P \wedge Q$, $P \vee Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$ given in table 2.1

P	Q	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

Table 2.1

Where T and F stand for Truth and false respectively.

2.4.2 Semantics

Semantics deals with the interpretation of formal language. Interpretation constitutes the domain of the wff, which is either T (true) or F (False). If there are 'n' distinct atoms in a wff, then there are 2^n distinct interpretations for that wff. Two wffs are equal if their truth tables are same.

As the zero order predicate calculus can not deal with individuals, functions and relations, we are switching over to First order predicate calculus.

2.5 First order predicate calculus

The beauty of the First order predicate logic [16] is that, it is possible to express individuals, functions, relations and much of mathematics in it. Almost any interesting reasoning problem about general commonsense

knowledge, can also be encoded into First order predicate calculus. We can make logical deductions of new statements from a set of given ones with the rules of inference it has got. It is most powerful to conduct deductions because of its generality and logical power.

2.3.1 Syntax

Syntax involves alphabet of symbols and a set of rules to construct various expressions from these symbols.

The basic alphabet consists of the following sets of symbols :

1. Predicate symbols : P^n, Q^n, R^n . . . where $n \geq 0$
   (For $n = 0$, Predicate symbols are treated as propositions).
2. Function Symbols : g^n, h^n, k^n . . . where $n \geq 0$
   (For $n = 0$, Function symbols are treated as constants).
3. Variables : x, y, z, . . .
4. Constants : a, b, c, . . .
5. Logical connectives : $\sim$, $\vee$, $\wedge$, $\Rightarrow$, $\exists$, $\forall$
6. Punctuation marks : , ( )

where 'n' denotes the number of arguments that a function or a predicate symbol can take. The classes of interesting expressions that can be constructed from the above symbols are as follows :

1. Terms
   a. each constant symbol is a term
   b. each variable is a term

c. If g is an n-place functions symbol, and t_1, t_2 and t_n are terms, then $g(t_1, t_2, .., t_n)$ is a term

No other symbol is a term.

2. Atomic formulas

a. All proposition letters are atomic formulas.

b. If $t_1, t_2 \ldots, t_n$ $(n \geq 1)$ are terms, then the expression $P_i^n (t_1, t_2, \ldots t_n)$ is atomic formula

c. No other expression is atomic formula.

3. Well-formed formulas (wffs)

a. every atomic formula is a wff.

b. If P is wff, then $(\sim P)$ is also wff.

c. If P and Q are wffs, then $(P \Rightarrow Q, \ P \Leftrightarrow Q)$ are also wffs.

d. If w is a wff and x is a free variable occuring in w, then $(\forall x)\ w$ and $(\exists x)\ w$ are wffs.

No other expression is a wff.

A wff is called a closed wff, if it does not contain free variables.

2.3.2 Semantics

The semantics of a statement is specified by an interpretation. An interpretation consists of (1) A non-empty set of objects called the domain (or universe), (2) an assignment of an object in the domain to each constant, (3) an assignment of an n-ary function on the domain to each n-ary function letter, and (4) an assignment

of n-ary relation on the domain to each n-ary predicate letter. A variable then ranges over the elements of the domain.

2.6 Satisfiability and validity of wff.

A closed wff (with out free variables) is true or false with respect to the given interpretation. We shall consider only closed wffs. An interpretation that makes a wff True is said to 'satisfy' the wff; or equivalently, the interpretation is said to be a 'model' of the wff. A wff is 'satisfiable' if and only if there exists an interpretation that satisfies the wff. A wff is 'logically valid' if and only if it is satisfied by all possible interpretations, or equivalently if the negation of the wff is 'unsatisfiable'.

The 'theorems' of a logical systems are usually intended to be the valid wff. However, since it is not practical in general to enumerate and test all possible interpretations, formal syntactic procedures called 'proof procedures' must be used to establish theorems. If every theorem produced by a proof procedure is really valid, the procedure is called 'sound'. If every valid formula can be demonstrated to a theorem, the procedure is 'complete'. In the desirable case that a proof procedure is both sound and complete, the theorem of the procedure coincide with the valid wff. A 'decision procedure' is a procedure that can decide in a finite number of steps whether or not any given wff is valid.

Unfortunately it is known that there are proof procedures for first order Predicate Calculus, but there is no decision procedure for first order predicate calculus. This means that 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 a decision procedure does not limit the applicability of first order logic as much as it may at first appear. Because of the time and space constraints on practical computation, the huristic power (to be discussed in IV Chapter) of proof procedure . . . i.e., its ability to prove useful theorems efficiently . . . is more important than its theoretical limitations. This issue is throughly discussed by Robinson in one of his papers. A decision procedure that requires enormous amount of time or intermediate storage is impossible, in practice, from a proof procedure that never terminates for some wffs.

A wff B is a 'logical consequence' of (follows from, semantically) a set of axioms (premises) C if and only if every model of C is a model of B. It can be easily shown that B is a logical sequence of C if and only if $C \Rightarrow B$ is logically valid, or, equivalently, if the statement $\sim[C \Rightarrow B]$ (logically equivalent to $C \wedge \sim B$) is unsatisfiable. In order to show that a wff unsatisfiable, we must demonstate that no interpretation exists for which it has value True (T). Though it look impossible in the

beginning, they are procedures to accomplish it which need that the wffs first be put in a special, convenient form called 'clause form' [13] .

2.7 Conversion of wff into clause form

As the complicated wffs with existential, universal, implication signs cannot be used directly to prove or verify a theorem, they are first converted into the convenient standard form called 'skolem conjunctive form' or 'skolem standard form' which is first introduced by Davis and Putnam [4] in Automatic Proof Procedures. Here, we define the prenex normal form, an important step in the convertion of a wff into a set of 'clauses'.

A wff in the first order logic is said to be in a prenex normal form if and only if it is in the following form :

$$(Q_1\ x_1)\ \ldots\ (Q_n\ x_n)\ (M)$$

where every $(Q_i\ x_i)$, $i = 1, \ldots, n$, is either $(\forall\ x_i)$ or $(\exists\ x_i)$, and M is a formula containing no quantifiers. $(Q_1\ x_1)\ \ldots\ (Q_n\ x_n)$ is called the prefix and M is called matrix of the wff. Then we convert it into a set of clauses.

A given wff is converted into a set of clauses by applying the following operations in sequence. We will explain this process with the wff.

$$(\forall\ x)\ \{\ P(x) \Rightarrow \{(\forall\ y)\ \{P(y) \Rightarrow P(f\ (x, y))\} \wedge \sim(\forall\ y)\ \{Q(x, y) \Rightarrow P(y)\}\}\}$$

Step-1 Eliminate implication signs : The implication sign is eliminated by making substitution '$\sim A \vee B$' form '$A \Rightarrow B$'. Substituting in our wff yields $(\forall x)$

$$(\forall x)\{\sim P(x) \vee \{(\forall y)\{\sim P(y) \vee P(f(x, y))\} \{\sim(\forall y)\{\sim Q(x, y) \vee P(y)\}\}\}$$

Step-2 Reduce scopes of negation signs : By repeatedly using the following substitutions, we can reduce the scope of each until it applies to a single predicate letter :

Substitute	$\sim A \vee \sim B$	for	$\sim(A \wedge B)$
"	$\sim A \wedge \sim B$	"	$\sim(A \vee B)$
"	A	"	$\sim\sim A$
"	$(\exists x)\ A$	"	$\sim(\forall x)\ A$
"	$(\forall x)\ A$	"	$\sim(\exists x)\ A$

using the above substitutions our wff becomes

$$(\forall x)\{\sim P(x) \vee \{(\forall y)\{\sim P(y) \vee P(f(x, y))\} A(\exists y)\{Q(x, y) A \sim P(y)\}\}\}$$

Step-3 Standardise variables : Standardising variables with in a wff means to rename the dummy variable to ensure that each quantifier has its own dummy variable. So, our wff becomes

$$(\forall x)\{\sim P(x) \vee \{(\forall y)\{\sim P(y) \vee P(f(x, y))\} A(\exists w)\{Q(x, w) A \sim P(w)\}\}\}$$

Step-4 Eliminate existential quantifiers : The rule for eliminating an existential quantifier from a wff is to replace each occurence of its existentially quantified variable by a Skolem functions. Function letters for skolem functions must be new, i.e., they can not be ones that already occur in the wff. Eliminating existential quantifiers, our wff becomes,

$$(\forall x)\{\sim P(x) \vee \{(\forall y)\{\sim P(y) \vee P(f(x, y))\} \wedge \{Q(x, g(x)) \wedge \sim P(g(x))\}\}\}$$

Step-5 Convert to Prenex form : Move all the universal quantifiers to the first of the wff. Our wff becomes

$$\underbrace{(\forall_x \forall_y)}_{\text{Prefix}} \underbrace{\{\sim P(x) \vee \{\{\sim P(y) \vee P(f(x, y))\} \wedge \{Q(x, g(x)) \wedge \sim P(g(x))\}\}}_{\text{matrix}}$$

Step-6 Put matrix in conjunctive normal form : Use repeatedly the following rule to put the wff in conjunctive normal form :

Substitute $\{A \vee B\} \wedge \{A \vee C\}$ for $A \vee \{B \wedge C\}$

our wff becomes

$$(\forall_x \forall_y) \{\{\sim P(x) \vee \sim P(y) \vee P(f(x, y))\} \wedge \{P(x) \vee Q(x, g(x))\} \wedge \{\sim P(x) \vee P(g(x))\}\}$$

Step-7 Eliminate universal quantifiers and '$\wedge$' (and) signs : Eliminating universal and '$\wedge$' (and) signs, our wff becomes the following clauses

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

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

$\sim P(x) \vee P(g(x))$

CHAPTER - III

Automatic theorem proving was first initiated by Herbrand in the early 1930's. He followed refutation procedure for proving theorems. However, Herbrand Procedure has one major drawback. It required to generate innumerable ground clauses and these clauses grow exponentially. With current computer technology, it is impossible even to store all clauses in computer. So the testing of its unsatisfiability is completely unthinkable.

In order to avoid the generation of innumerable clauses as required by the Herbrand's procedure, we shall introduce the Resolution principle, developed by Robinson [19]. This was proved to be effective, sound and complete.

3.1 The Resolution

The Resolution procedure finds proofs by refutation. To prove a theorem Q by refutation, one assumes that the theorem is not a logical consequence of the axioms B, and then derives a contradiction. The resolution procedure is a refutation algorithm that deduces from $(B \wedge \sim Q)$ an explicit contradiction. The search for a contradiction is an attempt to construct a model that satisfies $(B \wedge \sim Q)$. It has been shown that the resolution procedure deduces a contradiction if and only if $(B \wedge \sim Q)$ is unsatisfiable $(B \Rightarrow Q)$ is logically valid); thus, resolution is sound and complete proof procedure. To prove that a statement Q does not follow from set of axious B, one assumes it does and attempts to derive a

contradiction from $B \wedge \sim Q$. No decision procedure exists for the first order logic, so in general, for a given B and a given Q, one can not guarantee that the proof procedure will terminate in either the attempted proof of Q or the attempted disproof of Q from B.

In most automatic theorem proving, statements are converted into a standard quantifier-free form called clause form. Convertion of any wff into clause form is explained earlier. Each clause is a disjunction of literals; a literal is either an atomic formula or the negation of an atomic formula. A conjunction of several clauses may be written as a set of clauses.

The Resolution proof procedure uses statements in the standard clause form. First, the formula $B \wedge \sim Q$ (B is a set of axioms, $\sim Q$ is the negation of the theorem) is represented as a set of clauses. Then new clauses called resolvents are deduced from the starting clauses by the resolution rule of inference. The main theorem of the resolution states that if a resolvent is not satisfiable, then none of its antecedents are satisfiable. The goal of the procedure is to deduce the empty clause, represented as $\Box$, an evidence of contradiction that is not satisfiable. This demonstates that all its antecedents, including the starting wff, are not satisfiable.

The rule of Resolution is best illustrated in its propositional form : if $p \vee q$ and $\sim p \vee r$ are two wffs in which p is any proposition and q and r are any wffs, one

may deduce the wff q v r. More briefly, $(p \vee q) \wedge (\sim p \vee r) \Rightarrow (q \vee r)$.

The exact statement of the resolution rule requires that we introduce the notation of a substitution. A 'substitution' is a set of forms that are to be substituted for a set of variables occuring in given wff. A substitution σ may be written as a set, $\sigma = \{t_1/x_1, t_2/x_2, \ldots, t_n/x_n\}$ meaning that term t_1 is to be substituted for x_1, t_2 for x_2, etc. If L is a formula then $L\sigma$ denotes the formula resulting from performing the substitution σ on the formula L.

Two formulas L_1 and L_2 are said to 'unify' if there exists a substitution σ such that $L_1\sigma = L_2\sigma$. If $L' = L\sigma$, for any σ, then L' is said to be an 'instance' of L. The substitution σ is said to be the 'most general unifier' 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 $L_1\sigma = L_2\sigma$. Robinson has shown that if two formulas unify, there exists a most general unifier of the two formulas.

The heart of the Resolution process is the 'Unification Algorithm' that determines whether or not two formulas unify, and, if they do, finds the substitution set σ that is the most general unifier of the two formulas. This algorithm guarantees that in one sence each resolution inference step is as general as possible, since every less general unification is implied.

3.2 Unification Algorithm :

Here, we shall give a unification algorithm for finding a most general unifier for a finite unifiable set of non empty expressions. The algorithm will also detect when the set is not unifiable.

Unification Algorithm :

Step-1 Set K = 0, W_k = W, and $\sigma_K = \varepsilon$

Step-2 If W_k is a singleton, stop; σ_k is a most general unifier for W otherwise, find the disagreement set D_k of W_k.

Step-3 If there exist elements V_k and t_k in D_k such that V_k is a variable that does not occur in t_k, go to step 4. Otherwise stop; W is not unifiable.

Step-4 Let $\sigma_{k+1} = \sigma_k \{t_k/v_k\}$ and $W_{k+1} = W_k \{t_k/v_k\}$. Note that $W_{k+1} = W_{k+1}$

Step-5 Set K = K+1 and go to step-2.

The exact statement of the Resolution rule of inference begins as follows.

Let the prospective parent clauses be given by $\{L_i\}$ and $\{M_i\}$ and assume that the variables occuring in $\{M_i\}$ do not occur in $\{L_i\}$, and vice versa. Suppose that $\{l_i\} \subseteq \{L_i\}$ and $\{m_i\} \subseteq \{M_i\}$ are two subsets of $\{L_i\}$ and $\{M_i\}$, respectively, such that a most general unifier λ exists for the set $\{l_i\} \cup \{m_i\}$. That is, $\{m_i\}\lambda$ contains a single literal equal to the negation of the single literal* in $\{l_i\}\lambda$. Then we

* equal to the negation of the single literal

say that the two clauses $\{L_i\}$ and $\{M_i\}$ resolve and that the new clause

$$[\{L_i\} - \{l_i\}]\lambda \quad \cup \quad [\{M_i\} - \{m_i\}]\lambda$$

is a resolvent of the two clauses. The resolvent is an inferred clause. The process of forming a resolvent from two parent clauses is referred to as 'Resolution'. When two clauses are resolved, we many infer more then one resolvent because of various ways of choosing $\{l_i\}$ and $\{m_i\}$, but they are finite in number.

example : Consider the following set of formulas

F_1 : $(\forall_x)\,(C(x) \Rightarrow (W(x) \wedge R(x)))$

F_2 : $(\exists_x)\,(C(x) \wedge O(x))$

G : $(\exists_x)\,(O(x) \wedge R(x))$

show that G is a logical consequence of F_1 and F_2.

We transform F_1, F_2 and G into clause forms and obtain the following clauses

(1) $\sim C(x) \vee W(x)$

(2) $\sim C(x) \vee R(x)$ } From F_1

(3) $C(a)$

(4) $O(a)$ } From F_2

(5) $\sim O(x) \vee \sim R(x)$ From $\sim G$

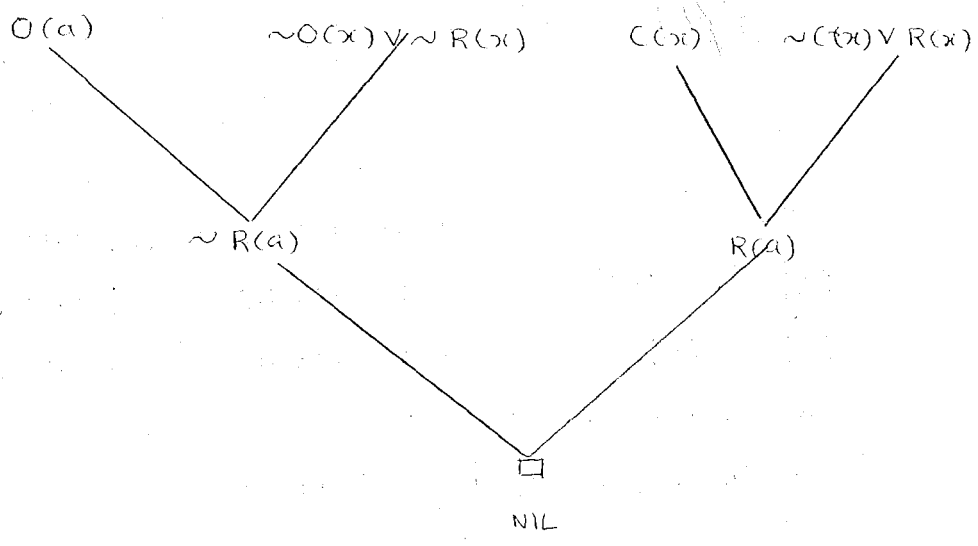
The above set of clauses is unsatisfiable. This we will prove by resolution.

(6) $R(a)$ a resolvent of (3) and (2)

(7) $\sim R(a)$ a resolvent of (5) and (4)

(8) $\square$ a resolvent of (7) and (6)

Something we can also represent as refutation Graph. in fig. 3.1



3.1 A refutation graph for the given example.

The resolution rule tells us how to derive a new clause from a specified pair of clauses containing a specified literal, but does not tell us how to choose which clauses to resolve. A mechanical attempt to resolve all possible pairs of clauses generally results in the generation of an unmanageably large number of irrelevant clauses. Therefore, various huristic search principles have been developed to guide and control the selection of clauses for Resolution. Among the most important of these are the breadth-First strategy [22] the set of support strategy [26], The linear input form strategy [13], the ancestry filtered form strategy [13] and simplification strategies [15].

All these strategies preserve completeness of the theorem prover. We discuss these strategies in detail in the next chapter.

3.3 Non-clausal Theorem Proving :

Recently Neil V Murray [13] developed a Proof procedure for proving theorems in First order predicate calculus, in which statements are not required to be in the clause form required by standard resolution theorem provers. Only condition is that the well-formed formulas (wffs) to be quantifier free and have distinct variables. All variables are implicitly universally quantified. More over, this system handles even the logical equivalence connective also. Non-clausal resolution is also effective, sound and complete.

3.3.1 The Formalism :

Throughout the discussion, we assume the commutative properties of $\wedge$, $\vee$ and $\Leftrightarrow$, T and F are atoms, and denote Truth and falsehood respectively. Any n-ary predicate symbol followed by n Terms is an atom. Well-formed formulas (wffs) are all and only those finite expressions defined by the following two rules.

1. Any atom is a well-formed formula (wff).
2. If A and B are wffs, then

   $\sim A$, $(A \vee B)$, $(A \wedge B)$, $(A \Rightarrow B)$, $(A \Leftrightarrow B)$

   are also well-formed formulas (wffs).

3.3.2 Polarity :

Let A, B, w be well-formed formulas. We say that w is positive in A if and only if at least one occurence of w in A is positive. w is negative in A if and only if at least one occurence of w in A is negative. The one occurence of w in w is positive. We determine the polarity of the constituents of a well-formed formula as follows :

(a) If W is positive (negative) in A, then w is negative (positive) in

$\sim A$ and $(A \Rightarrow B)$

(b) If w is positive (negative) in A, then W is positive (negative) in

$(A \vee B)$, $(A \wedge B)$ and $(B \Rightarrow A)$

(c) If w occurs in A, then w is positive and negative in $(A \Leftrightarrow B)$

Notation : Let A, w, a be well-formed formulas

$A\{a/w\}$ denote the well-formed formula obtained by replacing every occurence of w in A by a. We may have replacements with more than one component, i.e.

$\{A_1/W_1, A_2/W_2, \ldots, A_n/W_n\}$

We interpret this as the simultaneous replacement noting that both the A's and W's are well-formed formulas, and for $1 \leq i, j \leq n$, $i \neq j$ does not occur in W_j.

3.3.3 Reduction

We denote 'reduces to' by $\rightarrow$ and 'does not reduces to' by $\not\rightarrow$. Let A, B, C, w, a be well-formed formulas. We

have the following reduction rules :

$\sim T \rightarrow F$.	$\sim F \rightarrow T$.
$(T \wedge w) \rightarrow w$.	$(F \wedge w) \rightarrow F$.
$(T \vee w) \rightarrow T$.	$(F \vee w) \rightarrow w$.
$(T \Rightarrow w) \rightarrow w$.	$(F \Rightarrow w) \rightarrow T$
$(w \Rightarrow T) \rightarrow T$.	$(w \Rightarrow F) \rightarrow \sim w$
$(T \Leftrightarrow w) \rightarrow w$.	$(F \Leftrightarrow w) \rightarrow \sim w$

If W occurs in A and $W \rightarrow a$. then $A \rightarrow A\{a/w\}$

If $A \rightarrow B$ and $B \rightarrow C$, then $A \rightarrow C$.

A well-formed formula is said to be in reduced form whenever no reduction rule can be applied to it. In addition to $\rightarrow$ and $\not\rightarrow$, we will need the following metalinguistic relations :

(a) We use $A = B$ ($A \neq B$) to denote that A and B have (do not have) the same truth-value under all assignments of truth values to their atoms.

(b) We use $A \equiv B$ ($A \not\equiv B$) to denote that A and B are (are not) syntactically identical.

The above reduction rules are essentially those truth-functional simplifications that one would expect when dealing with the connectives in our formulism. In the next four lemmas we relate the process of reduction with the truth value and syntactic form of the well-formed formula involved.

Lemma - 3.3.3.1

Every well-formed formula W has a reduced form W' such that either $W' \equiv T$, or $W' \equiv F$, or T and F do not occur in w.

Lemma - 3.3.3.2

If all the atoms of a well-formed formula W are T or F, then the reduced form of W is either T or F.

Lemma - 3.3.3.3

If A reduces to B, then $A = B$ and $A \not\equiv B$.

Lemma - 3.3.3.4

Let A be any well-formed formula and $\{A_1, A_2, \ldots A_n\}$ be the set of all the distinct atoms in A. Let s be any assignment of truth-values to $\{A_1, A_2, \ldots A_n\}$.

If $A = F$, then

$$A\left\{s(A_1)/A_1, s(A_2)/A_2, \ldots, s(A_n)/A_n\right\} \rightarrow F$$

where $s(A_i)$ is the truth-value assigned to A_i by s.

3.3.4 Reduction, Polarity, and Replacement

In this section we state theorems which relate the replacement of a sub-well formed formula by a truth-value, with the resulting value of the entire well-formed formula and the polarity of the sub-well-formed formula. These theorems will be required for completeness.

Theorem - 3.3.4.1

Given well-formed formula A where $T \not\equiv A \not\equiv F$

If $A\{F/w\} = F(T)$ then w is positive (negative) in A

If $A\{T/w\} = F(T)$ then W is negative (positive) in A

Theorem - 3.3.4.1'

Given well-formed formula A where $T \not\leftarrow A \not\rightarrow F$

If $A\{F/w\} \rightarrow F(T)$, then w is positive (negative) in A.

If $A\{T/w\} \rightarrow F(T)$, then w is negative (positive) in A.

Theorem - 3.3.4.2

Given well-formed formulas A, B and atom w where $T \neq w \neq F$. Suppose that w occurs in B and $A \rightarrow B$ If w is positive (negative) in B, when w is positive (negative) in A.

Theorem - 3.3.4.3

If $A = F(T)$, $A \not\rightarrow F(T)$, and $A \neq F(T)$, then some atom a is both positive and negative in A.

3.3.5 The Proof Procedure

We now give a concise description of the proof procedure. It is non-clausal in essence similar to breadth-first binary resolution. The inference rule is called NC - resolution.

Given well-formed formulas U_i and U_j, we can determine if they contain atoms $L_1, \ldots, L_n$ such that $\{L_1, \ldots, L_n\}$ can be unified by mgu (most general unifier) M, i.e., $\{L_1, \ldots, L_n\} M = \{L\}$. We may also determine whether L occurs in $U_i M$ and $U_j M$ with opposite polarity. Now suppose L is positive in $U_i M$ and negative in $U_j M$. The following well-formed formula is an NC - resolvent of U_i and U_j :

$$U_i M \{F/L\} \vee U_j M \{T/L\}$$

If L is negative in $U_i M$ and positive in $U_j M$, then we have the following dual NC - resolvent

$$U_i M \{T/L\} \vee U_j M \{F/L\}$$

Each of these NC - resolvents is a logical consequence of U_i and U_j. Note that U_i and U_j may have only finitely many NC - resolvents. We may therefore compute the set of all NC - resolvents of pairs of well-formed formulas in some given set U of well-formed formulas and add the NC - resolvents to the original set. The iteration of this process finitely many times yeild a contradiction for all unsatisfiable sets U.

NC - resolution can be represented as a the transformation of inference nodes to failure nodes, in a semantic tree.

Example

Consider the two well-formed formulas

$$P(x) \Leftrightarrow R(a) \text{ and } P(b) \Rightarrow (Q(a, b) \vee \sim P(y))$$

We may take $L_1, L_2, L_3 = \{P(x), P(b), P(y)\}$
We get $L = P(b)$ after $M = \{b/x, b/y\}$
$P(b)$ is positive and negative in $P(b) \Leftrightarrow R(a)$, but is only negative in $P(b) \Rightarrow (Q(a, b) \vee \sim P(b))$.
We may infer the NC - resolvent.

$P(b) = R(a)\{F/P(b)\} \vee R(b) \Rightarrow (Q(a, b) \vee \sim P(b)$
$\{T/P(b)\}$ which reduces to
$\sim R(a) \vee Q(a, b)$.

CHAPTER - IV

4.1 Necessity of a strategy

The resolution rule which we discussed in the earlier chapter tells us only how to derive a new clause from a specified pair of clauses containing a specified literal, but does not tell us how to choose which clauses to resolve. This resolution rule is very well known as the 'unrestricted resolution rule'. A mechanical attempt to resolve on possible pairs of clauses generally results in the generation of an unmanageably large number of irrelevant clauses, though the resolution rule is more efficient than the earlier methods such as Herbrand's procedure used by Gilmore [5]. And most of the new clauses that are generated by the unlimited application of resolution rule are irrelevent and redundant. This straight forward application of resolution rule looks like a breadth-first search for a refutation. Such a search would start with a set of clauses S and add to this set all the resolvents between pairs of clauses in S to produce the set R(S). After that, all of the resolvents between pairs of clauses in R(S) would be added to produce the set R(R(S)) = R^2(S), and so on. This type of search is usually not practical as the set R(S), R^2(S), ... grow too fast. The following example will give a clear picture of the above discussion.

example : 1

Show that the following set is unsatisfiable by resolution

$$S = \{\sim P(x) \vee Q(x),\ \sim P(x) \vee Q(x),\ P(x) \vee \sim Q(x),\ \sim P(x) \vee \sim Q(x)\}$$

$R^0(S) =$ (1) $P(x) \vee Q(x)$
(2) $\sim P(x) \vee Q(x)$
(3) $P(x) \vee \sim Q(x)$
(4) $\sim P(x) \vee \sim Q(x)$ } S

$R^1(S)$: (5) $Q(x)$ from (1) and (2)
(6) $P(x)$ from (1) and (3)
(7) $Q(x) \vee \sim Q(x)$ from (1) and (4)
(8) $P(x) \vee \sim P(x)$ from (1) and (4)
(9) $Q(x) \vee \sim Q(x)$ from (2) and (3)
(10) $P(x) \vee \sim P(x)$ from (2) and (3)
(11) $\sim P(x)$ from (2) and (4)
(12) $\sim Q(x)$ from (3) and (4)

$R^2(S)$: (13) $P(x) \vee Q(x)$ from (1) and (7)
(14) $P(x) \vee Q(x)$ from (1) and (8)
(15) $P(x) \vee Q(x)$ from (1) and (9)
(16) $P(x) \vee Q(x)$ from (1) and (10)
(17) $Q(x)$ from (1) and (11)
(18) $P(x)$ from (1) and (12)
(19) $Q(x)$ from (2) and (6)
(20) $\sim P(x) \vee Q(x)$ from (2) and (7)

(21) $\sim P(x) \vee Q(x)$	from (2) and (8)
(22) $\sim P(x) \vee Q(x)$	from (2) and (9)
(23) $\sim P(x) \vee Q(x)$	from (2) and (10)
(24) $\sim P(x)$	from (2) and (12)
(25) $P(x)$	from (3) and (5)
(26) $P(x) \vee \sim Q(x)$	from (3) and (7)
(27) $P(x) \vee \sim Q(x)$	from (3) and (8)
(28) $P(x) \vee \sim Q(x)$	from (3) and (9)
(29) $P(x) \vee \sim Q(x)$	from (3) and (10)
(30) $\sim Q(x)$	from (3) and (11)
(31) $\sim P(x)$	from (4) and (5)
(32) $\sim Q(x)$	from (4) and (6)
(33) $\sim P(x) \vee \sim Q(x)$	from (4) and (7)
(34) $\sim P(x) \vee \sim Q(x)$	from (4) and (8)
(35) $\sim P(x) \vee \sim Q(x)$	from (4) and (9)
(36) $\sim P(x) \vee \sim Q(x)$	from (4) and (10)
(37) $Q(x)$	from (5) and (7)
(38) $Q(x)$	from (5) and (9)
(39) $\square$	from (5) and (12)

It can be observed from the above example that many irrelevent and redundant clauses have been generated. For example, (7), (8), (9) and (10) are tautologies (i.e., any wff of the form $P \vee \sim P$). As a tautology is true in any interpretation, if a tautology is removed from an unsatisfiable set of clauses, the remaining set still be unstatisfiable. Therefore, a tautogy is an irrelevant

clause and should not be generated. Even if one is generated, except in a very few cases, it should be deleted. Otherwise, it may interact with other clauses and produce other redundant clauses that we do not require at all. For example, (13) to (16), (20) to (23), (26) to (29), (33) to (36), and (37) to (38) are all such clauses. Moreover, the clauses $P(x)$, $Q(x)$, $\sim P(x)$, and $\sim Q(x)$ are repeatedly generated. In fact, to get the proof for S, we need only generate clauses (5), (12), and (39). To solve this redundancy problem and the decisions about which two clauses from a set of clauses S to resolve and which resolution of these clauses to perform, we shall consider various "strategies" developed in the next section.

4.2 Strategies

The various heuristic search principles are developed to guide and control the selection of clauses for resolution.

The goal of any strategy in most cases is to reduce the number of resolvents which must be generated from an unsatisfiable set of clauses before empty clause $\square$ is produced. Several strategies for selecting clauses have been developed for resolution; here we discuss some of the strategies and their application, first to Clausal and after to Non-Clausal theorem proving.

4.3 Strategies - Clausal Theorem proving

4.3.1 The Breadth first strategy :

In breadth - first strategy, all the first-level resolvents are computed first then the second-level resolvents and so on. Sometimes we may find answer with one or two levels of resolution of only specific clauses. Though the breadth first strategy is complete, it is most inefficient.

example 2 : Prove that the following set is unsatisfiable using breadth first strategy

$S = \{\sim K(x) \vee T(x),\ K(x),\ \sim T(x) \vee N(x),\ \sim P(x) \vee \sim N(x),\ P(x)\}$

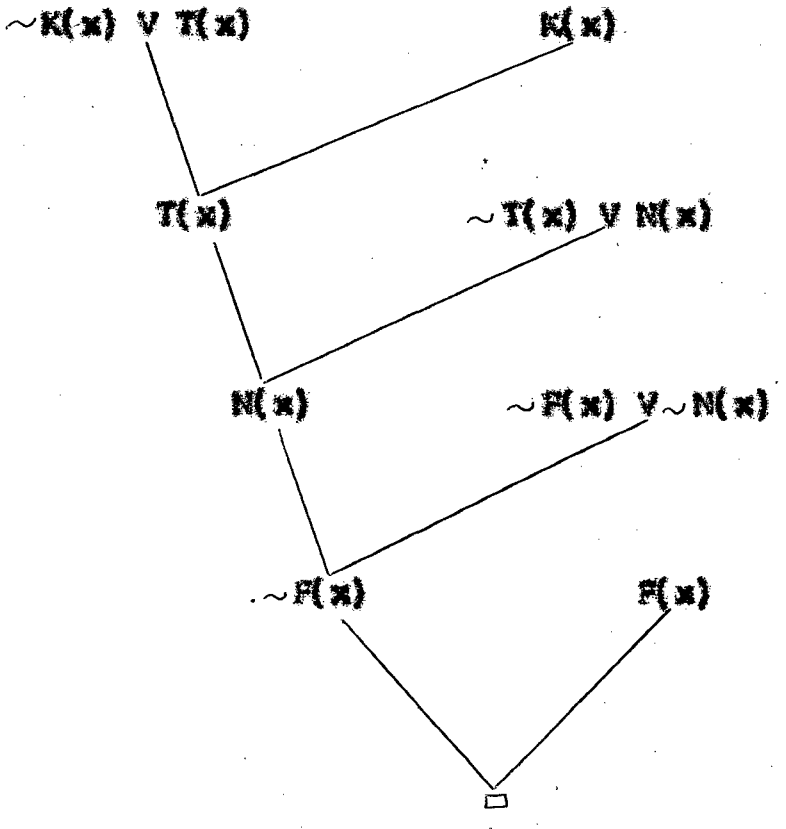


Fig. 4.1 Resolution refutation tree

Fig.4.2 shows the refutation graph for the same problem by using the breadth-first strategy. It can be observed from the diagram (4.2) that by using breadth first strategy we could produce the empty clause at third level itself.

4.3.2 Set of support strategy :

Set of support strategy [16] is normally used for proof finding programs. In this strategy at least one parent of each resolvent is selected from among the clauses resulting from the negation of the goal wff or from their descendants called the set of support clauses. Here,

Axiom - other clauses = Set of Support clauses

Resolution between two axioms or their factors is never allowed. This strategy is complete and more efficient [26] than breadth-first strategy. In set of support strategy, the growth of the clause set will be very slow and thus helps to moderate the usual combinatorial explosion. example will be given in the next section.

4.3.3 The unit preference strategy :

The unit preference strategy [27], modification of set of support strategy, essentially orders the clauses to be resolved by their length - i.e., by the number of literals they contains. Contradictions become apparent only when two 'unit' (one literal) clauses resolve together to produce the empty clause. Therefore, one might hope to discover a contradiction or empty clause

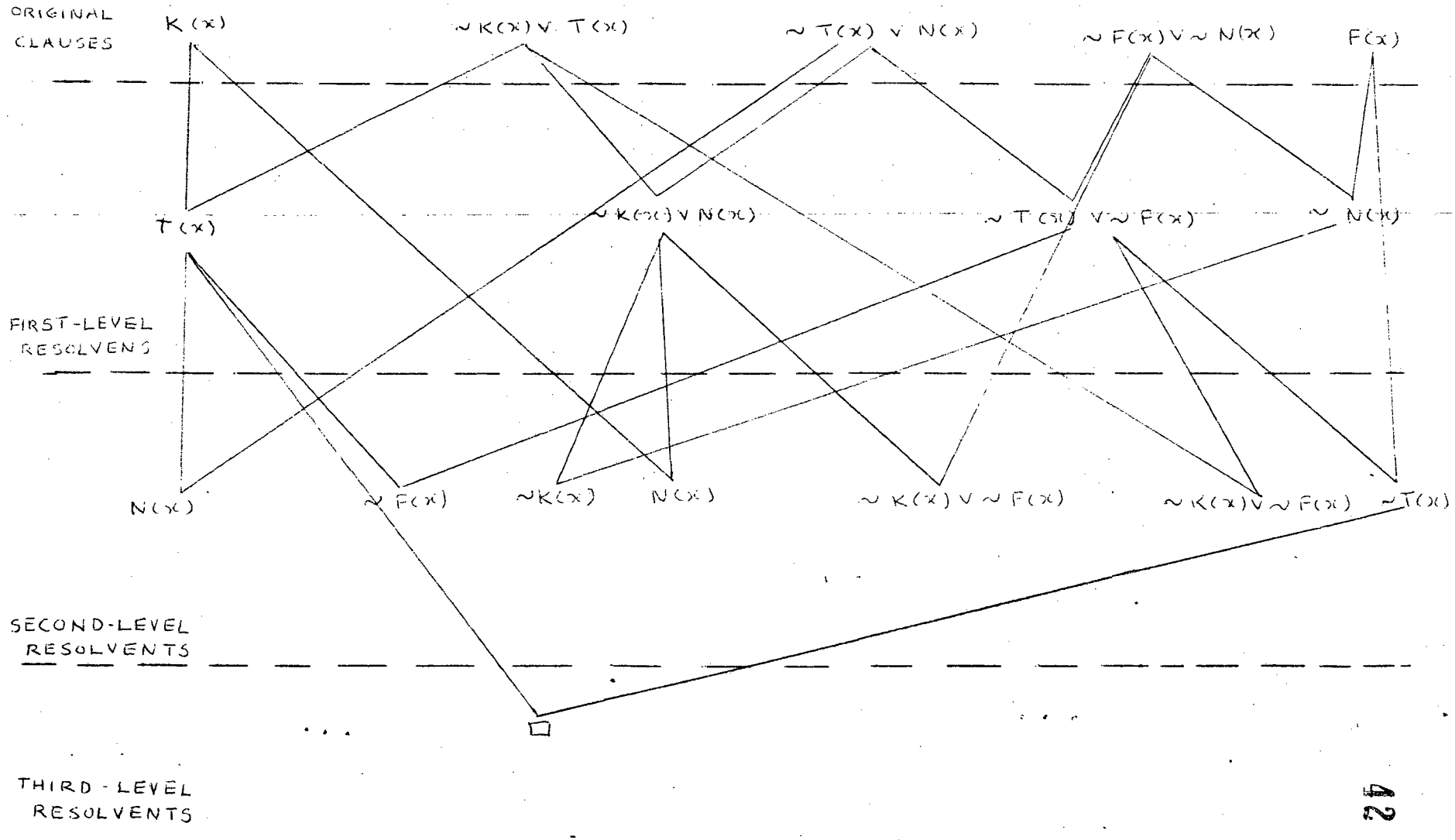

ORIGINAL CLAUSES
K(x)
~K(x) V T(x)
~T(x) V N(x)
~F(x) V ~N(x)
F(x)
T(x)
~K(x) V N(x)
~T(x) V ~F(x)
~N(x)
FIRST-LEVEL RESOLVENS
N(x)
~F(x)
~K(x)
N(x)
~K(x) V ~F(x)
~K(x) V ~F(x)
~T(x)
SECOND-LEVEL RESOLVENTS
THIRD-LEVEL RESOLVENTS

Fig. 4.2

in the least time by working first with the shortest clauses. This strategy says to first produce the shortest resolvent possible in which at least one of the 'parent' clauses is a unit. If no such resolutions are possible, attempt to produce the shortest possible resolvent or factor next. The strategy which combines the set of support and the unit preference strategy is more efficient than unit preference strategy.

Example-3 : As an example of proof using resolution, set of support strategy, and unit-preference strategy, let the axioms be

Axiom-1 $P(a)$

Axiom-2 $(\exists y)\ Q(y)$

Axiom-3 $P(a) \Rightarrow R(a)$

Axiom-4 $(\forall x)\ P(x) \wedge R(x) \Rightarrow Q(g(x))$

The axioms are converted to the following corresponding clauses :

clause-1	$P(a)$	from axiom-1
clause-2	$Q(b)$	from axiom-2
clause-3	$\sim P(a) \vee R(a)$	from axiom-3
clause-4	$\sim P(x) \vee \sim R(x) \vee Q(g(x))$	from axiom-4

The theorem to be proved from these axioms is

$(\exists x)\ Q(y(x))$.

The clause representing the negation of the theorem is

clause-5 $\sim Q(g(x))$ from negation of theorem

We show that this set of clauses is unsatisfiable. From the negation of the theorem, suppose clause 5 is selected as the only clause in the set of support. Following the unit-preference strategy, the first inference attempted is to resolve clause 5 with clause 1, a unit clause, which fails. Similarly, clause 5 does not resolve with clause 2. Then clause 5 fails to resolve with clause 3, a two clause (clause of two literals). Finally, clause 5 resolves with clause 4, producing

clause-6 $\sim P(x) \vee \sim R(x)$ from 4 and 5 ;

then clause 6 resolves with the unit clause 1, yielding

clause-7 $\sim R(a)$ from 1 and 6 ;

then

clause-8 $\sim P(a)$ from 3 and 7 ;

then

clause-9 contradiction from 1 and 8

We have an alternate proof if the unit-preference strategy is not used. The axioms and the negation of the theorem are the same as before. First, clause 6 can be produced from 4 and 5 as before.

clause-6 $\sim P(x) \vee \sim R(x)$ from 4 and 5

Then clause 6 and clause 3 resolve to produce

clause 7' $\sim P(a) \vee \sim P(x)$ from 3 and 6

By the other rule of inference, factoring, we have

clause-8' $P(a)$ from 7'

Finally,

clause-9 contradiction from 1 and 8'

completing the proof

As shown by the first proof of the above theorem, a proof is sometimes possible without factoring, but in general, factoring is necessary for completeness.

4.3.4 The Ancestry-Filtered form Strategy

A refutation graph is called to be in 'Ancestry - Filtered' (AF) form if each node in the graph corresponds to either of the following

(a) a base clause,

(b) an immediate descendant of a base clause

(c) an immediate descendant of two non-base clauses P and Q such that P is an ancestor of Q.

A base clause C in an AF-Form graph is called a 'top node' of every other node in the tree is either a base clause or a descendant of C. The theorem given below claims that an AF-Form refutation graph always exists for any unsatisfiable set of clauses. Therefore, a refinement strategy based on searching for AF-Form refutation graph is complete.

Theorem : Let G (NIL) be some refutation graph for an unsatisfiable set S of clauses and let C be a clause in S occuring in G(NIL). Then a refutation graph G'(NIL) in AF-form exists for S with C as a top node of G'(NIL).

If we denote $R_{AF}(S)$ the union of S with the set of all resolvents between pairs of S allowed by AF-form strategy, then by the above theorem we are guaranteed

that, if S is satisfiable, then there will exist some 'n' such that empty clause belongs to $R^n_{AF}(S)$.

The AF-form of the refutation graph is given in fig.4.3 where the clause D(x) is taken as base node.

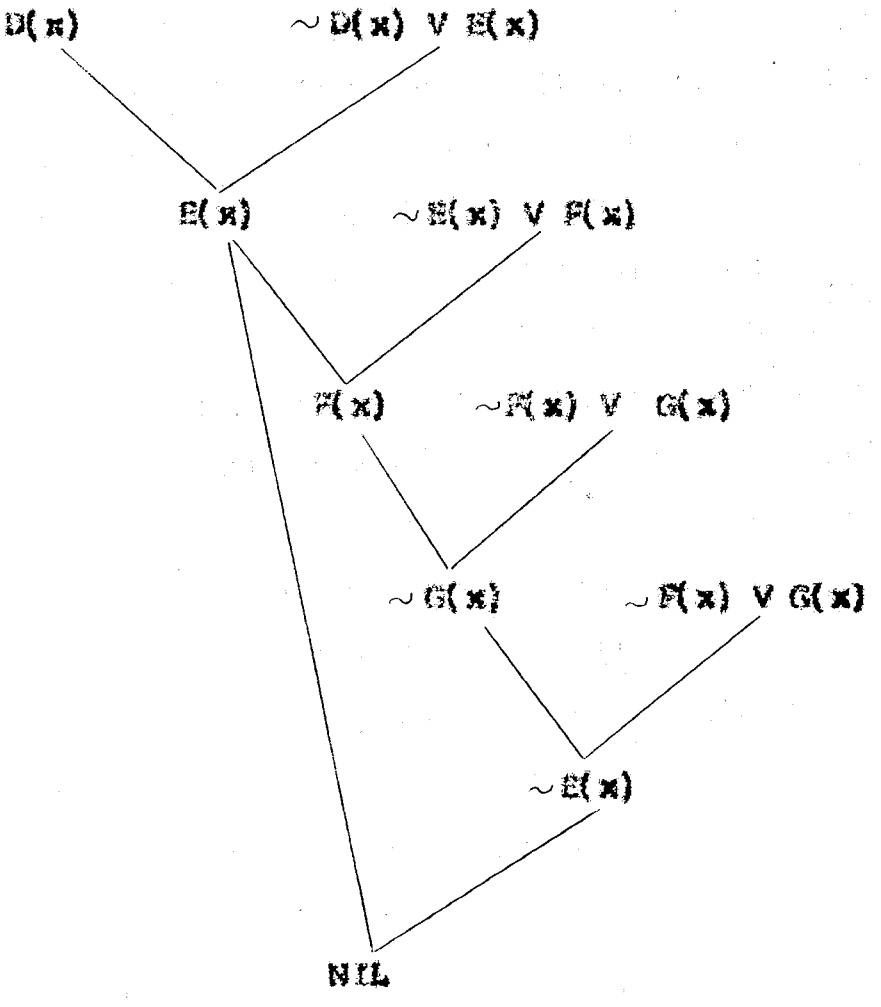


Fig. 4.3

4.3.3 Simplification strategies

Sometimes a set of clauses can be simplified by elimination of certain literals in the clauses or specific clauses. The best way is using resolution rule itself. These simplifications are such that the simplified set of clauses is unsatisfiable if and only if the original set is unsatisfiable. These simplification strategies help a lot in reducing the rate of growth of new clauses.

(a) Elimination of tautologies

Any clause containing a literal and its complement is called a tautology, may be eliminated, because any unsatisfiable set containing tautology is still unsatisfiable after removing it.

(b) Elimination of pure literal

A literal L is called 'pure' in a given set of clauses if it has no instance which is complementary to an instance of another literal in the set. Clauses consisting pure literals can be removed at any time and this removal preserves the unsatisfiability.

(c) Elimination by subsumption

By definition, a clause $\{L_i\}$ subsumes a clause $\{M_i\}$ if there exists a substitution θ such that $\{L_i\}\theta \subseteq \{M_i\}$. $\{M_i\}$ is called a subsumed clause. For example, $P(x)$ subsumes $P(a) \vee Q(y)$.

A clause in S, subsumed by another clause in S can be eliminated without effecting the unsatisfiability

of the remaining set. This strategy leads to substantial reductions in the number of resolutions to be performed for finding proof.

(d) Elimination by evaluation predicates

If a literal in a clause evaluates to T, the entire clause can be eliminated without effecting the unsatisfiability of the rest of the set. If a literal evaluates to F, then the occurance of just that literal in the clause can be eliminated. For example, $Q(x) \vee R(a) \vee G(3, 4)$ can be replaced by $Q(x) \vee R(a)$ as $G(3, 4)$ evaluates to F.

4.4 Strategies - Non-clausal theorem proving

4.4.1 Breadth - First Strategy :

example : 4 Show that the following set of Wffs is unsatisfiable.

$$S = \begin{cases} S(a) \Rightarrow T(y) \\ T(x) \Rightarrow Q(b) \\ \sim(G(y) \wedge Q(a)) \\ S(b) \wedge G(a) \end{cases}$$

We can write down wff $S(b) \wedge G(a)$ into two wffs, i.e., $S(b)$ and $G(a)$.

NC - Resolving the given wffs using breadth - first strategy, we get an empty set. The refutation graph is shown in Fig.4.4.

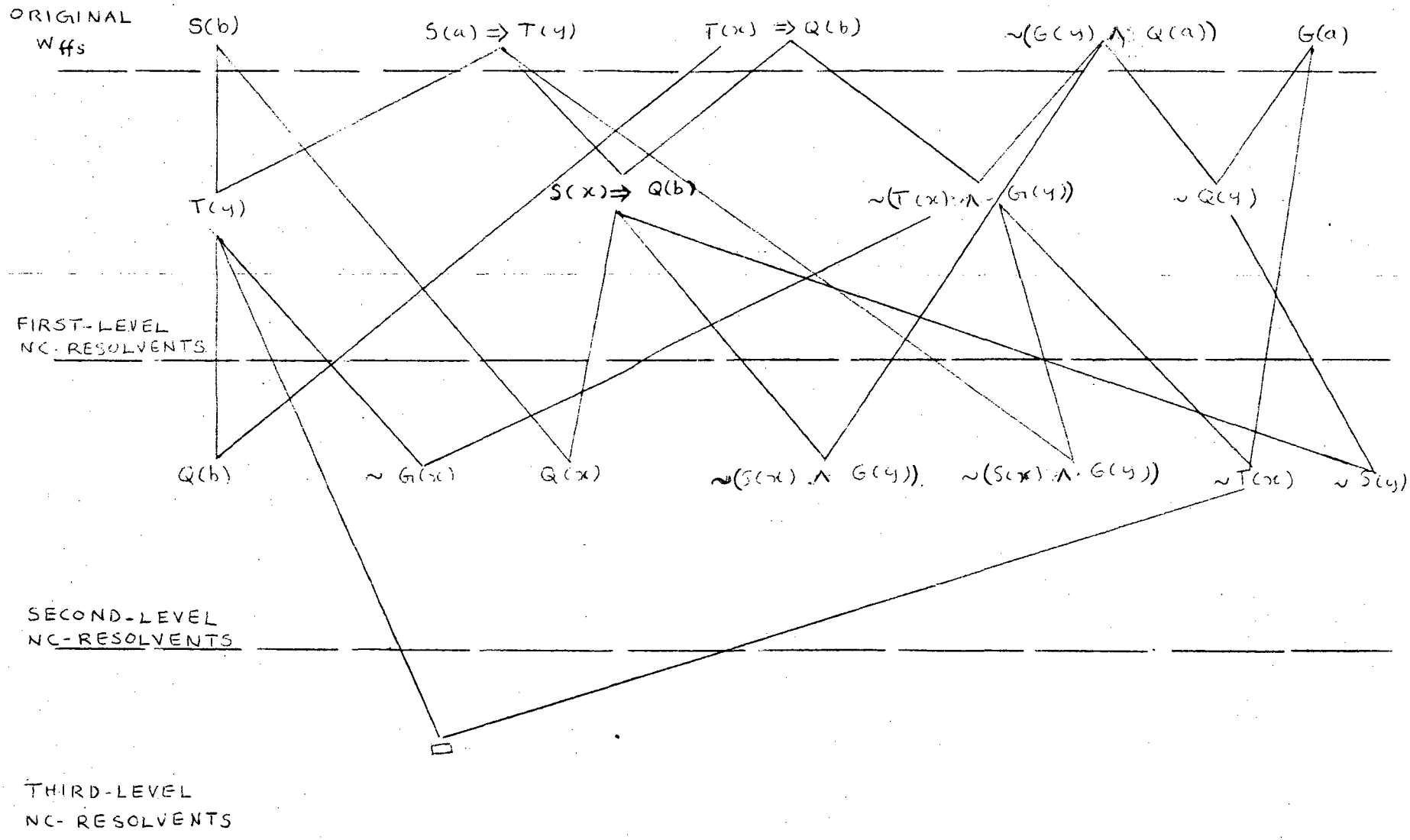

ORIGINAL
Wffs
S(b)
S(a) ⇒ T(y)
T(x) ⇒ Q(b)
~(G(y) ∧ Q(a))
G(a)
T(y)
S(x) ⇒ Q(b)
~(T(x) ∧ G(y))
~ Q(y)
FIRST-LEVEL
NC-RESOLVENTS
Q(b)
~ G(x)
Q(x)
~(S(x) ∧ G(y))
~(S(x) ∧ G(y))
~ T(x)
~ S(y)
SECOND-LEVEL
NC-RESOLVENTS
□
THIRD-LEVEL
NC-RESOLVENTS

Fig. 4.4

4.4.2 Set of Support and unit preference strategy :

Example : 5 Show that the following set of wffs is unsatisfiable using set of support and unit preference strategy.

$$S = \begin{array}{l} R(a) \wedge Q(b) \\ R(a) \Rightarrow S(a) \\ R(a) \wedge S(a) \Rightarrow Q(f(x)) \\ \sim Q(f(x)) \end{array}$$

Selecting $\sim Q(f(x))$ as the only Wff in the set of support and NC - Resolving the given wffs using the set of support and unit-preference strategy, we get the empty set. The refutation graph is shown in Fig.5.

4.4.3 Ancestry - filtered form strategy :

Example : 6 Show that the following set of Wffs is unsatisfiable using Ancestry-filtered strategy

$$S = \begin{cases} P(x) \Leftrightarrow Q(a) \\ \sim(P(x) \wedge Q(y)) \\ P(y) \vee Q(y) \end{cases}$$

NC - Resolving the above set of wffs using the Ancestry-Filtered Strategy, we get an empty set. The refutation graph is shown in Fig.6.

4.4.4 Simplification Strategies :

All the simplification strategies that are applicable to Clausal resolution are also applicable to NC - Resolution. Some more reduction rules are given in section (3.3.3).

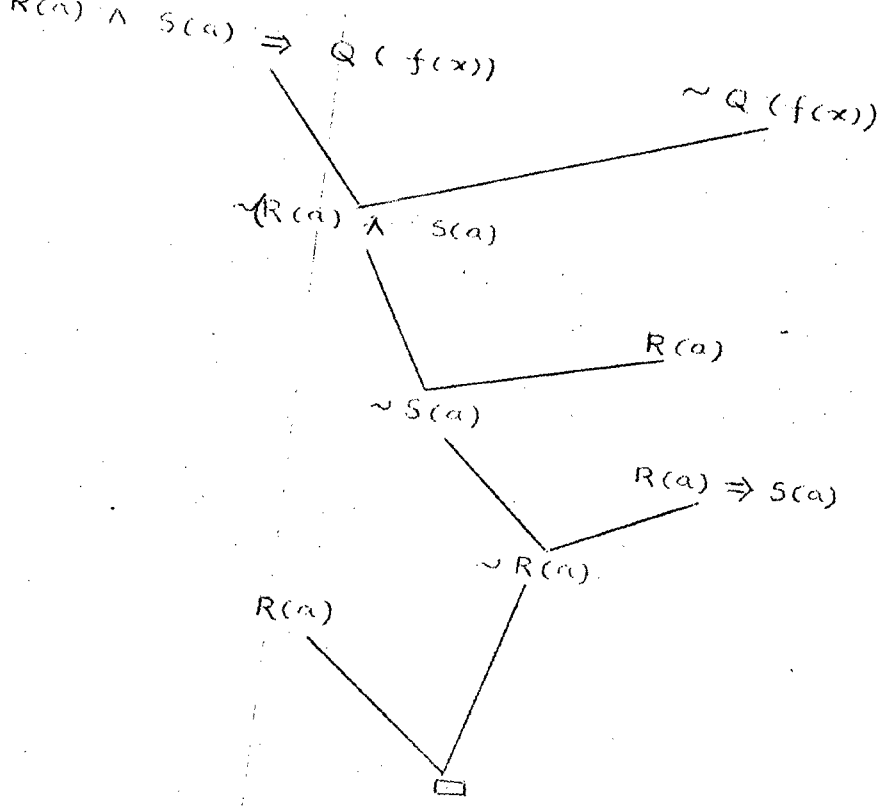


Fig. 5

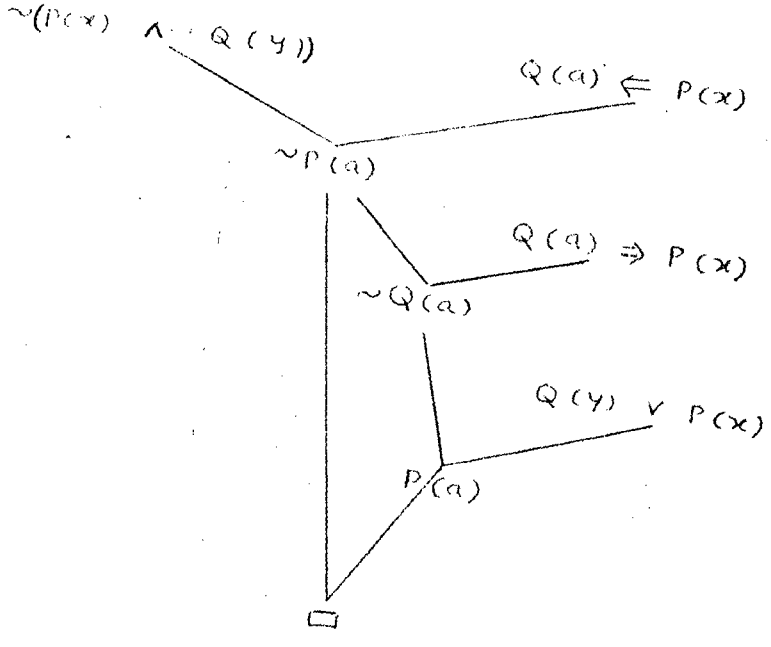


Fig. 6

4.5 Heuristic information

The blind search methods, whether breadth-first or depth-first, are exhaustive methods for finding path to a goal node. Though, these methods provide a solution to the path-finding problem, it is often impossible to use these methods because the search will expand too many nodes before a path is found. As there are always practical limits on the amount of time and storage available to expend on search, we must look for more efficient alternatives to blind search.

It is possible to state rules of thumb for many problems to help reduce the search. Any such technique used to speed up the search depends on special information about the problem. This information is called 'heuristic information' [3,8], serves to aid solving and the search procedures using it called 'heuristic search methods'.

A flexible way to use heuristic information is to use some criteria at every step. The measurement of these criteria are known as evaluation function. The purpose of an evaluation function is to provide a means for ranking those nodes that are candidates for expansion to determine which one is most likely to be on the best path to the goal. Evaluation functions have been based on a variety of ideas.

CHAPTER - V

Conclusion

New horizons opened in theorem proving after Robinson [19] developed Resolution principle. It was also shown to be highly efficient and very easily implemented on computers. Since then, many improvements of the resolution principle have been made. But all the resolution theorem proving procedures required the statements represented in first order predicate calculus to be in the clause form. Murry [13] developed a new version of the First order predicate calculus in which statements are not required to be in the clause form. Only condition is that the well-formed formulas (wffs) to be quantifier free and have distinct variables. This way all variables are implicitly universally quantified. In our study, we considered different strategies normally used for clausal theorem proving and could successfully apply most of the strategies we described, to Non-clausal theorem proving. Though most of the examples taken are simple and small, we believe that the application of strategies for simple problems may be useful to consider more complex problems.

Problems for research

Here we suggest some important research problems worthy of further work. Good solution to these problems

would certainly contribute to the field of artificial intelligence.

1. Automatic representation changes

A very important problem is that of creating a system that can automatically find considerable improvements in its representation of information. In a paper written by Amarel illustrating the importance of representation changes, discussed seven successively better representations for the missionary and cannibals puzzle. With each improved representation, the problem becomes easier. He also discussed the factors that make each change possible. Can such a process be automated ?

2. Automatic strategy changes

An important problem is devising a system that can automatically find substantial improvements in its problem solving strategies. Can a theorem prover be made to observe its axioms and performance and then find differences, metrices, indicators of relevance, or other means of successfully guiding search and selecting strategies ? Can a theorem prover be made to construct new strategies and/or prove new strategies to be better under particular conditions ?

3. Improved automatic theorem provers

In addition to the need for better theorem-proving formalisms there are innumerable possibilities for improvement within the resolution formalism.

One of the most important requirement for improving theorem provers is that of finding better proof strategies. It seems likely that no one fixed strategy will be best for all problems, so the key is finding flexible and suitable strategies. An important consideration in developing a new strategy is that the strategy should avoid redundancy. This can be done by two ways : (1) avoid the creation of new but unnecessary inferences and (2) create new inference but eliminate unnecessary ones. It seems especially difficult to change proof strategies and still avoid the creation of unnecessary inferences. It is easier to change proof strategies and eliminate unnecessary created inferences, although such a system will usually be less efficient.

REFERENCES

1. Barr, A., Feigenbaum and Edward, A., Hand Book of Artificial Intelligence, Vol.I, Pitman, London (1981).

2. Chandrasekaran, B., "Artificial Intelligence - The past decade", Advances in Computers, Vol.13 (1975), Academic Press, New York (1975).

3. Chang, C.L. and Lee, R.C., Symbolic Logic and Mechanical Theorem Proving, Academic Press, New York (1973).

4. Davis, M. and Putnam, H., "A computing procedure for quantification theory", J.ACM 7(3) (1960), pp.201-215.

5. Gilmore, PIC., "A proof method for quantification theory, its justification and realisation" IBM Journal of Research and Development (1960), pp. 28-35.

6. Glushkov, V.M., Introduction to Cybernetics, (Translated by George M. Kranc), Academic Press, New York (1966).

7. Harrison, M.C. and Rubin, N., "Another generalisation of resolution", J.ACM 25(3) (1978) pp.341-351.

8. Hunt, E.B., Artificial Intelligence, Academic Press New York (1975).

9. Joyner JR, W.H., "Resolution Strategies as Decition procedures", J.ACM 23(3) (1976), pp.

10. Kowalski, R. and Kuehner, D., "Linear Resolution with Selection function", Artificial Intelligence 2 (1972), pp.227-260.

11. LoveLand, D.W., Automatic theorem proving : A logical basis, North-Holland, Amsterdam (1977).

12. Luckham, D., "The Resolution Principle in Theorem-Proving", (Ed.) N. Collins and D. Michie, Machine Intelligence, Vol.1, (1967), pp. 47-61.

13. Murry, N.V., "Completely Non-clausal Theorem Proving", Artificial Intelligence 18 (1982), pp.67-85.

14. Newell, A., "Artificial Intelligence and the concept of Mind", (Ed.) Roger C. Schank and Kenneth Mark Colby, Computer Models of Thought and Language, W.H. Freeman and Company, San Francisco (1973), pp.1-60.

15. Nilsson, N.J., Principles of Artificial Intelligence, Springer-Verlag, New York, (1982).

16. Nilsson, N.J., Problem-solving methods in Artificial Intelligence, Mc Graw-Hill (1971).

17. Raphael, B., The Thinking Computer-Mind inside matter, W.H. Freeman and company, San Francisco (1976).

18. Robinson, J.A., Logic : Form and Function, Edinburgh University Press, Edinburgh (1979).

19. Robinson, J.A., "A machine oriented logic based on the resolution principle", J.ACM 12(1) (1965), pp.23-41.

20. Sciko1, S., "Variable range restriction in resolution Theorem proving", (ed.) E.W. Elcock and Donald Michie, Machine Intelligence Vol.8, Ellis Horwood Ltd, Chichester (1977), pp.73-85.

21. Simon, H.A., "What the knower knows : Alternative strategies for problem solving tasks", Human and Artificial Intelligence (Ed.), Friedhard Klix, North-Holland, Amsterdam (1979), pp.89-101.

22. Slagle, J.R., Artificial Intelligence : The Heuristic Programming approach, McGraw-Hill (1971).

23. Turing, A.M., "Computing Machinery and Intelligence", Computers and Thought, (Ed.) Edward A. Feigenbaum and Julian Feldman, McGraw-Hill (1963), pp.11-35.

24. Winston, H.P., (Ed.) Artificial Intelligence : An MIT Perspective, MIT Press, Mass. (1979).

25. Winston, H.P., Artificial Intelligence, Addison-Wesley (1977).

26. Wos, L., Robinson, G., and Carson, D., "Efficiency and completeness of set of support strategy in S theorem proving", J.ACM 12(4) (1965), pp.536-541.

27. Wos.L., Robinson, G., and Carson, D., "The unit preference strategy in theorem proving", Proc. AFIPS 1964 FJCC, Vol.26-Part II (1964), pp.615-621.