MASSACHUSETTS INSTITUTE OF TECHNOLOGY
A. I. LABORATORY

Artificial Intelligence
Memo No. 217 June 1971

COMPUTER PROOFS OF LIMIT THEOREMS

W.W. Bledsoe*, Robert S. Boyer*,
William H. Henneman

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported by the Advanced Research Projects Agency of the Department of Defense, and was monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0002; it was also supported by National Institute of Health Grant GM 15769-03.

*Massachusetts Institute of Technology and the University of Texas.

COMPUTER PROOF OF LIMIT THEOREMS

by

W. W. Bledsoe, Robert S. Boyer,
William H. Henneman

Massachusetts Institute of Technology
and the University of Texas

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported by the Advanced Research Projects Agency of the Department of Defense, and was monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0002, and was also supported by National Institute of Health Grant GM 15769-03.

1. Introduction

In this paper we describe some relatively simple changes that have been made to an existing automatic theorem proving program to enable it to prove efficiently a number of the limit theorems of elementary calculus. These changes include subroutines of a general nature which apply to all areas of analysis, and a special "limit-heuristic" designed for the limit theorems of calculus.

These concepts have been incorporated into an existing LISP program and run on the PDP-10 at the A.I. Laboratory, M.I.T., to obtain computer proofs of many of the limit theorems, including the theorem that ~~a~~ the limit of ~~a~~ the sum of two real functions is the sum of their limits, and a similar theorem about products. Also computer proofs have been obtained (or are easily obtainable) of the theorems that a continuous function of a continuous function is continuous, and that a function having a derivative at a point is continuous there, as well as limit results for polynomial functions.

The limit theorems of calculus present a surprisingly difficult challenge for general purpose automatic theorem provers. One reason for this is that calculus is a branch of analysis, and proofs in analysis require manipulation of algebraic expressions, solutions of inequalities, and other operations which depend upon the axioms of an ordered field. It is in applying these field axioms that automatic provers are usually forced into long and difficult searches. On the other hand, a human mathematician is often able to easily perform the necessary operations of analysis without being aware of the explicit use of the field axioms. One purpose of this paper is to describe ways in which automatic provers can also avoid the use of the field axioms and speed up proofs in analysis. Section 2 explains how this is done using a limited theory of types and routines for algebraic simplification and solving linear inequalities.

In Section 3 we present the limit-heuristic, give examples of its use, and discuss its "forcing" nature which enables it to curtail combinatorial searches.

The reader interested only in resolution based programs should skip Sections 4 and 5 and go directly to Section 6, where we explain how resolution programs can be altered to make use of the limit heuristic and other concepts.

In Section 5 we give a detailed description of a computer proof of the theorem that the limit of the product of two functions is the product of their limits. This proof was made by a program which is the same as that described in [1], except that the subroutine, RESOLUTION, in [1] has been replaced by a new subroutine called IMPLY. We have thus eliminated resolution altogether from our program replacing it by an "implication method" which we believe is faster and easier to use (though not complete). This implication method is described briefly in Section 4, and excerpts from actual computer proofs using it are given there and in Section 5.

It appears that some of these ideas may have wider implications than the limited scope in which they were used here. This is discussed in the comments of Section 7 and throughout the paper.

2. Types and Inequalities

In the work described in this paper we have used *membership types* whereby the type A is assigned to x whenever it is known that $(x \in A)$.

Let $<a \; b>$ denote the open interval from a to b , $\underline{R} = <-\infty \; \infty>$, $\underline{P} = <0 \; \infty>$, and $\underline{N} = <-\infty \; 0>$. We are primarily interested in *interval* types, including the types $\underline{R}$, $\underline{P}$, and $\underline{N}$. Thus in trying to prove

$$(0 < x \rightarrow Q(x))$$

we would assign the type $\underline{P}$ (or $<0 \; \infty>$) to x and then try to prove Q(x).

For example, suppose that we are to prove

(1) $$(0 < b \rightarrow \text{SOME } x \; (0 < x \wedge x < b)).$$ [1]

One valid approach is to solve for x in

(2) $$(0 < b \rightarrow 0 < x)$$

and then try to verify

(3) $$(0 < b \rightarrow x < b)$$

for that same x. But using matching we would get as a solution of (2) the substitution [b/x] [2] , and require

$$(0 < b \rightarrow b < b)$$

---

1. We use the words "SOME" and "ALL" as our existential and universal quantifiers. Thus "SOME x P(x)" means "for some x P(x)", and "ALL x P(x)" means "for all x P(x)".

2. We follow the usual practice of denoting a substitution by a list $[b_1/a_1, b_2/a_2, \ldots, b_n/a_n]$ where each a_i is to be replaced by the corresponding b_i.

in (3) which is impossible.

Of course (1) is unprovable without further hypotheses (or axioms) but it can be easily handled by the use of types (which implicitly assumes certain axioms). Our approach in proving (1) is to assign <0 ∞> to b, and then try to prove

(4) $\qquad$ SOME x $(0 < x \wedge x < b)$.

We first solve

(5) $\qquad$ $(0 < x)$

by assigning type <0 ∞> to x and then solve

(6) $\qquad$ $(x < b)$

by assigning the type <0 b> to x. The resulting type of x, <0 b>, was derived as the intersection of its initial type <0 ∞> gotten from (5), and the interval <-∞ b>, which would have been the type gotten from (6) alone. Since this intersection is not empty (because b has type <0 ∞>), it is assigned as the resulting type of x. Even though the variable x had already been "solved for" in (5) (typed), it remains a <u>variable</u> in the solution of (6) (though limited in scope) and therefore could be "solved for" again (retyped). In the examples of Section 5 some of the variables are retyped two or three times, and this greatly simplifies the proofs.

Types are used by the routines SOLVE< and SET-TYPE which are described below.

2.1 SOLVE<

This is a routine for solving linear inequalities. (SOLVE< A B) chooses a variable from A or from B and attempts to solve the inequality (A < B) in terms of that variable. If this fails it then chooses another variable and tries again. Since the terms and variables of A and B may be typed, this routine must take into consideration such types and reset the type of the variable when the solution is obtained. In fact the answer is completely given by the new types. The examples below best illustrate this point. If it can show that A is less than B, then the routine will return the answer "T" whether or not A and B have any variables.

Examples.

	INPUT: A	INPUT: B	OUTPUT: Value of (SOLVE< A B)	OUTPUT: New Type of x
1.	x (no types)	1	x	<-∞ 1>
2.	x Type x is <0 ∞>	1	x	<0 1>
3.	0	1	T	
4.	x·a+c Type a is <0 ∞>	(-x+d)	x	<-∞ $(\frac{d}{1+a} - \frac{c}{1+a})$>
5.	x Type x is <0 D_2> Type D_1 is <0 ∞> Type D_2 is <0 ∞>	D_1	x	(intersection <0 D_1><0 D_2>)

In this example the type of D in the answer could have been given as <0 (minimum $D_1 D_2$)> but we find the intersection form more convenient.

	A	B	Value of (SOLVE< A B)	New Type of x
6.	$\frac{a}{x}$ Type x is <0 ∞> Type a is <-∞ 0> Type b is <0 ∞>	b	x	<$\frac{a}{b}$ ∞>

In the actual theorem proving process, SOLVE< is applied to formulas that have been converted to quantifier free form by the introduction of skolem expressions.[3] Precautions are taken by SOLVE< to insure that it does not solve for a variable x in terms of a skolem expression in which x occurs. This is essentially the same precaution taken by J. A. Robinson in his Unification Algorithm [2].

For example, consider the false statement

SOME x ALL y (y < x) .

The skolem form of this is

(y x) < x .

The result of a call to (SOLVE< (y x) x) is NIL, since x occurs in the skolem expression (y x).

On the other hand, the theorem

SOME x ALL y SOME z (y < x+z)

which has skolem form

(y x) < x+z

can be proved by a call to (SOLVE< (y x) (x+z)) which correctly assigns type <(y x)-x ∞> to z.

---

3. A skolem expression is a term whose main function symbol is a skolem function. cf. ~~the footnote~~ in Section 4 which describes the elimination of quantifiers by the introduction of skolem functions.

Actually, ~~this~~ the routine SOLVE< just retypes a variable in a way that guarantees the solution of the desired inequality.

More extensive routines could easily be written (indeed have been written by others) to solve nonlinear inequalities, but these were not found necessary for proving the examples reported here.

<u>2.2 SOLVE=</u>. This is a routine for solving linear equations. Given two arithmetic expressions A and B, it selects a variable x from A or B and trys to solve the equation (A = B) in terms of x. If it succeeds, with answer y, it returns the substitution, [y/x] . Otherwise it selects another variable and trys again, returning NIL if all fail.

<u>2.3 SET-TYPE.</u> This is a subroutine which assigns types to certain skolem expressions. If a formula of the form $(A \in B)$ is in a conjunctive position of E (i.e., E can be expressed as $((A \in B) \wedge D)$ for some D), and if A is a skolem expression which does not occur in B, then (SET-TYPE E) assigns the type B to A and returns D, the formula gotten by removing $(A \in B)$ from E. If A already has type C, then SET-TYPE assigns the intersection $(B \cap C)$ as the type of A, if $(B \cap C)$ is non-empty. If $(B \cap C)$ is empty it returns E. If $(B \cap C)$ is not empty, but cannot be given specifically then the formula (intersection B C) is given as the type of A.

For example, if E is the formula

$$(A \wedge (x \in \underline{P} \wedge (B \rightarrow y \in \underline{R})))$$

then (SET-TYPE E), assigns $\underline{P}$ as the type of x, and returns

(1) $(A \wedge (B \rightarrow y \in \underline{R}))$.

If, in this example, x already had type $\underline{R}$,

then $\underline{P}$ is assigned as the new type of x;

if it already had type $\langle -1\ 1\rangle$ then it assigns type $\langle 0\ 1\rangle$ to x; if it already had type $\langle -\infty\ -1\rangle$ then it returns $(A \wedge (B \rightarrow y \in \underline{R}))$.

In a similar way, it assigns types to skolem expressions which satisfy certain inequalities. For example, if E is

$$(A < 0 \wedge (B < 1 \vee C))$$

then (SET-TYPE E) assigns type $\langle -\infty\ 0\rangle$ to A and returns

$$(B < 1 \vee C),$$

and if E is

$$(A < B \wedge C)$$

then (SET-TYPE E) assigns type $\langle -\infty\ B\rangle$ to A, and type $\langle A\ \infty\rangle$ to B and returns C. Similarly, (SET-TYPE $(A \neq 0)$) can be made to assign type (union $\langle -\infty\ 0\rangle$ $\langle 0\ \infty\rangle$) to A, but this sort of typing was not used in any of the examples given in this paper.

2.4 SIMPLIFY

This is an algebraic simplification routine which converts algebraic expressions into a canonical form, sorts its terms, and cancels complementary terms of the form $(a+(-a))$ and $(a \cdot \frac{1}{a})$. It is used in all of our routines which manipulate algebraic expressions. Such routines are not new to the literature.

Examples.

INPUT	OUTPUT
$(a\cdot(b+c))$	$(a\cdot b + a\cdot c)$
$(a\cdot b\cdot\frac{1}{a})$	b
$(-(a+\frac{1}{b})\cdot(b+c) + c\cdot a)$	$(-(a\cdot b)+(-1)+(-(c\cdot\frac{1}{b})))$
$(\lvert b+c-b\rvert + a)$	$(\lvert c\rvert + a)$
$(-\lvert(a\cdot\frac{1}{a}) - 1\rvert)$	0

3. Limit Heuristic

The limit heuristic rule defined below, in conjunction with the routines described in Section 2, is used to help prove limit theorems.

LIMIT-HEURISTIC: When trying to use a hypothesis of the type

$$|A| < E'$$

(and possibly other hypotheses) to establish a conclusion of the type

$$|B| < E,$$

first try to find a substitution σ which will allow B_σ[4] to be expressed as a non-trivial combination[5] of A_σ, $(B = K \cdot A + L)_\sigma$, and then try to establish the three new conclusions:

A. $(|K| < M)_\sigma$, for some M,

B. $(|A| < E/2 \cdot M)_\sigma$,

C. $(|L| < E/2)_\sigma$

Such a procedure is valid because if we can indeed find such a σ and prove A, B, and C, then we would have

$$\begin{aligned} |B|_\sigma &= |K \cdot A + L|_\sigma \\ &\leq (|K| \cdot |A| + |L|)_\sigma \\ &< M \cdot E/2M + E/2 \\ &= E. \end{aligned}$$

Of course, this is based on the triangle inequality, and uses the fact that $1/2 + 1/2 = 1$, $M \cdot 1/M = 1$ for $M \neq 0$, etc.

---

4. The notation B_σ denotes the result of applying the substitution σ to B.

5. The routine EXTRACT, described in Section 3.1 below, is used to express B in terms of A.

As an example, in proving the theorem that the limit of the product of two functions of real variables is the product of their limits, we find ourselves trying to establish a conclusion of the type

(1) $$|f(x) \cdot g(x) - L_1 \cdot L_2| < E.$$

Among our hypotheses is

(2) $$|f(x') - L_1| < E',$$

which can be used to help establish (1) (provided that we satisfy the conditions for (2)). If we apply the limit heuristic to (2) and (1) we find that for $\sigma = [x/x']$

$$(f(x) \cdot g(x) - L_1 \cdot L_2)$$

can be expressed as a combination of

$$(f(x') - L_1)_\sigma,$$

viz.,

$$g(x) \cdot (f(x) - L_1) + L_1 \cdot (g(x) - L_2),$$

and are able to establish the three subgoals:

A. $|g(x)| < M$, for some M.

B. $|f(x) - L_1| < E/2 \cdot M$.

C. $|L_1 \cdot (g(x) - L_2)| < E/2$.

Subgoal A follows from the hypothesis

(3) $$|g(x'') - L_2| < E''$$

(which also has conditions that must be satisfied). Subgoal B follows from (2), and subgoal C follows from (3).

The complete proof of the limit product theorem is given in Section 5 in great detail. The limit heuristic is used there not only to set up the three subgoals A, B, and C, but also to establish

A and C, by proposing further subgoals.

Because the limit heuristic enables our program to prove many theorems about limits, we regard it as a rather interesting trick. But more interesting and important than the fact that it works some problems is the principle behind it. That principle might be stated:

> To establish a conclusion C from several hypotheses, among which is H, force H to contribute all it can towards establishing C and leave a remainder to be established with the help of the other hypotheses.

The value of such a "forcing" technique is twofold. First, if one can truly make H contribute all it can towards C, then H is not needed to establish the remainder. That is, a reduction in the number of hypotheses is achieved while a significant step in the proof is made.

Second, it is implicit in the notion of "force" that certain facts are used to make an inference in a computational manner. For example, the limit heuristic "uses" many facts about algebra, such as the triangle inequality; but these facts are used to compute something, not to make random inferences. This strongly inhibits the generation of subgoals that occurs if one freely permits the application of axioms to his goals. We comment further on this "computational" aspect of the limit heuristic in Section 7.

We feel that such a forcing technique has applications in other areas of theorem proving where two or more hypotheses $H_1, H_2, \ldots H_n$ are needed to establish one conclusion C that cannot be logically divided.

In such applications the user must provide a heuristic which will enable the computer to determine how to get a partial result from H_1 and leave a remainder C' to be proved by the other hypotheses.

The limit heuristic uses the routine EXTRACT described below, which in turn uses the simplification routine described in Section 2.

3.1 EXTRACT

If there is a substitution σ for which B_σ can be expressed as a non-trivial combination of A_σ,

$$(B = K \cdot A + L)\sigma$$

then (EXTRACT A B) returns (K L σ), where σ is the most general such substitution. Otherwise NIL is returned.

A more precise definition follows the examples.

Examples

In the following, the symbols x, t, and h represent variables while all other symbols represent constants.

1. (EXTRACT A (K·A+L)) = (K L T).[6]

2. (EXTRACT A(t) A(t_0)) = (1 0 [t_0/t]).

3. (EXTRACT (f(x)-L_1) (f(x_0)+g(x_0) - (L_1+L_2))).

   = (1 (g(x_0) - L_2) [x_0/x]).

4. (EXTRACT (f(x)-L_1) (f(x_0)·g(x_0) - L_1·L_2)

   = (g(x_0) (L_1·g(x_0) - L_1·L_2) [x_0/x]).

5. (EXTRACT (f(x)-L_1) ($\frac{1}{f(x)}$ - $\frac{1}{L_1}$))[7] = (- $\frac{1}{f(x)\cdot L_1}$ 0 T).

6. (EXTRACT ($\frac{f(a+h)- f(a)}{h}$ - F') (f(x) - f(a)))

   = ((x-a) (x-a)·F' [h/(x-a)]).

7. (EXTRACT ((x_0-a) (x_0^2 -a^2)) = ((x_0+a) 0 T).

8. (EXTRACT (a·x_0+c) (b·x_0+d))

   = ($\frac{b}{a}$ (d - $\frac{b\cdot c}{a}$) T).

9. (EXTRACT (a·x_0 +c) (b·y_0+d)) = NIL.

Examples 3, 4, 5 are useful in proving limit theorems about the sum of two functions, the product of two functions (see Section 5), and

---

6. Throughout this paper we use the letter "T" to denote both "truth", and the empty substitution. This reserves "NIL" for denoting "false".

7. In this example, the second argument is first converted to (L_1· $\frac{1}{f(x)\cdot L_1}$ - f(x) · $\frac{1}{f(x)\cdot L_1}$), by use of a least common denominator.

the quotient of two functions. Example 6 is used in proving that a differentiable function is continuous.

Suppose there is a substitution σ and an expression x such that, A_σ and B_σ are polynomials in x, and B is linear in x . Then there are expressions a, c, b and d such that x does not occur in c, b, or d, and A_σ and B_σ can be reexpressed as

$$A_\sigma = a \cdot x + c \ ,$$

$$B_\sigma = b \cdot x + d \ ,$$

and (EXTRACT A B) returns the value $(\frac{b}{a} \quad (d - \frac{b\ c}{a})\ \sigma)$. If no such σ and x exist then EXTRACT returns NIL.

4. The Implication Method

At the heart of the program is a subroutine called IMPLY whose essential purpose is to handle logical deductions in the predicate calculus. It is a replacement for Resolution in [1]. We offer here a cursory description of its operation, sufficient to an understanding of the proofs in section 5.

The operation of IMPLY bears a closer resemblance to the proof techniques of the mathematician than does resolution. In general IMPLY examines the connectives in the formulas given as arguments to it and creates one or two subgoals. These subgoals are usually calls to IMPLY with new arguments which are closely related to but simpler than the original arguments. The resulting analysis of the formula to be proved is easy to follow.

This rather natural operation bears some responsibility for the development of the limit heuristic and the other techniques of this paper. In comparing the subgoals called by IMPLY with the methods of proof used in elementary calculus we established new subroutines and subgoals, such as the limit heuristic, sufficient to prove a number of theorems.

The subroutine IMPLY has two arguments:

E (the current formula under examination)

R (a reserve).

Usually E is of the form

$$(H \rightarrow C)$$

The answer to a call to IMPLY is either a substitution or NIL. The latter indicates failure to establish the subgoal. IMPLY attempts to find and return the most general substitution σ such that $(R \rightarrow E)_\sigma$ is true. If σ is the empty substitution then IMPLY returns T.

Table 1 gives rules describing some of the operations of IMPLY. These rules are applied in the order of their occurence in the table; if one fails, the next is tried; if all fail,IMPLY returns NIL. IMPLY returns the value given by the first rule which does not give NIL. In the following we use the shorter notation [E , R] for (IMPLY E R).

INPUT	OUTPUT
1. $[H \rightarrow C, R]$	
If $H \equiv C$, then	T
If there is a substitution which unifies H and C, (i.e., $H_\sigma \equiv C_\sigma$) then	σ
2. $[A \wedge B, R]$	
2.1 If $[A,R]$ yields $\sigma 1$[8] and 2.2 $[B,R]_{\sigma 1}$ yields $\sigma 2$ then	$(\sigma 1 \cup \sigma 2)$
3. $[A \vee B, R]$	
If $[A,R]$ yields $\sigma 1$, then	$\sigma 1$
If $[B,R]$ yields $\sigma 2$, then	$\sigma 2$
4. $[(A \rightarrow B) \rightarrow C, R]$	
4.1 If $[B \rightarrow C, R]$ yields $\sigma 1$ and 4.2[9] $[R \rightarrow A, NIL]_{\sigma 1}$ yields $\sigma 2$ then	$(\sigma 1 \cup \sigma 2)$
This rule is commonly known as backwards chaining.	
5. $[H \rightarrow (A \rightarrow B), R]$	$[H \wedge A \rightarrow B, R]$
6. $[A \vee B \rightarrow C, R]$	
6.1 If $[A \rightarrow C, R]$ yields $\sigma 1$ and $[B_{\sigma 1} \rightarrow C, R_{\sigma 1}]$ yields $\sigma 2$ then	$(\sigma 1 \cup \sigma 2)$
6.2 If $[B \rightarrow C, R]$ yields $\sigma 1$ and $[A_{\sigma 1} \rightarrow C, R_{\sigma 2}]$ yields $\sigma 2$ then	$(\sigma 1 \cup \sigma 2)$

---

8. When we use an expression like "[A,R] yields σ", it is to be understood that we also mean that σ is not NIL.

7. $[A \wedge B \rightarrow C, R]$

   If $[A \rightarrow C, R]$ yields $\sigma 1$ then $\sigma 1$

   If $[B \rightarrow C, R]$ yields $\sigma 2$ then $\sigma 2$

8. $[H \rightarrow A \wedge B, R]$

   8.1 If $\left\{ \begin{array}{l} [H \rightarrow A, R] \text{ yields } \sigma 1 \\ \text{and} \\ [H \rightarrow B_{\sigma 1}, R_{\sigma 1}] \text{ yields } \sigma 2 \end{array} \right.$ then $(\sigma 1 \cup \sigma 2)$

   8.2 If $\left\{ \begin{array}{l} [H \rightarrow B, R] \text{ yields } \sigma 1 \\ \text{and} \\ [H \rightarrow A_{\sigma 1}, R_{\sigma 1}] \text{ yields } \sigma 2 \end{array} \right.$ then $(\sigma 1 \cup \sigma 2)$

9. $[H \rightarrow A \vee B, R]$

   If $[H \rightarrow A, R]$ yields $\sigma 1$ then $\sigma 1$

   If $[H \rightarrow B, R]$ yields $\sigma 2$ then $\sigma 2$

10. $[H \rightarrow \sim A \vee B, R]$ $[H \wedge A \rightarrow B, R]$

11. $[\sim A \wedge B \rightarrow C, R]$ $[B \rightarrow A \vee C, R]$

12. $[\sim H \rightarrow C, R]$ $[R \rightarrow C \vee H, NIL]$

13. $[H \rightarrow \sim C, R]$ $[H \wedge C \rightarrow NIL, R]$

14. $[A = B \rightarrow C, R]$ $[R' \rightarrow C', NIL]$

    where R' and C' are gotten by replacing B by A in R and C.

15. $[H \rightarrow A = B, R]$ (SOLVE= A B)

    (i.e., if there is a substitution σ, which unifies A and B, then return σ)

Table 1 (concluded)

Before a formula E is sent to IMPLY it is first converted to a quantifier free form, but without converting it first to prenex normal form. The quantifier free form is achieved by using skolem functions, and is essentially the same as that used by Wang [3].[10] A call is then made to (IMPLY E NIL).

For example the formula

(1) $(P(y) \wedge \text{ALL } x\ (P(x) \rightarrow Q(x)) \rightarrow Q(y))$

is first converted to the skolem form

$$(P(y_0) \wedge (P(x) \rightarrow Q(x)) \rightarrow Q(y_0))$$

where y_0 is a skolem constant and x is a variable, and proved as follows.

1. (IMPLY $(P(y_0) \wedge (P(x) \rightarrow Q(x)) \rightarrow Q(y_0))$ NIL)

   1.1 (IMPLY $(P(y_0) \rightarrow Q(y_0))$ $(P(x) \rightarrow Q(x))$) Rule 7

   This fails.

   1.2 (IMPLY $((P(x) \rightarrow Q(x)) \rightarrow Q(y_0))$ $P(y_0)$) 7

      1.2.1 (IMPLY $(Q(x) \rightarrow Q(y_0))$ $P(y_0)$) 4.1

      This yields $\sigma = [y_0/x]$ by Rule 1.2

---

10. Specifically, if "positive" and "negative" are given the meaning as in Wang [3] pp. 9-10, then the elimination of quantifiers consists of deleting each quantifier and variable immediately after it, and replacing each variable v bound by a positive quantifier with a list whose first member is v and whose other members are those variables bound by negative quantifiers whose scope includes v. This list which replaces v is simply the application of a skolem function to certain arguments. With no ambiguity, but as an aid to memory, the skolem function is named v.

1.2.2. (IMPLY $(P(y_0) \rightarrow (P(x) \vee Q(y_0)))$ NIL)$_\sigma$ 4.2

(IMPLY $(P(y_0) \rightarrow (P(y_0) \vee Q(y_0)))$ NIL)

(IMPLY $(P(y_0) \rightarrow P(y_0))$ NIL) 9.1

This yields T by Rule 1.1

So the final answer to 1. is $[y_0/x]$, and the theorem is proved.

For the example

$$(\text{SOME } x\ (\text{ALL } y\ P(x,y)) \rightarrow \text{ALL } s\ (\text{SOME } t\ P(t,s)))$$

the skolem form is

$$(P(x_0, y) \rightarrow P(t, s_0)).$$

A call is made to IMPLY

(IMPLY $(P(x_0, y) \rightarrow P(t, s_0)$ NIL)

which yields $[x_0/t, s_0/y]$ by Rule 1.2. QED.

In trying to prove the non-theorem

$$(\text{ALL } y\ (\text{SOME } x\ P(x, y)) \rightarrow \text{SOME } t\ (\text{ALL } s\ P(t, s))),$$

the skolem form is

$$(P((x\ y), y) \rightarrow P(t, (s\ t))$$

where (x y) and (s t) are skolem expressions. A call to IMPLY

(IMPLY (P((x y), y) → P(t, (s t))) NIL)

fails; Rule 1.2 cannot be applied because the formulas P((x y), y) and P(t, (s t)) cannot be unified. A partial unification is given by [(x y)/t], but the resulting pair

$$P((x\ y),\ y), \quad P((x\ y),\ (s\ (x\ y)))$$

cannot be unified by [(s(x y))/y] since the variable y occurs in (s(x y)).

When attempting to prove an expression E with the help of axioms, $A_1, A_2, \ldots, A_n$, (where all free variables in the A_i have been universally quantified), a call is made to (IMPLY E' NIL) where E' is the skolemized form of

$$(A_1 \wedge A_2 \quad \ldots \wedge A_n \rightarrow E) .$$

In the operations described in Table 3, a resemblance can be seen between the method of Gentzen sequents (cf. Kleene's G3 [4]) and the subgoals which IMPLY sets up. The technique of finding a most general unifier is the Unification Algorithm of Robinson [2]. On the whole, IMPLY is closer to the system of Prawitz [6] than to resolution.

5. Examples of Computer Proofs.

Here we give excerpts from the proofs of five theorems, which were made by the program PROVER using IMPLY as its principal subroutine. PROVER is explained in [1] and IMPLY is described briefly in Section 4 above, but the reader familiar with Sections 2 and 3 should be able to follow these descriptions with no reference to [1] and little to Section 4.

In order to use the limit heuristic described in Section 3, we need to add the following rule to Table 1.

16. $[|A| < E' \rightarrow |B| < E, R]$

    If

    16.0 EXTRACT (A B) is (K L σ) (i.e. $(B = K\cdot A + L)_\sigma$) ,

    and if

    16.1 $[R \rightarrow |K| < M, \text{NIL}]_\sigma$ yields $\sigma 1$,[11]

    for some variable[12] M, and if

    16.2 $[|A| < E' \rightarrow |A| < E/2\cdot M, R]_{\sigma\sigma 1}$ yields $\sigma 2$,

    and if

    16.3 $[R \rightarrow |L| < E/2, \text{NIL}]$ yields $\sigma 3$,[13]

    then return the value $(\sigma \cup \sigma 1 \cup \sigma 2 \cup \sigma 3)$.

Also, we need two additional rules for solving inequalities; one rule for types, and one for equations.

17. $[H \rightarrow a < b, R]$	(SOLVE< a b)
18. $[a < b \rightarrow a' < c, R]$	$[(b < c) \vee (b = c), R]_\sigma$

If there is a substitution σ for which $(a = a')$.

---

11. In case K = 1, step 16.1 is omitted, and M is set to 1 in 16.2.

12. M is given type $\langle 0\ \infty \rangle$ and also M is made an additional argument of all skolem functions which already have at least one argument.

13. In case L = 0, step 16.3 is omitted.

19. [H → A ε B, R]

If A has type B then T

20. [a = b → c = d, R] (SOLVE= (a-b) (c-d))

These five rules are placed at the beginning of Table 1 (Section 4), in the order 17, 18, 19, 20, 16.

Also, a provision is made for assigning types to an expression A when it appears in the form (A ε B) or (A < B) in the hypothesis of the theorem being proved. This is accomplished when IMPLY is proving a subgoal of the form [H → C, R] by replacing H by (SET-TYPE H). Such calls to SET-TYPE need only be made in Rules 5, 10, 13, and before the first call to IMPLY, when new material is added to H. (see Section 2.3).

In what follows, R denotes the real numbers , P denotes the positives, and FRR denotes the functions on R to R. We use (Lim f a L) to denote $\lim_{x \to a} f(x) = L$. The standard definition of limit is:

(Lim f a L) <->

(a ε R) ∧ (L ε R) ∧ (f ε FRR) ∧

(ALL ε (0 < ε → (SOME δ (0 < δ ∧

(ALL x [(x ε R) ∧ (x ≠ a) ∧ |x-a| < δ → |f(x) - L| < ε])))

Example 1 (Limit of a product)

The program PROVER is given the formula

$$(\text{Lim } f\ a\ L_1 \wedge \text{Lim } f\ a\ L_2 \rightarrow \text{Lim } (f \cdot g)\ a\ (L_1 \cdot L_2))$$

The definition of limit is used to obtain

$$((a \in \underline{R} \wedge L_1 \in \underline{R} \wedge f \in FRR \wedge \text{ALL } E_1 \ (0 < E_1$$

$$\rightarrow \text{SOME } D_1 \ (0 < D_1 \wedge \text{ALL } x_1 \ (x_1 \in \underline{R} \wedge x_1 \neq a \wedge$$

$$|x_1 - a| < D_1 \rightarrow |f(x_1) - L_1| < E_1))))$$

$$
\begin{aligned}
\wedge \quad & (a \in \underline{R} \;\wedge\; L_2 \in \underline{R} \;\wedge\; g \in \mathrm{FRR} \;\wedge \\
& \quad \mathrm{ALL}\ E_2\ (0 < E_2 \;\rightarrow\; \mathrm{SOME}\ D_2\ (0 < D_2 \;\wedge \\
& \qquad \mathrm{ALL}\ x_2\ (x_2 \in \underline{R} \;\wedge\; x_2 \neq a \;\wedge\; |x_2 - a| < D_2 \\
& \qquad\quad \rightarrow |g(x_2) - L_2| < E_2)))) \\
\rightarrow \quad & (a \in \underline{R} \;\wedge\; (L_1 \cdot L_2) \in \underline{R} \;\wedge\; (f \cdot g) \in \mathrm{FRR} \;\wedge \\
& \quad \mathrm{ALL}\ E\ (0 < E \;\rightarrow\; \mathrm{SOME}\ D\ (0 < D \;\wedge\; \mathrm{ALL}\ x\ (x \in \underline{R} \;\wedge \\
& \qquad x \neq a \;\wedge\; |x-a| < D \;\rightarrow\; |(f\ g)(x) - L_1 \cdot L_2| < E)))))
\end{aligned}
$$

The first three parts of the conclusion $(a \in \underline{R})$ $(L_1 \cdot L_2) \in \underline{R}$ $(f \cdot g) \in \mathrm{FRR}$ are proved by the program using the hypotheses of the theorem.

The remainder of the theorem is prepared for IMPLY by replacing $(f \cdot g)(x)$ by $(f(x) \cdot g(x))$ and by eliminating the quantifiers and introducing skolem expressions.

(i) $$((a) \in (\underline{R}) \wedge (L_1) \in (\underline{R}) \wedge (f) \in (FRR) \wedge$$

$$(0 < E_1 \rightarrow (0 < (D_1\ E_1) \wedge$$

$$(x_1 \in (\underline{R}) \wedge x_1 \neq a \wedge |x_1 - (a)| < |(D_1\ E_1)$$

$$\rightarrow |(f)(x_1) - (L_1)| < E_1)))$$

$$((a) \in (\underline{R}) \wedge (L_2) \in (\underline{R}) \wedge (g) \in (FRR) \wedge$$

$$(0 < E_2 \rightarrow (0 < (D_2\ E_2) \wedge$$

$$(x_2 \in (\underline{R}) \wedge x_2 \neq (a) \wedge |x_2 - (a)| < (D_2\ E_2)$$

$$\rightarrow |(g)(x_2) - (L_2)| < E_2)))$$

$$\rightarrow (0 < (E) \rightarrow (0 < D \wedge$$

$$((x\ D) \in (\underline{R}) \wedge (x\ D) \neq (a) \wedge |(x\ D) - (a)| < D$$

$$\rightarrow |(f)((x\ D))\cdot(g)((x\ D)) - (L_1)\cdot(L_2)| < E)))$$

For readability and brevity, the skolem expressions are abbreviated in the following. Thus x is used in place of (x D), L_1 in place of (L_1), f(x) in place of (f)((x D)), and so on.

Thus we write the above expression as

(ii)

$$\alpha \begin{cases} (a \in \underline{R} \wedge L_1 \in \underline{R} \wedge f \in FRR \wedge \\ \quad (0 < E_1 \rightarrow (0 < D_1 \wedge \\ \quad\quad (x_1 \in \underline{R} \wedge x_1 \neq a \wedge |x_1 - a| < D_1 \\ \quad\quad\quad \rightarrow |f(x_1) - L_1| < E_1)))) \end{cases}$$

$$\beta \begin{cases} \wedge (a \in \underline{R} \wedge L_2 \in \underline{R} \wedge g \in FRR \wedge \\ \quad (0 < E_2 \rightarrow (0 < D_2 \wedge \\ \quad\quad (x_2 \in \underline{R} \wedge x_2 \neq a \wedge |x_2 - a| < D_2 \\ \quad\quad\quad \rightarrow |g(x_2) - L_2| < E_2)))) \end{cases}$$

$$\gamma \begin{cases} (0 < E \rightarrow 0 < D \wedge \\ \quad (x \in \underline{R} \wedge x \neq a \wedge |x - a| < D \\ \quad\quad \rightarrow |f(x) \cdot g(x) - L_1 \cdot L_2| < E)). \end{cases}$$

The computer continues to use the full skolem notation throughout its proof.

Before we follow the proof procedure for this theorem in great detail, we first sketch the proof that the computer will produce.

Given $E > 0$, choose M, M', E_1, and E_2 so that

$$M > 2 \cdot |L_2| ,$$

$$M' > |L_1| ,$$

$$E_1 < E/2 \cdot M ,$$

$$E_2 < \min (M/2, E/4 \cdot M') .$$

By hypothesis, there exist D_1 and D_2 such that $0 < D_1$ and $0 < D_2$, and for all x, if $x \neq a$ and $|x - a| < \min (D_1, D_2)$, then

$$|f(x) - L_1| < E_1 ,$$

and

$$|g(x) - L_2| < E_2 .$$

Furthermore, for all x, if $x \neq a$, and $|x - a| < \min(D_1, D_2)$, then since

$$|g(x) - L_2| < E_2 < M/2 ,$$

it follows that

$$|g(x)| < M/2 + |L_2|$$

$$< M/2 + M/2 ,$$

$$|g(x)| < M .$$

So let D be a number such that

$$0 < D < \min(D_1, D_2) .$$

If x is any number such that $x \neq a$ and $|x - a| < D$, then

$$|f(x) \cdot g(x) - L_1 \cdot L_2|$$

$$= |g(x) \cdot (f(x) - L_1) + L_1 \cdot (g(x) - L_2)|$$

$$\leq |g(x) \cdot (f(x) - L_1)| + |L_1 \cdot (g(x) - L_2)|$$

$$= |g(x)| \cdot |f(x) - L_1| + |L_1| \cdot |g(x) - L_2|$$

$$< M \cdot E/2 \cdot M + M' \cdot \min(M/2, E/4 \cdot M')$$

$$\leq E/2 + M' \cdot E/4 \cdot M'$$

$$< E. \qquad \text{QED.}$$

The key to this proof is the proper selection of M, M', E_1, E_2, and D. The computer makes precisely these same selections though its handling of types.

We now resume that description of the computer's procedure in finding its proof. A call is made to

$$(\text{IMPLY } (\alpha \wedge \beta \rightarrow \gamma) \text{ NIL})$$

where α, β, and γ are given in (ii) above.

SET-TYPE is applied to $(\alpha \wedge \beta)$, assigning type $\underline{R}$ to a, L_1, L_2, and type FRR to f and g, and the subformulas $(a \in \underline{R})$, $(L_1 \in \underline{R})$, $(L_2 \in \underline{R})$, $(f \in \text{FRR})$, and $(g \in \text{FRR})$, <u>are removed from α and β</u>.

Rule 5 is applied, converting the formula to

$$(\alpha \wedge \beta \wedge 0 < E \rightarrow 0 < D \wedge (x \in \underline{R} \wedge x \neq a \wedge |x-a| < D \rightarrow |f(x) \cdot g(x) - L_1 \cdot L_2| < E)).$$

SET-TYPE is applied to the hypothesis; E is assigned type $<0\ \infty>$ and $(0 < E)$ is removed.

Rule 8 calls imply on the two formulas

$$(\alpha \wedge \beta \rightarrow 0 < D)$$

and

$$(\alpha \wedge \beta \rightarrow (x \in R \wedge x \neq a \wedge |x - a| < D \rightarrow |f(x_1) \cdot g(x) - L_1 \cdot L_2| < E)).$$

The first call is satisfied by Rule 17, which uses SOLVE< to assign type $<0\ \infty>$ to D. The second results in an application of Rule 5, so the current subgoal is

$$(\alpha \wedge \beta \wedge (x \in \underline{R} \wedge x \neq a \wedge |x - a| < D) \rightarrow |f(x) \cdot g(x) - L_1 \cdot L_2| < E)$$

SET-TYPE is applied to the hypothesis; x is assigned type R and $(x \in \underline{R})$ is removed.

By Rule 7, the reserve $\underline{R}$ is set to

$$(\beta \wedge x \neq a \wedge |x - a| < D) ,$$

and

$$(\alpha \rightarrow |f(x) \cdot g(x) - L_1 \cdot L_2| < E)$$

becomes the current goal.

Rule 4 (backward chaining) is now applied. That is, the program tries first to establish the conclusion $|f(x) \cdot g(x) - L_1 \cdot L_2| < E$ from α. This is subgoal (1). When this subgoal is established, the program tries to satisfy the hypothesis of α, namely subgoal (2) below.

(1) $$(0 < D_1 \wedge (x_1 \in \underline{R} \wedge x_1 \neq a \wedge |x - a| < D \rightarrow |f(x_1) - L_1| < E_1) \rightarrow |f(x) \cdot g(x) - L_1 \cdot L_2| < E)$$

By Rule 7 the program first tries to prove

$$(0 < D_1 \rightarrow |f(x) \cdot g(x) - L_1 \cdot L_2| < E) .$$

But this fails. Therefore by Rule 7 (2nd part),

$$((x_1 \in R \wedge x_1 \neq a \wedge |x_1 - a| < D \rightarrow |f(x_1) - L_1| < E_1) \rightarrow |f(x) \cdot g(x) - L_1 \cdot L_2| < E)$$

becomes the current goal. (From now on we shall not mention those subgoals which are tried but not established.)

Again the program "chains backwards" using Rule 4. The current subgoal becomes (11) and the hypothesis

$$(x_1 \in \underline{R} \wedge X_1 \neq a \wedge |x_1 - a| < D)$$

is satisfied later at (12).

(11) $$(|f(x_1) - L_1| < E_1 \rightarrow |f(x)\cdot g(x) - L_1\cdot L_2| < E)$$

The program now tries to apply Rule 16, the limit heuristic. First

$$(\text{EXTRACT } (f(x_1 - L_1) \; (f(x)\cdot g(x) - L_1\cdot L_2))$$

is computed to be $(g(x)\ (g(x)\cdot L_1 - L_1\cdot L_2)\ \sigma)$, where $\sigma = [x/x_1]$. This follows from the equation

$$(f(x)\cdot g(x) - L_1\cdot L_2) = ((g(x)\cdot(f(x) - L_1) + (g(x)\cdot L_1 - L_1\cdot L_2)) \; .$$

Because the result of the call to EXTRACT is not NIL, Rule 16 is applicable. The program tries to establish the three subgoals (111), (112), (113), in accordance with Rules 16.1, 16.2, and 16.3. The current subgoal is

(111) $$(\beta \wedge x \neq a \wedge |x - a| < D \rightarrow |g(x)| < M)$$

where M is a new variable which is assigned type <0 ∞>. (Also M is made an additional argument in the skolem expressions $(D_1 E_1)$, $(D_2 E_2)$, (x D), in accordance with footnote 6 above. Although these new skolem expressions $(D_1\ E_1\ M)$, $(D_2\ E_2\ M)$, (x D M), will not appear in our descriptions since we are abbreviating them to D_1, D_2, x, they nevertheless play a crucial role. For example, in step (111.1) below the M in (x D M) prevents Rule 17 and SOLVE< from assigning type <|g(x D M)| ∞> as the answer to (111 1). See Section 2.1.)

By Rule 7, the reserve R is set to $(x \neq a \wedge |x - a| < D)$ and

$$(B \rightarrow |g(x)| < M)$$

becomes the current subgoal.

(Rule 4 is applied. (111 1) becomes the current subgoal and the hypothesis of B is satisfied later at (111 2).

(111 1) $$(0 < D \wedge (x_2 \in \underline{R} \wedge x_2 \neq a \wedge |x_2 - a| < D_2 \rightarrow |g(x_2) - L_2| < E_2) \rightarrow |g(x)| < M).$$

By Rule 7 the program tries

$$((x_2 \in \underline{R} \wedge x_2 \neq a \wedge |x_2 - a| < D_2 \rightarrow |g(x_2) - L_2| < E_2) \rightarrow |g(x)| < M).$$

Another application of Rule 4 sets up the two subgoals (111 11) and (111 12).

(111 11) $$(|g(x_2) - L_2| < E_2 \rightarrow |g(x)| < M)$$

Since (EXTRACT $(g(x_2) - L_2)$ $g(x)$) yields (1 L_2 $[x/x_2]$) the limit heuristic is applicable to (111 11). Because 1 is returned as the value of K from EXTRACT, only subgoals (111 111) and (111 112) are tried, in accordance with Rule 16. The current subgoal becomes

(111 111) $$(|g(x) - L_2| < E_2 \rightarrow |g(x) - L_2| < M/2).$$

By Rule 18, the program tries to establish

$$(E_2 < M/2) \vee (E_2 = M/2)$$

The first half of the disjunction is satisfied by a call to (SOLVE< E_2 M/2), giving type <-∞ M/2> to E_2. Thus subgoal (111 111) is established and the program tries to prove

(111 112) $(x \neq a \wedge |x - a| < D \rightarrow |L_2| < M/2)$.

Rule 17 is applied; (SOLVE< $|L_2|$ M/2) is called, resulting in the type $<2 \cdot |L_2| \;\; \infty>$ for M. Hence both subgoals of (111 11) are established.

The program now returns to the subgoal

(111 12) $$(x \neq a \wedge |x - a| < D \rightarrow x_2 \in \underline{R} \wedge x_2 \neq a \wedge |x_2 - a| < D_2)_\sigma ,$$

where $\sigma = [x/x_2]$. That is

$$(x \neq a \wedge |x - a| < D \rightarrow x \in \underline{R} \wedge x \neq a \wedge |x - a| < D_2).$$

This subgoal is established by several subcalls. The conclusion $(x \in \underline{R})$ follows since x has type $\underline{R}$. $(x \neq a)$ occurs in the hypothesis. And finally

$$(|x - a| < D \rightarrow |x - a| < D_2)$$

is established through Rules 18, 17, and a call to SOLVE<. As a result, the type of D is changed to <0 D_2>.

(111 2) $(x \neq a \wedge |x - a| < D \rightarrow 0 < E_2)$

is established by Rule 17. SOLVE< types E_2 as <0 M/2>. Recall that

E_2 was given type $<-\infty \quad M/2>$ at (111 111). Thus both subgoals of (111) have been established and the program returns to the second subgoal of the first use of the limit heuristic

(112) $$(|f(x) - L_1| < E_1 \rightarrow |f(x) - L_1| < E/2M).$$

This subgoal is quickly established using Rules 17,18 and (SOLVE $< E_1 \quad E/2M$), which assigns type $<-\infty \quad E/2M>$ to E_1.

The third subgoal of the first use of the limit heuristic is

(113) $$(\beta \wedge x \neq a \wedge |x - a| < D \rightarrow |g(x) \cdot L_1 - L_1 L_2| < E/2).$$

By Rule 7, the reserve B is set to $(x \neq a \wedge |x - a| < D)$, and the current subgoal becomes

$$(B \rightarrow |g(x)\ L_1 - L_1\ L_2| < E/2).$$

The program chains backwards twice.

(113 1) $$(0 < D_2 \wedge (x \in R \wedge x \neq a \wedge |x - a| < D_2 \rightarrow |g(x) - L_2| < E_2) \rightarrow |g(x) \cdot L_1 - L_1 \cdot L_2| < E/2)$$

(113 11) $$(|g(x) - L_2| < E_2 \rightarrow |g(x) \cdot L_1 - L_1 \cdot L_2| < E/2)$$

Since (EXTRACT $(g(x) - L_2)$ $(g(x) \cdot L_1 - L_1 \cdot L_2)$) yields ($L_1$ 0 T), the limit heuristic is again applicable, and subgoals (113 111), (113 112) and (113 113) are tried.

(113 111) $$(x \neq a \wedge |x - a| < D \rightarrow |L_1| < M')$$

becomes the current subgoal, where M' is a new variable of type $<0\ \infty>$. This goal is established by assigning type $<|L_1|\ \infty>$ to M', by Rule 17 .

(113 112) $$(|g(x) - L_2| < E_2 \rightarrow |g(x) - L_2| < (E/2)/2 \cdot M')$$

This subgoal is established by use of Rules 17, 18, and a call to (SOLVE< E_2 $E/4 \cdot M'$). E_2 is retyped as (intersection $<0\ \ M/2>$ $<-\infty\ \ E/4 \cdot M'>$). Recall that E_2 had been given type $<0\ \ M/2>$ to establish (111 2). Since the program does not know which of M/2 and E/4·M' is the smaller, the intersecticn is given as the answer, after it has checked that the intersection is non-empty.

The formula

(113 113) $$(x \neq a \wedge |x - a| < D \rightarrow |0| < E/4)$$

is the last subgoal of the last use of the limit heuristic. It is satisfied since E already has type $\langle 0\ \infty\rangle$.

The program now returns to

(113 12) $$(x \neq a \ \wedge \ |x - a| < D \ \rightarrow$$
$$x \in \underline{R} \ \wedge \ x \neq a \ \wedge \ |x - a| < D_2),$$

which is the same as (111 12). Also

(113 2) $$(x \neq a \ \wedge \ |x - a| < D \ \rightarrow \ 0 < E_2)$$

is the same as (111 2).

All of the subgoals of the first application of the limit heuristic at (1 1) have been established, giving as an answer to (1 1) the substitution $\sigma = [x/x_1 \ , \ x/x_2]$.

The program now tries to satisfy

(12) $$(\beta \ \wedge \ x \neq a \ \wedge \ |x - a| < D$$
$$\rightarrow \ x_1 \in \underline{R} \ \wedge \ x_1 \neq a \ \wedge \ |x_1 - a| < D_1) \ .$$

The substitution $[x/x_1]$ establishes the first two parts of the conclusion. To prove the third part, the program tries

$$(|x - a| < D \ \rightarrow \ |x - a| < D_1) \ ,$$

which results in the retyping of D as (intersection $\langle 0\ \ D_2\rangle$ $\langle -\infty\ \ D_1\rangle$). Recall that D previously had type $\langle 0\ \ D_2\rangle$.

Finally the subgoal

$$(\beta \ \wedge \ x \neq a \ \wedge \ |x - a| < D \ \rightarrow \ 0 < E_1)$$

is established by Rule 17 and a call to (SOLVE< 0 E_1) which retypes E_1

as $\langle 0 \quad E/2 \cdot M \rangle$. E_1 previously had type $\langle -\infty \quad E/2\ M \rangle$.

The proof is complete. We list here the final types assigned to the variables. Note that the program has made just those "choices" described in the sketch of the proof which was given earlier.

E_1	$\langle 0 \quad E/2 \cdot M \rangle$
E_2	(intersection $\langle 0 \quad M/2 \rangle$ $\langle -\infty \quad E/4 \cdot M' \rangle$)
D	(intersection $\langle 0 \quad D_2 \rangle$ $\langle -\infty \quad D_1 \rangle$)
M	$\langle 2 \cdot \lvert L_2 \rvert \quad \infty \rangle$
M'	$\langle \lvert L_1 \rvert \quad \infty \rangle$.

This proof may seem long and drawn out but these are essentially the steps a human prover would have to follow in <u>finding</u> and <u>exhibiting</u> a proof.

In the following examples we proceed directly to skolem form and consider only the proof of the main conclusions. Many steps in each proof are omitted.

The notation H_i is used to denote the hypothesis of Step i. Rule reference numbers are sometimes given to the right of formulas along with new type assignments.

<u>Example 2</u>. (composite continuous function theorem).

1. (g is continuous at a) $\wedge$ (f is continuous at g(a))

   $\rightarrow$ f:g is continuous at a.

2. Lim g a g(a) $\wedge$ Lim f g(a) f(g(a)) $\rightarrow$ Lim (f:g)a f(g(a)).

3. $(0 < E_1 \rightarrow (0 < D_1 \wedge (x_1 \in \underline{R} \wedge x_1 \neq a \wedge |x_1 - a| < D_1$

$\rightarrow |g(x_1) - g(a)| < E_1)))$

$\wedge (0 < E_2 \rightarrow (0 < D_2 \wedge (x_2 \in \underline{R} \wedge x_2 \neq a \wedge |x_2 - a| < D_2$

$\rightarrow |f(x_2) - f(g(a))| < E_2)))$

$\rightarrow$

$(0 < E \rightarrow (0 < D \wedge (x \in \underline{R} \wedge x \neq a \wedge |x - a| < D$

$\rightarrow |f(g(x)) - f(g(a))| < E)))$

In 3 the variables are E_1, x_1, E_2, x_2, D, and the skolem expressions are $(D_1\ E_1)$, $(D_2\ E_2)$, (E), (x D), (a), etc.

	CURRENT SUBGOAL	RULE	NEW TYPE ASSIGNMENTS
4.	$(H_3 \rightarrow 0 < D)$	5, 8	E $\langle 0\ \infty \rangle$
5.	(SOLVE< 0 D)	17	D $\langle 0\ \infty \rangle$
6.	$(H_3 \wedge x \neq a \wedge |x - a| < D \rightarrow \underbrace{|f(g(x)) - f(g(a))| < E)}_{Y}$		x $\underline{R}$
⋮			
7.	$(|f(x_2) - f(g(a))| < E_2 \rightarrow |f(g(x)) - f(g(a))| < E)$		
8.	$(E_2 < E \vee E_2 = E)$	18	
9.	(SOLVE< E_2 E)	9, 17	E_2 $\langle -\infty\ E \rangle$
10.	$(H_6 \rightarrow 0 < E_2)$, a condition from Step 7.		
11.	(SOLVE< 0 E_2)	17	E_2 $\langle 0\ E \rangle$
12.	$(H_6 \rightarrow x_2 \in \underline{R} \wedge x_2 \neq a \wedge |x_2 - a| < D_2)_\sigma$, a condition from Step 7, where $\sigma = [g(x)/x_2]$		

13. $(H_6 \rightarrow |g(x) - a| < D_2$ 8

14. $(|g(x_1) - g(a)| < E_1 \rightarrow |g(x) - g(a)| < D_2)$

15. (SOLVE< E_1 D_2) 18, 17, $\sigma = [x/x_1]$ E_1 <-∞ D_2>

16. $(H_6 \rightarrow |x - a| < D_1)$, a condition from Step 14.

17. $(|x - a| < D \rightarrow |x - a| < D_1)$ 7

18. (SOLVE< D D_1) 18, 17 D <0 D_1>.

<u>QED</u>

<u>Example 3</u>. (Differentiable functions are continuous).

$$\text{If } \lim_{h \to 0} \frac{f(a+h) - f(a)}{h} = F' \quad \text{then} \quad \lim_{x \to a} f(x) = f(a).$$

1. (Derivative f a F' $\rightarrow$ Continuous f a)

2. (Lim q 0 F' $\rightarrow$ Lim f a f(a)),

   where q(h) is the difference quotient $\frac{f(a+h) - f(a)}{h}$.

3. $(0 < E_1 \rightarrow (0 < D_1 \wedge (h \in R \wedge h \neq 0 \wedge |h| < D_1$

   $\rightarrow \left| \frac{f(a+h) - f(a)}{h} \right| - F' < E_1)))$

   $\rightarrow$

   $(0 \rightarrow E \rightarrow (0 < D \wedge (x \in R \wedge x \neq a \wedge |x - a| < D$

   $\rightarrow |f(x) - f(a)| < E)))$

In 3 the variables are E_1, h, D, and the skolem expressions are (D_1 E_1), (E), (x D), (a), (F'), etc.

⋮

4. $(H_3 \wedge x \neq a \wedge |x - a| < D \rightarrow |f(x) - f(a)| < E)$ x R

5. $(\left|\frac{f(a+h) - f(a)}{h} - F'\right| < E_1$

   $\rightarrow |f(x) - f(a)| < E)$ Rule 4

The limit heuristic Rule 16 is applied,

(EXTRACT ($\frac{f(a+h) - f(a)}{h}$ - F') (f(x) - f(a))) yields

((x - a) (x - a)·F' σ), where σ = [(x - a)/h].

5. $(H_4 \rightarrow |x - a| < M)$ 16.1

6. $(|x - a| < D \rightarrow |x - a| < M)$

7. (SOLVE< D M) 18, 17 D <0 M>

8. $\left(\left|\frac{f(x) - f(a)}{x - a} - F'\right| < E_1\right.$

$\rightarrow \left.\left|\frac{f(x) - f(a)}{x - a} - F'\right| < E/2\cdot M\right)$ Rule 16.2

9. (SOLVE< E_1 E/2·M) 18, 17 E_1 <-∞ E/2·M>

10. (H_4 → $|(x - a)\cdot F'| < E/2$) 16.3

11. ($|x - a| < D$ → $|(x - a)\cdot F'| < E/2$)

The limit heuristic is again used, EXTRACT yields (F' 0 T).

12. (H_4 → $|F'| < M$) 16.1

13. (SOLVE< $|F'|$ < M) 17 M <$|F'|$ ∞>

14. ($|x - a| < D$ → $|x - a| <$ E/4·M') 16.2
etc.

15. ($x \neq a \wedge |x - a| < D$

$\rightarrow h \in \underline{R} \wedge h \neq 0 \wedge |h| < D_1)_\sigma$ 4.2

a condition for Step 5. $\sigma = [(x - a)/h]$.

16. (H_{15} → $(x - a) \in \underline{R}$) 8

True by Rule 19 since both x and a have type $\underline{R}$.

17. ($x \neq a$ → $x - a \neq 0$) 8, 7

18. ($x - a = 0$ → $x = a$) 12, 13 (from Step 15)

19. (SOLVE= (x-a-0) (x-a)) 20 TRUE

20. ($|x - a| < D$ → $|x - a| < D_1$) 12, 13 (from Step 15)

21. (SOLVE< D D_1) 17, 18 D
(intersection <0 E/4·M'> <-∞ D_1>)

QED.

Example 4. $(\lim_{x \to a} x^2 = a^2)$.

1. $(f = \lambda\ x\ x^2 \rightarrow \text{Lim}\ f\ a\ (a \cdot a))$

2. $(0 < E \rightarrow (0 < D \wedge (x \in \underline{R} \wedge x \neq a \wedge |x - a| < D \rightarrow |x \cdot x - a \cdot a| < E)))$

In 2, D is a variable and (E), (x D), and (a) are skolem expressions.

SET-TYPE assigns type $<0\ \infty>$ to E.

3. $(0 < D)$ Rule 2

4. (SOLVE< 0 D) 17 D $<0\ \infty>$

5. $(x \neq a \wedge |x - a| < D$
$\rightarrow |x \cdot x - a \cdot a| < E)$ 2 x R

6. $(|x - a| < D \rightarrow |x \cdot x - a \cdot a| < E)$ 8

The limit heuristic is used, (EXTRACT (x - a) (x·x - a·a)) yields ((x+a) 0 T).

7. $(H_5 \rightarrow |x+a| < M)$ 16.1

The limit heuristic is used again, (EXTRACT (x-a) (x+a)) yields (1 2·a T).

8. $(|x - a| < D \rightarrow |x - a| < M/2)$ 16.1 (from Step 7)

9. (SOLVE< D M/2) 18, 17 D $<0\ M/2>$

10. $(H_7 \rightarrow |2 \cdot a| < M/2)$ 16.2 (from Step 7)

11. (SOLVE< $|2 \cdot a|$ M/2) 17 M $<2 \cdot |2 \cdot a|\ \infty>$

12. $(|x - a| < D \rightarrow |x - a| < E/2 \cdot M)$ 16.2 (from Step 5)

13. (SOLVE< D E/2·M) 17 D
(intersection $<0\ M/2>$ $<-\infty\ E/2 \cdot M>$)

QED

Example 5. (Limit of a quotient). The proof of this example is not complete.

1. $(\text{Lim } f \text{ a } L \wedge L \neq 0 \rightarrow \text{Lim } (1/f) \text{ a } (1/L))$.

2. $(0 < E_1 \rightarrow (0 < D_1 \wedge (x_1 \in \underline{R} \wedge x_1 \neq 0 \wedge |x_1 - a| < D_1$

$\rightarrow |f(x_1) - L| < E_1)))$

$\wedge\ L \neq 0 \rightarrow$

$(0 < E \rightarrow (0 < D \wedge (x \in \underline{R} \wedge x \neq 0 \wedge |x - a| < D$

$\rightarrow \left|\frac{1}{f(x)} - \frac{1}{L}\right| < E)))$

⋮

3. $(|f(x_1) - L| < E_1 \rightarrow \left|\frac{1}{f(x)} - \frac{1}{L}\right| < E)$

The limit heuristic Rule 16 is applied,

(SOLVE< $(f(x_1) - L)$ $(\frac{1}{f(x)} - \frac{1}{L})$) yields $(\frac{-1}{L \cdot f(x)}$ 0 $\sigma)$, where $\sigma = [x/x_1]$.

We are required by Rule 16 to establish the subgoals

(1) $(H_2 \rightarrow \left|\frac{-1}{L \cdot f(x)}\right| < M)$, 16.1

and

(2) $(|f(x) - L| < E_1 \rightarrow |f(x) - L| < E/2 \cdot M)$ 16.2

Subgoal (2) is easily established by assigning type $<-\infty \quad E/2 \cdot M>$ to E_1, but (1) presents difficulty. In fact the program is unable to give a proof without some axioms or a change in the program. See Section 7 for further comments on this example.

6. Resolution

In this section we show how the limit heuristic and the theory of types explained above can be used in Resolution based programs. This is done by giving some additional rules for resolution. These are:

6.1 SET-TYPE Rule

For each unit clause of the form

$$(x \in A)$$

where x is a skolem expression which does not occur in A, assign the type A to x. Also for each unit clause of the form

$$(x < a)$$

where x is a skolem function which does not occur in a, assign the type $<-\infty a>$ to x. Similarly for unit clauses of the form $(b < x)$ assign type $<b\infty>$ to x. In each of these cases, remove the unit clause. If x already has a type B and we are trying to assign a new type A, then assign the type $(A \cap B)$ if it is non-empty; if $(A \cap B)$ is empty, add the empty clause (i.e., the proof is finished); if it cannot be determined whether $(A \cap B)$ is empty, leave the original type as is and do not remove the unit clause. This SET-TYPE rule need only be applied at the beginning and after each new unit clause is generated.

6.2 SOLVE< Rule

For a clause of the form

$$D \vee (x \notin A)$$

(1) if x has type A then add D to the list of clauses, (2) if x is a variable and x does not occur in A, then assign the type A to x and add D to the list of clauses.

6.4 <u>TRANSITIVE Rule</u>

When attempting to resolve two clauses of the form $((a < b) \vee A)$ and $((a' < c) \vee B)$, where $a_\sigma = a'_\sigma$ for some substitution σ, if (SOLVE b c) is true, then add the resolvent $(A \vee B)_\sigma$ to the list of clauses.

6.5 <u>SOLVE= Rule</u>

For a clause of the form

$$D \vee (A \neq B),$$

if (SOLVE= A B) is true, with the value σ, then add D_σ to the list of clauses.

6.6 When attempting to resolve two clauses of the form

$$((a = b) \vee A) \text{ and } ((c \neq d) \vee B),$$

if (SOLVE= (a-c)(b-d)) is true, with value σ, then add $(A \vee B)_\sigma$ to the list of clauses.

Before going to our limit heuristic rule, we give some examples using the above three rules.

Example 1

$$(0 < a \rightarrow \text{SOME } x\ (0 < x \ \wedge \ x < a))$$

	Clauses	Clause References	Rule	New Type Assignments
1.	$0 < a_0$	From Theorem		NONE
2.	$0 \nless x \vee x \nless a_0$	From Theorem		NONE
3.		1	SET-TYPE	$a_0 \ \langle 0 \infty \rangle$
4.	$x \nless a_0$	2	SOLVE<	$x \ \langle 0 \infty \rangle$
5.	$\square$	4	SOLVE<	$x \ \langle 0 \ a_0 \rangle$

We could have removed $x \nless a_0$ first,

4.	$0 \nless x$	2	SOLVE<	$x \ \langle -\infty a_0 \rangle$
5.	$\square$	4	SOLVE<	$x \ \langle 0 \ a_0 \rangle$

Example 2

$$0 < D_1 \wedge 0 < D_2 \rightarrow \text{SOME } D\ (0 < D \wedge D < D_1 \wedge D < D_2)$$

Clauses	Clause References	Rule	New Type Assignments	
1. $0 < D_1$	From Theorem			
2. $0 < D_2$	From Theorem			
3. $0 \nless D \vee D \nless D_1 \vee D \nless D_2$	From Theorem			
4.	1	SET-TYPE	D_1	$<0\ \infty>$
5.	2	SET-TYPE	D_2	$<0\ \infty>$
6. $D \nless D_1 \vee D \nless D_2$	3	SOLVE<	D	$<0\ \infty>$
7. $D \nless D_2$	6	SOLVE<	D	$<0\ D_1>$
8. □	7	SOLVE<	D	(intersection $<0\ D_1><0\ D_2>$)

At steps 7 and 8, SOLVE< required the knowledge that D_1 and D_2 both had type $<0\infty>$.

Example 3

$$(x \in \underline{P} \wedge x \in \underline{N} \longrightarrow x \neq x)$$

Clauses	Clause References	Rule	New Type Assignments	
1. $x \in \underline{P}$	From Theorem			
2. $x \in \underline{N}$	From Theorem			
3. $x = x$	From Theorem			
4.	1	SET-TYPE	x	$<0\infty>$
5. □	2	SET-TYPE		

Example 4

$$(0 < a \wedge 0 < b \rightarrow (\text{SOMEz } (0 < z \;\wedge\; (c < z \rightarrow c < a)$$
$$\wedge\, (d < z \rightarrow d < b)))$$

	Clauses	Clause References	Rule	New Type Assignments
1.	$0 < a_0$			
2.	$0 < b_0$			
3.	$0 \nless z \vee c_0 < z \vee d_0 < z$			
4.	$0 \nless z \vee c_0 < z \vee d_0 \nless b_0$			
5.	$0 \nless z \vee c_0 \nless a_0 \vee d_0 < z$			
6.	$0 \nless z \vee c_0 \nless a_0 \vee d_0 \nless b_0$			
7.		1	SET-TYPE	a_0 <0 ∞>
8.		2	SET-TYPE	b_0 <0 ∞>
9.	$c_0 < z \vee d_0 < z$	3	SOLVE<	z <0 ∞>
10.	$c_0 < z \vee d_0 \nless b$	4	SOLVE<	
11.	$c_0 < z$	9,10	Rule 6.4	z <0 b_0>
12.	$c_0 \nless a_0 \vee d_0 < z$	5	SOLVE<	
13.	$c_0 \nless a_0 \vee d_0 \nless b$	6	SOLVE<	
14.	$c_0 \nless a_0$	12,13	Rule 6.4	z <0 b_0>
15.	□	11,14	Rule 6.4	z (intersection <0 b_0><0 a_0>)

By ordinary resolution we would require at least two axioms,

A1. $(0 < a \land 0 < b \to \text{SOME } z\ (0 < z \land z < a \land z < b))$

A2. $(x < y \land y < w \to x < w)$,

and a long and difficult sequence of resolution steps. This very example occurs as a disguised part of the proofs of most of the limit theorems, and therefore it is important to have an easy proof for it requiring no axioms.

<u>Example 5.</u>

$$(x < -1 \lor 1 < x \to 1 < |x|) .$$

This produces clauses

1. $x_0 < -1 \lor 1 < x_0$

2. $1 \not< |x_0|$

Since there are no unit clauses, we cannot apply SET-TYPE, and SOLVE< cannot handle 2 because there is no type assigned to x_0 . Thus the procedure seems to fail here unless we have more axioms. However, if we are employing the SPLITTING technique (see [1], end of Section 4), we know that resolving 1 and 2 is equivalent to resolving both

1'. $x_0 < -1$	1''. $1 < x_0$
2'. $1 \not< |x_0|$	2''. $1 \not< |x_0|$

(Note that we split Clause 1 since the two literals of 1 have no variable in common.) These are both easy.

3'.	1'	SET-TYPE	x_0	$<-\infty\ -1>$
4'. □	2'	SOLVE<		
3''.	1''	SET-TYPE	x_0	$<1\ \infty>$
4''. □	2''	SOLVE<		

If we do not SPLIT, then two axioms, $(1 < x \rightarrow 0 < x)$ and $(0 < x \rightarrow |x| = x)$ are required.

Ordinary resolution would require six axioms and a lengthly deduction.

<u>6.7 LIMIT-HEURISTIC Rule</u>. When attempting to resolve two clauses of the form

$$((|A| < E') \vee C_1)$$

$$(\sim(|B| < E) \vee C_2) ,$$

try to find a substitution σ which will allow B to be expressed as a non-trivial combination of A ,

$$(B = K \cdot A + L)_\sigma$$

and, if this is possible, add the following new "resolvent" clause to the clause list

$$(\sim(|K| < M) \vee \sim(|A| < E/2 \cdot M) \vee (|L| < E/2) \vee C_1 \vee C_2)_\sigma$$

where M is a new variable with type $<0\ \infty>$.[14]

The first part of 6.7 can be done by (EXTRACT A B). See Section 3.1. EXTRACT produces the desired K, L, and σ, where σ is the most general such substitution.

---

14. Also the variable M is made an additional argument of all skolem functions appearing in (1) which already have at least one argument.

Example 6. Given the clauses

1. $|f(x_1) - L_1| < E_1$

2. $|g(x_2) - L| < E_2$

3. $|f(x) + g(x) - L_1 - L_2| < E$,

where E_1, E_2, x_1, x_2 are variables, and E, E_1, E_2 each has type <0 ∞>.

Using Rule 6.7 on clauses 1 and 2 we get

4. (EXTRACT $(f(x_1) - L_1)(f(x) + g(x) - L_1 - L_2)$)

$= (1 \quad (g(x) - L_2)\ [x/x_1])$ (See Section 3.1).

5. $(\sim(|1| < M) \vee \sim|f(x) - L_1| < E/2\ M \vee \sim|g(x) - L_2| < E/2)$

6. $(\sim|f(x) - L_1| < E/2 \cdot M \vee \sim|g(x) - L_2| < E/2)$

From 5, using the SOLVE< Rule, type M is <1 ∞>.

Using Rule 6.4 on clauses 1 and 6 we first call

7. (SOLVE< E_1 $E/2 \cdot M$)

This results in assigning type <0 $E/2 \cdot M$> to E_1.

8. $(\sim|g(x) - L_2| < E/2)$ 6,7 Rule 6.4

Using Rule 6.4 on clauses 2 and 8 we call

9. (SOLVE< E_2 E/2)

This results in assigning type <0 E/2> to E_2.

10. □ 2,8 Rule 6.4

Example 6. (From the theorem that a function having a derivative at a point is continuous there).

Clauses

1. $\left|\frac{f(a+h) - f(a)}{h} - F'\right| < E_1$

2. $|f(x) - f(a)| \nless E$

3. $|x - a| < D$

where h, D and E_1 are variables, and the other terms have type **R**.

In attempting to resolve 1 and 2, the limit heuristic Rule 6.7, employs EXTRACT to obtain

$$(f(x) - f(a)) = [h \cdot (\frac{f(a+h) - f(a)}{h} - F') + h \cdot F']_\sigma$$

where σ is the substitution $[(x-a)/h]$. It therefore produces the new clause

4. $|x - a| \nless M \vee \left|\frac{f(x) - f(a)}{x - a} - F'\right| \nless \frac{E}{2 \cdot M} \vee |(x - a) \cdot F'| \nless \frac{E}{2}$

where M is a new variable of type $<0\ \infty>$. Rule 6.4 applied to clause 4, gives

5. $\left|\frac{f(x) - f(a)}{x - a} - F'\right| \nless \frac{E}{2 \cdot M} \vee |(x - a)\ F'| \nless \frac{E}{2}$

and D is assigned type $<0\ M>$. Rule 6.4 applied to 5 gives

6. $|(x - a) \cdot F'| \nless \frac{E}{2}$

and E_1 is assigned type $<-\infty\ \ E/2 \cdot M>$.

Again the limit heuristic Rule 6.7 is used on clauses 3 and 6. EXTRACT yields

$$(x - a) \cdot F' = F' \cdot (x - a) + 0$$

and the new clause

7. $|F'| \nless M' \ \vee \ |x - a| \nless \frac{E}{4 \cdot M'}$

is produced, where M' is a new variable of type <0 ∞>.

Rule 6.4 is applied to 7 to obtain

8. $|x - a| < \frac{E}{4 \cdot M'}$

and M' is assigned type <|F'| ∞>.

Finally, Rule 6.4 is applied to 8 to yield

9. □ QED.

This final step also assigned to D the type (intersection <-∞ E/4·M'> <0 M>).

Ordinary resolution would require several axioms for this proof and a very long deduction. This example constitutes a part of the proof that the limit of the sum of two functions is the sum of their limits.

7. Comments

One remark of note is that, except for the example on quotients, (mentioned below) these limit theorems were proved without the inclusion of axioms (reference theorems). This is desirable because for most automatic theorem proving programs, the axioms have to be selected by humans for each theorem being proved. Of course, we had to include the limit heuristic itself which acts like some axioms, but it does not hinder the proof of other theorems not requiring it,

because it does not release its action unless its need is detected. This is in the spirit of the "Big Switch" mentioned by Newall, Feigenbaum, and others.

It was surprising to us that so many theorems would follow from one heuristic. Will this happen in other areas of mathematics? Can we provide a series of big switches which will handle many areas of mathematics without excessive irrelevant computing? We doubt that it can be so simple, but nevertheless feel that such heuristics should be sought for other areas of mathematics. The success of such a collection of heuristics will depend in great part on the cleverness of the overseer program which directs the use of these heuristics. Hewitt's programming language PLANNER [5] might be well suited for writing such overseer programs, or for improving existing ones.

CALCULATE VERSUS PROVE

One thing that contributed to the success of this effort was the use of the routines SOLVE<, SOLVE=, and SIMPLIFY. The point is that they were used to calculate something rather than prove something. Since proving is inherently harder than calculation, we feel that such routines should be employed as much as possible. Think how difficult it would be in our proofs to employ a set of algebraic simplification axioms instead of using SIMPLIFY. Or suppose that instead of using EXTRACT to give us a linear decomposition, we tried to prove that such a linear decomposition exists. This suggests that more use ought to be made of calculation procedures within the proving mechanisms of automatic theorem provers. For example,

in proving theorems about	we might calculate
derivatives	limits
limits	solutions to equations
differential equations	derivatives
real functions	solutions to equations
measure theory	that two sets are equal
algebraic topology	group theoretic results
any field	a most general unifier

The unification algorithm is such an example, and it revolutionized automatic theorem proving when J. A. Robinson defined its role in resolution. A source of power to a mathematician is his ability to leave to calculation those things that can be calculated and thereby free his mind for the harder task of finding inferences.

TYPES

The use of membership types also helped considerably in proving these limit theorems. It is as if in proving,

(1) $\text{SOME } x\ (P(x) \wedge Q(x))$

we first find A, the set of all x for which P(x), and assign A as the type of x, and then find B the set of all x for which Q(x), and if $(A \cap B)$ is not empty, assign it as the type of x, and declare (1) to be true. This allows a maximum amount of freedom in the proving of Q(x) after P(x) has been proved; indeed x remains a <u>variable</u>, even though restricted, in the proof of Q(x).

This procedure worked well in our examples because linear inequalities are so easy to solve. We do not recommend that such a procedure should be used in all other situations, when theorems of type (1) are being proved, because it may be too difficult (or unnecessary) to solve for A the set of all x for which P(x) is true, before proving Q(x). We <u>do</u> suggest however that a procedure be followed that leaves x as a <u>variable</u>, though restricted, after P(x) has been proved and while Q(x) is being proved. Type theory might help attain such an objective.

Our present program will not prove limit theorems involving quotients, such as

(1) $\lim_{x \to a} f(x) = L \quad L \neq 0 \qquad \lim_{x \to a} \frac{1}{f(x)} = \frac{1}{L}$,

without the help of some axioms (see Example 5, Section 5). However, no axioms are needed for the proof of (1) if we add another heuristic to the program which is similar to the limit heuristic, but which is based upon the inequality

$$|x| - |y| \le |x-y|$$

instead of the triangle inequality

$$|x+y| \le |x| + |y|,$$

upon which the limit heuristic is based. In fact, it might be desirable to develop a more general heuristic, which not only encompasses both ideas, but also tries to attain such objectives as bounding an expression, e.g.

$$|g(x)| < M, \text{ for some } M,$$

and making an expression small, e.g.

$$|f(x) - L| < E, \text{ for a given } E.$$

Finally, it should be mentioned that the routines described in Section 2 are meant for general use in analysis and not just for limit theorems. It is hoped that routines of this kind can be used to make up an analysis prover in which relatively simple heuristics can be added for great effect.

References

1. W. W. Bledsoe, "Splitting and Reduction Heuristics in Automatic Theorem Proving," to appear in International Journal of Artificial Intelligence.

2. J. A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle," Jour. ACM. (12) (January, 1965), pp. 23-41.

3. Hao Wang, "Toward Mechanical Mathematics," IBM Jour. Res. Dev., 4, pp. 2-22.

4. S. C. Kleene, "Introduction to Metamathematics," Princeton Toronto New York, 1952.

5. Carl Hewitt, "Planner," M.I.T. A.I. Memo No. 168, August 1970.

6. Dag Prawitz, "An Improved Proof Procedure," Theoria, 26, pp. 102-39.