COMPUTER PROOFS OF LIMIT THEOREMS

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

Massachusetts Institute of Technology
and the University of Texas

ABSTRACT: Some relatively simple concepts have been developed which, when incorporated into existing automatic theorem proving programs (including those using resolution), enable them to prove efficiently a number of the limit theorems of elementary calculus, including the theorem that differentiate functions are continuous. These concepts include: (1) A limited theory of types, to designate whether a given variable belongs to a certain interval on the real line, (2) An algebraic simplification routine, (3) A routine for solving linear inequalities, applicable to all areas of analysis, and (4) A "limit heuristic", designed especially for the limit theorems of calculus.

jL. 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 the limit of 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 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 LIJ, 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 ∈ A).

Let <a b> denote the open interval from a to b, R = <-∞ ∞> , P = <0 ∞> , and N = <-∞ 0> . We are primarily interested in interval types, including the types R, P, and N. Thus in trying to prove

(0 < x → Q(x))

we would assign the type P (or <0 ∞>) to x and then try to prove Q(x).

For example, suppose that we are to prove

(1) (0 < b → SOME x (0 < x ∧ x < b)).[1]

One valid approach is to solve for x in

(2) (0 < b → 0 < x)

and then try to verify

(3) (0 < b → x < b)

for that same x. But using matching we would get

---

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)".

as a solution of (2) the substitution [b/x],[2] and require (0 < b -> b < b) 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 type <0 »> to b, and then try to prove

(4) SOME x (0 < x ^ x < b) .

We first solve

(5) (0 < x)

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

(6) (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 variable 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	New Type of x given by (SOLVE< A B)
1.	x	1	<-∞ 1>
2.	x Type x is <0 ∞>	1	<0 1>
3.	0	1	(the value T is returned)
4.	x·a+c Type a is <0 ∞>	(-x+d)	<-∞ $(\frac{d}{1+a} - \frac{c}{1+a})$>
5.	x Type x is <0 D_2> Type D_1 is <0 ∞> Type D_2 is <0 ∞>	D_1	(intersection <0 D_1><0 D_2>)

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

6. x <a ←∞ b>

Type x is <0 »>

Type a is <-« 0>

Type b is <0 ∞>

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.

Actually, 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

2. We follow the usual practice of denoting a substitution by a 11st [b!/a_{1f} b_2/a_2»... ,b_n/a_n] where each ai is to be replaced by the corresponding Bi.

3. A skolem expression is a term whose main function symbol is a skolem function, cf. footnote 11 in Section 4 which describes the elimination of quantifiers by the introduction of skolem functions.

solve nonlinear inequalities, but these were not found necessary for proving the examples reported here.

2.2 SOLVE=. 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.

2.3 SET-TYPE. This is a subroutine which assigns types to certain skolem expressions. If a formula of the form (A B) is in a conjunctive position of E (i.e., E can be expressed as ((A ε B 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 ε B) from E. If A already has type C, then SET-TYPE assigns the intersection (B∩ C) as the type of A, if (BO C) is non-empty. If (B n C) is empty it returns E. If (B ∩ 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 A (x ε P, A (B - y c R)))

then (SET-TYPE E), assigns F^ as the type of x, and returns

(1) (A A (B > y , R)) .

If, in this example, x already had type F*, then P^ is assigned as the new type of x; if it already had type <-I 1> then it assigns type <0 1> to x; if it already had type <-«■ -1> then it returns (A A (X ε P v (B - y r R))) .

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

(A < 0 A (B < 1 V O)

then (SET-TYPE E) assigns type <-» 0- to A and returns

(B < 1 ∨ C) ,

and if E is

(A < B ∧ C)

then (SET-TYPE E) assigns type <-∞ B> to A, and type <A <*>> to B and returns C. Similarly, (SET-TYPE (A *f* 0)) can be made to assign type (union < -. 0><0 *>) 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--). 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

2- 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 o which will allow B_o to be expressed as a non-trivial combination[5] of A_0, (B = K-A + L)o, 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 find such a o and prove A, B, and C, then we would have

$$\begin{aligned} B_0 &= K\text{-}A + L_o \\ &\leq (|K|\text{-}|A| + |L|)_o \\ &< M\text{-}E/2M + E/2 \\ &= E . \end{aligned}$$

Of course, this is based on the triangle inequality, and uses the fact that 1/2 + 1/2 = 1, M-I/M =1 for M > 0 , etc.

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

4. The notation B_0 denotes the result of applying the substitution o to B.

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

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 $\alpha = [x/x']$

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

can be expressed as a combination of

$$(f(x') - L_1)_{\alpha} ,$$

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 loqically divided. In such applications the user must provide a heuristic which will enable the computer to determine how to qet a partial result from Hj and leave a reaminder 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 o for which B_n can be expressed as a non-trival combination of A_0 ,

$$(B = K\cdot A + L)_0$$

then (EXTRACT A B) returns (K L n), where o 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. $(\text{EXTRACT } A \ (K\cdot A+L)) = (K\ L\ T)$.[6]

2. $(\text{EXTRACT } A(t)\ A(t_0)) = (1\ 0\ [t_0/t])$.

3. $(\text{EXTRACT } (f(x)-L_1)\ (f(x_0)+g(x_0) - (L_1+L_2))) = (1\ \ (g(x_0) - L_2)\ \ [x_0/x])$.

4. $(\text{EXTRACT } (f(x)-L_1)\ \ (f(x_0)\cdot g(x_0) - L_1\cdot L_2) = (g(x_0)\ \ (L_1\cdot g(x_0) - L_1\cdot L_2)\ \ [x_0/x])$.

5. $(\text{EXTRACT } (f(x)-L_1)\ \ (\frac{1}{f(x)} - \frac{1}{L_1}))$[7]

$= (- \frac{1}{f(x)\cdot L_1}\ \ 0\ \ T)$.

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

$= ((x-a)\ \ (x-a)\cdot F'\ \ [h/(x-a)])$.

7. $(\text{EXTRACT } ((x_0-a)\ \ (x_0^2 - a^2)) = ((x_0+a)\ 0\ T)$.

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

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

8. (EXTRACT $(a \cdot x_0+c)$ $(b \cdot x_0+d)$)

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

9. (EXTRACT $(a \cdot x_0+c)$ $(b \cdot y_0+d)$) = 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 the quotient of two functions. Example 6 is used in proving that a differentiate function is continuous.

Suppose there is a substitution α and an expression x such that, Aα and Bα 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_a and B_0 can be re-expressed as

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

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

and (EXTRACT A B) returns the value $(\frac{b}{a}$ $(d - \frac{b \cdot c}{a})$ $\sigma)$. 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 * 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 o such that (R -+ E) is true. If $_0$ Is the empty substitution then[a] 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 ⊢ C, R]	
If H ≡ C , then	T
If there is a substitution σ which unifies H and C, (i.e., $H_\sigma \equiv C_\sigma$) then	σ
2. [A ∧ B, R]	
2.1 [A,R] yields σ1	
If and 2.2 [$B_{\sigma 1}$,R] yields σ2	then $(\sigma 1 \circ \sigma 2)$[8]
3. [A ∨ B, R]	
If [A,R] yields σ1, then	σ1
If [B,R] yields σ2, then	σ2
4. [A → B) → C, R]	
4.1 [B → C, R] yields σ1	
If and 4.2[9] [R → $A_{\sigma 1}$, NIL] yields σ2	then $(\sigma 1 \circ \sigma 2)$
This rule is commonly known as backwards chaining.	
5. [H ⊢ (A → B), R]	[H ∧ A → B, R]
6. [A ∨ B → C, R]	
6.1 [A → C, R] yields σ1	
If and 6.2 [$B_{\sigma 1}$ → C, R] yields σ2	then $(\sigma 1 \circ \sigma 2)$

Table 1
Some of the rules defining IMPLY.

---

8. When we use an expression like "[A,R] yields σ", it is to be understood that we also mean than σ is not NIL. $(\sigma 1 \circ \sigma 2)$ denotes $(\sigma 1 \sigma 2 \cup \sigma 2)$.

9. If 4.2 fails but [R → $A_{\sigma 1}$ ∨ C, NIL] yields σ3 or [(A → B) ∧ R → $A_{\sigma 1}$, NIL] yields σ3, then IMPLY returns $(\sigma 1 \circ \sigma 3)$.

7. [H ⊢ A ∨ B, R]
   If [H ⊢ A, R] yields σ1 then σ1
   If [H ⊢ B, R] yields σ2 then σ2

8. [H ⊢ A ∧ B, R]
   8.1 If [H ⊢ A, R] yields σ1 and then (σ1 ∘ σ2)
   8.2[10] [H ⊢ B_{σ_1}, R] yields σ2

9. [A ∧ B ⊢ C, R]
   If [A ⊢ C, R ∧ B] yields σ1 then σ1
   If [B ⊢ C, R ∧ A] yields σ2 then σ2

10. [H ⊢ ∿A ∨ B, R] [H ∧ A ⊢ B, R]

11. [∿A ∧ B ⊢ C, R] [B ⊢ A ∨ C, R]

12. [∿H ⊢ C, R] [R ⊢ C ∨ H, NIL]

13. [H ⊢ ∿C, R] [H ∧ C ⊢ NIL, R]

14. [A = B ⊢ C, R] [R' ⊢ C', NIL]
    where R' and C' are gotten by replacing B by A in R and in C.

15. [H ⊢ A = B, R] (SOLVE= A B)
    (i.e., if there is a substitution α, which unifies A and B, then return a)

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].[11] A call is then made to (IMPLY E NIL).

10. It is possible for IMPLY to yield a substitution which assigns to a variable x more than one value: a/x, b/x, a ≠ b. If this happens and if IMPLY tries to substitute for x in another expression (as it might do using Rule 8.2, 6.2, 2.2, or 4.2), then IMPLY returns NIL.

If Rule 8.2 fails on the ≠I given by Rule 8.1 (i.e., if [H -▶ B , R] returns NIL), then the program "backs up" and recomputes 8.1 trying to find another solution σ1' of [H -+ A, R] for which [H -* B .i,R] can succeed. A similar backing up proceeaure is used in Rules 2, 4, and 6.

11. 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

For example the formula

(1) (P(y) ∧ ALL x (P(x) → Q(x)) → Q(y))

is first converted to the skolem form

(P(y_0) ∧ (P(x) → Q(x)) → Q(y_0)),

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

1. (IMPLY (P(y_0) ∧ (P(x) → Q(x)) → Q(y_0)) NIL)
   1.1 (IMPLY (P(y_0) → Q(y_0)) (P(x) → Q(x)))
   (by Rule 9). This fails.
   1.2 (IMPLY ((P(x) → Q(x)) → Q(y_0)) P(y_0))
   (by Rule 9).
   1.2.1 (IMPLY (Q(x) → Q(y_0)) P(y_0))
   (by Rule 4.1).
   This yields σ = [y_0/x] by Rule 1.
   1.2.2 (IMPLY (P(y_0) → P(x)$_\sigma$) NIL) Rule 4.
   This yields T by Rule 1.

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

For the example

(SOME x (ALL y P(x,y)) → ALL s (SOME t P(t,s)))

the skolem form is

(P(x_0, y) → P(t, s_0)).

A call is made to IMPLY

(IMPLY (P(x_0, y) → P(t, s_0)) NIL)

which yields [x_0/t, s_0/y] by Rule 1. QED.

In trying to prove the non-theorem

(ALL y (SOME x P(x, y)) → SOME t (ALL s P(t, s))),

the skolem form is

(P((x y), y) → 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. 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), P((x y), (s (x y)))

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

When attempting to prove an expression E with the help of axioms, A_1, A_2, ..., 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 skolem form of

(A_1 ∧ A_2 ∧ . . . ∧ A_n → E).

In the operations described in Table 3, a resemblance can be seen between the method of

simply the application pf a skplem function to certain arguments, with no ambiguity, but as an aid to memory, the skolem function is named v.

Gentzen sequents (cf. Kleene's G3 [4]) and the subgoals which IMPLY sets up. The technique of 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 must add the following rule to Table 1.

16. [|A| < E' → |B| < E, R]

If
16.0 (EXTRACT A B) is $(K\ L\ \sigma)$
(i.e., $(B = K\cdot A + L)_\sigma$),
and if
16.1 $[R \rightarrow (|K| < M)_\sigma,$ NIL] yields $\sigma 1$[12],
for some variable M[13] and if
16.2 $[|A| < E' \rightarrow |A| < E/2\cdot M, R]_{(\sigma\circ\sigma 1)}$[8]
yields $\sigma 2$, and if
16.3 $[R \rightarrow (|L| < E/2)_{(\sigma\circ\sigma 1\circ\sigma 2)},$ NIL]
yields $\sigma 3$,
then return the value $(\sigma\circ\sigma 1\circ\sigma 2\circ\sigma 3)$[14]

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

17. [H → a < b, R] (SOLVE< a b)

18. [a < b → a' < c, R]
$[(b < c)_\sigma \vee (b = c)_\sigma, R]$
if there is a substitution for which $(a = a')_\sigma$.

19. [H → A ε B, R]
If A has type B then return T.

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

These five reles 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 a type to an expression A when it appears in the form (A > 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, i.e., 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

(Lim f a L_1 ∧ Lim g a L_2
→ Lim (f·g) a (L_1·L_2))

The definition of limit is used to obtain

((a ε R ∧ L_1 ε R ∧ f ε FRR ∧
ALL E_1 (0 < E_1 → SOME D_1 (0 < D_1 ∧
ALL x_1 (x_1 ε R ∧ x_1 ≠ a ∧ |x_1-a| < D_1
→ |f(x_1) - L_1| < E_1))))
∧ (a ε R ∧ L_2 ε R ∧ g ε FRR ∧
ALL E_2 (0 < E_2 → SOME D_2 (0 < D_2 ∧
ALL x_2 (x_2 ε R ∧ x_2 ≠ a ∧ |x_2-a| < D_2
→ |g(x_2) - L_2| < E_2))))
→
(a ε R ∧ (L_1·L_2) ε R ∧ (f·g) ε FRR ∧
ALL E (0 < E → SOME D (0 < D ∧
ALL x (x ε R ∧ x ≠ a ∧ |x -a| < D
→ |(f·g)(x) - L_1·L_2| < E)))))

The first three parts of the conclusion, (a L R), (U-Lp) e R, and (f-g) t FRR are proved by the program using the hypotheses of the theorem and the closure properties of R_ and FRR.

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

((a) ε (R) ∧ (L_1) ε (R) ∧ (f) ε (FRR) ∧
(0 < E_1 → (0 < (D_1 E_1) ∧
(x_1 ε (R) ∧ x_1 ≠(a) ∧ |x_1 - (a)| < ($D_1$$E_1$)
→ |(f)(x_1) - (L_1)| < E_1))))

∧((a) ε (R) ∧ (L_2) ε (R) ∧ (g) ε (FRR) ∧
(0 < E_2 → (0 < (D_2 E_2) ∧
(x_2 ε (R) ∧ x_2≠(a) ∧ |x_2 - (a)| < ($D_2$$E_2$)
→ |(g)(x_2) - (L_2)| < E_2))))

(i)
→

12. In case K = 1, Step 16.1 is omitted, and M is set to 1 in 16.2.
13. M is given type 0 - and also M is made an additional argument of all skolem functions which already have at least one argument.
14. In case L = 0, Step 16.3 is omitted.

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

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

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

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

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

But 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

$$|f(x) - L_1| < E_1 \quad \text{and} \quad |g(x) - L_2| < E_2.$$

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

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

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

$$|f(x)\cdot g(x) - L_1\cdot L_2|$$
$$= |g(x)\cdot(f(x) - L_1) + L_1\cdot(g(x) - L_2)|$$
$$\leq |g(x)\cdot(f(x) - L_1)| + |L_1\cdot(g(x) - L_2)|$$
$$= |g(x)|\cdot|f(x) - L_1| + |L_1|\cdot|g(x) - L_2|$$
$$< M\cdot E/2\cdot M + M'\cdot\min(M/2,\ E/4\cdot M')$$
$$\leq E/2 + M'\cdot E/4\cdot M' < E. \quad \text{QED.}$$

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 through its handling of types.

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

(IMPLY ($\alpha \wedge \beta \rightarrow \gamma$) NIL)

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

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

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

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

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

SET-TYPE assigns type $<0\ \infty>$ to D_1 and

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

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

Again the program "chains backward" 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 $<0 \infty>$. (Also M is made an additional argument in the skolem expressions $(D_1 \ E_1)$, $(D_2 \ E_2)$, and $(x \ D)$, in accordance with Footnote 13 above. Although these new skolem expressions $(D_1 \ E_1 \ M)$, $(D_2 \ E_2 \ M)$, and $(x \ D \ M)$, will not appear in our descriptions since we are abbreviating them to D_1, D_2, and 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 $< |g(x \ D \ M)| \ \infty>$ as the answer to (111 1). (See Section 2.1).

By Rule 9, the reserve R is set to $(x \neq a \wedge |x - a| < D)$ and

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

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

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

By Rule 9 the program tries

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

after assigning type $< 0 \infty >$ to D_2.

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 $<-\infty \ M/2>$ 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< $\left|\frac{L_2}{L_2}\right|$ M/2) is called, resulting in the type $< 2\cdot\left|\frac{L_2}{L_2}\right|_{\infty}>$ 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 $<-\infty \ M/2>$ at (111 111^2). 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 E/2M), which assigns type $<-\infty \ E/2M>$ to E_1.

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

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

By Rule 9, the reserve R is set to $(x \neq a \wedge |x - a| < D)$, and the current subgoal becomes

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

The program chains backward twice.

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

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

Since (EXTRACT $(g(x) - L_2)$ $(g(x)\cdot L_1 - L_1L_2)$) 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 $(intersection <0\ M/2^2> <-\infty\ E/4\cdot M'^2>)$. 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 $E/4\cdot M'$ is the smaller, the intersection 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 $<0\ \infty>$.

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 (11) have been established, giving as an answer to (11) 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 $(intersection\ <0\ \ D_2> <-\infty\ D_1>)$. Recall that D previously had type $<0\ \ D_2>$.

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 $<0\ \ E/2\cdot M>$. E_1 previously had type $<-\infty\ E/2\cdot M>$. QED.

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	$<0\ \ E/2\cdot M>$
E_2	$(intersection\ <0\ \ M/2> <-\infty\ E/4\cdot M'>)$
D	$(intersection\ <0\ \ D_2> <-\infty\ D_1>)$
M	$<2\cdot|L_2|\ \infty>$
M'	$<|L_1|\ \infty>$.

This proof may seem long and drawn out but these are essentially the steps a human prover would have to follow in finding and exhibiting 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. Rule reference numbers are sometimes given to the right of formulas along with new type assignments.

Example 2. (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. $(0 < E_1 \rightarrow (0 < D_1 \wedge (x_1 \in \underline{R} \wedge x_1 \neq a \wedge |x_1 - a| < D_1 \rightarrow |g(x_1) - g(a)| < E_1)))$
   $(0 < E_2 \rightarrow (0 < D_2 \wedge (x_2 \in \underline{R} \wedge x_2 \neq g(a) \wedge |x_2 - g(a)| < D_2 \rightarrow |f(x_2) - f(g(a))| < E_2)))$
   $(0 < E \rightarrow (0 < D \wedge (x \in \underline{R} \wedge x \neq a \wedge |x - a| < D \rightarrow |f(g(x)) - f(g(a))| < E)))$

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 NO. and NEW TYPE ASSIGNMENTS
4. $(H_3^{15} \rightarrow 0 < D)$	5, 8
	E $<0\ \ \infty>$
5. (SOLVE< 0 D)	17
	D $<0\ \ \infty>$

15. The notation H_i is used to denote the hypothesis of Step i.

6. $(H_3 \wedge x \neq a \wedge |x-a| < D \rightarrow |f(g(x)) - f(g(a))| < E)$ 8.2 x 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) 7, 17 E_2 <-∞ E>

10. $(H_6 \rightarrow 0 < E_2)$, a condition from Step 7.

11. (SOLVE 0 E_2) 17 E_2 <0 E>

12. $(H_6 \rightarrow x_2 \in \underline{R} \wedge x_2 \neq a \wedge |x_2 - a| < D_2)_\sigma$, where $\sigma = [g(x)/x_2]$. A condition from Step 7

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. $(|x - a| < D \rightarrow |x - a| < D_1)$ 9

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

Example 3. (Differentiable functions are continuous).

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

$$\text{then } \lim_{x \to a} f(x) = f(a).$$

1. (Derivative f a F' → Continuous f a)

2. (Lim q 0 F' → Lim f a f(a)), where q(h) is the difference quotient $\frac{f(a+h) - f(a)}{h}$.

3. $(0 < E_1 \rightarrow (0 < D_1 \wedge (h \in \underline{R} \wedge h \neq 0 \wedge |h| < D_1 \rightarrow \left|\frac{f(a+h) - f(a)}{h} - F'\right| < E_1)))$

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

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

⋮

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

(x has been given type $\underline{R}$)

5. $(\left|\frac{f(a+h) - f(a)}{h} - F'\right| < E_1 \rightarrow |f(x) - f(a)| < E)$ using Rule 4 and others

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 $\sigma = [(x - a)/h]$.

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

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

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

9. $(\left|\frac{f(x) - f(a)}{x - a} - F'\right| < E_1 \rightarrow \left|\frac{f(x) - f(a)}{x - a} - F'\right| < E/2{\cdot}M)$, Rule 16.2

10. (SOLVE E_1 E/2·M) 18, 17 E_1 <-∞ E/2M> Rule 16.3

11. $(H_4 \rightarrow |(x - a)\cdot F'| < E/2)$

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

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

13. $(H_4 \rightarrow |F'| < M)$ Rule 16.1

14. (SOLVE |F'| M) 17 M <|F'| ∞>

15. $(|x - a| < D \rightarrow |x - a| < E/4{\cdot}M')$, 16.2 etc.

16. $(x \neq a \wedge |x - a| < D \rightarrow h \in \underline{R} \wedge h \neq 0 \wedge |h| < D_1)_\sigma$, 4.2 a condition for Step 5. $\sigma = [(x - a)/h]$.

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

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

18. $(x \neq a \rightarrow x - a \neq 0)$, from Step 16. 8, 9

19. $(x - a = 0 \rightarrow x = a)$ 12, 13

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

21. $(|x - a| < D \rightarrow |x - a| < D_1)$ 8 from Step 16

22. (SOLVE D D_1) 17, 18

D is assigned type (intersection <0 E/4·M'> D_1). QED.

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

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

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.

First SET-TYPE assigns type <0 ∞> to E. Then

3. (0 < D) Rule 8.1

4. (SOLVE· 0 D) Rule 17
D 0 ·

5. $(x \neq a \wedge |x - a| \cdot D \rightarrow |x \cdot x - a \cdot a| \cdot E)$ 8.2,5
from Step 2. x is assigned type R.

6. $(|x - a| \cdot D \cdot |x \cdot x - a \cdot a| \cdot E)$ 9

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

7. $(H_5 \cdot |x+a| \cdot M)$ 16.1
The limit heuristic is used again, (EXTRACT (x-a) (x+a)) yields (1 2·a T).

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

9. (SOLVE· D M/2) 18, 17
D 0 M/2

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

11. (SOLVE· |2·a| M/2) 17
M <2·|2·a| ··

12. $(|x - a| \cdot D \rightarrow |x - a| \cdot E/2 \cdot M)$ 16.2
from Step 6.

13. (SOLVE· D E/2·M) 17
D is assigned type
(intersection ·0 M/2· ·-·· E/2·M·). QED.

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

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

2. $(0 \cdot E_1 \rightarrow (0 \cdot D_1 \wedge (x_1 \cdot \underline{R} \wedge x_1 \neq a \wedge |x_1 - a| \cdot D_1 \rightarrow |f(x_1) - L| \cdot E_1))) \wedge L \neq 0 \rightarrow (0 \cdot E \rightarrow (0 \cdot D \wedge (x \cdot \underline{R} \wedge x \neq a \wedge |x - a| \cdot D \cdot \left|\frac{1}{f(x)} - \frac{1}{L}\right| \cdot E)))$

⋮

3. $(|f(x_1) - L| \cdot E_1 \rightarrow \left|\frac{1}{f(x)} - \frac{1}{L}\right| \cdot E)$
The limit heuristic (Rule 16) is applied, (EXTRACT $(f(x_1)$ -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) $(H_2 \rightarrow \left|\frac{-1}{L \cdot f(x)}\right| \cdot M)$, 16.1

and

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

Subgoal (2) is easily established by assigning type E/2-M> to E, , but (1) presents difficulty. In fact the program is unable to give a proof of (1) 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 expldined 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.
Tor each unit clause of the form (x ■ A), where x is a skoleni 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 ■--a to x. Similarly for unit clauses of the form (b x) assign type -b - to x. In each of these cases, remove the unit clause. If x already has a type B and we are try ing to assign it a new type A, then assign the type (A ∩ B) if it is non-empty; if (A ∩ B) is empty, add the empty clause (i.e., the proof is finished); if it cannot be determined whether (A ∩ B) is empty, leave the original type as is and do not remove the unit clause. This SET-TYPE rule need only beapplied at the beginning and after each new unit clause is generated.

6.2 MEMBER Rule.
For a clause of the form

$$D \vee (x \not\in A)$$

(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.

6.3 TRANSITIVE Rule.
When attempting to resolve two clauses of the form $((a \cdot b) \vee A)$ and $((a' \not\in c) \vee B)$, where $a \sigma = a' \sigma$ for some substitution σ, if (SOLVE< $^{\sigma}$b c) $^{\sigma}$ yields σ', then add the resolvent $(A \vee B)_{\sigma\sigma'}$ to the list of clauses.

6.4 SOLVE< Rule.
For a clause of the form

$$D \vee (A \not\in B),$$

if (SOLVE< A B) yields the value σ, then add D_σ to the list of clauses.

6.5 SUBTRACTION Rule.
When attempting to resolve two clauses of the form

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

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

6.6 SOLVE= Rule.
For a clause of the form

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

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

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

Example 1. (0 < a → SOME x (0 < x ∧ x < a)).

Clauses	Clause Reference	Rule and new Type Assignments
1. $0 < a_0$	}	From Theorem
2. $0 \not< x \vee x \not< a_0$	}	From Theorem
3.	1	SET-TYPE a_0 <0 ∞>
4. $x \not< a_0$	2	SOLVE< x <0 ∞>
5. □	4	SOLVE< x <0 a_0>.
We could have removed $x \not< a_0$ first,		
4. $0 \not< x$	2	SOLVE< x <-∞ a_0>
5. □	4	SOLVE< x <0 a_0>

Example 2.

(0 < D_1 ∧ 0 < D_2 → SOME D (0 < D ∧ D < D_1 ∧ D < D_2))

1. $0 < D_1$	}	From Theorem
2. $0 < D_2$	}	From Theorem
3. $0 \not< D \vee D \not< D_1 \vee D \not< D_2$	}	From Theorem
4.	1	SET-TYPE D_1 <0 ∞>
5.	2	SET-TYPE D_2 <0 ∞>
6. $D \not< D_1 \vee D \not< D_2$	3	SOLVE< D <0 ∞>
7. $D \not< D_2$	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 ∞>.

Example 3. (x ε P ∧ x ε N → x ≠ x)

1. x ε P	}	From Theorem (x is a skolem constant)
2. x ε N	}	From Theorem (x is a skolem constant)
3. x = x	}	From Theorem (x is a skolem constant)
4.	1	SET-TYPE x <0 ∞>
5. □	2	SET-TYPE

Example 4.

(0 < a ∧ 0 < b → (SOME z (0<z ∧ (c<z → c<a) ∧ (d<z → d<b)))

1. 0 < a	}	From Theorem. a, b, c, d are skolem constants. z is a variable.
2. 0 < b	}	From Theorem. a, b, c, d are skolem constants. z is a variable.
3. $0 \not< z \vee c < z \vee d < z$	}	From Theorem
4. $0 \not< z \vee c < z \vee d \not< b$	}	From Theorem
5. $0 \not< z \vee c \not< a \vee d < z$	}	From Theorem
6. $0 \not< z \vee c \not< a \vee d \not< b$	}	From Theorem
7.	1	SET-TYPE a <0 ∞>
8.	2	SET-TYPE b <0 ∞>
9. c < z ∨ d < z	3	SOLVE< z <0 ∞>
10. c < z ∨ d ≮ b	4	SOLVE<
11. c < z	9,10	Rule 6.3 z <0 b>
12. c ≮ a ∨ d < z	5	SOLVE<
13. c ≮ a ∨ d ≮ b	6	SOLVE<
14. c ≮ a	12,13	Rule 6.3 z <0 b>
15. □	11,14	Rule 6.3 z (intersection <0 b><0 a>).

By ordinary resolution Example 4 would require at least two axioms,

A1. (0 < a ∧ 0 < b → SOME z (0 < z ∧ z < a ∧ z < b))

A2. (x < y ∧ y < w → 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.

We now give the LIMIT-HEURISTIC Rule.

6.7 LIMIT-HEURISTIC Rule.

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 for some substitution σ, then add the following new "resolvent" clause to the clause list

$$(1) \quad (\sim(|K| < M) \vee \sim(|A| < E/2 \cdot M) \vee \sim(|L| < E/2) \vee C_1 \vee C_2)_\sigma,$$

where M is a new variable with type <0 ∞>[16].

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.

---

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

Example 5. Given the clauses

1. $|f(x_1) - L_1| < E_1$
2. $|g(x_2) - L_2| < E_2$
3. $|f(x) + g(x) - L_1 - L_2| \not< E$,

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

Rule 6.7 is used on clauses 1 and 3;
(EXTRACT $(f(x_1)-L_1)$ $(f(x)+g(x)-L_1-L_2)$)
yields (1 $(g(x) - L_2)$ $[x/x_1]$) (see Section 3.1);
and the following resolvent is produced

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

5.	$\sim(\|f(x) - L_1\| < E/2\cdot M) \vee \sim(\|g(x) - L_2\| < E/2)$	4	SOLVE< M <1 ∞>
6.	$\sim(\|g(x) - L_2\| < E/2)$	1,5	Rule 6.3 E_1 <0 E/2M>
7.	□	2,6	Rule 6.3

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

1. $\left|\frac{f(a+h) - f(a)}{h} - F'\right| < E_1$
2. $|f(x) - f(a)| \not< E$
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 express

$$(f(x) - f(a))$$

as

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

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

4. $|x - a| \not< M \vee \left|\frac{f(x) - f(a)}{x - a} - F'\right| \not< E/2\cdot M \vee |(x - a)\cdot F'| \not< E/2$

where M is a new variable of type <0 ∞>. Rule 6.3 applied to 3 and 4 gives

5. $\left|\frac{f(x) - f(a)}{x - a} - F'\right| \not< E/2M \quad |(x - a)\cdot F'| \not< E/2$

and D is assigned type <-∞ M>. Rule 6.3 applied to 1 and 5 gives

6. $|(x - a)\cdot F'| \not< E/2$

and E_1 is assigned type <-∞ E/2·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. $|F'| \not< M' \vee |x - a| \not< E/4\cdot M'$

is produced, where M' is a new variable of type <0 ∞>. Rule 6.4 is applied to 7 to obtain

8. $|x - a| \not< E/4\cdot M'$

and M' is assigned type <|F'| ∞>.

Finally, Rule 6.3 is applied to 3 and 8 to yield □. 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.

The clause (1) which is added by the Limit-heuristic Rule 6.7 can be thought of as a kind of "catalyst" clause, because it speeds up the derivation of □ (without the necessity for additional axioms). It might be useful to produce similar catalyst clauses in other situations where difficult proofs are required.

7. Comments.

One remark is that, except for the example on quotients, (mentioned again 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 heuristics with 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] or the Stanford Research Institute language QA4 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 these routines 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

in place of the routine SIMPLIFY. Or suppose that instead of using EXTRACT to give a decomposition, we tried to prove that such a 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 and inequalities
differential equations	derivatives
real functions	solutions to equations
measure theory	that two sets are equal
algebraic topology	group theoretic results
anything	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.

MEMBERSHIP TYPES

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

(2) SOME x (P(x) A Q(x))

we first find A, the set of all x for which P(x) is true and assign A as the type of x, and then find B the set of all x for which Q(x) is true and if (AAB) is not empty, assign it as the type of x, and declare (2) 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 variable, even though restricted, in the proof of Q(x). This idea is somewhat related to constraint methods used by Fikes in [7].

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 (2) 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 do suggest however that a procedure be followed that leaves x as a variable, 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

(3) $\lim_{x \to a} f(x) = L \wedge L \neq 0 \rightarrow \lim_{x \to a} \frac{1}{f(x)} = \frac{1}{L}$,

without the help of some axioms (see Example 5, Section 5). However, no axioms are needed for the proof of (3) if we add another heuristic to the program which is similar to the limit heuristic, but which is based upon the inequality

$$|x| - |y| \leq |x - y|$$

instead of the triangle inequality

$$|x + y| \leq |x| + |y|,$$

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.,

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

and making an expression small, e.g.,

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

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

8. Acknowledgement. The 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 Defence, and was monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0002, and was also supported by National Institutes of Health Grant GM 15769-03.

References.

1. W. W. Bledsoe, "Splitting and Reduction Heuristics in Automatic Theorem Proving," Artificial Intelligence 2 (1971) pp 55-77.

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, D. van Nostrand, 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.

7. R. E. Fikes, "REF-ARF: a system for solving problems stated as procedures," Artificial Intelligence 1 (1970) pp.27-120.