A Man-Machine
Theorem Proving System

W. W. Bledsoe and Peter Bruell

University of Texas, Austin, Texas

ATP-4

A MAN-MACHINE THEOREM PROVING SYSTEM

W. W. Bledsoe and Peter Bruell
University of Texas, Austin

[RACT: This paper describes a man-machine theorem ving system at the University of Texas (Austin) ch has been used to prove a few theorems in eral topology. The theorem (or subgoal) being ved is presented on the scope in its natural form that the user can easily comprehend it and, by a ies of interactive commands, can help with the of when he desires. A feature called DETAIL is loyed which allows the human to interact only when ded and only to the extent necessary for the proof.

The program is built around a modified form of LY, a natural-deduction-like theorem proving nique which has been described earlier.

A few examples of proofs are given.

1. Introduction.

Some workers in automatic theorem proving, in- ding the authors, believe that it will be many rs (if ever) before machines alone can prove ficult theorems in mathematics. Thus some, who e to see machines used as practical assistants to e mathematicians, have redirected their attention man-machine theorem provers [3, 4, 5] and theorem of checking [6, 7, 8].

The present paper describes a man-machine theo- proving system at the University of Texas which been used to prove a few theorems in general ology. Our system is organized in the same general as those of Guard [3], Luckham [4], and Huet [5], with many major differences. For example, kham and Huet use variations of Resolution as ir principal rules of inference whereas we use a ified form of IMPLY [1], whic.. is a natural-deduc- n-type method.

Also our system displays formulas on the scope a natural, easy to read, manner and has available ariety of interactive commands the user can employ help with the proof. Among these is a feature led DETAIL which allows the human to interact only n needed and only as much as is required for the of.

As yet this system has proved no new theorem in ology. The program is still in the state of devel- ent and it will be sometime before we can deter- e whether it can materially help a mathematician ve new theorems.

This paper describes the facility, the inter- ive commands available to the user mathematician, modified version of IMPLY which is used, and es a few examples of proofs of theorems.

2. The Facility and Interactive Commands.

The facility consists of a Datapoint 3300 ter- al connected to the CDC 6600 computer via the UT eractive (time-sharing) system TAURUS [11]. A hematician (the user) sits at the terminal, types a theorem to be proved and occasionally helps the gram with the proof by providing information he ls is needed and answering questions the program es.

The computer program consists of a large auto- ic theorem prover and a subroutine for interacting with the mathematician. The theorem prover, which is described in Section 3, is written in LISP and is based on IMPLY (see Section 4 of [1]) and the methods given in [1] and [2]. It has the ability to prove theorems on its own; human intervention is used to increase its power and speed up proofs.

The DETAIL Feature.

One of the principal difficulties with most man-machine provers is in knowing when and how the man should intervene. Firstly the human may have trouble in reading and comprehending the text on the scope, and secondly, he doesn't know when the machine should be helped, and how much he should do. He does not want to make a lot of unneeded entries, and if he makes a mistake he wants to easily recover.

The first difficulty is solved in the system described here by employing the human oriented language IMPLY and in displaying the theorem on the scope in a "pretty-print" form. This is further described below.

The second difficulty is handled by a procedure which allows the computer by itself to first try to prove the theorem (or subgoal). If it succeeds, then all is well, but if it fails within a prescribed time-limit, it prints on the scope the statement of the theorem and the word "FAILURE" and awaits a command from the user. If he commands "DETAIL" then it will proceed (again) with its proof to the point where the current goal is split into subgoals. At that point it prints on the scope the statement of the new subgoal for which it failed and stops, and the whole process can be repeated.

At any of these stops the user can employ a variety of other commands such as DEFN, PUT, USE, etc. (which are described below) to help with the proof. In this way he can easily isolate the difficulty and make only those entries needed by the machine in its proof. Indeed he can start the machine on the proof of a theorem without enough hypotheses (reference theorems) and supply them only when and if they are needed in the proof.

The following is a symbolic example for explaining the DETAIL process. Real examples are given in Section 4.

Suppose the machine is able to convert the example $(H \to C)$ into two subgoals $(H \to D)$ and $(H \to E)$ by defining C as $(D \wedge E)$ and suppose it can prove $(H \to D)$ but not $(H \to E)$, but that it can prove $(H \wedge H_2 \to E)$. Then the dialogue would be as follows:

Human:		$(H \to C)$
Machine:	(1)	$(H \to C)$ FAILED
Human:		DETAIL
Machine:		$(H \to D \wedge E)$
	(11)	$(H \to D)$ PROVED
Human:		GO
Machine:	(12)	$(H \to E)$ FAILED
Human:		DETAIL
Machine:	(12)	$(H \to E)$ FAILED

Human: USE H_2.

(note: here H_2 is some lemma or axiom)

Machine: (12) $(H \wedge H_2 \rightarrow E)$.
PROVED.

Machine: (1) $(H \wedge H_2 \rightarrow C)$
PROVED.

Notice that the only <u>real</u> human intervention was at the step where he commanded (USE H_2), and that help was given only when needed.

<u>The Interactive Commands</u>.

The following is a listing of some of the interactive commands available to the user. A few of these are further explained below and in [12]. In the following the word "theorem" is sometimes used to represent the current subgoal being proved.

NAME OF COMMAND	USER TYPES	THE MACHINE'S RESPONSE
PUT	PUT x = ( )	The machine replaces each occurrence in the theorem being proved by ( ).
DEFN	D A	It replaces all occurrences of A by its (stored) definition.
USE	USE N	It fetches theorem number N from memory and adds it to the hypothesis of the current theorem.
	USE ( )	It adds ( ) to the hypothesis.
LEMMA	LEMMA ( )	It first proves ( ) and then calls USE ( ).
PROCEED	GO	It proceeds with the proof with no changes by the user.
DETAIL	DETAIL	(see explanation above) It finds the first subgoal of the current goal, displays it on the scope, and stops.
COUNT	CNT N	It increases the time-limit on the current subgoal by a factor N.
FAIL	F	It fails the current subgoal (i.e., returns NIL).
ASSUME	A	It assumes the current subgoal to be proved and proceeds.
BACKUP	REJECT	It returns NIL and backs up in the proof to the previous goal.
REORDER	(N → M)	It reorders the goal, placing hypothesis number N first and conclusion number M first.
DELETE	DELETE N M ...	It deletes hypotheses number N, M,...
PRETTY-PRINT	TP	It prints the theorem on the scope in a easily readable form (see Example below).
	TP F	If PUT F = ( ) has been used earlier, it prints the theorem on the scope with each occurrence of ( ) replaced by the symbol F.
	TP F G ...	Similarly for F, G, etc.
	TPC F	Similarly for conclusion only.
	TPH F	Similarly for hypothesis only.
HISTORY	RUN HISTORY	The machine redoes the steps in the proof down to the current point, but eliminates unproductive steps.
ADD-REDUCE	ADD-REDUCE ( )	( ) is (permanently) added to the REDUCE table.
ADD-PAIRS	ADD-PAIRS ( )	( ) is (permanently) added to the PAIRS Table
ADD-DEFN	ADD-DEFN (A ( ))	( ) is added to the definition table as the definition of the atom A.

Computation can also be stopped at any point by the use of an interrupt which will cause the program to return to the beginning of the function IMPLY and halt.

Most of the commands described above are retractable. If a command has changed the theorem in any way the machine displays the changed version and then asks "OK???" The program will then make the change permanent only if the user types "OK."

The machine theorem prover used in this system has been revised (from [1]) to provide a more parallel type of search. This is described below.

As in [2] the presentation of the theorem on the scope is in its original natural form for easy reading by the mathematician. No unnatural conversions, such as changing $(A \rightarrow B)$ to $(\sim A \vee B)$, are made. Additionally, when a symbol, say F, has been replaced by a long expression ( ), the mathematician can, by typing TP F, cause the presentation on the screen to be in terms of F instead of ( ), thus making it easier for him to comprehend.

Such conveniences are necessary to make possible real-time communication between the mathematician and the computer's prover.

Skolem functions are used to eliminate quantifiers but the expressions are left in their natural form (see p. 37 of [1] and p. 18 of [10]), easily readable by the human. Printing of theorems and subgoals on the scope is done with skolem arguments suppressed to further improve readability.

<u>PUT</u>. One of the principal difficulties encountered in automatic theorem proving (indeed in human theorem proving) is the problem of instantiating a variable. For example, it is essentially trivial to instantiate the variable x as x_0 in

$$(x_0 \in A \rightarrow \exists x\ (x \in A))^*$$

*The quantifier " " is retained here for ease of presentation. Such quantifiers are replaced by skolem expressions in actual computer proofs, as indicated above.

: it is far more difficult to find an acceptable lue for G in the expression

$$) \; H \to \exists G \begin{array}{l}(G \text{ is a closed cover } \wedge \\ G \text{ is a locally finite } \wedge \\ G \text{ is a refinement of } F_0),\end{array}$$

ere H is a given hypothesis. In such a situation nachine prover might eventually find and verify a tisfactory G, depending on the nature of H, but s work can be tremendously reduced if the mathe- tician would indicate a value for G. For example, might put

$$) \quad G = \{C: \exists A \; (A \in G' \wedge C = \text{Closure } A)\}.$$

en (1) becomes

$$\to (\{C: \exists A \; (A \in G' \wedge C = \text{Closure } A)\} \text{ is a closed cover}$$

$$) \wedge [\qquad\qquad\qquad] \text{ is locally finite}$$

$$\wedge [\qquad\qquad\qquad] \text{ is a refinement of } F_0),$$

iich can now be split (by the computer) into three ibgoals. The first subgoal of (3) becomes

$$H \to ([\qquad\quad] \text{ is a closed cover})$$

iich is converted to

$$H \to (C \in [\qquad\quad] \to C \text{ is closed}$$

$$4) \qquad \wedge([\qquad\quad] \text{ covers } X)\,.$$

his is again split; the first subgoal becomes

$$\to (C \in \{C: \exists A \; (A \in G' \wedge C = \text{Closure } A)\} \to C \text{ is}$$

losed) which is reduced by the computer to

$$H \to \exists A \; (A \in G' \wedge C = \text{Closure } A) \to C \text{ is closed}$$

nd then to

$$5) \quad H \wedge A \in G' \to \text{Closure } A \text{ is closed.}$$

The subgoal (5) is now easily proved by the com- iuter referring to a REDUCE table (see p. 57 of [2] ind Section 3) which shows that Closure A is always :losed.

Similarly, the second subgoal of (4) is reduced to

$$(6) \quad H \wedge x \in X \to \exists A \; (A \in G' \wedge x \in \text{Closure } A)$$

which again is easily proved if H contains the iypothesis

$$G' \text{ is an open cover.}$$

The other subgoals of (3) are handled similarly, using other hypotheses from H.

Thus the very difficult problem (1) has been reduced to a series of easier problems by the human action (2) and some machine manipulations. It is true that the mathematician is required to provide the most difficult step in the proof but then the computer does the rest, proving a series of smaller theorems and verifying that the mathematician's choice for G was indeed correct. If he made a wrong choice at (2) he might want to intervene later, backup, and try a different value for G.

The PUT feature, though quite simple, is a very powerful device. It alone makes a tremendous difference in the number of theorems the computer program can prove.

DEFN. When the mathematician desires that a certain expression, such as "Reg.", be defined, he types

D Reg

and the machine immediately replaces each occurrence of "Reg" (in the subgoal being proved) by its definition.

When an expression is replaced by its definition, the particular skolemization of that definition will depend on its position in the formula. For example, the expression

$$(\text{Reg} \to C)$$

would be replaced by

$$[x \in A \wedge \text{open } A \to \text{open } B(x,A) \wedge x \in B(x,A)$$

$$\wedge \text{ Clsr } B(x,A) \subseteq A] \to C,$$

whereas

$$(H \to \text{Reg})$$

would be replaced by

$$H \to [x_0 \in A_0 \wedge \text{open } A_0 \to \text{open } B \wedge x_0 \in B$$

$$\wedge \text{ Clsr } B \subseteq A_0].$$

An option is provided so that DEFN can be applied to only parts of the expression. Thus for example, "Reg" might be replaced by its definition in the conclusion but not the hypothesis.

PRETTY-PRINT. The command TP causes the machine to print the current theorem (subgoal) in a parsed, easy to read form. For example, if the theorem is

```
(→(∧(OC(FSD1))(∧(REG)(OCLFR)))(∧(CC G)(∧
(REF G(FSD1))(LF G))))
```

the command TP will cause to be printed on the scope:

```
            (OC(F))
          ∧
            (REG)
          ∧
            (OCLFR)
        →
            (CC G)
          ∧
            (REF G(F))
          ∧
            (LF G)
```

Note that the skolem constant (FSD1) has been printed as (F), though its complete form is retained by the program.

Now if the command

PUT G = {C: Closed C}

is used, the conclusion of (1) is altered accordingly. The command TPC if issued now will cause

```
        (CC{C: Closed C})
      ∧
          (REF{C: Closed C} (F))
        ∧ (LF{C: Closed C})
```

to be printed, whereas TPC G causes

$$(CC\ G) \wedge (REF\ G\ (F)) \wedge (LF\ G)$$

ɔ be printed.

ISTORY. If commanded the program keeps a record ɪistory) of each step it has taken in the proof of a ɪeorem, including steps where the human intervenes ɪt excluding unproductive steps. This history can ɛ used by the mathematician later, upon the command ʀUN HISTORY N", to rerun all or part of the proof ɪthout interruption, and to try if desired a ɪfferent line of proof at any step.

3. The Machine Prover

The prover used by this system consists mainly : a modified form of IMPLY (Section 4 of [1]), with ɪe addition of REDUCE (see p. 57 of [2]), and other ɪncepts from [2] and [17].

Two of the principal differences in the present rsion is that IMPLY is now the main routine (instead CYCLE), and REDUCE is now applied inside IMPLY. e SPLIT functions (p. 56 of [2]) are an integral rt of IMPLY itself. Also IMPLY has been given a eadth-first search capacity (see below), and the ck-up feature (see Footnote 11 of [1]) has been moved and replaced by a human back-up capability.

PLY. IMPLY is a natural deduction type system which ocesses formulas in their "natural" form (see also , 10]). It consists partially of a few rewrite les such as

$$(H \rightarrow (B \rightarrow C)) \Rightarrow (H \wedge B \rightarrow C)$$

$$(H \rightarrow (A \leftrightarrow B)) \Rightarrow (H \wedge A \rightarrow B) \wedge (H \wedge B \rightarrow A)$$

ɪch convert the expression being proved from one ɪm to another. Its main function is to split a ɪl into subgoals

$$(H \rightarrow A \wedge B) \Rightarrow (H \rightarrow A) \quad \text{and} \quad (H \rightarrow B)$$

ːkchain, substitute equals, and forward chain (new ːition). A fundamental part of IMPLY is a matching ːtine (unification): if τ is a most general ːfier of A and A' then the subgoal

$$(A \rightarrow A')$$

judged "TRUE" with τ being returned to be applied further subgoals.

UCE. REDUCE consists wholly of a set of rewrite es which converts parts of formulas. It contains ɛial heuristics for set theory, topology, etc. example

$$(t \in A \cap B) \Rightarrow (t \in A \wedge t \in B)$$

$$(t \in A \cup B) \Rightarrow (t \in A \vee t \in B)$$

$$(t_0 \in \sigma F_0) \Rightarrow (A \in F_0 \wedge t_0 \in A)$$

$$(\text{same as } \exists A\ (A \in F_0 \wedge t_0 \in A)$$

$$\text{or} \quad (t_0 \in \sigma F_0) \Rightarrow (A_0 \in F_0 \wedge t_0 \in A_0)^*$$

$$(\text{Choice } A \in A) \Rightarrow (A \neq \emptyset)$$

$$t \in \{x: P(x)\} \Rightarrow P(t)$$

$$(\text{range } \lambda x\ \mu x) \Rightarrow \{y: \exists x\ (y = \mu x)\}$$

etc.

REDUCE helps convert expressions into forms which are more easily provable by IMPLY. It also is a convenient place to store facts that can be used by the machine as they are needed. For example REDUCE returns "TRUE" when applied to such formulas as (Closed Clsr A), (Open X), (Open $\emptyset$), (Open interior A), ($\emptyset \subseteq A$), etc.

Forward Chaining. It seems that unrestrained forward chaining is a poor idea in automatic theorem proving because it tends to produce an excessive number of useless hypotheses (lemmas). Consequently, our earlier versions of IMPLY relied heavily on backward chaining. However, the use of the man-machine system (especially the PUT feature) on theorems in topology has brought to our attention the power of forward chaining in many proofs, especially in cases where the chaining expression is a ground (all constant) formula. We therefore have provided ground forward chaining as a new rule in IMPLY.

RULE (forward chaining). If P_0 is a ground expression (i.e., contains no variables) which is an instance of P (i.e., there is a substitution τ for which $P_0 = P\tau$) then the goal

$$(H \wedge (P \rightarrow Q) \wedge P_0 \rightarrow C)$$

is converted to the new goal

$$(H \wedge (P \rightarrow Q) \wedge P_0 \wedge Q\tau \rightarrow C).$$

This rule need only be applied at the time something new is added to the hypothesis, such as when an expression $(H \rightarrow (A \rightarrow B))$ is converted to $(H \wedge A \rightarrow B)$, or when another forward chaining step has just been completed.

This rule has been further extended in the system to provide for so-called "PEEK forward chaining", which works as follows:

RULE (PEEK forward chaining). If P_0 is a ground expression, $P_0 = P\tau$, A has the definition $(P \rightarrow Q)$, then the goal

$$(H \wedge A \wedge P_0 \rightarrow C)$$

is converted to the new goal

$$(H \wedge A \wedge P_0 \wedge Q\tau \rightarrow C).$$

Note that the machine "peeks" at the definition of A to see if forward chaining is possible, but then returns A to its original form. This variation is very useful (see Example 2, (111 H1)). Returning A to its original form makes the theorem much easier to comprehend for the mathematician reading the display on the scope.

Forward chaining still tends to clutter up the scope with useless hypotheses, and the user occasionally finds it useful to remove some of them by the command DELETE. More importantly the user, when he gives the computer a theorem to prove, need not list all required lemmas but can give them only as they are needed in the proof, and thereby can eliminate much irrelevant forward chaining.

Breadth-First-Search. One of the difficulties with the previous version of IMPLY was that its search was essentially "depth-first." For example, in proving

$$(H(x) \rightarrow P(x)) \wedge P(x_0) \rightarrow P(x_0)$$

it would back chain off of

---

ce REDUCE is now called from inside IMPLY, it (RE-) must eliminate quantifiers and skolemize in the se of reducing formulas. As was explained in Sec- 2 under DEFN, the exact form of this skolemization nds on the position of the expression in the theorem.

$$(H(x) \rightarrow P(x))$$

d try to prove $H(x_0)$, before finally getting ound to the trivial proof $(P(x_0) \rightarrow P(x_0))$.

A human, acting more intelligently, would sually glance across the hypotheses, and notice x_0) before trying to establish $H(x_0)$.

A more serious difficulty is encountered in stantiating definitions, in that not enough direc- on is provided as to which definition to instantiate rst. As a general rule, an expression such as "reg" ould not be replaced by its definition unless it l1 "do some good." Otherwise a glut of new symbols nper both man and machine. Also it is usually tter to instantiate definitions in the conclusion fore those in the hypothesis, and to instantiate finitions of "strange" terms such as "paracompact" fore those of ordinary terms such as "closed" or ."

We have attempted to remedy these two diffi- ties and have also added another feature called IRS" which tries if possible to apply that othesis which is like the desired conclusion, even n a complete match cannot be made.

The following is a rather sketchy description of revised IMPLY program, which gives only the vor of it. A detailed description is given in].

When a theorem (or subgoal)

$$(H \rightarrow C)$$

given for IMPLY to prove, it first calls REDUCE, a applies its own rewrite rules, and SPLITS it if ropriate. Next it does a breadth-first search trying the following seven steps in the order icated. If any step fails it goes to the next; success of a step usually results in another call the function IMPLY.

1. Match
2. Match and Backchain
3. PAIRS
4. PEEK
5. Define C
6. Define strange terms
7. Define any term.

These are described in more detail below. With exception of step 5 each of the steps listed lves a call from IMPLY to a function called HOA. basically happens is that IMPLY splits the rem into subgoals on the basis of the OR-AND cture of C, and HOA attempts to use the hypo- es to prove these subgoals.

1. Try matching the conjuncts of H with C. That is if H is of the form $H_1 \wedge H_2 \wedge H_3$ it tries to match C with one of the H_i.
2. Same as 1., except that backchaining is allowed. For example, in

$$H_1 \wedge (H_2 \rightarrow C') \rightarrow C$$

it would first try matching C and H_1, and then if that failed, it would try to match C and C' and backchain.

3. Try PAIRS. If the main connective of C matches the main connective of a conjunct H_i of H (but C and H_i do not match), then consult the PAIRS table for conditions which would yield

$$(H_i \rightarrow C),$$

and try to prove those conditions.
For example, given

$$(\text{Ref } G_0F_0) \rightarrow (\text{Ref } H_0J_0)$$

(here (Ref G_0F_0) means that G_0 is a refinement of F_0), PAIRS would consult the PAIRS table and find

$$(\text{Ref } H_0G_0) \wedge (\text{Ref } F_0J_0)$$

there. If it can establish this new subgoal, the proof is concluded. If that fails it will look for another entry on the PAIRS table concerning Ref.

4. PEEK at definitions in H. Here we do not arbitrarily instantiate definitions of expressions in H, but rather do so only if we find in a conjunctive position of H a possible match for C. For example, in

$$(\text{reg} \wedge \text{open cover } F_0 \rightarrow \text{cover } G_0)$$

we first look at the definition of reg and find no reference to "cover", so we leave reg as it was and try the definition of open cover F_0, which is

$$\text{Cover } F_0 \wedge F_0 \subseteq T.$$

Since "cover" appears in a conjunctive position, we retain this definition, and our theorem becomes

$$(\text{reg} \wedge (\text{Cover } F_0 \wedge F_0 \subseteq T) \rightarrow \text{Cover } G_0).$$

Starting again at Step 1 we eventually consider PAIRS (Step 3) on

$$(\text{Cover } F_0 \rightarrow \text{Cover } G_0),$$

which may or may not succeed. If it fails, the theorem is returned to its original form

$$(\text{reg} \wedge \text{Open Cover } F_0 \rightarrow \text{Cover } G_0)$$

and Step 5 is then tried.

5. Instantiate the definition of the main connective of C.
6. Instantiate the definition of the first "strange" symbol in H.
7. Instantiate the definition of any symbol in C.

Equality Substitution. IMPLY employs a form of equality substitution whereby if given the goal

$$(a = b \wedge A \rightarrow C)$$

the program first tries to prove

$$(A' \rightarrow C')$$

where A' and C' are obtained from A and C by replacing all occurrences of a by b, and then if that fails, tries replacing b by a.

This has been sufficient for many applications, but more sophisticated methods may be needed such as those used by Nevins in [9], Slagle in [14], or the "equality-term-locking" of [16], or others, which provide guidance for which of a or b should be replaced by the other.

4. Examples.

The examples we have explored are mostly from Kelley's General topology [13], though in fact any reasonably precise text would do.

We have taken examples from various parts of the book. Example 2 is a theorem about paracompactness. The examples tried so far have been about just one topology. This is convenient since it allows fixed symbols $\mathcal{T}$ and X for the topology $\mathcal{T}$ on the space X. The space X is assumed to be non-empty. The definitions used by the computer are stored (permanently) in its memory.

The theorem labels used in the following examples are also those used by the computer. They help inform the user where he is in the proof. For example, if a goal has theorem label (1 2) and it SPLITS, then the two parts will be labeled (1 2 1) and (1 2 2). If back chaining is used on a theorem labeled L, then the two steps are labeled (LB) and (LH).

The presentation on the scope is always in the "pretty-print" format depicted on page 11. But to conserve space we have here shown our examples in a more compact form, and some lines of the proof are omitted.

In this presentation, an "h" at the left indicates a human input, an "Ed" indicates an editorial comment, and an "m" indicates machine output. The m's are omitted in our description after the first few lines of each example.

In the examples

The expression	Means
$\mathcal{T}$	The topology (family of open sets) on the space X
Cover G	$X \subseteq \bigcup_{A\in G} A$
Oc G	$G \subseteq \mathcal{T} \wedge$ Cover G
cc G	$G \subseteq$ Closed $\wedge$ Cover G
Reg	The topology is regular (defined below)
Lf G	G is locally finite
Ref G F	G is a refinement of F
Oclfr	Every open cover of X has an open locally finite refinement
Occlfr	Every open cover of X has a closed locally finite refinement
Nb N x	N is a neighborhood of x
subsets A	The set of subsets of A
σG	$\bigcup_{A\in G} A$

EXAMPLE 1. (Theorem 1, p. 38 of [13]). A set is open if and only if it contains a neighborhood of each of its points.

DIALOGUE

h $(\text{Open } A \leftrightarrow x \in A \rightarrow \exists N(\text{Nb } N\ x \wedge N \subseteq A)$

Ed This is the statement of the theorem to be proved.

m (1) $(\text{Open } A \leftrightarrow x \in A \rightarrow \exists N\ (\text{Nb } N\ x \wedge N \subseteq A)$
FAILED TIMELIMIT ..

Ed The two dots, .., indicated that the machine has stopped for human input. The computer has failed to prove (1) in its prescribed timelimit.

h DETAIL

m (1 1) $(\text{Open } A_0 \rightarrow (x_0 \in A_0 \rightarrow \text{Nb } N\ x_0 \wedge N \subseteq A_0))$
FAILED ..

Ed The computer has taken the skolemized expression (1 1) as the first half of (1). A_0 and x_0 are skolem constants and N is a variable for which a value must be found that satisfies the conclusion.

h DETAIL

m $(\text{Open } A_0 \wedge x_0 \in A_0 \rightarrow \text{Nb } N\ x_0 \wedge N \subseteq A_0)$

m (1 1 1) $(\text{Open } A_0 \wedge x_0 \in A_0 \rightarrow \text{Nb } N\ x_0)$
FAILED ..

h DETAIL

Ed It now defines "Nb".
$(\text{Open } A_0 \wedge x_0 \in A_0 \rightarrow \text{Open } A \wedge A \subseteq N \wedge x_0 \in A)$

(111 1) $(\text{Open } A_0 \wedge x_0 \in A_0 \rightarrow \text{Open } A)$
PROVED .. (A_0/A)

h GO

(111 2) $(\text{Open } A_0 \wedge x_0 \in A_0 \rightarrow A_0 \subseteq N)$
$(\text{Open } A_0 \wedge x_0 \in A_0 \rightarrow (t_0 \in A_0 \rightarrow t_0 \in N))$

(111 2) (" $\wedge\ t_0 \in A_0 \rightarrow t_0 \in N)$
PROVED .. (A_0/N)

h GO

(111 3) $(\text{Open } A_0 \wedge x_0 \in A_0 \rightarrow x_0 \in A_0)$
PROVED ..

h GO

(1 1 2) $(\text{Open } A_0 \wedge x_0 \in A_0 \rightarrow A_0 \subseteq A_0)$
PROVED ..

Ed This was proved by REDUCE which knows that $A \subseteq A$ for any A.

h GO

(1 2) $((x \in A_0 \rightarrow \text{Nb } N(x)x \wedge N(x) \subseteq A_0) \rightarrow \text{Open } A_0)$
FAILED ..

Ed This is the second half of (1). Note that the skolemization is different from that in (1 1).

h DETAIL

Ed The machine (at Step 5 of IMPLY), "defines" open. Note that in this case a useful characterization is given in place of a bonafide definition.

$((x \in A_0 \rightarrow \text{Nb } N(x)x \wedge N(x) \subseteq A_0) \rightarrow$

$(F \subseteq \mathcal{T} \wedge A_0 = \sigma F)$

$\vee\ (\text{Open } B \wedge \text{Open } D \wedge A_0 = B \cap D))$

2 1) $((\quad) \to F \subseteq \mathcal{T} \wedge A_0 = \sigma F)$

FAILED ..

h DETAIL

:1 1) $((\quad) \to F \subseteq \mathcal{T})$

FAILED ..

Ed At this point the human user realizes that he must help the machine by giving a value for F.

h PUT $F = \mathcal{T} \cap$ subsets A_0

$((\quad) \to (\mathcal{T} \cap \text{subsets } A_0) \subseteq \mathcal{T}$

$((\quad) \to (B_0 \in (\mathcal{T} \cap \text{subsets } A_0) \to \text{Open } B_0))$

$((\quad) \wedge (\quad " \quad) \to \text{Open } B_0))$

$((\quad) \wedge (\text{Open } B_0 \wedge B_0 \subseteq A_0) \to \text{Open } B_0))$

.21 1) $((x \in A_0 \to \text{Nb } N(x)x \wedge N(x) \subseteq A_0) \to (\mathcal{T} \cap \text{subsets } A_0) \subseteq \mathcal{T})$

PROVED ..

h GO

.21 2) $((x \in A_0 \to \text{Nb } N(x)x \wedge N(x) \subseteq A_0) \to A_0 = \sigma(\mathcal{T} \cap \text{subsets } A_0))$

FAILED ..

h DETAIL

$((\quad) \to [A_0 \subseteq \sigma(\mathcal{T} \cap \text{subsets } A_0) \wedge \sigma(\ ) \subseteq A_0])$

:1 21) $((\quad) \to A_0 \subseteq \sigma(\mathcal{T} \cap \text{subsets } A_0))$

$((\quad) \to [(t_0 \in A_0) \to t_0 \in \sigma(\ )])$

$((\quad) \wedge t_0 \in A_0 \to t_0 \in \sigma(\mathcal{T} \cap \text{subsets } A_0))$

Ed $t_0 \in A_0$ is forward chained into the existing hypothesis to yield $(\text{Nb } N(t_0)t_0 \wedge N(t_0) \subseteq A_0)$ which is added to the hypothesis.

$((\quad) \wedge \text{Nb } N_0 \, t_0 \wedge N_0 \subseteq A_0 \wedge t_0 \in A_0 \to t_0 \in \sigma(\mathcal{T} \cap \text{subsets } A_0))$

Ed where N_0 is written for the skolem expression $N(t_0)$.

$((\quad) \wedge " \wedge " \wedge \to \text{Open } A \wedge A \subseteq A_0 \wedge t_0 \in A$

:1 211) $((\quad) \wedge \text{Nb } N_0 \, t_0 \wedge N_0 \subseteq A_0 \wedge t_0 \in A_0 \to \text{Open } A)$

Ed It now "peeks" into the definition of "Nb", finds "open" there, and hence uses the definition of Nb.

$((\quad) \wedge (\text{Open } A_1 \wedge A_1 \subseteq N_0 \wedge t_0 \in A_1) \wedge N_0 \subseteq A_0 \wedge t_0 \in A_0 \to \text{Open } A)$

PROVED .. (A_1/A)

h GO

(121 212) $((\quad) \wedge " \wedge A_1 \subseteq N_0 \wedge " \wedge N_0 \subseteq A_0 \to A_1 \subseteq A_0)$

Ed This subgoal easily follows by the transitivity of "$\subseteq$". Such a rule could be build in as in [14], or we could use a locked axiom as in Chapter 7 of [15]. Another possibility is to "forward chain" $(A_1 \subseteq N_0)$ into $(N_0 \subseteq A_0)$ to get $(A_1 \subseteq A_0)$. Some such device will probably be used in future programs, but here we employed the command "USE", and the program easily completed the proof.

h USE $(A \subseteq B \wedge B \subseteq C \to A \subseteq C)$

EXAMPLE 2. (Theorem 28(b $\to$ c), p. 156 of [13]). If X is a regular topological space and each open cover of X has a locally finite refinement, then each open cover of X has a closed locally finite refinement.

DIALOGUE

h $(\text{Reg} \wedge \text{Oclfr} \to \text{Occlfr})$

m (1) $(\text{Reg} \wedge \text{Oclfr} \to \text{Occlfr})$

m (FAILED TIMELIMIT ..

Ed The computer has tried and failed to prove (1) in its prescribed timelimit.

h DETAIL

m $(\text{Reg} \wedge \text{Oclfr} \to (\text{Oc } F_0 \to \text{cc } G \wedge \text{Ref } GF_0 \wedge \text{Lf } G))$

$(\text{Reg} \wedge \text{Oclfr} \wedge \text{Oc } F_0 \to \text{cc } G \wedge \text{Ref } GF_0 \wedge \text{Lf } G)$

(1 1) $(\text{Reg} \wedge \text{Oclfr} \wedge \text{Oc } F_0 \to \text{cc } G)$

FAILED TIMELIMIT ..

Ed The computer has defined Occlfr, moved the expression Oc F_0 to the hypothesis, and split the conclusion, giving (1 1) as the first subgoal of (1) on which it failed.

Ed Here F_0 is a skolem constant and G is a variable which is to be chosen to satisfy the theorem. The most important and hardest thing the program has to do is to find an acceptable G.

h DETAIL

$(\text{Reg} \wedge \text{Oclfr} \wedge \text{Oc } F_0 \to \text{Cover } G \wedge G \subseteq \text{Closed})$

(1 1 1) $(\text{Reg} \wedge \text{Oclfr} \wedge \text{Oc } F_0 \to \text{Cover } G)$

FAILED ..

Ed At this point the human operator realizes that he must help by giving a value for G. He does this in three steps below, by first asking that the expression Oclfr be defined, and then giving values for the variables F' and G.

(Reg $\wedge$ [Oc F' $\rightarrow$ Cover G_0 $\wedge$ Ref G_0F' $\wedge$
Lf G_0] $\wedge$ Oc F_0 $\rightarrow$ Cover G)

h PUT F' = {B': Open B' $\wedge$ $\exists$B (B $\in$ F_0 $\wedge$
Clsr B' $\subseteq$ B)}

Ed In this writeup we have denoted by G_0 the skolem expression G(F'). The machine retains its complete skolem expressions but prints only (G) on the scope for ease of reading.

Ed Since the new entry [$\rightarrow$] in the hypothesis is an implication, and since F' has been given a value, the machine first tries proving OcF' before proceeding. This is done in (111 H) below. If it succeeds it will then retain the hypothesis

(Cover G_0 $\wedge$ Ref G_0F' $\wedge$ Lf G_0)

instead of [$\rightarrow$].

m TRY PROVING HYPOTHESIS

(111 H) (Reg $\wedge$ Oc F_0 $\rightarrow$ Oc F')

Ed We are writing F' here but the machine retains the value of F' given above.

(111 H) (Reg $\wedge$ Oc F_0 $\rightarrow$ Oc F')
FAILED TIMELIMIT ..

h CNT 4

Ed The operator has increased the timelimit by a factor of 4 (for this goal only) and this causes success. More details of this part of the proof will be given below.

(111 H) (Reg $\wedge$ Oc F_0 $\rightarrow$ Oc F')

m ESTABLISHED HYPOTHESIS

(1 1 1) (Reg $\wedge$ [Cover G_0 $\wedge$ Ref G_0F' $\wedge$ Lf G_0] $\wedge$
$\wedge$ OcF_0 $\rightarrow$ Cover G)

Ed It is now ready to continue with the proof of (1 1 1). The human makes his final input. Bar G_0 is the set of closures of members of G_0.

h PUT G = Bar G_0

(1 1 1) (Reg $\wedge$ Cover G_0 $\wedge$ Ref G_0F' $\wedge$ Lf G_0 $\wedge$ Oc F_0
$\rightarrow$ Cover (Bar G_0))

m TRY PAIRS (cover)

(111 P) (Reg $\wedge$ Ref G_0F' $\wedge$ Lf G_0 $\wedge$ Oc F_0
$\rightarrow$ Ref G_0(Bar G_0))

Ed When PAIRS is presented with (COVER G_1 $\rightarrow$ Cover G_2) it suggests trying Ref G_1G_2. Ref G_1G_2 means that G_1 is a refinement of G_2, i.e., each member of G_1 is a subset of a member of G_2.

Ed (111 P) is proved by REDUCE which has on its table that Ref F(Bar F) for any F.

(1 1 1) (Reg $\wedge$ Cover G_0 $\wedge$ Ref G_0F' $\wedge$ Lf G_0 $\wedge$ Oc F_0
$\rightarrow$ Cover (Bar G_0))
PROVED ..

h GO

(1 1 2) (Reg $\wedge$ Cover G_0 $\wedge$ Ref G_0F' $\wedge$ Lf G_0 $\wedge$ Oc F_0
$\rightarrow$ (Bar G_0) $\subseteq$ Closed)
PROVED ..

Ed Note that the new value (Bar G_0) is given for G, and also the three new hypotheses obtained for the proof of (1 1 1) are retained in (1 1 2).

Ed Again REDUCE proves (1 1 2) since it knows that Bar F $\subseteq$ Closed for any F.

h GO

(1 1) (Reg $\wedge$ Cover G_0 $\wedge$ Ref G_0F' $\wedge$ Lf G_0 $\wedge$ Oc F_0
$\rightarrow$ cc(Bar G_0))
PROVED ..

h GO

(1 2) (Reg $\wedge$ Cover G_0 $\wedge$ Ref G_0F' $\wedge$ Lf G_0 $\wedge$ Oc F_0
$\rightarrow$ Ref(Bar G_0)F_0)

m TRY PAIRS (Ref)

Ed The machine uses PAIRS as before to easily complete the proof of (1 2).

(1 3) (Reg $\wedge$ Cover G_0 $\wedge$ Ref G_0F' $\wedge$ Lf G_0 $\wedge$ Oc F_0
$\rightarrow$ Lf(Bar G_0))

Ed This is the last subgoal of (1). Lf F means the family F is locally finite

m TRY PAIRS (Lf)

(1 3 P) (" $\rightarrow$ (Bar G_0 = Bar G_0))

Ed When PAIRS tries (Lf F $\rightarrow$ Lf G) it suggests trying to prove that (G = Bar F) because it knows that (Lf F $\rightarrow$ Lf Bar F). If such an entry had not been on the PAIRS table then the user might have intervened with the command (USE (Lf F $\rightarrow$ Lf(Bar F)) which would have produced the same result. The PAIRS table is a convenient way to store such Lemmas.

(1 3) (Reg $\wedge$ " $\rightarrow$ Lf(Bar G_0))
PROVED ..

h GO

(1) (Reg $\wedge$ Oclfr $\rightarrow$ Occlfr)
PROVED ..

Ed Q.E.D.

Ed We now list some (but not all) details of the proof of (111 H). Recall the value of F'.

(111 H) (Reg $\wedge$ Oc F_0 $\rightarrow$ Oc F')

(111 H1) (Reg ∧ Oc F_0 → Cover F') h

(Reg ∧ Oc F_0 → X ⊆ σF')

Ed Recall that $\sigma F' = \bigcup_{A \in F'} A$ h

h

(Reg ∧ Oc F_0 → (x_0 ∈ X → x_0 ∈ σF'))

(*) (Reg ∧ Oc F_0 ∧ x_0 ∈ X → x_0 ∈ σF')

Ed When the expression $x_0 \in X$ is switched over to the hypothesis, forward chaining is performed by the machine (see Section 3). It PEEKS into the definition of Oc F_0, and finds h

Ed Oc F_0 ≡ Cover F_0 ∧ F_0 ⊆ $\mathcal{T}$ m

≡ X ⊆ σF_0 ∧ F_0 ⊆ $\mathcal{T}$

(**) ≡ (x∈X → B(x) ∈ F_0 ∧ x ∈ B(x)) ∧ (D ∈ F_0 → Open D)

Ed It therefore substitutes x_0 for x and obtains the additional hypothesis (B(x_0) ∈ F_0 ∧ x_0 ∈ B(x_0)). Thus (*) becomes (Reg ∧ Oc F_0 ∧ x_0 ∈ X ∧ B_1 ∈ F_0 ∧ x_0 ∈ B_1 → x_0 ∈ σF')

Ed Where B_1 is the skolem expression B(x_0). The addition of $B_1 \in F_0$ to the hypothesis causes further forward chaining into (D ∈ F_0 → open D) of (**) to yield open B_1, which in turn, with $x_0 \in B_1$, is forward chained into

Reg ≡ (x ∈ A ∧ open A→x ∈ B(x,A) ∧ Open B(x,A) ∧ Clsr B(x,A) ⊆ A)

to yield three more hypotheses. Thus writing B_2 for B(x_0,B_1), (111 H1) becomes

(111 H1) (Reg ∧ Oc F_0 ∧ x_0 ∈ X ∧ B_1 ∈ F_0 ∧ x_0 ∈ B_1 ∧ Open B_1 ∧ x_0 ∈ B_2 ∧ Open B_2 ∧ Clsr B_2 ⊆ B_1 → x_0 ∈ σF') m

Ed Which is now easily proved by the program

The following example is given to show the use of ', ADD-DEF, ADD-REDUCE, REDUCE, and the PEEK feature REDUCE. Many (most) steps of the dialogue are omitted the writeup.

MPLE 3. (Th. 15, P. 49 of [8]). Suppose the topo-y $\mathcal{T}$ has a countable base. Then each open cover a set has a countable subcover. m

(cbl F ∧ Base F) ∧ A ⊆ X ∧ G ⊆ $\mathcal{T}$ ∧ A ⊆ σG → ∃H (H ⊆ G ∧ cbl H ∧ A ⊆ σH) h

m

(cbl F_o ∧ Base F_o ∧ A_o ⊆ X ∧ G_o ⊆ $\mathcal{T}$ ∧ A_o ⊆ σG_o → H ⊆ G_o ∧ cbl H ∧ A_o ⊆ σH) m

DETAIL m

1) (cbl F_o ∧ ... → H ⊆ G_o) h

DETAIL (1 2)

H ⊆ G_o h

FAILED (1 2 1)

m

h

(PUT F_1 = {A' ∈ F_o: ∃B' ∈ G_o A' ⊆ B'})

(PUT f = (λA ∈ F_1 choice {B ∈ G_o: A ⊆ B})

(PUT H = Range f)

(1 1) (· · · → Range f ⊆ G_o)

FAILED

At this point the user realizes that the computer does not have the definition of Range.

(ADD-DEF (Range g) {y: ∃ x ∈ domain g y = g(x)})

(· · · → {y: · · ·} ⊆ G_o)

(· · · →

(B_o ∈ {y: · · ·} → B_o ∈ G_o))

Reduce

(· · · ∧ A_1 ∈ domain f ∧ B_o = f(A_1) → B_o ∈ G_o)

Here the machine knows nothing about λ expressions so it can reduce neither domain f nor f(A_1). So the user gives the following information.

(ADD-REDUCE ((λx ∈ A B) x) B), λ-conversion

(ADD-REDUCE (domain (λx ∈ A B)) A).

Now the machine in trying to reduce (domain f) "peeks" at the definition of f, finds it to be a λ-expression, and reduces (domain f) to F_1.

In converting f(A_1) it again "peeks" at the definition of f to reduce f(A_1) to (Choice {B ∈ G_o: A_1 ⊆ B}).

(· · · ∧ A_1 ∈ F_1 ∧ B_o = Choice {B ∈ G_o: A_1 ⊆ B} → B_o ∈ G_o)

Sub =

(· · · ∧ A_1 ∈ F_1 ∧ " → Choice {B ∈ G_o: A_1 ⊆ B} ∈ G_o) etc.

etc.

FAILED

(ADD-REDUCE (choice A ∈ B) (A ⊆ B))

(· · · ∧ A_1 ∈ F_1 ∧ " → {B ∈ G_o: A_1 ⊆ B} ⊆ G_o)

(· · · " → (B_2 ∈ G_o ∧ A_1 ⊆ B_2 → B_2 ∈ G_o))

PROVED

GO

(· · · → cbl Range f ∧ A_o ⊆ σ Range f)

DETAIL

(· · · → cbl Range f)

FAILED

(USE ∀f' (function f' ∧ cbl domain f' → cbl Range f')

Back Chain

(cbl Range f' → cbl Range f) f/f'

(121 H) (· · · → function f ∧ cbl domain f)

(121 H1) (· · · → function f)

(· · · → function $\lambda A \in F_1$

choice $\{B \in G_o : A \subseteq B\}$)

(USE (function $\lambda x \in A$ P))

m (· · · → function $\lambda A \in F_1$

choice $\{B \in G_o : A \subseteq B\}$)

m PROVED

(121 H2) (· · · → cbl domain f)

(· · · → cbl domain ($\lambda A \in F_1$ choice { }))

(cbl F_o ∧ · · · → cbl F_1)

h (ADD-PAIRS (cbl F) (cbl G) ($G \subseteq F$))

(121 H2P) (· · · → $F_1 \subseteq F_o$)

Ed This is easily proved and finishes the proof of (1 2 1).

(122) (cbl F_o ∧ Base F_o ∧ ... → $A_o \subseteq \sigma$ Range f)

PROVED..

QED

5. Remarks.

Many of the abilities which are built into this man-machine facility have been developed only after a period of trial and error. In fact the reason for many of these is to provide for more ease in checking out and changing the program. We expect the program to continue to change as it is tried on more and more examples, hopefully evolving into a system which will be useful to the researcher in topology. So far this is not the case, we have handled only well known theorems. Our next step involves work on the system by some practicing topologist. This should help determine whether such a system might have any practical value in the near future.

An interesting point is this. Even though the mathematician is able to intervene at any point in the proof, he is nevertheless very annoyed when he has to do so in a trivial way. When, for example, he PUTS the values for F' and G in Example 2, he feels he has done enough and rightfully expects the computer to do the rest. Thus even in a man-machine system, the theorems that the machine alone is required to prove are far from trivial. In fact experience so far shows that they are on a par with the hardest theorems being proved today by automatic theorem provers.

Therefore, it is felt that any improvement in machine-alone programs is truly worthwhile to the man-machine effort.

Acknowledgment.

Various people both at U.T. and elsewhere have greatly influenced our thinking about automatic theorem proving and interactive systems. We want to especially thank Bill Henneman, Robert Anderson, Dave Luckham, Vesko Marinov, Bill Bennett, Mike Ballentyne, and Howard Ludwig.

This work was supported in part by NSF Grant GJ-32269 and NIH Grant 5801 GM 157-69-05.

References

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

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

3. J. R. Guard, F. C. Oglesby, J. H. Bennett, and L. G. Settle, Semi-automated Mathematics, JACM 16 (1969), 49-62.

4. John Allen and David Luckham, An Interactive Theorem-Proving Program, Machine Intelligence 5 (1970), 321-336.

5. G. P. Huet, Experiments with an Interactive Prover for Logic with Equality, Report 1106, Jennings Computing Center, Case Western Reserve University.

6. John McCarthy, Computer Programs for Checking Mathematical Proofs, Proc. Amer. Math. Soc. on Recursive function Theory, held in Ney York, April, 1961.

7. Paul W. Abrahams, Machine Verification of Mathematical Proof, Doctoral Dissertation in Mathematics, MIT, May, 1963.

8. W. W. Bledsoe and E. J. Gilbert, Automatic Theorem Proof-Checking in Set Theory: A Preliminary Report, Sandia Corp. Report SC-RR-67-525, July, 1967.

9. Arthur J. Nevins, A Human Oriented Logic for Automatic Theorem Proving, MIT AI Memo No. 268, October, 1972.

10. Raymond Reiter, The Use of Models in Automatic Theorem Proving, Dept. of Computer Science, University of British Columbia, September, 1972.

11. TAURUS, described in Users Manual, Computation Center, University of Texas, Austin.

12. Peter Bruell, A Description of the Functions of The Man-Machine Topology Theorem Prover, (under preparation).

13. John L. Kelley, General Topology, van Nostrand, 1955.

14. James R. Slagle, Automatic Theorem Proving with Built-in Theories Including Equality, Partial Ordering, and Sets, JACM 19 (1972), 120-135.

15. Robert Boyer, Locking: A Restriction on Resolution, Ph.D. Dissertation, Mathematics Dept., University of Texas, Austin, 1971.

16. Dallas S. Lankford, Equality Atom Term Locking, Ph.D. Dissertation, Mathematics Dept., University of Texas, Austin, 1972.

17. George Ernst, The Utility of independent subgoals in Theorem Proving, Information and Control, vol. 18, 3, 1971.