Discussions on Automatic Theorem Proving

W. W. Bledsoe

November 1973

University of Texas, Austin, Texas

ATP-10

Nov 1973

Discussions on Automatic Theorem Proving

W. W. Bledsoe

This is a set of notes and ideas which were presented at a talk in the Department of Computational Logic, University of Edinburgh, Scotland, June 13, 1973.

Some of these ideas have been presented before [2], but many were developed at the University of Edinburgh during a visit there, May 15 - June 14, 1973. During this period many discussions were held with Bernard Meltzer and his staff in which several suggestions were made regarding these matters.

The talk of June 13 followed an earlier one at Edinburgh in which the work on automatic theorem proving at the University of Texas at Austin, was reviewed [1].

Theme.

Man-machine provers are here to stay. As the user interacts with the computer, the presentation on the terminal scope must be easy for him to follow; he should find it easy to intervene only when his help is needed. Purely syntactic approaches are doomed to failure. Semantic information, in terms of semantic tables and clever programs for their use, must be utilized, along with provisions for easily adding to and changing these tables. Much can be learned by looking at the way humans work and by working with many examples.

Outline

The talk centered around ways of improving automatic theorem proving

programs, with particular emphasis on the man-machine program developed by Bledsoe, et al, at Texas [2].

This is discussed below in a series of topics, A - K.

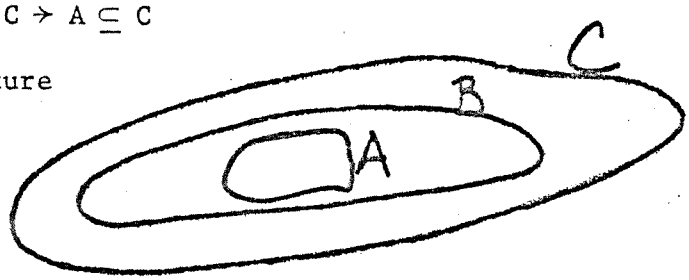
A. Graphic Representations.

This work follows some earlier ideas of Bill Henneman and myself but has also been highly influenced by recent discussions with Alan Bondy [3]. (See also [4]).

When a mathematician is asked to prove the theorem

(1) $A \subseteq B \wedge B \subseteq C \rightarrow A \subseteq C$

he merely draws a picture



and says "behold." The picture (on paper or in his head) carries the transitivity law as a bonus. He does not have to remind himself that (1) holds, he just uses it automatically when the need arises.

Similarly if F, G, H are families

(2) $F \subseteq\subseteq G \wedge G \subseteq\subseteq H \rightarrow F \subseteq\subseteq H$

where " $\subseteq\subseteq$ " is the refinement relation, (i.e. $F \subseteq\subseteq G$ if each member of F is a subset of a member of G).

One wants to be able to use (1), (2), and other such transitivity laws whenever they are needed. But experience with automatic provers (especially resolution) has shown that when these laws are added as hypotheses (as additional clauses in revolution) they greatly complicate and lengthen the proof, because they interact with so many other hypotheses (and even interact with themselves). Boyer (Chap. 7 of [5]) and Slagle [6] have offered solutions to this difficulty, but I prefer a "graphic" solution in the spirit of Bundy [3].

Thus if A_0, B_0, and C_0 are skolem constants, and $A_0 \subseteq B_0$ is ever asserted to be true, we simply "draw" in the computer the connection

$$A_0 \xrightarrow{\subseteq} B_0$$

which means that $A_0 \subseteq B_0$. Later if we learn that $B_0 \subseteq C_0$ we extend the connection

$$A_0 \xrightarrow{\subseteq} B_0 \xrightarrow{\subseteq} C_0$$

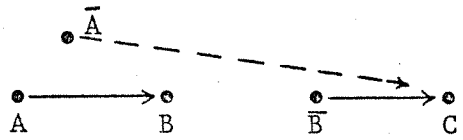
and automatically connect A_0 and C_0, as well. Of course these connections are made by property lists, giving A_0 the property "subset of B_0", and B_0 the property "superset of A_0", etc.

Notice that this places the emphasis on the objects A_0, B_0, C_0, instead of the predicates $\subseteq$, $\subseteq\subseteq$, $\leq$, etc. Present provers seem to be based on the idea of predicates with the objects (terms) as secondary. Maybe we should devise a prover based on objects with predicates as secondary.

Now let us consider the theorem

(3) $$A \subseteq B \wedge \bar{B} \subseteq C \longrightarrow \bar{A} \subseteq C$$

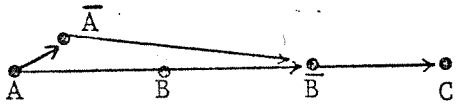
where $\bar{B}$ represents the closure of B. Represented graphically we have



We want the dotted line to hold. By using the fact that $-$ is an "increasing" function we get

$$\bar{A} \subseteq \bar{B} \qquad \text{(Since } A \subseteq B\text{)}$$

and hence we close the gap

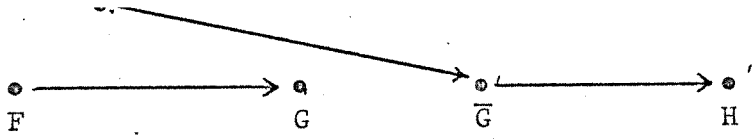


and get $\bar{A} \subseteq C$. (We have also added the lines $A \subseteq \bar{A}$ and $B \subseteq \bar{B}$ but they were not needed in this particular proof.)

Similarly

$$F \subseteq\subseteq G \wedge \bar{G} \subseteq\subseteq H \longrightarrow \bar{F} \subseteq\subseteq H$$

where $\bar{G}$ represents the set of closures of members of G.



Note how geometric the proof is. We move in a "direction" from A to B , from A to $\overline{A}$, from F to $\overline{F}$, etc. This "geometry" helps the intuition of man and should therefore help that of the computer.

Whole networks of connecting links can be build up between objects, not only for these connectors but for others such as $\in$, = , etc. Then when we want to know if something is true for object A_0 we first interrogate the graph to see if it has already been established before we proceed with a regular proof.

A similar geometric interpretation can be given the following theorem:

$$(Po \wedge (Px \rightarrow P\,f\,x) \rightarrow P\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,f\,o)$$

Notice that one does not count the f's, he just notices that he has Po and that he can "move" from any x to fx, and hence can have Pfo, Pffo, etc.

I believe much in mathematical proofs is influenced by such geometrical intuitions. These graph type programs are a step toward programs that make more use of such interpretations.

(Note: While the author was in Edinburgh, Ballantyne and Bennett, working independently, developed ideas similar to these and more. See [4].)

B. Counterexamples and Graphics.

Counterexamples play a large role in a mathematician's life. When working on a conjecture he often alternates between trying to prove it and trying to construct a counterexample. In fact many of his proofs are constructions that show that counterexamples are impossible.

As an example consider the "theorem"

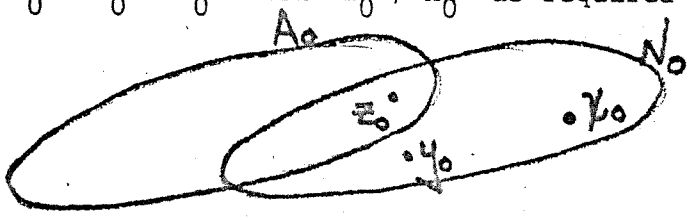
(1) $\quad \text{accum}(\text{accum } A_0) \subseteq \text{accum } A_0,$

Where Accum D means the accumulation points of D.

Df. $x \in \text{Accum } D \equiv$ for each neighborhood N of x there is a point of D for which $y \in N$ and $y \neq x$.

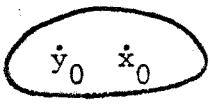
We might give the following "proof":

1. Suppose $x_0 \in \text{accum}(\text{accum } A_0)$
2. We must show that $x_0 \in \text{accum } A_0$
3. So let N_0 be a neighborhood of X_0
4. We must show that there is a $Z_0 \in A_0 \cap N_0$ for which $Z_0 \neq x_0$
5. Since $x_0 \in \text{accum}(\text{accum } A_0)$ we know there is a $y_0 \in (\text{accum } A_0) \cap N_0$ with $y_0 \neq x_0$
6. And since $y_0 \in \text{accum } A_0$ we know there is a $z_0 \in A_0 \cap N_0$ with $z_0 \neq y_0$
7. Thus $Z_0 \in A_0 \cap N_0$ with $Z_0 \neq x_0$ as required by Step 4. Q E D.



But (1) is false! Where did we err? The picture seems to verify our proof. In fact we don't know that $Z_0 \neq X_0$. We only know that $Y_0 \neq X_0$ and $Z_0 \neq Y_0$. The above "proof" has in fact been the construction of a counterexample:

Let X be the space of two points,



With the only open sets being $\emptyset$ (empty set), and $\{x_0 , y_0\}$.

And let A_0 be $\{x_0\}$. Thus

$x_0 \in \text{accum } A_0$, $y_0 \in \text{accum } A_0$, $\{y_0\} = \text{accum } A_0$,

$x_0 \in \text{accum}(\text{accum } A_0)$.

Thus $\text{accum}(\text{accum } A_0) = \{x_0\} \not\subseteq (\text{accum } A_0) = \{y_0\}$.

We now change (1) a little to make it true.

(2) $\text{accum}(\text{accum } A_0) \subseteq \text{accum } A_0 \cup A_0$

A proof similar to the above will succeed for (2). We change only Steps 1 and 7 as follows:

1. Suppose $x_0 \in \text{accum}(\text{accum } A_0)$.

   If $x_0 \in A_0$ we are finished, so assume

   $x_0 \notin A_0$.

7. Since $z_0 \in A_0$ and $x_0 \notin A_0$ (from Step 1) we know that $z_0 \neq x_0$.

   Thus $z_0 \in (A_0 \cap N_0)$ with $z_0 \neq x_0$ as required by Step 4. Q E D.

This technique of "constructing-a-counterexample" proof is used quite often by mathematicians, and would be profitable in computer provers.

Cases. Notice also we used a "by cases" argument in Step 1, and continued the proof under the assumption that $x_0 \notin A_0$. A convenient way of recording such a condition is to record that fact on the property lists of x_0 and A_0 (See Graphic Representations).

The principal use of counterexamples in proofs should be for the elimination of unfruitful search branches in the proof tree. This was used for dramatic success by Gelerntner in his geometry program. Also Hennemann (unpublished work at MIT) has collected and used a set of groups of orders

up to 30 in computer proofs. Similar counterexample sets should be developed for topology, analysis, and other areas of mathematics for use in proofs.

C. Types.

In set theory we have the following three types (and many others)

a b c	x y z	points
A B C	X Y Z	sets
F G H		families of sets

It is very convenient for the mathematician to use different letter sets for different types (as above) because then the notation itself carries much of the information, saving added hypotheses and confusion. Type theory has already been introduced into authomatic theorem proving for higher order logic [7,8,9,10], and in proving limit theorems [11]. Surely it will find further utility later.

Here we wish to introduce another idea regarding types. We do this by an example. First we give two definitions:

Df. $OCF \equiv$ F is an open cover of the space X (ie, each member of F is open, and each $x \in X$ is in some member of F)

Df. regular $\equiv \forall x \forall A(x \in A \wedge \text{open } A \rightarrow \exists B((x \in B \wedge \text{open } B \wedge \overline{B} \subseteq A))$

If we are given the two hypotheses

(1) $OCF_o \wedge$ regular

it is not immediately clear, especially to the computer, how they can be combined to give additional information. On the other hand if we rewrite the definition of regular in the equivalent form

Df. regular $\equiv \forall F(OCF \rightarrow \exists G(OCG \wedge \overline{G} \subseteq\subseteq F))$, then "$OCF_o$" immediately forward chains into "regular" to produce

$$\exists G(OCG \wedge \overline{G} \subseteq\subseteq F_o),$$

which might be used to produce further results.

This example comes up in the proof of a theorem about paracompactness in topology, which is described in the appendix. When the "set-class" definition

of regularity is used the proof of this difficult theorem is considerably simplified, and is done by computer alone except for one (essential) human intervention (see Appendix).

Notice that what happened was that we changed the type of the definition of regular from

point-set

to

set-class.

One could have available in the computer both definitions and let the one triggered by the particular application be the one which is used or one could have an automatic converter.

A Mathematician seems to do this kind of type conversion in his head as the need arises, and one would expect in the long run for the computer to do likewise; however, as an interim measure, such conversions could be made by man ahead of time, and stored for computer usage.

D. List of all Theorem.

It is intriguing to contemplate the day when a mathematician will have at his disposal, on an interactive computer, the list of all theorems ever proved.

Such a list would not be too awfully difficult to produce for a small team given a modest grant. We couldn't get all proved theorems but could get a close approximation. Such a list might be made available to essentially everyone by the ARPA net.

However, I do not propose accumulating such a list at this time because we do not yet know what to do with it.

Certainly present day computer provers are unable to cope with such lists (they are usually swamped with only a few axioms). But it is time (following the lead of workerd in date retrieval and question answering) to start developing provers which do work with such large data bases.

As an interim solution one could devise interactive commands that allowed the user to see "pertinent" parts of the theorem file.
For example, he could ask for all theorems on "regularity" (topological) and the response might be-

3563.

Meaning that there were 3563 of them. At which time he could restrict his inquiry to "regularity" and "hausdorff", and get a response of a smaller number, say 356. Further restriction could bring numbers, like 10, in which case he could ask to see all 10 of the theorems.

Along with each theorem could be stored an optional list of special cases of it which would help to explain it to the user. Someday we might have the computer generate there special cases.

It would be handy for the user to use a light pen to direct the computer through a large list of theorems, quickly rejecting large subsets of them with certain properties, and zeroing in on those with desired characteristics.

E. Book Order.

The problem of automatic provers using a large data base of theorems and definitions is a severe one. (see list of all theorems). There are too many possibilities to try.

One way that might alleniate the difficulty is to arrange the known theorems and definitions in a linear order, as they are in a book. Then if the prover is stuck and needs more information about a concept (a predicate) P, it searches back through the list in reverse book order, looking for theorems about P. When it comes to the definition of P it stops.

One might try retailoring the theorems in the list to make them more useful to the prover, marking lemmas as only of local interest, so that the prover can try using only the "big" theorems in later proofs. Such retailoring might be done automatically, or partially so.

F. Paired Hypotheses.

In mathematics one if familiar with theorems of the form

(1) $$P(x_1) \wedge P(x_2) \rightarrow P(f(x_1,x_2)).$$

For example:

$$\text{open } A \wedge \text{open } B \rightarrow \text{open } (A \cap B)$$

$$\text{open } A \wedge \text{open } B \rightarrow \text{open } (A \cup B)$$

$$NbAx \wedge NbBx \rightarrow Nb(A \cap B)x \qquad \text{(Nb means neighborhood)}$$

$$x \leq 0 \wedge y \leq 0 \rightarrow x + y \leq 0$$

Provers of the future should be, above all, pattern recognizers. In the above the prover would recognize the paired hypothesis structure, and look in a PAIRED-HYP table (similar to PAIRS table (See [2])) to see the desired conclusion is at hand.

For example:

(2) $$NbA_o x \wedge NbB_o x \rightarrow Nb(A_o \cap B_o)x$$

could be referred to a PAIRED-HYP table entry

$$(Nb(NBAx \wedge NBBx \rightarrow Nb(A \cap B)x)T),$$

which would assert its truthfulness.

However, suppose we had to prove (2) and there was no table entry for Nb. Then the program should expand the definition of Nb and try again. Getting

(3) $$([\text{open } D_1 \wedge x \in D_1 \wedge D_1 \subseteq A] \wedge [\text{open } D_2 \wedge x \in D_2 \wedge D_2 \subseteq B] \rightarrow [\text{open } D \wedge x \in D \wedge D \subseteq A \cap B])$$

This raises three similar paired hypothesis subgoals

(4) $$(\text{open } D_1 \wedge \text{open } D_2 \rightarrow \text{open } D)$$

(5) $$(x \in D_1 \wedge x \in D_2 \rightarrow x \in D)$$

(6) $$(D_1 \subseteq A \wedge D_2 \subseteq B \rightarrow D \subseteq A \cap B)$$

Now if we have a PAIRED-HYP entry

(7) $(\text{open}(\text{open } E \wedge \text{open } E' \to \text{open}(E \cap E'))"T")$,

then (4) can be matched in (7) by the substitution D_1/E, D_2/E', $D_1 \cap D_2/D$, proving (4), and reducing (5) and (6) to

(5') $(x \in D_1 \wedge x \in D_2 \to x \in D_1 \cap D_2)$

(6') $(D_1 \subseteq A \wedge D_2 \subseteq B \to D_1 \cap D_2 \subseteq A \cap B)$

which are proved by conventional methods (See [2]).

Thus we could have the following algorithm:

ALGORITHM. When presented with a paired hypothesis theorem

$$P(x_1) \wedge P(x_2) \wedge \cdots \to P(f(x_1,x_2))$$

1. Look in the PAIRED-HYP table
2. If not found there, expand the definition of P in all three places, and return to 1.

If this algorithm is applied to the above example (2), it would also expand the definition of $\subseteq$ in (6') getting an easy to prove theorem. However, the techniques of [2] will easily prove (5') and (6'), without this further expansion.

G. Family Closure

Much of mathematics is devoted to proving closure properties for certain functions and predicates. For example

$$\text{Open } A \wedge \text{Open } B \to \text{Open } (A \cap B)$$

$$\text{Group } G \wedge \text{Group } H \to \text{Group } (G \cap H)$$

$$\text{Countable } A \wedge \text{Countable } B \to \text{Countable } (A \cup B)$$

$$A \subseteq C \wedge B \subseteq C \to A \cup B \subseteq C$$

Many of these properties are known theorems. When they are not known they can often be reduced to other closure theorems at a lower level by instantiating a definition. We now define the closure of a predicate P with respect to a function f.

For a unary predicate P and a binary function f, define

Df. $$\mathscr{C}(P,f) \equiv \forall A \forall B(P(A) \wedge P(B) \to P(A,B))$$

Example. $$\mathscr{C}(\text{Countable}, \cup) \equiv \forall A \forall B(\text{cbl } A \wedge \text{cbl } B \to \text{cbl}(A \cup B))$$

We can generalize this as follows:

Df. If P is an n-ary predicate and f is a k-ary function, let

$$\begin{aligned}\mathscr{C}(P,f,m) \equiv \forall A_1 \ldots \forall A_k (P(x_1, \ldots x_{m-1}, y_1, x_{n+1} \ldots x_n) \\ \wedge P(\ " \quad , y_2, \quad " \ ) \\ \vdots \\ \to P(\ " \quad , f(y_k, \ldots, y_k) \ "))\end{aligned}$$

Example. $$\mathscr{C}(\text{Nb}, \cap, 1) \equiv \forall A \forall B(\text{NbA}x \wedge \text{Nb}\,Bx \to \text{Nb}(A \cap B)x)$$

We could further generalize this to $\mathscr{C}(P,f,m,m')$.

Example. $\mathscr{C}(\subseteq, \cap, 1,2) \equiv \forall A \forall B \forall C \forall D(A \subseteq B \wedge C \subseteq D \rightarrow A \cap C \subseteq B \cap D)$

FAMILY CLOSURE HEURISTIC

When proving $\mathscr{C}(P,f,m)$

(i) Change $\mathscr{C}(P,f,m)$ to its defined form (see p. 2)

(ii) Expand the definition of P in (1) $P(x_1,\ldots x_{m-1},y_1,x_{m+1},\ldots,x_m)$

(2) $\vdots$

(k) $P(\quad " \quad ,y_k, \quad " \quad)$

(*) $P(\quad " ,f(x_1,\ldots,x_k), " \quad)$

(iii) If (*) asks for a set B, then get $B_1,B_2,\ldots,B_k$ from the corresponding places in (1),...(k), and put $f(B_1,\ldots,B_k)/B$.

(iv) Verify that this value of B satisfies the theorem.

As an alternate to (iii) we could have

(iii') First invoke $\mathscr{C}(Q,f,j)$ for each Q in the definition of P in the hypothesis, where j is the place in Q corresponding to m in P. (These j's can be given as extra information in the definition table).

(This is like forward chaining)

Example. Th. $\mathscr{C}(Nb, \cap,1)$

(i) $\forall A \forall B(Nb\ Ax \wedge Nb\ Bx \rightarrow Nb(A \cap B)x)$

$(Nb\ A_0\ x_0 \wedge Nb\ B_0\ x_0 \rightarrow Nb(A_0 \cap B_0)x_0)$ Skolem form

(ii) $[(Open\ A_1 \wedge A_1 \subseteq A_0 \wedge x_0 \in A_1)$

$\wedge(Open\ B_1 \wedge B_1 \subseteq B_0 \wedge x_0 \in B_1)$

$\longrightarrow (Open\ B \wedge B \subseteq (A_0 \cap B_0) \wedge x_0 \in B)]$

(where $A = A(A_0,x_0)$ is a skolem function, etc.)

(iii) Note that a B is needed; B corresponds to A_1 and B_1, so put

$B = f(A_1,A_2) = A_1 \cap B_1$.

(iv) Easily verify

$$(\text{Open } A_1 \wedge A_1 \subseteq A_0 \wedge x_0 \varepsilon A_1 \wedge \text{Open } A_2 \wedge A_2 \subseteq B_0 \wedge x_0 \varepsilon B_1$$

$$\longrightarrow \text{Open } (A_1 \cap B_1) \wedge (A_1 \cap B_1) \subseteq (A_0 \cap B_0) \wedge x_0 \varepsilon A_1 \cap B_1)$$

Example. Using (iii'). Th. $\mathscr{C}(\text{Nb}, \cap, 1)$

(i) same

(ii) same

Nb Ax
P ≡ Nb
m = 1

(iii') Forward chain

Open $A_1 \wedge$ Open B_1	gives	Open $(A_1 \cap B_1)$
$A_1 \subseteq A_0 \wedge A_2 \subseteq B_0$	gives	$A_1 \cap A_2 \subseteq A_0 \cap B_0$
$x_0 \varepsilon A_1 \wedge x_0 \varepsilon B_1$	gives	$x_0 \varepsilon A_1 \cap B$.

Open A'	$\wedge$ A' $\subseteq$ A ($\subseteq$ A'A)	$\wedge$ x ε A' (ε x A')
Q ≡ Open	Q ≡ $\subseteq$	Q ≡ ε
j = 1	j = 1	Q = 2

we get

$$(\ldots\wedge \text{ Open } (A_1 \cap B_1) \wedge A_1 \cap A_2 \subseteq A_0 \cap B_0 \wedge x_0 \varepsilon A_1 \cap B_0$$

$$\longrightarrow \text{Open } B \wedge B \subseteq A_0 \cap B_0 \wedge x_0 \varepsilon B.$$

(iv) Which is easily proved by $A_0 \cap B_0/B$.

Df. If F is a family and f is a k-ary function, let

$$\mathscr{C}f(F,f) \equiv \forall A_1 \ldots \forall A_k(A_1 \varepsilon F \wedge \ldots \wedge A_k \varepsilon F \rightarrow f(A_1,\ldots,A_k) \varepsilon F)$$

Example. Let Nbb x = {A:Nb A x}.

Then $\mathscr{C}f$(Nbb x , $\cap$) is a theorem.

HEURISTIC for $\mathscr{C}f(F,f)$

(i) Change to $(A_1 \varepsilon F \wedge A_2 \varepsilon F \wedge \ldots \wedge A_k \varepsilon F \rightarrow f(A_1,\ldots,A_k) \varepsilon F)$.

(ii) Reduce $A_i \varepsilon F$ to $P(x_1,\ldots,x_m,y_i,\ldots,x_n)$, for $i = 1,\ldots,k$.

(iii) Define P( ) in terms of Q .

(iv) Involk $\mathscr{C}(Q,f,j)$ as before.

Example. $\mathscr{C}f$(Nbb x, $\cap$)

(i) $(A \varepsilon \text{Nbb } x \wedge B \varepsilon \text{Nbb } x \rightarrow A \cap B \varepsilon \text{Nbb } x)$.

(ii) $(\text{Nb } A\ x \wedge \text{Nb } B\ x \rightarrow \text{Nb } (A \cap B)\ x)$

etc as before.

Reversing in $\mathscr{C}(P,f,m)$ and $\mathscr{C}f(P,f)$

In case the definition of P is given in terms of P' the reverse of P , then use f' , the reverse of f , instead of f in the output of the algorithms.

Example. Open' $\equiv$ closed, closed' = open

$\cup' \equiv \cap$, $\cap' \equiv \cup$

$\sigma' \equiv \pi'$, $\pi' \equiv \sigma$

$\mathscr{C}f$(closed,$\cup$)

is converted to $\mathscr{C}f$(open', $\cap'$) ,

$\mathscr{C}f$(open, $\cap$)

H. Generating Theorems. An orderly way to generate theorem (by computer) would be to proceed though a list of closure theorems (forward, or backwards).

$\mathscr{C}$(open, $\cup$)	define
$\mathscr{C}$(open, $\cap$)	define
$\mathscr{C}$(closed, $\cup$)	Use $\mathscr{C}$(open, $\cap$) and Reverse
$\mathscr{C}$(closed, $\cap$)	Use $\mathscr{C}$(open, $\cup$)
$\mathscr{C}$(Nb, $\cap$, 1)	Use $\mathscr{C}$(open, $\cap$), $\mathscr{C}$($\subseteq$, $\cap$, 1, 2), $\mathscr{C}$(ε, 2)
$\mathscr{C}$(Nb, $\cup$, 1)	Use " $\cup$ " $\cup$ " "
$\mathscr{C}$(countable, $\cup$)	Won't work without augmentation
$\mathscr{C}$(finite, $\cup$)	"
$\mathscr{C}$f(Nbb x, $\cap$)	Use $\mathscr{C}$(Nb, $\cap$, 1)

Also we could try certain commutative and distributive laws:

$$f(g(A)) = g(f(A)),\ f(g(A,B)) - g(f(a),f(b)) .$$

If f is an enlarger (i.e., $A \subseteq f(A)$)
and g is a decreaser (" , $g(A) \subseteq A$) ,
then we can expect

$$f(g(A)) \subseteq g(f(A))$$

or $f(g(A)) \leq g(f(A))$.

I. Table Function Evaluation

For certain functions, we define them in the computer by a Table. To prove a theorem for such a function f :

(a) If the argument of f is a known constant c then look up f(c) and prove the theorem for that.

(b) If the argument of f is a skolem constant x_0 then prove $f(x_0)$ for all allowable x_0's in the table.

Example. Suppose Reverse (g) is defined by the following table: (here f is "reverse")

$$\text{rev } \cup \equiv \cap$$

$$\text{rev } \cap \equiv \cup$$

$$\text{rev } \sigma \equiv \pi$$

$$\text{rev } \pi \equiv \sigma$$

Then a theorem: $\phi(\text{rev } \cap)$ is proved as $\phi(\cup)$

whereas a theorem: $\phi(\text{rev } x_0)$ is proved as $\phi(\cap)$ $\phi(\cup)$ $\phi(\pi)$ $\phi(\sigma)$

or a theorem: $[\phi(x_0) \rightarrow \psi(\text{rev } x_0)]$ is proved as

$$[\phi(\cup) \rightarrow \psi(\cap)] \wedge [\phi(\cap) \rightarrow \psi(\cup)] \wedge \ldots \wedge [\phi(\pi) \rightarrow \psi(\sigma)]$$

J. Target

In some proofs we can see that certain things have to be equal and we simply force them to be by whatever manipulation is necessary.

For example in the following proof we could have expanded the definitions of interior and acc, and got

$$(\exists D(\text{open } D \wedge x \in D \wedge D \subseteq B)$$

$$(1) \quad \longleftrightarrow \quad \exists D(\text{open } D \wedge x \in D \wedge \forall y(y \notin (X \sim B) \vee y \notin D \vee y = x)))$$

This essentially forces

$$(2) \qquad (D \subseteq B \longleftrightarrow \forall y(y \in B \vee y \notin D \vee y = x))$$

so we might as well try to prove (2).

When we find we cannot prove (2) we then try proving (2) with the additional hypothesis

$$\text{open } D \wedge x \in D$$

and succeed. But in both cases we had recognized the target (2) that had to be proved and insisted on seeing it through.

Notice that (2) was got from (1) by simply deleting equal parts so one would expect a computer to do such targeting.

Theroem. interior x B $\longleftrightarrow \sim$ acc x(X $\sim$ B)

Proof.

$\sim$ acc x(X $\sim$ B)

$\longleftrightarrow \sim \forall D(\text{open } D \wedge x \in D \rightarrow \exists y(y \in (X \sim B) \wedge y \in D \wedge y \neq x))$

$\longleftrightarrow \exists D(\text{open } D \wedge x \in D \wedge \forall y(y \notin (X \sim B) \vee y \notin D \vee y = x))$

$\longleftrightarrow \exists D(\text{open } D \wedge x \in D \wedge \forall y(y \in D \rightarrow y \in B \wedge y = x)$

$\longleftrightarrow \exists D(\text{open } D \wedge x \in D \wedge D \subseteq B)$

$\longleftrightarrow$ interior x B $\quad Q \in D$

K. Over director.

The major difficulty with any automatic theorem prover is the program not knowing what to do next. In the man-machine mode the human can assume the role of a director and force the machine along productive lines toward completing the proof. A hope is that computer programs can be made to do this in the future. Of course, the general problem solver gives the framework for this in concept, but in fact one does not know how to build the GPS tables to make it work.

In the UT-theorem prover a theorem label (TL) is carried which pretty well defines the part of the proof that the program is working on. For example if TL is (1 2 1) the program is trying to proof the first subgoal of the second subgoal of the first subgoal of the main theorem. Similarly the letter P is used in the label to denote that the pairs heuristic has been used.

It is proposed that this theorem label can be used to direct the proof. When a line of proof seems unproductive, the proof is stopped there and its theorem label is stored for possible use later. Each time the program is stopped in this way, a subroutine looks at all the old theorem labels and the immediately available new ones and decides which one should next be pursued further. In this way the proof is forced into a breath first search mode, which could be helpful especially if the overdirector can cleverly choose the next path to take.

References

1. W. W. Bledsoe, "Some Ideas on Automatic Theorem Proving", Computer Science Department Memo, University of Texas, May 14, 1973.

2. W. W. Bledsoe and Peter Bruell, "A Man-Machine Theorem Proving System", Third International Joint Conference on Artificial Intelligence, Stanford, California, August 20-23, 1973. Accepted for Artificial Intelligence Journal.

3. Alan Bundy, "Doing Arithmetic with Diagrams", IJCAI, 130-138 (1973).

4. Michael Ballantyne and William Bennett, "Graphing Methods for Topological Proofs", Computer Science Department Memo, University of Texas, August, 1973.

5. Robert Boyer, Locking: A Restriction on Resolution, Ph.D. Dissertation, Mathematics Department, University of Texas, Austin, 1972.

6. J. R. Slagle, "Automatic Theorem Proving with Built-in Theories Including Equality, Partial Ordering and Sets," J. ACM 19, No. 1, 120-135.

7. J. A. Robinson, "Mechanizing Higher Order Logic", Machine Intelligence 4, 151-170 (1969).

8. P. B. Andrews, "Resolution in Type Theory", Report 70-72, Department of Mathematics, Carnegie-Mellon University, July 1970.

9. G. P. Huet, "A Mechanization of Type Theory", IJCAI, 139-146 (1973).

10. T. Pietrzykowski and D. Jensen, "A Complete Mechanization of ω-order Type Theory", Association for Comp. Mach. National Conference, vol. 1, 82-92 (1972).

11. W. W. Bledsoe, R. S. Boyer, and W. H. Henneman, "Computer Proofs of Limit Theorems", Artificial Intelligence 3 (1972), 27-60.