A COMPLETENESS THEOREM AND A COMPUTER PROGRAM FOR FINDING THEOREMS DERIVABLE FROM GIVEN AXIOMS

LEE CHAR-TUNG

DEGREE DATE: 1967

68-10,359

LEE, Char-tung, 1939-
A COMPLETENESS THEOREM AND A COMPUTER PROGRAM FOR FINDING THEOREMS DERIVABLE FROM GIVEN AXIOMS.

University of California, Berkeley, Ph.D., 1967
Engineering, electrical

University Microfilms, Inc., Ann Arbor, Michigan

A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms

By

Char-tung Lee

B.S. (National Taiwan University) 1961
M.S. (University of California) 1963

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Engineering

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Approved:

Jones R. Slagle

Volker Strassen

Eugene Wong

Committee in Charge

Degree conferred DEC 16 1967
Date

ABSTRACT

This thesis concerns itself with the problem of finding interesting theorems derivable from a set of axioms within the first order predicate calculus. The author uses an inference rule called the resolution principle and proved a completeness theorem of the resolution principle for finding theorems.

A program was written and tested on 45 theorems. The program generated 41 of them. The author draws many conclusions from the experiments with this program. He also makes some suggestions for future work along this line.

ACKNOWLEDGMENTS

The author wishes to show his hearty gratitude to Dr. James R. Slagle for his patience and guidance without which this thesis would not have been possible. It was due to a course taught by Dr. James Slagle that the author began to be interested in the field of artificial intelligence. He will always remember his long discussions with Dr. Slagle; these discussions often lasted longer than one hour. In a country where everyone seems to be so busy, the author is indeed extremely fortunate to have Dr. Slagle as his advisor.

The author would also like to thank Professor John McCarthy of Stanford University who allowed the author to use the PDP-6 time sharing system and Mr. Steve Russell for his guidance on how to use the PDP-6 system.

This work was supported in part by the National Science Foundation under Grant No. GK-716.

TABLE OF CONTENTS

	Page
ABSTRACT	i
CHAPTER 1. INTRODUCTION	1
Section 1: Artificial Intelligence and Reasoning Forward	1
Section 2: The Nature of the Progrm	2
Section 3: Experimental Results and Its Findings	4
Section 4: Historical Background of Theorem Proving by Machine	4
CHAPTER 2. THE FORMAL AXIOMATIC THEORY AND THE RESOLUTION PRINCIPLE	7
Section 1: Preliminare Definitions	7
Section 2: Satisfiability, Unsatisfiability in the First Order Axiomatic Theory	9
Section 3: The Functional Normal Form of a Prenex Nofmal Formula	10
Section 4: The Negation of a Clause	12
Section 5: The Resolution Principle	15
CHAPTER 3. THE COMPLETENESS THEOREM OF THE RESOLUTION PRINCIPLE FOR THEOREM FINDING	17
Section 1: The Effectiveness of the Resolution Principle	17
Section 2: The Herbrand Theorem	19
Section 3: The Proof of the Completeness Theorem	21
CHAPTER 4. EXPERIMENTS AND FINDINGS WITH DIFFERENT VERSIONS OF THE PROBRAM	24
Section 1: The Sub-clause Principle	24
Section 2: The Idealized Program	26
Section 3: PG1 - The Idealized Program	26

Section 4: PG2 - No Checking of Triviality 30
Section 5: "The Unit Resolution Principle" 31
Section 6: PG3 - Employing Unit Resolution - Principle and Bound of a Clause 32
Section 7: The "Axiom Resolution Strategy" and PG4 33
Section 8: PG5 - Adding Interesting Theorems to the Axioms 37
Section 9: PG6 and Its Experiments 38
CHAPTER 5. THE TECHNIQUE TO SINGLE OUT INTERESTING THEOREMS 39
Section 1: The Recognition of an Important Theorem 39
Section 2: The "Unit Condensation Procedure" 40
Section 3: The Substitution Procedure 42
Section 4: An Example 44
CHAPTER 6. CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH 48
Section 1: Conclusions 48
Section 2: Suggestions for Future Research 50
BIBLIOGRAPHY 52

CHAPTER 1

INTRODUCTION

In this chapter, the author describes briefly the nature, the principle and the performance of a program that finds theorems. A historical background of this research is also given.

Section 1: Artificial Intelligence and Reasoning Forward

Although it is difficult to define the word "intelligence" precisely, it is generally agreed that one's intelligence is not measured by how much he knows, but rather by how capable he is to develop new ideas. For example, consider a president who is in the process of making an important decision. His staff members have already furnished him all the information he needs and made suggestions as to what kind of actions to take. As a good president, he should be able to perceive the consequences of these actions, not only those obvious ones, but also those that ordinary people can not easily see. In other words, an intelligent person can look much farther ahead.

It is surprising to note that this kind of reasoning forward, like predicting consequences from a set of suggestions or deriving theorems from a set of axioms, is an intellectually difficult problem. A mathematics student would be considered a genius if he could derive three or four important theorems out of the basic group theory axioms, say fifteen minutes after he is taught these axioms.

The author, however, believes that to find important theorems of a set of axioms, does not require ingenuity. Thanks to the invention of high speed digital calculating machines and the development of mathematical logic, the whole process can be mechanized.

To show the marvelous combined capability of mathematical logic and digital computers, the author has written a computer program whose input is a set of mathematical axioms and the output is a set of theorems. In this chapter, the author will describe the principle and nature of this program. In the second chapter, the author will introduce the first order predicate calculus and an inference rule called the "resolution principle." In the third chapter, the author presents his completeness theorem of the resolution principle for theorem finding. In chapter four the author discusses experiments and findings with different versions of the program and in chapter five, the author describes the techniques to single out interesting theorems. The last chapter, chapter 6, is devoted to the conclusions and a discussion of future problems.

Section 2: The Nature of the Program

Since not every reader is familiar with digital computers, this project will be discussed in concrete terms. First, suppose we have a set of I.B.M. cards containing punches representing (in a suitable notation) a certain set of mathematical axioms. Now feed these cards into a digital computer that contains the author's program. In fifteen minutes, the computer will print out theorems of these axioms. Furthermore, this program will print out those which it considers to be important.

The reader may imagine himself to be a teacher who is teaching a student, in this case a digital computer, some axioms of a certain mathematical system. He teaches the machine by typing in these axioms through a teletypewriter. As soon as he finishes typing the last symbol, he gives a signal to tell the machine to think a little bit about these axioms.

The response is almost immediate. On the same typewriter, the machine is typing back theorems that can be deduced out of these axioms. At first, you may laugh at the theorems since they look so obvious to you. As time goes on, you will be bewildered and begin to notice that the theorems are becoming more and more difficult. Finally the theorems are so unexpected and interesting that you begin to wonder if you have already created a monster, not because of his physical power, but for his highly developed intelligence.

For readers who might like to know more about this program, the following questions are designed to clarify some of their doubts.

(1) What computer language does this program use?

This program uses a symbolic language, called LISP [10], which is proved to be very powerful in symbol manipulation.

(2) What form of logic does this program use?

The program uses the first order predicate calculus as a logical basis.

(3) What inference rule does the program use?

The inference rule used in this program is called "resolution principle" and was proposed by J. Robinson [17].

(4) Exactly what does the program do?

This program does two things: generating theorems and picking out the interesting ones.

(5) Does the author believe that this program behaves more intelligently than human beings with ordinary intelligence?

Yes, definitely. No human being with ordinary intelligence can figure out as many theorems as this program can, if given the same amount of time.

(6) What computer was used? How much memory core was used?

This program was tested on the PDP-6 time sharing system of the Computer Science Department of Stanford University. 24 K of core memory was used.

Section 3: Experimental Results and Its Findings

This program was tested on forty five theorems from Lederman's book [9] on group theory. Of these forty five theorems, this program only failed to generate four of them. It also successfully picked out all of the forty one theorems, among others, as interesting ones.

Of these four theorems, the common characteristic is that they require more than fourteen axioms. The limitation of memory spaces is concluded to be the greatest obstacle of theorem finding. The author does not believe any small improvement in this respect can improve the performance of the machine significantly. Some drastic changes are needed in order that this program can handle a large number of axioms.

Section 4: Historical Background of Theorem Proving by Machine

It was in 1930 that Hilbert started his very ambitious project to create a formal axiomatic theory (see chapter 2) which is consistent and complete in the sense that every formula is theoretically decidable. It was then found out that the consistency of a major part of classical mathematics can be reduced to that of number theory. Unfortunately, this project had to come to a halt because of the famous Gödel's proof. Gödel stated that it is impossible to prove the consistency of a formal theory which includes number theory by constructing methods "formalizable" within itself. The result of Gödel's theory is even more disastrous than it appears. It shows that so long as the system of number theory is

consistent, there is always a true statement of number theory that is not provable. One might think a way out by adjoining the unprovable true statement to the axioms of number theory. "This simply invites more trouble," said Gödel, since the extended formal system must still necessarily be incomplete and there will still be an undecidable statement.

The final blow to the dream of mechanizing mathematics by machine came in 1936 as Church proved that for any machine, there exists a statement in the first order predicate calculus for which the machine can not decide whether it is true or not. This theorem shows the machine can not do everything in mathematics. But, as evidenced by recent developments, machines have been shown to be able to take care of a great deal of mathematical manipulation.

Traditionally, researchers in the field of artificial intelligence have been interested in solving mathematical problems [1,2,3,4,5,6,7, 11,12,13,14,15,17,18,19,20,21,22,23,24,25,26,28,29,30]. In 1956, Newell, Shaw and Simon wrote the famous program called "Logic Theorist," [12]. The aim of this research is not to find methods which guarantee solutions in propositional calculus, but rather to find out how a mathematician would prove a theorem in this elementary logic. Thus it is not surprising that this program is quite human like but not too effective, since it only proved seventy percent of the theorems in Chapter II in Principia Mathematica [27]. Actually, it is well known that propositional calculus is decidable and there should be no trouble at all for a computer to prove a hundred percent of the theorems in propositional calculus. In 1959, Gelernter [4,5] succeeded in programming a computer to prove theorems in geometry. His program, called "Geometry Theorem Proving Machines,"

can do a job as well as a high school student. In 1962, J. Slagle [20,21] demonstrated that a computer can solve elementary integration problems at the level of a freshman student.

All of the programs mentioned above are, in certain senses, for special purposes. Since most branches of mathematics can be expressed in the first order predicate calculus, proving mathematical theorems is equivalent to testing the consistency of a set of formulas in the first order predicate calculus. (This will be discussed in the next chapter.)

The year of 1960 was perhaps the most productive year in this field of research. Prawitz [14,15], Gilmore [6,7], Wang [24,25,26] and Davis [1,2] all proposed different proof procedures, but none of them seemed to be very effective.

In 1965, Robinson [17] proposed his excellent "resolution principle" and really made a dramatic and significant contribution to theorem proving. His inference rule is not only complete, but also psychologically understandable by human beings. Various strategies concerning resolution principle have been proposed ever since then [11,18,19,23,28, 29,30].

CHAPTER 2

THE FORMAL AXIOMATIC THEORY AND THE RESOLUTION PRINCIPLE

In this chapter, the author describes the so-called "formal axiomatic theory" and an inference rule called the resolution principle. The reader should have at least some background in propositional calculus or Boolean algebra.

Section 1: Preliminary Definitions

In order to achieve absolute precision to the presentation of mathematical theory, symbols are used extensively. A formal axiomatic theory thus carries symbolization to its ultimate by suppressing all words in favor of symbols. In this project, we use the first order predicate calculus as its logical basis.

Before precisely defining the first order predicate calculus, it will be appropriate for us to introduce it informally. In the first order predicate calculus, we express a sentence (or a statement) by explicitly expressing the relationship between its subject and its predicate. This is done by using predicate symbols. For example, P( ) may represent "is a real number." Then P(5) means that "5 is a real number." The predicate symbol does not necessarily have only one argument as in the above example. The sentence "x+y = z" can be expressed as P(x,y,z).

But the use of predicates alone is not adequate. We do not know whether P(x) means "for all x, P(x) holds," or "there exists an x, such that P(x) holds." To eliminate the confusion, we introduce "quantifiers." The idea of "for all x" is represented by the symbol (x). Thus (x)P(x) means that "for all x, P(x) holds."

The idea of "there exists an x" is symbolized by (EX). Thus (EX)P(x) means "there exists an x, such that P(x) holds."

To be more precise, in a formal axiomatic theory, formulas are certain strings of symbols. The primitive symbols are the following:

1. Punctuation symbols:

   , ( )

2. Logical symbols:

   V, &, ∿, E

3. Symbols representing variables:

   $x, y, z, u, v, r, s, t, x_1, y_1, z_1, \ldots$, etc.

4. Symbols representing functions:

   $a, b, c, d, e, f, g, h, i, j, k, a_1, b_1, c_1, \ldots$, etc.

   where every symbol represents a function of degree n, $n \geq 0$.

   If n = 0, the function symbol represents an individual constant.

5. Symbols representing predicates:

   $P, Q, M, P_1, Q_1, M_1, \ldots$, etc.

   where every symbol represents a predicate with n arguments, $n \geq 0$.

We can now define the "well formed formula" by first defining "terms" and "atomic formulas."

Definition: Terms. A variable is a term and a string of symbols consisting of a function symbol of degree $n \geq 0$ followed by n terms is a term.

Definition: Atomic formula. The expression $P_i(r_1, r_2, \ldots, r_n)$ is an atomic formula if P_i is a predicate symbol and $r_1, r_2, \ldots, r_n$ are terms. Again, $r_1, r_2, \ldots, r_n$ are called the arguments of P_i.

Definition: Well formed formulas: (w.f.f.)

(a) An atomic formula is a w.f.f.

(b) If R is a w.f.f., so is ∿R, (x_i)R and (EX_i)R.

(c) If R and S are w.f.f.'s, then so are (R&S) and (RVS).

Definition: A formula is said to be quantifier free if it does not contain any of (x_i) or (EX_i).

Definition: A formula is a prenex normal formula, or is said to be in prenex normal form if it starts with a sequence of (x_i) and (EX_i) in which no variable occurs more than twice (called the prefix) and the sequence is followed by a quantifier free w.f.f. (called matrix).

Definition: A literal is an atomic formula or $\sim R$, where R is an atomic formula.

Definition: A clause is a disjunction of literals in which no atomic formula occurs twice.

Definition: A prenex conjunctive formula is a prenex normal formula whose matrix is in conjunctive normal form.

Section 2: Satisfiability, Unsatisfiability in the First Order Axiomatic Theory

In order to give the sentence "T is a theorem of $A_1, A_2, \ldots, A_n$" a precise definition, one has to define "satisfiability" precisely. It is assumed that for every w.f.f. R, there is a non-empty set of elements, called a universe U. An assignment of values over U to each function symbol and predicate symbol occurring in R consists of the following:

(1) To each function symbol of degree n, we assign a function of degree n, ranging over U, whose values are also in U.

(2) To each predicate symbol with n arguments, we assign a function of degree n ranging over U, whose values are T or F.

Let $R(x_1, x_2, \ldots, x_k)$ be a w.f.f. Given any assignment of R over a universe U, the value T or F will be assigned to $R(t_1, t_2, \ldots, t_k)$

for each ordered k-tuplet $(t_1, t_2, \ldots, t_k)$ of elements of U.

A w.f.f. is called "satisfiable" if there is some assignment of R over some universe U, such that R is assigned T.

R is "unsatisfiable" if it is not satisfiable.

We can now define "theorem" in terms of "unsatisfiable."

Definition: Suppose we are given a set of w.f.f.'s $A_1, A_2, \ldots, A_n$, we say that a w.f.f. R is a theorem, or a logical consequence of $A_1\&A_2\ldots\&A_n$ if $A_1\&A_2\ldots\&A_n\&\sim R$ is unsatisfiable.

Definition: If w.f.f. B is a logical consequence of w.f.f. A, we say that A implies B.

Definition: A w.f.f. A is said to be equivalent to a w.f.f. B if A implies B and B implies A.

Since there is an algorithm to obtain a prenex normal form of any w.f.f., we may assume that $A_1\&A_2\ldots\&A_n$ is a prenex normal formula. Furthermore, we can even assume that the matrix is in conjunctive normal form. For both of these assumptions, one can consult Hilbert & Ackerman [8].

However, the prefix of the formula we mentioned above still contains existential quantifiers. For reasons that will appear clear later, the existential quantifiers should be eliminated. This will be discussed in the following section.

Section 3: The Functional Normal Form of a Prenex Normal Formula

Definition: Functional normal form of a prenex normal formula. (This is a notation used by Quine. See [16].) Let R be in prenex normal form. The functional normal form of R is obtained by deleting all existential quantifiers and replacing the occurrences of their variables by function symbols according to the following scheme: if the universal quantifiers to the

left of (EX_{ij}) are $(x_{q_1})(x_{q_2})\ldots(x_{q_p})$, then (EX_{ij}) is dropped and X_{ij} in the matrix of R is replaced by $f_{ij}(x_{q_1},x_{q_2},\ldots,x_{q_p})$. The w.f.f. R' thus obtained has only universal quantifiers in the prefix and is called the functional normal form of R.

That we can delete the existential quantifiers of a w.f.f. without affecting its satisfiability can be seen by the following example: Suppose the given prenex formula is

$$(x_1)(EX_2)(EX_3)(X_4)R(x_1,x_2,x_3,x_4) \tag{1}$$

where the matrix $R(x_1,x_2,x_3,x_4)$ is quantifier free. Then the formula is satisfiable if

$$(x_1)(x_4)R(x_1,f_2(x_1),f_3(x_1),(x_4) \tag{2}$$

is satisfiable. This is because (2) implies (1). Furthermore, if (1) is true in some universe U (under some assignment), then there are functions f_2 and f_3 over U such that (2) is true in U under the same assignment. Thus if (1) is satisfiable, so is (2).

Since we can always obtain the prenex conjunctive normal form of a w.f.f. and since we can always delete existential quantifiers, we can assume that the axioms are given as a set of quantifier free clauses. For example, the closure axiom in group theory can be put in the following form:

$$P(x_1,x_2,f(x_1,x_2)) \quad .$$

The reader should bear in mind that every variable in the above formula is universally quantified. Thus the real formula should be

$$(x_1)(x_2)P(x_1,x_2,f_3(x_1,x_2)) \quad .$$

Section 4: The Negation of a Clause

The ability to delete the existential quantifiers of a prenex normal formula simplifies our problem. Instead of representing our axioms as a set of w.f.f.'s, we may assume that our axioms are represented as a conjunction of clauses. By the same token, we can assume our theorem is also a clause.

However, this transformation does cause some complications. The following example shows some of these complications:

We have two axioms as below:

$$(x_1)(x_2)(\sim P(x_1,x_2) \vee P(x_2,x_1)) \tag{3}$$

$$(x_1)(\exists x_2)P(x_1,x_2) \tag{4}$$

The clauses corresponding to (3) and (4) are

$$\sim P(x_1,x_2) \vee P(x_2,x_1) \tag{5}$$

$$P(x_1,f_2(x_1)) \tag{6}$$

One logical consequence of (5) and (6) is

$$P(f_2(x_1),x_1) \tag{7}$$

One logical consequence of (3) and (4) is

$$(x_1)(\exists x_2)P(x_2,x_1) \tag{8}$$

It is sometimes rather misleading to think that (7) is equivalent to (8). A careful examination of these two formulas will show that (7) implies (8), but (8) does not imply (7).

To obtain the functional normal form of (8), we must necessarily avoid using old function symbols which occurred in (5) and (6). We can use a new symbol, say $g(x_1)$ to replace x_2 in (8). So the clause corresponding

to (8) is

$$P(g(x_1),x_1) \tag{9}$$

To obtain the negation of (7), we first have to attach universal quantifiers to it. Thus (7) becomes

$$(x_1)P(f_2(x_1),x_1) \quad . \tag{10}$$

Then we find the negation of (10)

$$(EX_1)\sim P(f_2(x_1),x_1) \tag{11}$$

The elimination of existential quantifiers in (11) leads to

$$\sim P(f_2(a),a) \tag{12}$$

which is the desired negation of (7).

To obtain the negation of (9), we must remember that it is obtained by deleting existential quantifiers from formula (8). So we first obtain the negation of (8)

$$(EX_1)(X_2)\sim P(X_2,X_1) \tag{13}$$

The deletion of (EX_1) from (13) leads to the negation of (8). It is

$$\sim P(x,b) \tag{14}$$

Remark: The new function symbols of a clause must be deleted before obtaining the negation of a clause.

A problem arises from the above discussion. That is, given a clause with new function symbols, can we always find the negation of it. The problem is actually whether we can remove these new function symbols or not.

Unfortunately, there exist clauses whose new function symbols can not be removed. For example, consider the following two clauses:

$$P(f(x_1),f(x_2)) \tag{15}$$

$$P(f(x_1),g(x_2)) \tag{16}$$

where f and g are supposed to be new function symbols. It is obvious that we can not delete these new function symbols.

The following algorithm serves to delete the new function symbols occurring in a clause. If there is no way to delete them, this algorithm will also establish this fact.

Algorithm: Given a clause as input, proceed as follows:

Step 1: Set i = 1. Set P = the null sequence. Go to Step 2.

Step 2: Let $f_{i1},f_{i2},\ldots,f_{im}$ be the new function symbols with the lease number of variables. If for some f_{ij}, f_{ij} occurs more than once and it contains different sets of variables, stop. Otherwise, go to Step 3.

Step 3: If $f_{i1},f_{i2},\ldots,f_{im}$ all have the same set of variables, go to Step 4. Otherwise, stop.

Step 4: For all j, replace $f_{ij}(x_{i1},x_{i2},\ldots,x_{in})$ by y_{ij} in the clause. Go to Step 5.

Step 5: Set $P_i = (x_{i1})(x_{i2})\ldots(x_{in})(Ey_{i1})(Ey_{i2})\ldots(Ey_{im})$. Set $P = P_iUP$. Go to Step 6.

Step 6: If there are no more new function symbols in the formula, stop. Otherwise, set i = i+1. Go to Step 2.

That this process will stop in a finite number of steps can be seen from the fact that this clause contains only a finite number of new function symbols.

If this algorithm stops at Step 2 or Step 3, it means that it is impossible to replace the new function symbols in this clause. If the algorithm stops at Step 6, the "P" thus obtained is the prefix and the clause with all new function symbols removed can be considered as the matrix. Thus

we have obtained the desired prenex normal formula.

The above discussion shows that we do not know how to negate all the clauses containing new function symbols. From now on, we restrict ourselves to all the clauses that we can negate.

Section 5: The Resolution Principle

The resolution principle is an inference rule which can be used to deduce one logical consequence out of two clauses. The deduced logical consequence is also a clause. For precise definition of resolution principle, the reader should consult [17]. Right now, it suffices to say that the resolution principle involves two ideas:

(a) The syllogism principle of propositional calculus. Thus from pVq and $\sim pVr$, where p, q and r are atomic formulas, one can obtain qVr.

(b) Instantiation principle of the predicate calculus. From $P(v_1,v_2,\ldots,v_m)$, one can infer $P(t_1,t_2,\ldots,t_m)$ where each v_i is a variable and each t_i is a term. For the definition of "term" consult Section 1 of this chapter.

Given two clauses, the resolvent, if any, of a literal ℓ_1 or one clause and a literal ℓ_2 in another clause is a logical consequence of these two clauses. To obtain this resolvent, one proceeds as follows:

(a) Rename variables so that all variables in one clause are distinct from all the variables in the other clause.

(b) Find the minimal substitution, if any, which makes the literals identical but opposite in sign.

(c) Make these substitutions throughout both clauses.

(d) Cancel the literals ℓ_1 and ℓ_2 which were now made identical but opposite in sign. The resolvent is the disjunction of the literals

remaining in the second clause.

Although the resolution principle was used to prove theorems, it should be obvious that it can also be used to deduce theorems.

CHAPTER 3

THE COMPLETENESS THEOREM OF THE RESOLUTION PRINCIPLE FOR THEOREM FINDING

In this chapter, the author proves the completeness theorem for the resolution principle for theorem finding. An important theorem, called the Herbrand Theorem, is first introduced.

Section 1: The Effectiveness of the Resolution Principle

As we discussed in the last chapter, we can always transform a set of w.f.f.'s into a set of clauses. Therefore, from now on, we assume that we are given a set of clauses as axioms. Furthermore, we say that the theorems are also in the form of clauses. The reader should note that no generality is lost in doing so.

It is perhaps legitimate to ask the following question: "Given a set of clauses, can the resolution principle deduce all the theorems?" To answer this question, the author proved a "completeness theorem" of the resolution principle in finding theorems which will be discussed in the next chapter. More definitions and theorems have to be introduced before the final formal proof. Meanwhile we will just present this theorem together with one simple example to help the readers understand the true meaning and significance of this theorem.

Definition: If S is any set of clauses, then the resolution of S, denoted by R(S), is the set consisting of the members of S together with all the resolvents of the pairs of members of S.

Definition: If S is any set of clauses, then the n-th resolution of S, denoted by $R^n(S)$, is defined for $n \geq 0$ as follows: $R^0(S) = S$, and for $n \geq 0$, $R^{n+1}(S) = R(R^n(S))$.

The completeness theorem which the author proved can be stated as follows:

Theorem: Given a set S of clauses, if a clause C is a logical consequence of S, then for some $n \geq 0$, there exists a clause $T \in R^n(S)$, such that T implies C.

The following example serves the purpose of showing how the resolution principle works in deducing theorems.

Example: We are given clauses C_1, C_2 and C_3.

C_1: $\sim a \vee b \vee c$

C_2: $\sim a \vee \sim b$

C_3: $a \vee c$

Using resolution principle, we can deduce the following theorems:

T_1: $\sim a \vee c$

T_2: $\sim b \vee c$

T_3: c .

However, it is easy to see that both T_4 and T_5 are implied by T_3. In fact, T_1 and T_2 although deduced by resolution principle, are also implied by T_3.

We can therefore define the triviality of a theorem as follows:

Definition: A theorem is trivial if it is implied by another theorem, but is not equivalent to it. A theorem is non-trivial if it is not trivial.

In the above example, T_3 is the only non-trivial theorem and the completeness theorem guarantees that we can obtain T_3 by using resolution principle.

In short, the resolution principle does not generate all the theorems out of a set of clauses. But it does generate all the non-trivial theorems of a set of clauses. Note that not all the theorems deduced by resolution principle are non-trivial. However, these trivial theorms can be checked out quite easily.

The proof of this theorem is quite involved. First, we need the concept of "Herbrand Universe" which will be introduced in the next section.

Given a set of clauses S, we are interested in the set which contains all the clauses that can be generated by using resolution principle. Let us denote this set by Q(S). In propositional calculus, Q(S) is easy to find, since there exists an n, $n \geq 0$, such that the following sequence

$$S \subseteq R(S) \subseteq R^2(S) \ldots R^n(s)$$

terminates. Q(S) is thus the particular $R^n(S)$ which is equal to $R^{n+1}(S)$. However, in the first order predicate calculus, n does not necessarily exist such that $R^{n+1}(S) = R^n(S)$, for otherwise, we would have a decision procedure to decide the satisfiability of a set of formulas. Q(S), in this case, should be defined as follows:

Definition: For a set of clauses S, Q(S) is defined as follows:

(1) If clauses C and D belong to Q(S), then the resolvents of C and D belong to Q(S).

(2) The set S belongs to Q(S).

Section 2: The Herbrand Theorem

Definition: The Herbrand Universe: Let R be a prenex normal formula. Let S be the functional normal form of R. Let f_{i1}, f_{i2},..., f_{im} be all the function symbols in R and n_k be the number of variables of f_{ik}, (k = 1,2,...,m). The Herbrand Universe of R, is defined to contain, to begin with, all the individual constants of S (or just "a" if there occurs none). Further,

if it contains $t_1, t_2, \ldots, t_{nk}$, then it contains $f_{ik}(t_1, t_2, \ldots, t_{nk})$ for $k = 1,2,\ldots,m$.

Definition: Herbrand instance of a w.f.f. R: The Herbrand instance of a w.f.f. R is to mean any result of substituting, for the occurrences of the variables in R, the terms from the Herbrand Universe of R.

The significance of the above definitions can be seen from the following Skolem-Herbrand-Gödel Theorem. For the proof of this theorem, see [16,19].

Skolem-Herbrand-Gödel Theorem: The w.f.f. R is satisfiable if the conjunction of all the Herbrand instances of R is truth-functionally satisfiable.

The above theorem enables us to use techniques used in propositional calculus, namely truth table technique, to determine the satisfiability of a w.f.f. in the first order predicate calculus. Thus we can define the term "satisfiable" in terms of ideas from propositional calculus.

Given a set of clauses $C_1, C_2, \ldots, C_n$, let $A_1, A_2, \ldots, A_m$ be all the distinct atomic formulas which occur in S or whose complements occur in S. Let $M_i = \{M_{i1}, M_{i2}, \ldots, M_{ij}, \ldots\}$ be the set of all the Herbrand instances of A_i.

An interpretation I is constructed as follows:

(1) For all i and j, no M_{ij} or $\sim M_{ij}$ are members of I simultaneously (satisfiable).

(2) For all i and j, either M_{ij} or $\sim M_{ij}$ is in I (complete).

An interpretation I is said to satisfy a clause C if I contains at least one member of every Herbrand instance of C. Thus an interpretation I does not satisfy a clause C if I consists entirely of the complements of

the literals of at least one of the Herbrand instances of C. An interpretation is said to satisfy a set of clauses if it satisfies every clause of the set.

A clause C is a logical consequence of a set of clauses $C_1, C_2, \ldots, C_n$ if there is no interpretation that satisfies $C_1 \& C_2 \ldots \& C_n \& \sim C$.

Section 3: The Proof of the Completeness Theorem

Theorem: Given a set of clauses $S = C_1 \& C_2 \ldots \& C_n$, if a clause C is a logical consequence of S, then for some $n \geq 0$, there exists a clause $T \in R^n(S)$, such that T implies C.

Proof:

(1) Let $C = L_1 \vee L_2 \ldots \vee L_k$.

(2) Let $\sim C = L_1' \& L_2' \ldots \& L_k'$.

(3) Let the conjunction of Q(S) and $\sim C$ be denoted as Q. For the definition of Q(S) see Section 1 of this chapter.

(4) Let A be the set of all the Herbrand instances over Q of the set of all the distinct atomic formulas which occur in Q(S) or whose complements occur in Q(S).

(5) Let G be the set of all the distinct Herbrand instances of $L_1', L_2', \ldots, L_k'$.

(6) Let $B = \{B_1, B_2, \ldots, B_j, \ldots\}$ be the set of all the members of A which do not occur in G or whose components do not occur in G.

(7) We construct an interpretation as follows:

$$I_0 = \{G\}$$

For $j > 0$, $I_j = I_{j-1} \cup \{B_j\}$ unless for some $n \geq 0$ there is a clause in $R^n(S)$, which has a Herbrand instance that consists entirely of the complements of the literals in

the set $I_{j-1} \cup \{B_j\}$ in which case, $I_j = I_{j-1} \cup \{\sim B_j\}$.

(8) If {B} has m members, then $I = I_m$.

(9) If {B} has countably infinite number of members, then I has countably infinite number of members.

(10) If, for every $n \geq 0$, there does not exist any clause T in $R^n(S)$, such that T implies C, then the interpretation I should satisfy both $\sim C$ and Q(S).

(11) For if it does not, then there is a least j, $j > 0$ such that there is some clause, say C, that has a Herbrand instance consisting entirely of the complements of the literals in the set I_j.

(12) In this case, according to the definition of I_j, $I_j = I_{j-1} \cup \{\sim B_j\}$.

(13) Let H_C be that particular Herbrand instance of C.

(14) By the leastness of j, H_C contains B_j.

(15) This means that C contains a literal, say E, and there is a particular substitution λ_e such that $E\lambda_e = B_j$.

(16) But there must be another clause, say D, which has a Herbrand instance consisting entirely of the complements of the literals of the set $I_{j-1} \cup \{B_j\}$.

(17) Let H_D be this particular Herbrand instance of D.

(18) By the leastness of j, H_D contains $\sim B_j$.

(19) This means that D contains a literal, say F, and there is a substitution λ_F such that $F\lambda_F = \sim B_j$.

(20) Thus E and F are most generally unifiable and have opposite signs. (For the definition of "unifiable," see [17].)

(21) Consider the resolvent of C and D which deletes both literals E and F. Call this resolvent of C and D "T".

(22) Next consider the resolvent of H_C and H_D which deletes B_j and $\sim B_j$. Call this resolvent of H_C and H_D "M".

(23) "M" obviously consists of entirely the complements of the literals of I_{j-1}.

(24) This is impossible unless M consists entirely of the complements of the literals in the set G.

(25) Thus T has a Herbrand instance which consists entirely of the complements of the set {G}.

(26) Thus T&$\sim$C is unsatisfiable and T implies C.

(27) So for some $n \geq 0$ $R^n(S)$ must necessarily contain T, such that T implies C. Q.E.D.

CHAPTER 4

EXPERIMENTS AND FINDINGS WITH DIFFERENT VERSIONS OF THE PROGRAM

The "theorem finding" program consists of two major parts, the program that generates theorems and the program that singles out the interesting ones. The second part will be described in Chapter 5. In this chapter, the author will show how the original program was gradually modified to its present form. The best version of the program is called PG5. PG5 employs the unit resolution principle, the axiom resolution strategy, and various other techniques which are all described in the following sections.

Section 1: The Sub-clause Principle

Before defining the idealized program, we should discuss more about "triviality" as defined in Chapter 3. To say that a theorem is trivial is to say that it is implied by another theorem. Thus a program to check triviality must necessarily be a theorem proving program. Unfortunately, there is no decision procedure for theorem proving in the first order calculus, as proved by Church. In other words, no matter what inference rule one uses, a theorem proving program may, in some cases, go on forever without telling us anything.

However, the lack of decidability of the first order predicate calculus does not prevent us from doing something about checking triviality. In certain cases, the triviality of a theorem can be easily checked. The principle concerning these special cases is called the "Sub-clause Principle." For details of this principle, the reader is urged to consult [17].

The principle is based on the fact that CVD is implied by C where C and D are atomic formulas. We have to make some definitions first.

Definition: If C and D are two clauses, we say that C is a sub-clause of D if there is a substitution 6 such that C6 $\subseteq$ D. If C is a sub-clause of D, then D is said to be a super-clause of C.

Again, for the definition of "substitution," please consult [17].

The Sub-clause Principle can be stated as below:

Sub-clause Theorem: If C is a super-clause of D, then C is implied by D.

The proof of the above principle is not to be given in this thesis. The reader can find the proof of this theorem in [17] under the title of "Subsumption Theorem."

The Sub-clause Principle allows us to check out some trivial theorems rather quickly. For example, if we are given the following two theorems:

T_1: $\sim P(x) \vee P(f(x))$

T_2: $\sim P(a) \vee P(f(a))$

The Sub-clause Principle would tell us immediately that T_2 is trivial because T_1 becomes identical to T_1 if x is substituted by a. But we must understand that a lot of theorems are not implied by other theorems through Sub-clause Principle. The following case will be a good example. We have two axioms:

T_1: $\sim P(x) \vee P(f(x))$

T_2 $\sim P(x) \vee P(f^2(x))$

T_2 is a theorem of T_1, but it is not implied by T_1 through the Sub-clause Principle. In fact, T_2 is a resolvent of T_1 with itself. The

reader is encouraged to verify this by himself.

From now on, whenever the author mentions checking triviality, he means checking triviality through the Sub-clause Principle.

Section 2: The Idealized Program

By an idealized program, we mean a program that checks the triviality of every theorem deduced. In other words, whenever a theorem is deduced, we check whether it is a super clause of another clause. If yes, this clause is deleted. Otherwise, we further check whether any other theorems deduced are super clauses of this clause. If yes, these trivial clauses are deleted. The whole process is described in the following diagram: (See next page.)

In the following sections, the author describes several important experiments. The various versions of this program discussed later are designated as PG1, PG2,..., PG6.

Section 3. PG1 - The Idealized Program

Let us call the idealized program PG1. The first experiment was designed to test the ability of PG1 to generate theorems. The particular theorem selected was called Theorem TG1.

Theorem TG1: In an associative system with left and right solutions, there exists a right identity element.

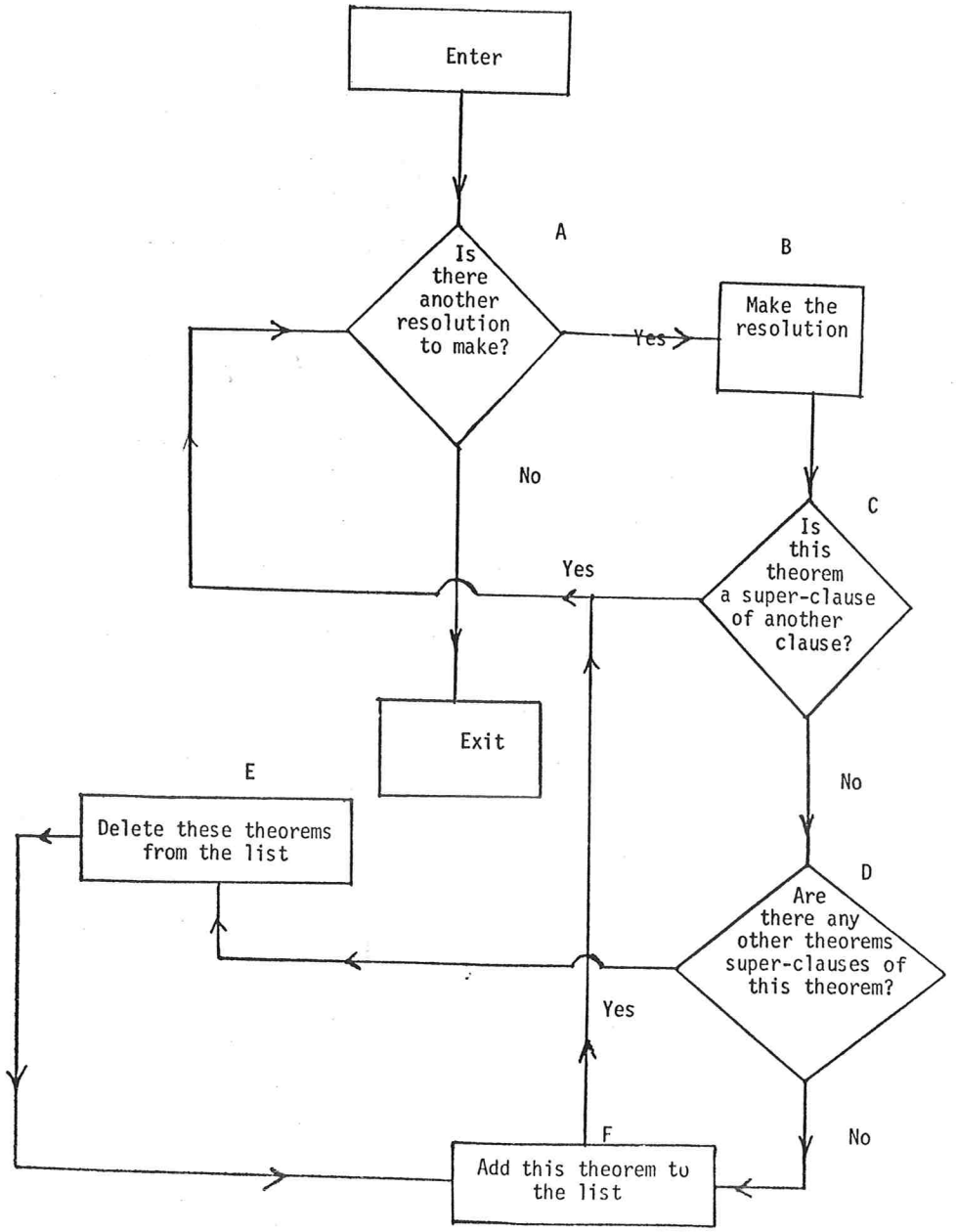
This theorem involves three axioms, namely:

A1: $\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(x,v,w) \vee P(u,z,w)$ **(associativity)**

A_2: $P(g(x,y),x,y)$ **(left solution)**

A_3: $P(x,h(x,y),y)$ **(right solution)**

FLOW DIAGRAM



The target theorem in this case is

T: $P(x,h(y,y),x)$ (right identity)

Several points should be pointed out before presenting the result of the experiment.

(1) The associative axiom is as follows:

$$(x\cdot y)\cdot z = x\cdot(y\cdot z)$$

Expressed symbolically, it involves two clauses:

$$\sim P(x,y,u)\vee\sim P(y,z,v)\vee\sim P(x,v\cdot w)\vee P(u\cdot z,w)$$

$$\sim P(x,y,u)\vee\sim P(y,z,v)\vee\sim P(u\cdot z,w)\vee P(x,v\cdot w)$$

In this case, we only use the first clause because we are only interested in the theorem concerning the right identity. The second clause is needed in case that we want to generate the theorem concerning the left identity. In such a case, the target theorem is

$$P(g(y,y),x,x)$$

(2) At the first glance, the target theorem seems to be obtained from the following formula:

$$(EX)(y)P(y,x,y) \qquad (1)$$

The corresponding clause of (1) is

$$P(y\cdot e.y) \qquad (2)$$

However, the use of the resolution principle does not generate any clause containing new function symbols. Our program can never generate (2). Instead, it will generate

$$P(x\cdot h(y,y)\cdot x) \qquad (3)$$

which is our target theorem.

The important thing is that (3) implies (2). To prove this, we first find the negation of (1) and obtain

$$(X)(Ey)\sim P(y,x,y) \tag{4}$$

The functional normal form of (4) is

$$\sim P(f_1(x),t,f_1(x)) \tag{5}$$

(3) is contradictory to (5). Therefore (3) implies (2).

(3) One can say that this program is indeed superior to a human being, because (3) furnishes more information than (2) does. It not only tells us the existence of such a right identity, but also how to find the right identity. The following table defines an associative system with left and right identities.

	a	b	c
a	a	b	c
b	b	c	a
c	c	a	b

One can easily see that

$$h(a,a) = h(b,b) = h(c,c) = a \quad .$$

It is also easy to prove that the element "a" is indeed a right identity.

The idealized program immediately shows that it is too expensive to check the triviality of every theorem. This program only generated 24 theorems in 30 minutes. Obviously, most of the time was spent on checking triviality of theorems. The output also shows that not a single theorem is checked out because of triviality. To see how much the program will be influenced if no checking of triviality is made, PG1 was modified to PG2.

The result of testing PG2 is presented in Section 4.

Section 4: PG2 - No Checking of Triviality

The first experiment concerning PG2 was to test whether PG2 could generate TG1 or not. PG2 successfully generated TG1. The following data are the results of this first experiment of PG2.

(a) 72 theorems were generated before TG1 was generated.

(b) The total time to generate these 72 theorems is approximately 2 minutes, which confirmed the belief that PG1 spent most of the time on checking triviality of theorems.

The second experiment on PG2 was to generate Theorem TG2.

Theorem TG2: In an associative system with a left inverse and a left identity, the left inverse is also a right inverse.

The axioms involved are:

A1: $\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(x,v,w) \vee P(u,z,w)$ }

A2: $\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(u \cdot Z,W) \vee P(X,v,w)$ } (associativity)

A3: $P(I(X),X,e)$ (left inverse)

A4: $P(e,X,X)$ (left identity)

This time, PG2 failed to generate this target theorem. Several important findings of this experiment are summarized below:

(a) 520 theorems were generated when the memory core became saturated.

(b) The program stopped at the stage of $R^3(S)$, which means that the program could not ever go any further than the third level of resolution.

(c) By examining the clauses generated, it was found that most of the trouble came from resolving A1 and A2. These two

axioms produce a great many theorems; none of which is interesting.

PG2 was thus modified to PG3 in which a limit was put on the number of literals a clause can contain. If the number of literals of a clause exceeds a certain limit, PG3 will not generate it. Besides, PG3 also has a new feature which will automatically avoid generating some of the trivial theorems. This will be discussed in Section 5.

Section 5: "The Unit Resolution Principle"

Definition: A unit is a clause containing exactly one literal.

The unit resolution principle concerns the inference of a clause from a set of units and a clause. See [23]. The following example suffices to explain it.

Example: We have clauses

$$\sim P(x,y) \vee \sim P(y,z) \vee P(x,z) \quad (1)$$

$$P(a \cdot b) \quad (2)$$

$$P(b \cdot c) \quad (3)$$

Suppose we resolve (1) and (2) by deleting the first literal of (1); we obtain

$$\sim P(b,z) \vee P(a,z) \quad (4)$$

Resolving (4) and (3) gives

$$P(a,c) \quad (5)$$

If we resolve (1) and (3) by deleting the second literal of (1), we obtain

$$\sim P(x,b) \vee P(x,c) \quad (6)$$

Resolving (6) and (2) gives

$$P(a \cdot c) \tag{7}$$

Thus, in this case, different ways of resolution may lead to the same clause. In general, there are at most k! different ways to deduce the same clause from a clause $\ell_1 V \ell_2 \ldots V \ell_m$ and a set of units U_1, $U_2, \ldots, U_k$ where $m \geq k$, if every U_i is used at most once. The unit resolution principle recognizes this fact and avoids the repetition.

Generally, mathematical axiom systems contain quite a few units. Without employing this principle, much of the computer time and memory space will be wasted on deducing these trivial theorems.

Another important aspect of units is that most units deduced are trivial. While it is difficult to check the triviality of a clause containing more than one literal, it is not difficult at all to check the triviality of a unit. We therefore limited ourselves only to checking the triviality of units in some versions of the program.

Section 6: PG3 - Employing Unit Resolution - Principle and Bound of a Clause

The first experiment of PG3 used TG2 as its target theorem. The new features of PG3 are:

(a) Every clause containing more than 4 literals is not to be generated.

(b) The triviality of a theorem is checked only if it is a unit.

(c) Unit resolution principle is used.

PG3 successfully generated TG2 and we can summarize the results of this experiment as follows:

(a) 552 theorems were generated before the memory space was saturated.

(b) The program was pushed to $R^6(S)$, indicating that PG3 is superior to PG2.

(c) 17 trivial units were deleted from the memory.

PG3 was also used in another important experiment. In this experiment, the purpose was to see how an increase of memory space will influence the performance of the program. Three sets of axioms S_1, S_2 and S_3, were used. Table 6.1 shows S_1, S_2, and S_3.

The memory core was first set to 24K. Then it was increased to 32K. PG3 was tested to see at what level of resolution it would stop for these two different sets of memory space.

The result of this experiment can be summarized in Table 6.2. In Table 6.2, the depth of the program means the value n of $R^n(S)$ at which PG3 stopped. For example, when we say the level is 4, we mean that memory became full when the program was generating theorems in $R^4(S)$.

Table 6.2 shows that the performance of the program would not be greatly improved by merely increasing the size of the memory space. This is quite an important discovery.

Section 7: The "Axiom Resolution Strategy" and PG4

Since the accumulation of theorems is a great problem, the author tried an entirely new idea, which the author calls "axiom resolution strategy." This strategy would only resolve axioms with axioms, or theorems with axioms. In other words, no theorem is used to resolve with another theorem. In this way, a theorem can be deleted from the memory immediately after it has been used to resolve with all of the axioms.

S_1

$\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(x,v,w) \vee P(u,z,w)$

$P(x,h(x,y),y)$

$P(g(x,y),x,y)$

S_2

$\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(u,z,w) \vee P(x,v,w)$

$\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(x,v,w) \vee P(u,z,w)$

$P(IX),X,e)$

$P(e,X,X)$

S_3

$\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(u,z,w) \vee P(x,v,w)$

$\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(x,v,w) \vee P(u,z,w)$

$P(I(X),X,e)$

$P(e,X,X)$

$\sim P(X,y,u) \vee \sim P(X,y,v) \vee M(u,v)$

$\sim M(x,y) \vee \sim M(y,z) \vee M(X,z)$

$\sim M(x,y) \vee M(y,X)$

$M(x,x)$

Here $M(x,y)$ means $x = y$.

TABLE 6.1. SYSTEMS OF AXIOMS TESTED IN THE SECOND EXPERIMENT OF PG3.

Axioms Tested	No. of theorems generated		Depth of the program	
	24K	32K	24K	32K
S_1 (3 axioms)	516	651	7	8
S_2 (4 axioms)	552	683	6	6
S_3 (8 axioms)	537	672	4	4

TABLE 6.2. RESULTS OF THE SECOND EXPERIMENT OF PG3.

This means that a much smaller number of clauses will occupy the memory and thus memory saturation is avoided.

The program that uses "axiom resolution strategy" is called PG4. PG4 is a modification of PG3. Its features are

(a) "Axiom resolution strategy" is used.

(b) "Unit resolution principle" is used.

(c) The triviality of a clause is checked only if it is a unit.

(d) No clause containing more than 6 literals is to be generated.

The first experiment of PG4 was to use axiom system S_1. PG4 successfully generated TG1 by using S_1.

The second experiment of PG4 was to use S_2. PG4 not only generated TG1 and TG_2, but also generated TG3.

Theorem TG3: In an associative system with a left inverse and a left identity, the left identity is also a right identity.

It should be noted here that TG3 is quite a deep theorem. It is found in $R^9(S_2)$ with respect to PG4.

The third experiment was to use axiom system S_3. The target theorem was TG4.

Theorem TG4: In a group, if $x \cdot z = y \cdot z$, then $x = y$.

But, the author was disappointed to note that in using System S_2, PG4 did not generate Theorem TG5.

Theorem TG5: In a group, if the square of every element is an identity, then the group is Abelian.

A modification of PG4 led to PG5, which successfully generated TG5.

Section 8: PG5 - Adding Interesting Theorems to the Axioms

Aside from having a program to generate theorems, PG5 also has a program to determine the "interest" of a theorem. Since the determination of "interest" is quite complicated, we will devote the whole of Chapter 5 to its discussion. Meanwhile, it suffices to know that it is possible to determine the "interest" of a theorem. In PG5, whenever a theorem is decided to be interesting, it is added to the list of axioms, thus stored in the memory permanently. PG5 is superior to PG4 so long as the criteria for interest are good.

PG5 successfully generated TG5. In fact, the crucial theorem in this case is TG3. PG5 generated TG3, determined that it was an interesting one, and then added it to the axiom system. TG5 was generated after TG3 was added to the system of axioms.

So far the best version of the program is PG5. PG5 was later tested on 40 other theorems from Lederman's book. In sum, PG5 generated 41 of all the theorems tested and failed to generate 4 of them.

One aspect of how PG5 was used to generate these 41 theorems is that the axiom system is not necessarily an independent one. Previously proved theorems can be used as axioms. Actually, it is a waste of time if we use independent systems of axioms all the time; human beings seldom do this.

Another version of this program is called PG6. Although PG6 is a failure, we can still draw some important conclusions from the experiments of PG6.

Section 9: PG6 and Its Experiments

Program PG6 is based on one simple idea: "Interesting theorems are generated by interesting theorems." Thus, PG6 starts with a set of axioms, say S. Then PG6 generates a set of theorems out of S. The number of theorems PG6 generates is easily controlled by the programmer. From all of these generated theorems, PG6 selects those that it considers interesting and adds them to the system of axioms S. All of the others are deleted from the memory.

Supposedly, PG6 is a simulation of a human's thought process. It is impossible for a human being to memorize all of his thinking; thus he only remembers the important ones, just as PG6 does.

PG6 was tested on TG1, TG2, TG3, and TG4. Surprisingly, PG6 failed to generate any of them. However, we can still draw an important conclusion; that is, although "interesting" theorems play important roles in generating interesting theorems, they are not the on-y ones that are needed. Obviously, those uninteresting theorems are also needed. Without these uninteresting ones, the interesting ones can not be generated.

CHAPTER 5

THE TECHNIQUE TO SINGLE OUT INTERESTING THEOREMS

In this chapter we will discuss one of the most difficult, yet the most important, problems of theorem finding: how to single out interesting theorems. The main idea is to condense a long clause to a short one so that its real meaning can be easily determined. The author calls this procedure the "Unit Condensation Procedure."

Section 1: The Recognition of an Important Theorem

It is our experience that even a small set of axioms will produce a large number of clauses before an interesting one is deduced. If a program can only grind out theorems but can not single out the interesting ones, this program is useless. However, to determine the importance of a theorem is the most difficult part of this research.

An absolutely objective guideline to determine the importance of a theorem is probably impossible to find because an important theorem is hard to define in the first place. The criterion that a deep and short theorem is interesting is not sufficient. A deep theorem is one that can be obtained only after a long proof. Many interesting theorems are neither short nor deep.

However, the author has found that human beings are quite capable in determining the importance of a theorem. That is why the author made an investigation of human thought processes in this regard. He observed that a human being can quickly grasp the real meaning of a theorem by reducing a long theorem to a short one. The special technique a human being uses may vary from theory to theory, yet the reducing mechanism is quite general. In the following sections, a more detailed discussion will be given.

Section 2: The "Unit Condensation Procedure"

In many cases, the idea of "equality" plays an important role when a human tries to simplify a clause. In group theory, for example, we have an axiom

$$\sim P(x,y,u) \vee \sim P(x,y,v) \vee M(u,v) \tag{1}$$

where M(u,v) means u = v and P(x,y,z) means $x \cdot y = z$.

In group theory, we also have

$$P(e,X,X) \tag{2}$$

as well as $P(I(X),X,e)$ (3)

as axioms.

Thus we have two important theorems:

$$\sim P(e,X,u) \vee M(X,u) \tag{4}$$

$$\sim P(I(X),X,u) \vee M(u,e) \tag{5}$$

concerning equality in group theory.

These two theorems are used quite extensively to simplify a long clause. For example, consider the following clause

$$\sim P(e,z_1,z_2) \vee \sim P(I(z_4),z_4,z_3) \vee P(z_3,z_1,z_2) \tag{6}$$

This clause involves two conditions and one conclusion, namely:

If $e \cdot z_1 = z_2$ (7)

and $I(z_4) \cdot z_4 = z_3$ (8)

then $z_3 \cdot z_1 = z_2$ (9)

However, it can be simplified according to the following reasoning:

Step 1: If $e \cdot z_1 = z_2$, then $z_1 = z_2$. So delete the first condition by substituting $z_1 = z_2$ and the meaning of the clause becomes:

If $I(z_4) \cdot z_4 = z_3$, then $z_3 \cdot z_1 = z_1$

Step 2: If $I(z_4) \cdot z_4 = z_3$, then $z_3 = e$. So substitute $z_3 = e$ and the clause is reduced to the following simple statement

$$e \cdot z_1 = z_1 \tag{10}$$

By now it is not difficult to see (10) is trivial, so we can say that (6) is uninteresting since it does not contain any new information.

This simplification procedure can be easily mechanized. It involves only two steps:

Step 1: Note the "unit" type literals in a clause. In group theory, it means the noting of negative instances of

$$P(e \cdot X \cdot y) \tag{11}$$

and $$P(I(X), y, e) \tag{12}$$

Later, as more theorems are produced, we should also note negative instances of

$$P(X \cdot e \cdot y) \tag{13}$$

and $$P(y \cdot I(X), e) \tag{14}$$

Step 2: Resolve this clause with the unit to delete the noted literal.

This simplification procedure is called "unit condensation procedure" by the author since it uses units to simplify a long clause. Here, the resolution principle is again shown to be a powerful inference rule in that the deduction is not only comprehensible by human beings, but also descriptive of a human's thought process.

As can be expected, there are many cases in which this procedure does not apply. In these cases, human beings can still determine the importance of a theorem using more complicated procedures. We will discuss one of these procedures and its mechanization in the next section.

Section 3: The Substitution Procedure

Consider the following clause

$$\sim P(z_1,z_2,e) \vee \sim P(z_2,z_3,e) \vee \sim P(z_4,z_1,e) \vee \sim P(z_5,z_4,z_6) \vee P(z_6,z_3,z_5) \qquad (15)$$

The real meaning of this clause is:

If $z_1 \cdot z_2 = e$ (16)

$z_2 \cdot z_3 = e$ (17)

$z_4 \cdot z_1 = e$ (18)

and $z_5 \cdot z_4 = z_6$ (19)

then $z_6 \cdot z_3 = z_5$ (20)

Here, no simple reduction can be made as in the example of the last section. But we can substitute (19) into (20) and (20) becomes

$$(z_5 \cdot z_4) \cdot z_3 = z_5 \qquad (21)$$

By the associative law, we obtain

$$z_5 \cdot (z_4 \cdot z_3) = z_5 \qquad (22)$$

(22) suggests trying to prove $z_4 \cdot z_3 = e$. If this can be proved from (16), (17) and (18), this theorem does not say anything significant. In this particular case, this can easily be proved and (15) is considered uninteresting.

It should be noted that it is not necessarily easy to establish that $z_4z_3 = e$. In that case, (15) should be considered an interesting one.

Let us consider another clause which has a pattern similar to (16):

$$\sim P(z_1,z_2,e)\vee\sim P(z_5,z_3,e)\vee\sim P(z_4,z_6,e)\vee\sim P(z_2,z_3,z_4)\vee P(z_5,z_1,z_6) \qquad (23)$$

In this case, no substitution whatsoever can be made. Although it is a long statement, it is still a "rich" one in the sense that it contains a great deal of information. Indeed, (23) is one of the most exciting theorems of the group theory. To see this, we let $z_2 = z_1$, $z_5 = z_3$ and $z_6 = z_4$. Then (23) becomes

$$\sim P(z_1,z_1,e)\vee\sim P(z_3,z_3,e)\vee\sim P(z_4,z_4,e)\vee\sim P(z_1,z_3,z_4)\vee P(z_3,z_1,z_4) \qquad (24)$$

In group theory, a well-known theorem is "If $x^2 = e$ for all x in a group, then this group is commutative." It is not difficult to show that (24) implies this theorem.

The complete simulation of the simplification procedure is too expensive. Experimental results show that the majority of clauses in which substitution can be made, as in (22), turn out to be uninteresting ones. A crude decision procedure to single out theorems which can not be further simplified works satisfactorily.

In short, our decision procedure to determine the importance of a theorem is as follows:

Step 1: Check if a clause can be simplified by using the unit condensation procedure. If not, go to Step 3. Otherwise, simplify it and go to Step 2.

Step 2: If the simplified clause is a unit, it is considered "interesting" only if it is non-trivial. If the simplified clause is not a unit, go to Stpe 3.

Step 3. Check if the clause can be further simplified by substitution. If yes, this clause is considered uninteresting. Otherwise, it is considered interesting.

Section 4: An Example

We shall present an example on how a theorem can be found. The reader who tries to follow this example should first note the following:

(1) The program used is PG5.

(2) This example does not give all the theorems the program generated. Only the clauses relevant to the target theorem are presented.

Example: The target theorem is:

In an associative system with left identity and left inverse, the left identity is also a right identity.

The program is given 3 axioms which are expressed in 4 clauses.

1: $\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(x,v,w) \vee P(u,z,w)$
2: $\sim P(x,y,u) \vee \sim P(y,z,v) \vee \sim P(u,z,w) \vee P(x,v,w)$ } (associativity)

3: $P(e,x,x)$ (left identity)

4: $P(I(x),x,e)$ (left inverse)

The program first checks clause 1 and clause 2. It immediately finds out that the resolvant of clause 1 and clause 2 contains 6 literals which exceeds the limit, so it ignores them.

To generate the resolvant of clause 1 and clause 3, a renaming should first be done so that the two clauses will contain entirely different sets of variables. To do this, the program makes the following renaming of variables:

In clause 1,

$$x = x_1,\ y = x_2,\ u = x_3,\ z = x_4,\ v = x_5,\ w = x_6,$$

In clause 3,

$$x = y_1$$

Thus clause 1 and clause 3 become:

1' $\sim P(x_1,x_2,x_3) \vee \sim P(x_2,x_4,x_5) \vee \sim P(x_1,x_5,x_6) \vee P(x_3,x_4,x_6)$

3' $P(e,y_1,y_1)$

There are three resolvants out of clause 1 and clause 3. But only the one which deletes the second literal of clause 1 is relevant to the generation of our target theorem. To carry out resolution, the program matches the second literal of 1' with the first literal of 3' and finds out that the following substitution should be made:

$$x_2 = e,\ x_4 = x_5 = y_1$$

The resolvant is thus obtained by making this substitution throughout the whole clause 1' and deleting the second literal. The reader can verify for himself that the resolvant is as follows:

5' $\sim P(x_1,e.x_3) \vee \sim P(x_1,y_1,x_6) \vee P(x_3,y_1,x_6)$

To unify the notation, the program renames the variable again by making the following substitution:

$$x_1 = z_1,\ x_3 = z_2,\ y_1 = z_3,\ x_6 = z_4.$$

The new clause now becomes

5. $\sim P(z_1,e,z_2) \vee \sim P(z_1,z_3,z_4) \vee P(z_2,z_3,z_4)$. (from 1 & 3)

Again, it should be noted that the program also generates many other clauses at this level, say by resolving clause 1 and clause 4, clause 2 and clause 3, etc. But the program will not resolve these clauses with themselves, instead it will resolve clause 5 with an axiom. As before, clause 5 will not be resolved with clause 1 or clause 2 because of the bound put on the number of literals a clause can contain. So it will be resolved with both clause 3 and clause 4. The following clause is then generated:

6. $\sim P(I(z_1),e,z_2) \vee P(z_2,z_1,e)$ (from 4 & 5)

An important thing to note here. Clause 6 can be used to resolve with clause 4. The resolvant is

6' $P(e,e,e)$ (from 4 & 6)

The reader is reminded here that the above clause is a unit. So its triviality is now checked. Using Sub-clause Principle, the program immediately finds out that clause 6' is implied by clause 3. Thus clause 6' is deleted from the memory.

The program now continues to generate the following sequence of clauses:

7. $\sim P(I(z_1),z_2,z_3) \vee \sim P(z_2,z_4,e) \vee \sim P(z_3,z_4,z_5) \vee P(z_5,z_1 e)$ (from 2 & 6)

8. $\sim P(z_1,z_2,e) \vee \sim P(e,z_2,z_3) \vee P(z_3,z_1,e)$ (from 4 & 7)

9. $\sim P(z_1,z_2,e) \vee P(z_2,z_1,e)$ (from 3 & 8)

By now, clause 9 is considered interesting, so it is added to the axioms. In order to avoid confusion, we shall still keep the same label.

The reader should be reminded that clause 9 is now stored permanently in the memory and considered an axiom.

Clause 9 is resolved with clause 4 and one of the most important results is obtained. This is

10. $P(z_1,I(z_1),e)$ (from 4 & 9)

Clause 9 is the well known theorem TG2. It is a unit and is checked again to see whether it is trivial or not. Since it is not trivial, it is considered interesting and added to the system of axioms.

11. $\sim P(I(z_1),z_2,z_3) \vee \sim P(e,z_2,z_4) \vee P(z_1,z_3,z_4)$ (from 2 & 10)

12. $\sim P(e,z_1,z_2) \vee P(z_1,e,z_2)$ (from 4 & 11)

Clause 12 is an interesting theorem. It also generates our target theorem.

13. $P(z_1,e,z_1)$ (from 3 & 12).

CHAPTER 6

CONCLUSIONS AND SUGGESTIONS FOR FUTURE RESEARCH

This chapter is devoted to the conclusions and suggestions for future research. Most of these conclusions are drawn from the experiments of this research and from the author's experience in formalized axiomatic methods.

Section 1: Conclusions

(1) Generally speaking, judging from the performance of PG5, one can conclude that for a small number of axioms, this program is definitely superior to human beings. It should be considered as something better than a machine with "common sense," since not every human being with common sense can derive so many important theorems in such a short time.

(2) As shown by Table 5.2, a mere increase of memory space is not a good solution to improve the capability of the program. Although PG5 avoids memory saturation, it obviously also misses a great number of theorems. Probably a better inference rule, which saves memory space, is the only solution to this problem.

(3) As shown by the results of the experiments of PG6, uninteresting theorems should still be used to derive interesting theorems. Probably this is also the case in a human's though process. If we write down in detail the proof procedure of a very difficult theorem, we will find out that in most steps of the proof we are using only quite easy axioms or theorems. For example, the axioms concerning equality are obvious and familiar to most of us, yet these axioms are often very useful.

(4) The greatest failure of this research is that the author failed to obtain a general theory on how an important theorem is derived. The present technique is to generate theorems first and pick out the interesting ones next. If we can find some principle to guide the direction of deduction, we might be able to save a lot of time and memory space.

(5) However, putting a bound on the number of literals of a clause is equivalent to a very crude type of guidance, in the following senses:

(a) Putting a bound passively avoids generating theorems in the wrong direction. Consider A1 and A2 in the example given at the end of Chapter 5. If there is no bound, many theorems will be generated by resolving A1 with A2, and by resolving the resolvents of A1 and A2. The number of literals of these theorems will increase indefinitely. The reader is urged to derive some of these theorems and he will probably arrive at the same conclusion--that these theorems are really uninteresting ones.

(b) Let us consider the example in Chapter 5 again. Denote the number of literals of the ith clause by m_i. We are interested in the pattern of the sequence of m_i's, from i = 5 to i = 13. In fact, we can observe the following sequence:

m_5	m_6	m_7	m_8	m_9	m_{10}	m_{11}	m_{12}	m_{13}
3	2	4	3	2	1	3	2	1

The above sequence shows that the program tends to generate shorter clauses, unless it comes to a lower limit. This is accomplished simply by putting a bound on m_i.

Section 2: Suggestions for Future Research

(1) When a student is taught a certain mathematical theory, he is taught not only in abstract terms, but also with concrete examples. For example, in group theory, we usually are taught a table describing binary operation. In learning about linear vector space, we start with a diagram of vectors. Experience tells us that the establishing of models in learning abstract mathematical theories is extremely helpful in visualizing the true meaning of the theory. Recently J. Slagle [23] proposed a "semantic resolution principle" which uses models for theorem proving. How this can be used to find theorems more effectively is an open problem and should be an interesting one.

(2) We would like to see if this program can be useful in domains outside of mathematics, social science, for example. However, we are using the so-called "formal axiomatic method." In a formal axiomatic method, the meaning of symbols is suppressed entirely. This certainly will create a problem that some of the logical consequences might be utterly meaningless. The following example illustrates this point.

Consider the case that we have three axioms:

(i) If x has y z's, and z has v w's, then x has f(y,v) w's. Here f(y,v) denotes multiplication.

(ii) Man has 2 hands.

(iii) Hand has 5 fingers.

These three axioms can be put into three clauses as below:

$$\sim P(x,y,z) \vee \sim P(z,v,w) \vee P(x,f(y,v),w) \quad (1)$$

$$P(\text{man}, 2, \text{hand}) \quad (2)$$

$$P(\text{hand}, 5, \text{finger}) \quad (3)$$

where P(x,y,z) means x has y z's.

One logical consequence of this system is the resolvent of (1) and (2) by deleting the second literal of (1).

$$\sim P(x,y,man) \vee P(x,f(y,2),hand) \qquad (4)$$

(4) is a theorem of this system, but it makes no sense. For all practical purposes, the machine should be instructed to avoid generating this clause. A machine that generates this kind of clause can hardly be considered to have common sense although it does possess elementary deducing capability.

(3) It would be delightful to have a program having the ability to draw some conclusions by examining a finite set of data. This program now produces many theorems. Hopefully, one day, another program can discover some general laws from the theorems and thus suggest new definitions to describe these general laws. An example would be the case that the machine discovers the following relations in group theory,

$$e^2 = 3$$

$$e^3 = e$$

$$\vdots$$

The machine, after recognizing this special pattern, should try to prove that for every n, $e^n = e$. In such a case, one may claim that machines can discover new definitions too.

BIBLIOGRAPHY

1. Davis, M. and Putnam, H. A computing procedure for quantification theory. JACM (July, 1960), pp. 201-205.

2. Davis, M. Eliminating the irrelevant from mechanical proofs. Proc. of Symposia in Applied Mathematics, American Mathematics Society (1963), pp. 15-30.

3. Feigenbaum, E. and Feldman, J. (Editors). Computers and Thought, McGraw-Hill, New York (1963).

4. Gelernter, H. Realization of a geometry theorem proving machine. Proc. of the International Conference on Information Processing (1959). Reprinted in [3].

5. Gelernter, H., Hansen, J, and Loveland, D. Empirical explorations of the geometry theorem machine. Proc. of the Western Joint Computer Conference (1960), pp. 143-147. Reprinted in [3].

6. Gilmore, P. A program for the production of proofs for theorems derivable within the first order predicate calculus from axioms. Proc. of the International Conference on Information Processing (1959).

7. Gilmore, P. A proof method for quantification theory: its justification and realization. IBM Journal of Research and Development (Jan. 1960), pp. 28-35.

8. Hilbert, D. and Ackermann, W. Principles of Mathematical Logic, Chelsea Publishing Company, New York (1950).

9. Ledermann, W. "Introduction to the Theory of Finite Groups," Oliver and Boyd, London (1964).

10. McCarthy, J. "LISP 1.5 Programmer's Manual," the M.I.T. Press, Cambridge, Mass. (1962).

11. Meltzer, B. Theorem proving for computers. Some results on resolution and renaming. The Computing Journal (1966).

12. Newell, A., Shaw, J., and Simon, H. Empirical explorations of the logic theory machine. Proc. of the Western Joint Computer Conference (1957), pp. 218-239.

13. Pitrat, J. Realization of a program which chooses the theorems it proves. Proc. of International Conference on Information Processing, Vol. 2 (1965), pp. 324-325.

14. Prawitz, D., Prawitz, H. and Vogera, N. A mechanical proof procedure and its realization in an electronic computer. JACM Vol. 7 (1960), pp. 102-108.

15. Prawitz, D. An improved proof procedure. Theoria, Vol. 26 (1960), pp. 102-139.

16. Quine, W. A proof procedure for quantification theory. Journal of Symbolic Logic, Vol. 20, No. 2 (June, 1955), pp. 141-149.

17. Robinson, J. A machine oriented logic based on the resolution principle. JACM, Vol. 7 (Jan, 1965), pp. 23-41.

18. Robinson, J. Automatic deduction with hyper resolution. International Journal of Computer Mathematics, Vol. 2 (1965), pp. 227-234.

19. Robinson, J. A review of automatic theorem proving. Annual Symposia in Applied Mathematics, Vol. XIX (1966).

20. Slagle, J. "A heuristic program that solves symbolic integration problems in freshman calculus (SAINT)," doctoral dissertation, M.I.T., Cambridge, Mass. (1961).

21. Slagle, J. A heuristic program that solves symbolic integration problems in freshman calculus, JACM (Oct. 1963), Reprinted in [3].

22. Slagle, J. A multipurpose, theorem proving, heuristic program that learns. Proc. of the IFIP Congress, Vol. 2 (1965), pp. 323-324.

23. Slagle, J. Automatic theorem proving with renamable and semantic resolution. JACM (Oct. 1967).

24. Wang, H. Towards mechanical mathematics. IBM Journal of Research and Development, Vol. 4 (1960), pp. 2-22.

25. Wang, H. Proving theorems by pattern recognition. I. Communications ACM, Vol. 3 (1960), pp. 220-234.

26. Wang, H. Proving theorems by pattern recognition, II. Bell Systems Tech. J., Vol. 40 (1961), pp. 1-41.

27. Whitehead, A. and Russell, B. Principia Mathematica, Cambridge Univ. Press, Cambridge (1913).

28. Wos, L., Carson, D. and Robinson, G. The unit preference strategy in theorem proving. Proc. of the Fall Joint Computer Conference, (1964), pp. 616-621.

29. Wos, L., Robinson, G. and Carson, D. The efficiency and completeness of the set of support strategy in theorem proving. JACM (Oct. 1965), pp. 536-541.

30. Wos, L., Robinson, G., Carson, D. and Shella, L. The concept of demodulation in theorem proving. JACM (Oct. 1967).

Institut für Computersprachen
Abteilung für Anwendungen der
Formalen Logik
Technische Universität Wien
A-1040 Wien, Resselgasse 3/3