Non-resolution Theorem Proving[1]

W. W. Bledsoe

Department of Mathematics, The University of Texas at Austin, 78712, U.S.A.

ABSTRACT

This talk reviews those efforts in automatic theorem proving, during the past few years, which have emphasized techniques other than resolution. These include: knowledge bases, natural deduction, reduction, (rewrite rules), typing, procedures, advice, controlled forward chaining, algebraic simplification, built-in associativity and commutativity, models, analogy, and man–machine systems. Examples are given and suggestions are made for future work.

1. Introduction

Automatic theorem proving was born in the early 1930s with the work of Herbrand, but did not get much interest until high speed digital computers were developed. Earlier work by Newell, Simon, Shaw, and Gelernter in the middle and late 1950s emphasized the heuristic approach, but the weight soon shifted to various syntactic methods culminating in a large effort on resolution type systems in the last half of the 1960s. It was about 1970 when considerable interest was revived in heuristic methods and the use of human supplied, domain dependent, knowledge.

It is not my intention here to slight the great names in automatic theorem proving, and their contributions to all we do, but rather to show another side of it. For recent books on automatic theorem proving see Chang and Lee [19], Loveland [44], and Hayes [31]. Also see Nilsson's recent review article [61].

The word "resolution" has come to be associated with general purpose types of theorem provers which use very little domain dependent information and few if any special heuristics besides those of a syntactic nature. It has also connoted the use of clauses and refutation proofs.

There was much hope in the late 60's that such systems, especially with various exciting improvements, such as set of support, model elimination, etc., would be powerful provers. But by the early 70's there was emerging a belief that resolution type systems could never really "hack" it, could not prove really hard mathematical theorems, without some extensive changes in philosophy.

[1] This is an edited version of an invited lecture at the 4th International Joint Conference on Artificial Intelligence, held in Tbilisi, Georgia, U.S.S.R., September 1975.

This report is about this other non-resolution effort. But we do not just want to emphasize non-*resolution*, but rather to emphasize the efforts that are less syntactic in nature, that use heuristics and user supplied knowledge, which is often domain dependent. Our belief is that other purely syntactic methods such as Gentzen systems will fare only about as well as resolution systems, unless they employ some of the kinds of concepts we mention below. Also much improvement in resolution systems can be gained by using such concepts, and this has been done in many cases.

The author was one of the researchers working on resolution type systems who "made the switch". It was in trying to prove a rather simple theorem in set theory[2] by paramodulation and resolution, where the program was experiencing a great deal of difficulty, that we became convinced that we were on the wrong track. The addition of a few semantically oriented rewrite rules and subgoaling procedures [7] made the proof of this theorem, as well as similar theorems in elementary set theory, very easy for the computer. Put simply: the computer was not doing what the human would do in proving this theorem. When we instructed it to proceed in a "human-like" way, it easily succeeded. Other researchers were having similar experiences.

This is not really a general review in any fair sense. Rather it is a list of things I feel are important, with a real bias toward my work and that of my students and friends.

A list of references is given at the end of the paper.

2. Concepts

We will now list some of the concepts and techniques that we have in mind, that seem to hold promise in automatic theorem proving, and briefly discuss them. Of course no such list could be complete and we apologize for glaring omissions. Also these concepts are not mutually exclusive, some being special cases of others.

The word "knowledge" is a key to much of this modern theorem proving. Somehow we want to use the knowledge accumulated by humans over the last few thousand years, to help direct the *search* for proofs. Once a proof has been found it is a relatively simple matter to verify its validity by purely syntactic

KNOWLEDGE
- Build-in man's knowledge
- Often Domain-specific
- Contextual and Permanent

AVOID LISTING OF AXIOMS
- Clogs up the system

EASY TO USE and CHANGE

FIG. 1. Basic concepts.

[2] The family of subsets of $(A \cap B)$ is the same as the intersection of the family of subsets of A and the family of subsets of B. This example is treated later.

procedures. So in a sense all of our concepts have to do with the storage and manipulation of knowledge of one sort or another. See Fig. 1.

The use of knowledge and built-in procedures partially eliminates the need for long lists of axioms, which tend to slow up proofs and use excessive amounts of memory. Such knowledge must be organized in a way that is easy to use and change.

Fig. 2 shows the 13 concepts we will discuss in the succeeding pages.

1. Knowledge Base
2. Reductions (rewrite rules)
3. Algebraic Simplification
4. Built-in Inequalities (and total ordering)
5. Natural Systems
6. Forward Chaining
7. Overdirector
8. Types
9. Advice
10. Procedures (and Built-in Concepts)
11. Models (and counterexamples)
12. Analogy
13. Man–machine

FIG. 2. Concepts.

2.1. Knowledge base

We store information in a knowledge base (or data base), process that information to obtain other information (by procedural forward chaining, etc.), and interrogate the data base when necessary to answer questions. A central idea here is, that facts are stored about "objects" rather than "predicates". For example the hypothesis Open(A_0) would be stored with "Open" as a property of "A_0" rather than with "A_0" as a property of "Open". (Objects are the skolem constants arising in a proof.) Also knowledge is stored about concepts. This knowledge can be stored in procedures or in lists or other structures.

The planner—QA4 type systems are ideally suited for using these concepts. See for example Winograd's Thesis [84], especially Sections 3.1.3–3.3.1 for an excellent description of some of these concepts.

Some concepts associated with a knowledge base are shown in Fig. 3. Demons are routines that watch the knowledge base and only act whenever certain properties become true of the data base. Languages like Microplanner, Conniver, and QA4 greatly facilitate the use of demons.

Some parts of the base remains static (as for example, properties of continuous functions) while other parts such as information about objects in the proof are dynamic and should be carried in a contextual data base.

Contains FACTS about Concepts and Objects
Facts are manipulated during proof to obtain new facts (contextual)
Procedural forward chaining
Static Information
Look up Answers
Object oriented: Open (A_0)
  Property list A_0 ≀ open
Demons
Examples
  Partial Sets
  Monads
  Reduction Rules
} See later
"Graph" Provers
MODELS
FRAMES

FIG. 3. Knowledge base.

The "graph" provers of Bundy [17], Ballantyne and Bennett [4] use such a data base, as do the provers of Winograd [84], Goldstein [28], and others. Minsky's frames [55] appear to offer good advice for organizing data for a knowledge base.

Shortly we will show an example from analysis [5] which utilizes a data base with many of the concepts we have discussed in this paper.

2.2. Reduction

A reduction is a rewrite rule,

$$A \dashrightarrow B.$$

For example, the rule

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

requires that we change all subformulas of the form $t \in (A \cap B)$ into the form $(t \in A \wedge t \in B)$, (but never rewrite the latter into the former). Such rules are

IN	OUT
$t \in (A \cap B)$	$t \in A \wedge t \in B$
$t \in (A \cup B)$	$t \in A \vee t \in B$
$t \in \{x : P(x)\}$	$P(t)$
$t \in \text{subsets}(A)$	$t \subseteq A$
$t \subseteq A \cap B$	$t \subseteq A \wedge t \subseteq B$
$t \in \bigcup_{\alpha \in F} g(\alpha)$	$\exists \alpha (\alpha \in F \wedge t \in g(\alpha))$
.	
.	
.	

FIG. 4. Some REDUCE rules (rewrite rules).

semantic; their inclusion, and their use is not based upon their syntactic structure but on their meaning. The user supplies these rules. Fig. 4 lists some such rules, and Fig. 5 gives some definitions.

$A = B$	$A \subseteq B \wedge B \subseteq A$	(set equality)
$A \subseteq B$	$\forall x(x \in A \rightarrow x \in B)$	
	skolem form	
	$(x_0 \in A \rightarrow x_0 \in B)$	in "conclusion"
	$(x \in A \rightarrow x \in B)$	in "hypothesis"
Subsets (A)	$\{B : B \subseteq A\}$	
sb(A)	Subsets (A)	

FIG. 5. Some definitions.

The use of REDUCTIONS is best illustrated by an example from [7]. See Fig. 6. Here the formula "subsets(A)" means the set of all subsets of A; we have shortened it to "sb(A)" for this proof.

THEOREM. $\forall A \forall B$ (subsets $(A \cap B)$ = subsets $(A) \cap$ subsets (B))

PROOF. sb$(A \cap B)$ = sb$(A) \cap$ sb(B)	THE GOAL		(1)
$[\text{sb}(A \cap B) \subseteq \text{sb}(A) \cap \text{sb}(B)] \wedge [\ldots \supseteq \ldots]$,		defn of =	
	SUBGOAL 1		
$[\text{sb}(A \cap B) \subseteq \text{sb}(A) \cap \text{sb}(B)]$			(11)
$[t_0 \in \text{sb}(A \cap B) \Rightarrow t_0 \in (\text{sb}(A) \cap \text{sb}(B))]$,		defn of $\subseteq$	
$[t_0 \subseteq (A \cap B) \Rightarrow t_0 \in \text{sb}(A) \wedge t_0 \in \text{sb}(B)]$		Reduce	
$[t_0 \subseteq A \wedge t_0 \subseteq B \Rightarrow t_0 \subseteq A \wedge t_0 \subseteq B]$		Reduce	
"T"			
$[\text{sb}(A) \cap \text{sb}(B) \subseteq \text{sb}(A \cap B)]$	SUBGOAL 2		(12)
"T" Similarly			

FIG. 6. An example of a proof.

IN	OUT
$\emptyset \subseteq A$	"T"
$A \subseteq A$	"T"
$\emptyset \in \omega$	"T"
$A \in \emptyset$	"False"
$0 \leqslant 1$	"T"
Open $\emptyset$	"T"
$A \subseteq \bar{A}$	"T"
.	
.	
.	
We would not include	
$P(y) \wedge \forall x\,(P(x) \rightarrow Q(x)) \rightarrow Q(y)$	"T"
because it is too complex.	

FIG. 7. Storing unit facts as reductions.

The reduction rules and definitions given in Fig. 4 and 5 are used in the proof. Notice how easy and "human-like" the proof proceeds when reductions are used. A corresponding resolution proof (without built-in partial ordering) required 14 clauses and a lengthy deduction.

Reductions also offer a convenient way for storing unit facts that can be easily used during proofs. See Fig. 7.

We are concerned with the four types of REDUCTION shown in Fig. 8.

REDUCTION (Rewrite Rules)
Conditional Reduction
Controlled Definition Instantiation
Complete Sets of Reductions

FIG. 8. Four kinds of REDUCTION.

2.2.1. Controlled definition instantiation

In this example we did *not* instantiate all definitions possible, but rather followed the rule: when all other strategies fail, instantiate the definition of the main connective of the conclusion. See Fig. 9.

EXAMPLES. Do not expand definitions in:

$$(A \subseteq B \wedge x \in A \wedge \text{Open } A \Rightarrow \text{Open } A).$$

Do expand the definition of "OCCLFR" in

$$(\text{Regular } (\mathcal{T}) \wedge \text{OCLFR}(\mathcal{T}) \Rightarrow \text{OCCLFR}(\mathcal{T}))$$

if other attempts fail.

FIG. 9. Controlled definition instantiation.

Instantiating all definitions can badly clutter up the proof and is often not needed. In general, definition instantiation should be carefully controlled.

2.2.2. Conditional reduction

This is a slight generalization of the reduction concept, whereby the program will perform the reduction only if a given stated condition is true in the *data base*. We do *not* want a large effort expended to determine whether the condition is true, because reductions are supposed to be performed quickly, so we verify the condition in the data base (rather than call the prover itself for this purpose). See Fig. 10 for a simple example.

2.2.3. Complete sets of reductions (written by D. Lankford).

Instead of using a reduction $(A \dashrightarrow B)$ one could get the same effect by adding the formula $(A = B)$ as another hypothesis. But computational experience [5, 9, 30, 37, 39A, 40, 42, 56] has shown it is desirable to use equations as rewrite rules or to incorporate them into a normal form algorithm, in order to reduce the computation time and storage space needed. To what extent this can be done has

IN	CONDITION	OUT
INTERIOR (A)	OPEN (A)	A
CLOSURE (A)	CLOSED (A)	A
$\|A\|$	$A \geqslant 0$	A
$\|A\|$	$A < 0$	$-A$

EXAMPLE THEOREM. $2 \leqslant I \wedge 4 \leqslant J \leqslant 9 \wedge (0 \leqslant K \wedge L \leqslant 10 \rightarrow P(K, L)) \rightarrow P(I, |J|)$.

PROOF. The hypothesis $2 \leqslant I$ and $4 \leqslant J \leqslant 9$ are stored in the data base on the property lists of I and J. The term $|J|$ is reduced (rewritten) to J after a check in the data base verifies the condition $J \geqslant 0$.

Backchaining on the third hypotheses now gives the subgoal ($0 \leqslant I$ and $J \leqslant 10$) which is easily verified by the data base.

FIG. 10. Conditional reductions and an example.

been the object of considerable research. Fig. 11 lists some of the principal workers in this area with brief descriptions of their contributions. The pioneering works of Knuth and Bendix [40] and Slagle [73] focused on certain sets of rewrite rules for special study, those which determine normal forms for the corresponding equational theories. These decision procedures, called *complete sets of reductions* by Knuth and Bendix [40] and *sets of simplifiers* by Slagle [73], are defined by two properties: the finite termination property—no term can be infinitely reduced, and the unique termination property—any two sequences of immediate reductions, starting with the same term and terminating in irreducible terms, terminate with identical terms. Knuth and Bendix [40] developed an effective procedure which often derives a complete set of reductions from a given set of equations. Fig. 12 shows the end result of their derivation of a complete set of reductions for group theory beginning with the three left minimal axioms. Slagle [73] initiated the study of refutation complete methods for combining complete sets of reductions with resolution and paramodulation. Lankford [41, 42] generalizes a synthesis of their methods. Fig. 13 gives a proof in group theory using the Knuth-Bendix reductions.

Unfortunately, complete sets of reductions fail to exist for some equational theories, such as commutative group theory and ring theory, for the simple reason that some equations, such as commutative equations, cannot be expressed as rewrite rules with the finite termination property. For example, in proving a theorem in group theory like that in Fig. 14, when the hypothesis $(x+(x+x) = 0)$ is added to the axioms of group theory, the whole set can no longer be converted to a complete set of reductions. The existence of undecidable word problems, such as the theory defined by four equations found by Matijasevic [49A], shows that this limitation cannot be entirely overcome by a modification of the notion of complete sets of reductions. In addition, many important theories contain

Ballantyne and Bledsoe [5]: Metatheoretical concepts of non-standard analysis encoded as rewrite rules.

Bledsoe, et al. [?]: Ring theory normal form generator.

Guard, et al. [30]: Semi-automatic proof of SAM's lemma.

Knuth [39A]: Automatic proof of a conjecture from central groupoid theory.

Knuth and Bendix [40]: Complete sets of reductions; A class of finite termination tests; a test dependent unique termination algorithm.

Lankford [41, 42]: A test independent unique termination algorithm; an enlarged class of finite termination tests; refutation completeness results for complete and incomplete sets of reductions.

Nevins [56]: An experimental anticipation of many of the concepts related to complete sets of reductions.

Plotkin [63]: Refutation completeness results for theories having normal forms by specialization of the unification algorithm.

Slagle [73]: Complete sets of reductions; refutation completeness results for complete sets of reductions for fully narrowed input sets.

Winker [83]: Dynamic demodulation; a class of finite termination tests.

FIG. 11. Workers in the area of complete sets of reductions.

KB1	$x+0 \rightarrow x$
KB2	$0+x \rightarrow x$
KB3	$x+(-x) \rightarrow 0$
KB4	$(-x)+x \rightarrow 0$
KB5	$(x+y)+z \rightarrow x+(y+z)$
KB6	$-0 \rightarrow 0$
KB7	$-(-x) \rightarrow x$
KB8	$-(x+y) \rightarrow (-y)+(-x)$
KB9	$x+((-x)+y)) \rightarrow y$
KB10	$(-x)+(x+y) \rightarrow y$

FIG. 12. Knuth and Bendix's complete sets of reductions for a group.

THEOREM.	$\cdot[(D+(C+(-C)))+(-(0+(-A)))] \in H \rightarrow (D+A) \in H$	
Proof.		
	$\cdot[(D+(C+(-C)))+(-(-A))] \in H \rightarrow (D+A) \in H$	KB2
	$\cdot[(D+0)+(-(-A))] \in H \rightarrow (D+A) \in H$	KB3
	$\cdot[D+(-(-A))] \in H \rightarrow (D+A) \in H$	KB1
	$\cdot[D+A] \in H \rightarrow (D+A) \in H$	KB7
	TRUE	MATCH

FIG. 13. A proof in group theory using complete sets of reductions.

equations that occur in non-unit clauses. And the well-known result of Birkhoff on equational definability shows that this situation cannot be avoided. Nevertheless, we believe that modifications and adaptations of the concept of complete sets of reductions [41, 42, 63, 73] will lead to significant improvement in the treatment of equality when combined with heuristic methods and built-in procedures [5, 9, 56, 63, 76].

For example, consider the theorem that in a group with $x+x+x = 0$, $h(h(x, y), y) = 0$ where $h(x, y) = x+y+(-x)+(-y)$. This theorem has received repeated attention in the literature [61A, 67A] as a difficult theorem for mechanical

THEOREM. $x+(x+x) = 0 \rightarrow h(h(a, b),a) = 0$.

Proof. The conclusion is reduced to

$$a+b+(-a)+b+a+(-b)+(-a)+(-b) = 0 \qquad \text{(associate to the right).}$$

The hypothesis $(x+(x+x) \rightarrow 0)$ is added as another rewrite rule.

11	$x+(x+x) \rightarrow 0$	added.

In the first round, three new reductions are generated, and one equality:

N1	$x+x+x+y \rightarrow y$	11, KB5
N2	$x+y+x+y+x+y \rightarrow 0$	11, KB5
N3	$-x \rightarrow x+x$	11, KB10
E1	$x+y+x+y = y+y+x+x$ (this cannot be made a reduction).	N3, KB8

N1-3 Act upon KB1-10, 11 and eliminate all of KB1-10, 11 except

KB1	$x+0 \rightarrow x$
KB2	$0+x \rightarrow x$
KB5	$(x+y)+z \rightarrow x+(y+z)$
11	$x+x+x \rightarrow 0$.

FIG. 14. First round of the proof using Lankford's method (using Fig. 12).

In the second round, three new reductions are generated, and no new equations:

N4	$x+y+x+y+x+y+z \rightarrow 0$
N5	$x+y+z+x+y+z+x+y+z \rightarrow 0$
N6	$x+y+x+y+x \rightarrow y+y$.

The Goal:

$$a+b+(-a)+b+a+(-b)+(-a)+(-b) = 0$$

is proved in this round by applying N3, E1, N1, N6 and 11.

Steps: 2.

Formulas saved: 7.

Time: 30 seconds.

FIG. 15. Second round of the proof.

(and human!) theorem provers, and was only recently proved fully automatically without hints by Nevins [56]. Lankford's methods [42] in Fig. 14 and 15 were implemented by Ballantyne and Lankford [5A] and show considerable improvement over Nevin's methods, primarily because of Nevin's treating associativity by unification rather than as a rewrite rule.

Fig. 16 gives a challenging problem for automatic provers from ring theory which seems beyond present methods.

THEOREM. *In a RING*

$$x^3 = x \rightarrow ab = ba.$$

(Recall that a ring is +commutative.)

FIG. 16. A challenging problem for automatic provers.

2.3. Algebraic simplification

There is a strong need to avoid adding the field axioms for the real numbers as hypotheses to a theorem being proved, because this greatly slows proofs. The associativity and commutativity axioms for + and · are especially troublesome, so several efforts have been made to "build these in".

Some references to this work are: Slagle and Norton [74, 75]; QA4; QLISP [65, 68]; Plotkin [63]; Frönig [25]; Stickel [77]; Bledsoe, et al. [9, 12].

Of course much is learned from the researchers working in the field of symbol manipulation and algebraic simplification, where these methods have been applied to other problems in physics and mathematics. However, automatic theorem proving presents difficulties not covered by that work.

For example, the theorem

$$P(a+b+c) \rightarrow P(b+a+c)$$

is easily handled by using a canonical form, but the theorem

$$P(k+2) \rightarrow P(b+5)$$

where k is a variable and b is a constant, presents more difficulty. An ordinary unification algorithm

$$\text{UNIFY}(k+2, b+5)$$

would put b for k, b/k, and fail. An "Algebraic Unifier", could write the equation

$$k+2 = b+5,$$

and "solve for" k, getting $k = b+3$, and return $(b+3)$ for k, to successfully complete the proof.

A similar approach works on the example

$$\text{UNIFY}(B[k+1] = \text{Amax}(B, j, k+1),$$
$$A_0[i_0] = \text{Amax}(A_0, 1, i_0))$$

where B, j, k are variables, and A_0, i_0 are constants. This example is from the field of program verification (see [13], pp. 27–28).

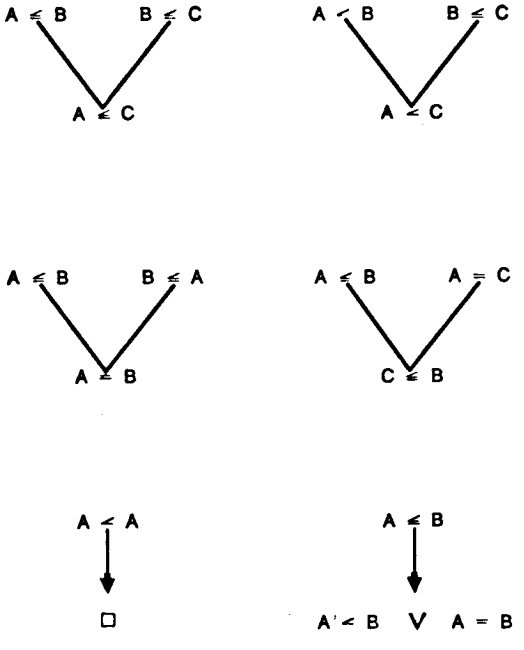
Data types such as sets, bags, and triples [68, 65] handle some of these problems.

2.4. Built-in inequalities (and total orderings)

Again we must avoid the explicit use of such axioms as the transitivity axiom

$$(x \leqslant y \wedge y \leqslant z \rightarrow x \leqslant z).$$

Bledsoe et al. [8, 9, 13, 29] employ "interval types" for dealing with certain inequalities, and Slagle and Norton [75] have built-in axioms for handling total and partial ordering (including inequalities). See Fig. 17.



ALLOW UNIFICATION IN ALL OF THESE CASES

FIG. 17. Slagle and Norton's build-in partial and total ordering.

The following is a theorem from program verification which was proved by Slagle and Norton's program. This theorem, which is a verification condition from Hoare's FIND program, and others like it have been proved by the "interval type" methods of Bledsoe and Tyson [8, 13], and by others.

THEOREM.

$$\begin{aligned} & j < i \\ & \wedge\ m \leqslant p \leqslant q \leqslant n \\ & \wedge\ \forall x \forall y (m \leqslant x < i \wedge j < y \leqslant n \rightarrow A[x] \leqslant A[y]) \\ & \wedge\ \forall x \forall y (m \leqslant x \leqslant y \leqslant j \rightarrow A[x] \leqslant A[y]) \\ & \wedge\ \forall x \forall y (i \leqslant x \leqslant y \leqslant n \rightarrow A[x] \leqslant A[y]) \\ & \quad \rightarrow A[p] \leqslant A[q]. \end{aligned}$$

Also Slagle and Norton have built-in partial ordering for handling some problems in set theory.

2.5. Natural systems

We have chosen to emphasize the so called "natural" systems in this report. I would not like to define the term, but will only give examples. In general we are not talking about refutation systems such as resolution, though we sometimes do proofs by contradiction [66]. They are sometimes called goal oriented systems, or Gentzen type systems.

We are given a goal G and a hypothesis H and wish to show that G follows from H,

$$(H \rightarrow G)$$

or more generally to find a substitution θ for which

$$(H\theta \rightarrow G\theta)$$

is a propositionally valid formula. A set of rules is given for manipulating H and G to obtain the desired θ. For example, $(P(a) \wedge (P(x) \rightarrow Q(x)) \Rightarrow Q(a))$ has the solution $\theta \equiv a/x$.[3] The Rules in Fig. 18 and 19 are from the IMPLY System described in [9, 12]. They are given more precisely and completely in [12]. See Fig. 20 for a simple example.

I4. $(H \Rightarrow A \wedge B)$ "SPLIT"
If $(H \Rightarrow A)$ returns θ
and $(H \Rightarrow B\theta)$ returns λ
then return $\theta \circ \lambda$.

I3. $(H_1 \vee H_2 \Rightarrow C)$ "CASES"
If $(H_1 \Rightarrow C)$ returns θ
and $(H_2\theta \Rightarrow C)$ returns λ
then return $\theta \circ \lambda$.

I5. $(H \Rightarrow C)$
Put C': = REDUCE(C): H': = REDUCE(H)
Call $(H' \Rightarrow C')$.

I7. $(H \Rightarrow (A \rightarrow B))$ "PROMOTE"
Call $(H \wedge A \Rightarrow B)$.

I13. $(H \Rightarrow C)$
Put C': = DEFINE(C)
Call $(H \Rightarrow C')$.
(See [12] for the ordering of these rules)

FIG. 18. IMPLY rules. A partial set from [12].

[3] Sometimes θ must be more complicated, (see App. 3 of [12]) as in the example $(P(x) \rightarrow P(a) \wedge P(b))$ which has the solution $\theta \equiv a/x \vee b/x$.

H2. $(H \Rightarrow C)$ "MATCH"
If $H\theta \equiv C\theta$, return θ,

H6. $(A \wedge B \Rightarrow C)$ "OR-FORK"
If $(A \Rightarrow C)$ returns θ (not NIL), return θ,
Else Call $(B \Rightarrow C)$

H7. $H \wedge (A \rightarrow D) \Rightarrow C$ "BACK-CHAIN"
If $(D \Rightarrow C)$ returns θ,
and $(H \Rightarrow A\theta)$ returns λ,
then return $\theta \circ \lambda$

H9. $H \wedge (a = b) \Rightarrow C$ "SUB="
Put $a' :=$ CHOOSE (a, b), $b' :=$ OTHER (a, b)
Call $(H(a'/b') \Rightarrow C(a'/b'))$.

FIG. 19. IMPLY rules (contd.).

THEOREM. $(P(a) \wedge \forall x(P(x) \rightarrow Q(x)) \rightarrow Q(a))$.

$P(a) \wedge (P(x) \rightarrow Q(x)) \Rightarrow Q(a))$
$(P(a) \Rightarrow (Q(a))$ H6
FAILS
$(P(x) \rightarrow Q(x)) \Rightarrow Q(a))$
$(Q(x) \Rightarrow Q(a))$ H7
Return $\theta \equiv a \mid x$ H2
$(P(a) \Rightarrow P(x)(a \mid x))$
Return TRUE.

Returns $a \mid x$.

FIG. 20. An example proved by IMPLY.

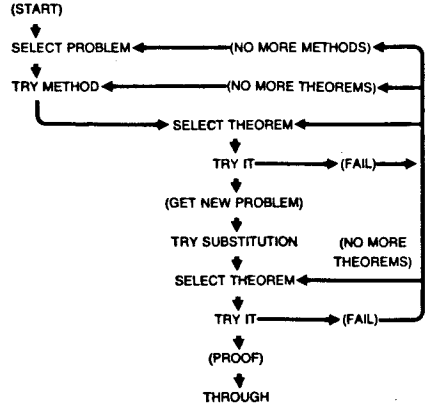


FIG. 21. General flow diagrams of the LOGIC THEORIST [59].

Newell, Simon, and Shaw's logic theorist [59, 60], and Gelernter's geometry machine [26], were natural (or goal directed) systems, although we see that they included various other features. See Fig. 21 and 22. Reiter's MATH-HACK

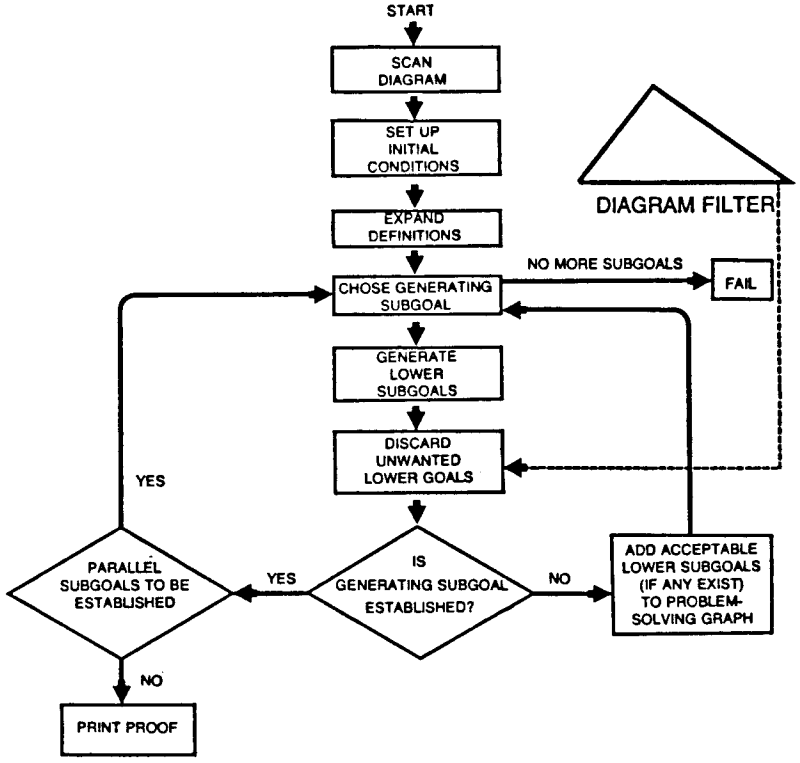


FIG. 22. Simplified flow chart for the geometry-theorem proving machine (Fig. 3 of [26]).

NSS—Logic Theorist [60]
Gelernter's Geometry machine [26]
Reiter's math-hac [66]—much like [12] but more
Maslov [48]
Bibel [6]
Ernst [23]
Boyer-Moore [14]
Nevins [56–58]
Planner [34, 78]
Conniver [52]
QA4, QLISP [68, 65]
Goldstein [28]
Ullman [80]

FIG. 23. Other natural systems.

system [66] is much like that of [12] but has the important addition of models which we will mention later, and other features.

Other natural systems include the Planner–Conniver–QA4 group, and those of Maslov, Goldstein, Nevins, Bibel, Boyer-Moore, Ernst, [48, 49, 28, 34, 78, 52, 68, 65, 6, 14, 23, 46, 15, 30] and others. See Fig. 23.

What are the advantages (if any) of the natural systems? There may be none—especially in the long run—and especially if the techniques we emphasize here are built into the resolution systems. But we feel that this is not easy to do. Specifically we feel that the natural systems are:

- Easier for human use.
- Easier for machine use of knowledge.

See Fig. 24 and 25.

HUMAN USE

Bring to bear knowledge from pure mathematics in the same form used there.

Recognize situations where such knowledge can be used.

Professional mathematician will want to participate.

Easier to design, augment, work upon.

Essential for man–machine interaction (where the man is a trained mathematician).

FIG. 24. Advantages of natural systems.

MACHINE USE

Automatically limits the search. Does not start all proofs of the theorem. (Syntactic search strategy.)

A natural vehicle to hang on heuristics, knowledge, semantics. (Semantic search strategies.)

Easier to combine procedures with deduction.

Contextual data base problem. (One data base would be needed for each clause.)

New Languages (PLANNER, QA4). Ease the implementation.

FIG. 25. Advantages of natural systems (contd.).

We include here two quotes regarding the advantages of natural systems.

"There is a naturalness with which systems of natural deduction admit a semantic component with the result that a great deal of control is gained over the search for a proof. It is precisely for this reason that we argue in favour of their use in automatic theorem-proving, in opposition to the usual resolution-based systems, which appear to lack any kind of reasonable control over dead-end searches." Raymond Reiter [66].

"A point worthy of stress is that a deductive system is not "simpler" merely because it employs fewer rules of inference. A more meaningful measure of simplicity is the ease with which heuristic considerations can be absorbed into the system." Arthur Nevins [56].

2.6. Forward chaining

Forward chaining is accomplished when one hypothesis is applied to another to obtain an additional hypothesis. See the example in Fig. 26.

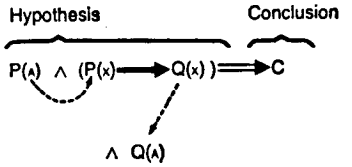


FIG. 26. Forward chaining. The additional hypothesis $Q(A)$ is obtained.

Since such a process can, in some cases, result in an infinite repetition it is important that it be controlled by a cut-off mechanism. Also we have found other controls desirable, such as allowing only those new hypotheses which are *ground* formulas.

Procedural forward chaining is also used, where a procedure is invoked which manipulates items in the data base (or in the hypothesis) to produce new items. This is exhibited, for instance, in the non-standard analysis example given below and in [5].

The early programs of Newell, Simon and Shaw, used forward chaining, as did many others. Fig. 27 lists some more recent examples where extended use of forward chaining produced surprising results. Nevin's remark [58, pp. 2, 3] on how his geometry prover was so greatly improved by the use of forward chaining, is of particular interest on this point.

Bundy [17]—Doing arithmetic with diagrams.
Siklossy, Rich, and Marinov [71]—British Museum.
Ballantyne and Bennett [4]—Topology.
Nevins [58]—Plane Geometry.
Ballantyne—Non-Standard-Analysis [5].

FIG. 27. Extensive use of forward chaining.

2.7. Overdirector

Every prover has a control routine which directs the search tree. See Newell, Simon, and Shaw's control structure was shown in Fig. 21.

This overdirector can bring to bear strategies or experts (see [28]), heuristics, and advice tables, controlled backup, etc. as it sees fit.

It is important that such an overdirector have the flexibility to switch from one line of attack to another, and back again, as the proof proceeds, thus providing a parallel search capability. This of course, requires a (controlled) back-up mechanism such as that possessed by Conniver. Unrestricted back-up is intolerable. A contextual data base, which can be consulted by the overdirector to help it decide whether and how much to back-up, or what other line of attach to take, is an indispensable part of the prover we have in mind. The concepts of Conniver [52] and QA4 [68, 65] apply here.

EXAMPLE. (From Non-Standard Analysis.) The following example is given here to exhibit the use of some of the concepts we have described above. These techniques have been used by Mike Ballantyne to prove by computer (not interactively) several difficult theorems in intermediate analysis. See [5] for a complete description of this work.

The reader need not be conversant with non-standard analysis (or even intermediate analysis) to follow the example given in Fig. 28 and 29.

Notice that the proof follows the general procedure described by the rules of Figs 18–19. First, the fact that f is continuous, is noted in the data base and the hypothesis $\text{Cont}(f, S_0)$ is dropped. Next the term "Compact" is defined (in non-standard terms), and the formula $x_0 \in f(S_0)$ is "promoted" to the hypothesis by Rule 17 of Fig. 18, and then reduced to produce the new hypothesis.

$$V_0 \in S_0 \wedge x_0 = f(V_0).$$

Forward chaining then gives the additional hypothesis: $\text{st}(V_0) \in S_0$.

At this point we leave the rules of Fig. 18 and 19, and work with the data base. See [5] for details.

Various routines such as EL, STANDARD, FINITE, CONTINUOUS, are used to put items into the data base and to manipulate them to obtain others. This is called procedural forward chaining. For example the program detects $(V_0 \in S_0)$ and $(\text{st}(V_0) \in S_0)$ in the hypothesis and calls EL which builds a set S_0' in the data base with the elements V_0, and $\text{st}(V_0)$, and drops $(V_0 \in S_0)$ and $(\text{st}(V_0) \in S_0)$ from the hypothesis. This set S_0' with only two elements represents the set S_0 which may be infinite.

Similarly the monad M_1:$(\text{st}(V_0), V_0)$ is built in the data base. The reader needs only to understand that continuous functions map monads into monads, (and not what a monad is) and hence that the monad M_1 is mapped into the monad M_2:$(f(\text{st}(V_0)), f(V_0))$.

The hypothesis $(x_0 = f(V_0))$ is used to generate the reduce rule R_1 and another hypothesis generates R_2. Thus the goal $\text{st}(x_0) \in f(S_0)$ is easily converted to the new

5.2. THEOREM. *If f is continuous on a compact set S, then $f(S)$ is compact.*

Proof.

- $\text{Cont}(f, S_0) \wedge \text{Compact}(S_0) \Rightarrow \text{Compact}(f(S_0))$

  Note: f is continuous on S_0. (In Data Base)
- $\text{Compact}(S_0) \Rightarrow \text{Compact}(f(S_0))$
- $(x \in S_0 \rightarrow \text{st}(x) \in S_0) \Rightarrow (x_0 \in f(S_0) \rightarrow \text{st}(x_0) \in f(S_0))$ — definition
- $(x \in S_0 \rightarrow \text{st}(x) \in S_0) \wedge x_0 \in f(S_0) \Rightarrow \text{st}(x_0) \in f(S_0)$ — promote (See Fig 18).
- $(x \in S_0 \rightarrow \text{st}(x) \in S_0) \wedge (V_0 \in S_0 \wedge x_0 = f(V_0) \Rightarrow \text{st}(x_0) \in f(S_0)$ — reduce

  $(x \in S_0 \rightarrow \text{st}(x) \in S_0) \wedge (V_0 \in S_0 \wedge x_0 = f(V_0)$
  $\wedge \underline{\text{st}(V_0) \in S_0} \Rightarrow \text{st}(x_0) \in f(S_0)$ — forward chain

FIG. 28. An example of a proof by Ballantyne's prover.

	DATA BASE	AGENT
•	S_0': $(\text{st}(V_0), V_0)$	EL
	Type: $\text{st}(V_0)$, Standard	Standard
	Type: $\text{st}(V_0)$, Finite	Standard
	V_0 , Finite	
	M_1: $(\text{st}(V_0), V_0)$	Finite
	R_1: $x_0 \dashrightarrow f(V_0)$	Equals
•	M_2: $(f(\text{st}(V_0)), f(V_0))$	Continuous
•	R_2: $\text{st}(f(V_0)) \dashrightarrow f(\text{st}(V_0))$	Continuous
	$f(S_0)'$: $(f(\text{st}(V_0)), f(x_0))$	Continuous
•	$(x \in S_0 \rightarrow \text{st}(x) \in S_0) \Rightarrow \text{st}(x_0) \in f(S_0)$	
•	$(x \in S_0 \rightarrow \text{st}(x) \in S_0) \Rightarrow \text{st}(f(V_0)) \in f(S_0)$	R_1
•	$(x \in S_0 \rightarrow \text{st}(x) \in S_0) \Rightarrow f(\text{st}(V_0)) \in f(S_0)$	R_2
•	TRUE	$f(S_0)'$

FIG. 29. Proof (contd.).

goal $f(\text{st}(V_0)) \in f(S_0)$, which is readily verified *by inspection*; i.e., the program notes that the set $f(S_0)'$ in the data base, contains the item $f(\text{st}(V_0))$.

In summary, one sees the manipulation of a data base and the execution of a few logical operations, to produce the proof of this theorem.

2.8. Types

The concept of typing plays a fundamental role in mathematics and computer science. Using a letter e for the identity element of a group, lower case letters x, y, z, for members of the group, and capital letters G, H, for groups and subgroups, is immensely helpful to humans in proving theorems.

Similar typing is helpful in automatic provers. Other data types such as integer, real, negative, complex, bags, sets, types, interval types, infinitesimals, infinitely large, etc., can be advantageous in certain applications. See Fig. 30.

e	identity in a group
x, y, z	members of a group
G, H	groups, subgroups
x, y, z	points
A, B, C	sets
F, G, H	Families
$\mathcal{T}$	topology
P, Q	predicates
x, y	reals
z	complex
I, J, K	Integers
ε, δ	Infinitesimals
r, s, t	standard reals
x, y, z	non-standard reals
ω	infinitely large integers
Bags, sets, types	
Interval types	

FIG. 30. Some examples of types.

2.9. Advice

One of the most powerful things a human can do to aid the prover is to provide "advice" for the use of a theorem or lemma. Carl Hewitt's PLANNER [34] exploits this idea.

For example in Fig. 31 we see an example of Winograd's [84] where, to determine that a thing x is a thesis we are "advised" to either verify that it is long, or that it contains a persuasive argument. This is given in Micro-planner language in Fig. 32.

Another such advice lemma is given in Fig. 33, and this is used in Fig. 34 to prove a theorem. This proof also clearly emphasizes the need for simplification routines and eqation solving routines in proofs in analysis.

GOAL	VERIFY
(Thesis x)	(Long x) (Use: contents—check, count pages)
	or
	(x contains y)
	and
	(argument y)
	and
	(persuasive y)

FIG. 31. Winograd's example.

```
(DEFINE THEOREM EVALUATE
                                        ;EVALUATE is the name we are
                                        ;giving to the theorem
 (THCONSE(X Y)
                                        ;this indicates the type of
                                        ;theorem and names its
                                        ;variables
  (THGOAL(# THESIS $?X))
                                        ;show that X is a thesis
                                        ;the " $?" indicates a variable
  (THOR
                                        ;THOR is like "or", trying things
                                        ;in the order given until one works
   (THGOAL(# LONG $?X)(THUSE CONTENTS-CHECK COUNTPAGES))
                                        ;THUSE says to try the theorem
                                        ;named CONTENTS-CHECK first,
                                        ;then if that doesn't work, try
                                        ;the one named COUNTPAGES
   (THAND
                                        ;THAND is like "and"
    (THGOAL(# CONTAINS $?X $?Y))
                                        ;find something Y which is
                                        ;contained in X
    (THGOAL(# ARGUMENT $?Y))
                                        ;show that it is an argument
    (THGOAL(# PERSUASIVE $?Y)(THTBF THTRUE))))))
                                        ;prove that it is persuasive, using
                                        ;any theorems which are applicable
```

FIG. 32. PLANNER Representation of the advice lemma in Fig. 31 (Fig. 53 of [84]).

GOAL	VERIFY	
$(\lvert A\rvert \leqslant \varepsilon)$	$(A = B+C)$	1
	and	
	$(\lvert B\rvert \leqslant \varepsilon_1)$	2
	and	
	$(\lvert C\rvert \leqslant \varepsilon_2)$	3

and	
$(\varepsilon_1+\varepsilon_2 \leqslant \varepsilon)$	4
or	
(Other advice)	
$(A = B \cdot C)$	1
and	
$(\lvert B\rvert \leqslant \varepsilon_1)$	2
and	
$\lvert C\rvert \leqslant \varepsilon_2)$	3
and	
$(\varepsilon_1 \cdot \varepsilon_2 \leqslant \varepsilon)$	4
or	
(Other advice)	

FIG. 33. An advice lemma.

THEOREM. $|a| \leqslant E^{\alpha}-1 \wedge |b| \leqslant E^{\beta}-1 \wedge 1+c = (a+1)(b+1) \rightarrow |c| \leqslant E^{\alpha+\beta}-1$

Proof. Goal: $|c| \leqslant E^{\alpha+\beta}-1$.

Parts 1–4 of the advice lemma (Fig 33) are used to convert this to subgoals (1)–(4) below.

(1) $c = B+C$

To solve this, convert the hypothesis $1+c = (a+1)(b+1)$ to $c = a \cdot b+a+b$ and substitute $a \cdot b/B$, $a+c/C$ (i.e. substitute $a \cdot b$ for B and $a+b$ for C)

(2) $\lvert a \cdot b\rvert \leqslant \varepsilon_1$	use parts 1′–4′ of the advice lemma.	
(21) $a \cdot b = B \cdot C$	$a/B, b/C$	
(22) $\lvert a\rvert \leqslant \varepsilon_{11}$	$E^{\alpha}-1/\varepsilon_{11}$	
(23) $\lvert b\rvert \leqslant \varepsilon_{12}$	$E^{\beta}-1/\varepsilon_{12}$	
(24) $(E^{\alpha}-1) \cdot (E^{\beta}-1) \leqslant \varepsilon_1$	$(\ ) \cdot (\ )/\varepsilon_1$	
(3) $\lvert a+b\rvert \leqslant \varepsilon_2$	use parts 1–4 of the advice Lemma (again).	
(32) $\lvert a\rvert \leqslant \varepsilon_{21}$	$E^{\alpha}-1/\varepsilon_{21}$	
(33) $\lvert b\rvert \leqslant \varepsilon_{22}$	$E^{\beta}-1/\varepsilon_{22}$	
(34) $(E^{\alpha}-1)+(E^{\beta}-1) \leqslant \epsilon_2$	$(E^{\alpha}-1)+(E^{\beta}-1)/\varepsilon_2$	
(4) $(E^{\alpha}-1) \cdot (E^{\beta}-1)+((E^{\alpha}-1)+(E^{\beta}-1)) \leqslant E^{\alpha+\beta}-1$	use simplification.	
$E^{\alpha+\beta}-1 \leqslant E^{\alpha+\beta}-1$	TRUE	

FIG. 34. The proof of an example (due to Overbeek) using the advice lemma of Fig. 33.

The concept depicted in Fig. 33, might be generalized in a manner shown in Fig. 35. Then perhaps an instantiation of it (like Fig. 33) could be saved by the program for future use.

GOAL	VERIFY
$P(C)$	Find $P(A)$ in Hypothesis
	and
	Express C in terms of A, $C = f(A, B)$
	and
	Find $P(\alpha) \wedge P(\beta) \rightarrow P(f(\alpha, \beta))$
	and
	Goal $P(B)$
	or
	(Other Advice)

FIG. 35. A more general advice lemma.

2.10. Procedures (and built-in concepts)

These have been discussed already, especially in the non-standard analysis example given in Section 7.

Fig. 36 lists some of these concepts and examples. An "expert" is a set of procedures for solving one type of problem. See Goldstein [28].

Strategies
Heuristics
  Syntactic
  Semantic (domain dependent)
Experts
EXAMPLES:
  Induction
  Built-in partial and total ordering, inequality, associativity, etc.
  "Solvers"
  Goldstein's geometry prover
  Limit heuristic
  PAIRS heuristic
CONCEPTS:
  Follow a plan rather than search.
  Calculate an answer rather than prove a formula.

FIG. 36. Some procedures.

In Fig. 37 we see an INDUCTION heuristic being applied [7]. In general when a heuristic is to be applied, the program detects a pattern and consults a list of recommendations. In this example it detects the presence of ω in the theorem being proved and proceeds as shown. Details of this proof are omitted.

THEOREM. $\omega = \bigcup_{\alpha \in \omega} \alpha$

(1) $(\omega \subseteq \bigcup_{\alpha \in \omega} \alpha) \wedge (\bigcup_{\alpha \in \omega} \alpha \subseteq \omega)$. *Defn of* = *SUBGOAL 1*

SUBGOAL 1

(11) $(\omega \subseteq \bigcup_{\alpha \in \omega} \alpha)$ *EASY*

SUBGOAL 2

(12) $(\bigcup_{\alpha \in \omega} \alpha \subseteq \omega)$

$(t_0 \in \bigcup_{\alpha \in \omega} \alpha \rightarrow t_0 \in \omega)$. *Defn of* $\subseteq$

$(\alpha_0 \in \omega \wedge t_0 \in \alpha_0 \rightarrow t_0 \in \omega)$. *REDUCE*

The proof fails by the normal procedures. It detects the presence of ω, and decides to try INDUCTION. Pre-INDUCTION converts it to the form:

$$(\alpha_0 \in \omega \rightarrow \underbrace{(t_0 \in \alpha_0 \rightarrow t_0 \in \omega)}_{P(\alpha_0)}).$$

It now trys:

$$P(0) \text{ and } (P(\alpha_0) \rightarrow P(\alpha_0+1)).$$
$$\cdot 1 \qquad \cdot 2$$

(12.1) $(t_0 \in 0 \rightarrow t_0 \in \omega)$ SUCCEEDS

(12.1) $(\alpha_0 \in \omega \wedge (t_0 \in \alpha_0 \rightarrow t_0 \in \omega) \rightarrow \underbrace{(t_0 \in (\alpha_0+1) \rightarrow t_0 \in \omega)}_{t_0 = \alpha_0 \vee t_0 \in \alpha_0})$.
SUCCEEDS

FIG. 37. The use of an INDUCTION heuristic to prove a theorem.

Fig. 39 shows some strategies from Goldstein's geometry prover [28], and Fig. 40 shows an example where the PAIRS heuristic [10] is being applied. In this example a *partial match* was obtained between the two formulas

$$\text{Cover}(G_0) \text{ and } \text{Cover}(\bar{\bar{G}}_0),$$

which triggered the program to consult the PAIRS table (see Fig. 41) for advice. The first advice given from the PAIRS table, namely $(G_0 \subseteq \bar{\bar{G}}_0)$, failed, but the second one, $(G_0 \subseteq\subseteq \bar{\bar{G}}_0)$, succeeded.

STRATEGY EQTR13

TO-PROVE: Triangle XYZ = Triangle UVW

ESTABLISH: 10 seq XZ = seq VW
20 angle XYZ = angle UVW
30 angle YZX = angle VWU

REASON: congruence by asa

(This is like backchaining)

CONVERSION ANGLE-BISECTOR

GIVEN: seq DB bisects angle ABC

ASSERT: angle ABD = angle CDB

FORGET: given

(This is like reduce)

COROLLARY EQTRI-2

GIVEN: Triangle XYZ = Triangle UVW

ASSERT: Angle XYZ = angle UVW
angle YZX = angle VWU
angle ZXY = angle WUV

FIG. 39. Some strategies from Goldstein's Prover.

THEOREM. $\forall G$ (*Cover* $(G) \rightarrow$ *Cover* $(\bar{\bar{G}})$)

(1) *Cover* $(G_0) \rightarrow$ *Cover* $(\bar{\bar{G}}_0)$

No Match

Partial Match: Use PAIRS heuristic

Consult PAIRS table under "Cover"

(1.1) *Try* $(G_0 \subseteq \bar{\bar{G}}_0)$ *Fails*

(1.2) *Try* $(G_0 \subseteq\subseteq \bar{\bar{G}}_0)$ "T" *by REDUCE*

or

$(A_0 \in G_0 \Rightarrow \underbrace{C \in \bar{\bar{G}}_0 \wedge A_0 \subseteq C}_{(A_0 \in G_0 \Rightarrow B \in G_0 \wedge C = \bar{B} \wedge A_0 \subseteq C)})$ *defn of* $\subseteq\subseteq$

$(A_0 \in G_0 \Rightarrow B \in G_0 \wedge A_0 \subseteq \bar{B})$ *sub* =

SUBGOAL 1

(1.21) $(A_0 \in G_0 \Rightarrow B \in G_0)$ A_0/B

SUBGOAL 2

(1.22) $(A_0 \in G_0 \Rightarrow A_0 \subseteq \bar{A}_0)$ "T" *by REDUCE*

QED.

FIG. 40. An example using Bledsoe's PAIRS heuristic.

PAIRS table—

IN	*Pattern*	*Recommendations*
Cover	(Cover $(G) \rightarrow$ Cover (F))	$[(G \subseteq F)(G \subseteq\subseteq F) \ldots]$ .1 .2
Countable	(Countable $A \rightarrow$ Countable B)	$[(B \subseteq A)$ .1
	$(\exists f(f$ is a function $\wedge$ domain $f \subseteq A \wedge B \subseteq$ range $f)) \ldots]$ .2	

DEFINITION table—

$\bar{A} \equiv$ The Closure of A. (note: $A \subseteq \bar{A}$).

$\bar{G} \equiv \{\bar{A} : A \in G\}$

cover $(G, X) \equiv X \subseteq \bigcup_{\alpha \in G} \alpha$

$G \subseteq\subseteq F$ (G is a refinement of F)

$\equiv \forall A \in G\, \exists C \in F (A \subseteq C)$

REDUCE TABLE (single entry)

IN	*OUT*
$A \subseteq \bar{A}$	"T"
$G \subseteq\subseteq \bar{G}$	"T"

FIG. 41. A PAIRS table, definitions, and reductions, used in Fig. 40.

2.11. Models

In Fig. 22, we saw the flow chart of Gelernter's geometry prover, with its famous "diagram filter", being used to discard unwanted subgoals. This is an excellent example of a MODEL or counterexample being used to help with a proof. Since models and counterexamples play such crucial roles in mathematics it is not surprising that they have been found useful in automatic provers. We expect their role to be expanded.

Fig. 42 shows Reiter's Rule 4 (see [66]) and an explanation of how the model M is used in the execution of this rule; and Fig. 43 gives an example of a theorem being proved by his system. In this example the Model M might be, for example, the Klein-four Group (Fig. 44) in which the goal (122), $BB^{-1} = B^{-1}$ clearly fails. More complicated models are needed for other proofs, especially where commutativity is *not* assumed.

4. $\psi \vdash A \wedge B$

If $\psi \vdash A$ returns σ_1, $M \models_E B\sigma_1$,

and $\psi \vdash B\sigma$, returns σ_2

$\sigma_1\sigma_2$

Suppose that, during an attempted proof of A, x is instantiated by the term t. At this point, make the semantic test $M \models_E B(t)$. If successful, proceed with the proof of A. Otherwise, A's proof has obviously gone astray and must be redirected. Thus, rather than patiently waiting for A to deliver a (possibly wrong) σ_1, the wff B should be continuously semantically monitoring the proof of A, thereby minimizing the risk of receiving an incorrect σ_1. We believe that this kind of parallel processing of dependent subgoals will considerably alleviate the problem of back-up encountered by purely syntactic theorem-provers.

FIG. 42. Reiter's rule 4 in MATH-HACK for the use of models, and his accompanying comment.

THEOREM. *If S is a subset of a group such that $xy^{-1} \in S$ whenever x and $y \in S$, then $x^{-1} \in S$ whenever $x \in S$.*

Proof. $ex = x \wedge xe = x \wedge xx^{-1} = \underset{\alpha}{e} \wedge x^{-1}x = e \wedge b \in S$

$$\overline{\wedge\, (x \in S \wedge y \in S \wedge xy^{-1} = z \rightarrow z \in S)} \vdash b^{-1} \in S.$$

Backchain on α to get the subgoal

(1) $\vdash x \in S \wedge y \in S \wedge xy^{-1} = b^{-1}$

(11) $\vdash x \in S$ $\quad b/x$

(12) $\vdash y \in S \wedge by^{-1} = b^{-1}$

(121) $\vdash y \in S$ $\quad b/y$

(122) $\vdash bb^{-1} = b$

Fails in the model, so back up to (1). Reorder subgoals.

(11) $\vdash xy^{-1} = b^{-1}$ $\quad e/x, b/y$

(12) $\vdash e \in S \wedge b \in S$

Easily proved.

FIG. 43. An example proved by Reiter's system.

For Reiter example

	e	a	b	c
e	e	a	b	c
a	a	e	c	b
b	b	c	e	a
c	c	b	a	e

FIG. 44. The Klein-four group.

Some others using models and counterexamples in automatic proofs are:

Gelernter [26]—Geometry.
Slagle [72]—Resolution.
Reiter [66]—Groups.
Nevins [58]—Geometry.
Siklossy [70]—Robots (DISPROVER).
Winograd [84]—Block's world.
Ballantyne [3]—Topology.
Henschen [33]—Groups.

2.12. Analogy

Perhaps the biggest error made by researchers in automatic theorem proving has been in essentially ignoring the concept of *analogy* in proof discovery. It is the very heart of most mathematical activity and yet only Kling [39] has used it in an automatic prover. His paper showed how, with the use of knowledge, a proof in group theory could be used to help obtain a similar proof in ring theory.

We strongly urge that other workers in this field familiarize themselves with Kling's work and extend and apply them more effectively.

2.13. Man-machine

One of the most irksome things about current automatic theorem provers is the apparent need for the human user to prove the theorem himself before he gives it to the computer to prove. This is necessary because he must determine (for the computer) what axioms, or supporting theorems, are needed in the proof, and if he puts in too many, the proof will bog down. See [12, p. 45].

This problem is partially eliminated by the use of the various concepts mentioned above, such as procedures and REDUCTION tables, which effectively carry the information needed from some of these reference theorems, and are able to give this information when needed without slowing the system down. The remainder of the difficulty can be eliminated by having the human user insert reference theorems only when they are needed. See Fig. 45.

Also present systems cannot prove very hard theorems, so they don't get involved in interesting mathematics. We take as a maxim:

> Automatic provers will not compete successfully with humans for the next 100 years. Therefore the most effective systems will be those in which the computer acts as an *assistant* to the human user.

Thus it is imperative that this work attracts researchers from pure mathematics, and therefore, that interactive programs be made convenient for the *user*, not the programmer.

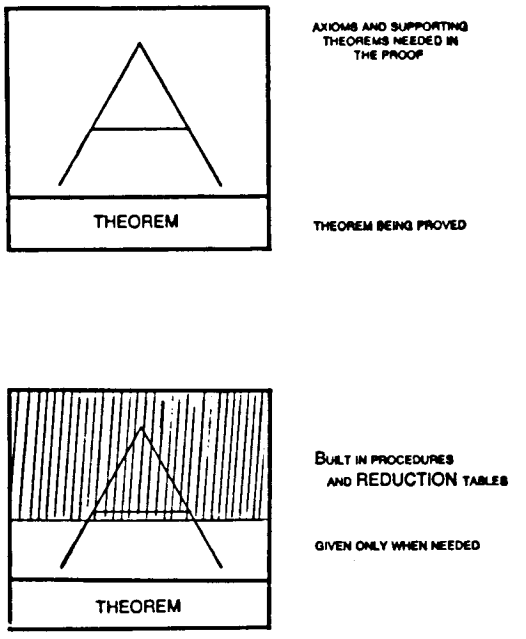


FIG. 45.

Some of the needs of the user mathematician are listed on Fig. 46. Point 3 Fig. 46 is important because a mathematician will not long use a system which repeatedly requires him to give trivial information to the system.

1. READ and easily COMPREHEND the scope.
2. FOLLOW the PROOF.
3. HELP COMPUTER *only* when needed.
4. Axioms and Reference theorems:
   (i) Built-in (some),
   (ii) Others added only when needed.
5. Convenient Commands.

FIG. 46. Some needs of a mathematician interacting with a mechanical prover.

We feel that a well-built system can be exercised on a *large* number of examples, thereby obtaining much valuable information on the utility of concepts in the program.

By running a large number of examples, the user can learn by experience, those places where he needs to improve the automatic part of the system, places where a little extra programming can greatly reduce the load on the human user.

This objective has been partly attained in an interactive program verification system [29, 13] which has been running for the last year in Ralph London's laboratory of the Information Science Institute, Los Angeles, and is now also running at the University of Texas. Peter Bruell, Mabry Tyson, and Larry Fagan were instrumental in developing this system. Much more needs to be done on it to make it truly effective.

Others who have (earlier) worked on interactive systems include: Guard, et al. [30]; Allen and Luckham [2]; Huet [37] and others.

3. Programming Languages

The new programming languages, such as PLANNER [34], MICRO-PLANNER [78], QA-4 [68], Q-LISP [65], and PLASMA [35], which have been proposed and/or implemented during the last few years have much to offer automatic theorem proving. Especially are they rich in concepts such as: Knowledge, data base, procedures, goal oriented, automatic backup, pattern directed invocation, demons, data types.

Also these languages have built-in structures and controls to handle the kinds of things we propose.

However, we do not believe that the lack of use of these programming languages in current automatic prover has hurt their performance. No proof of a hard theorem has been omitted because the user did not use one of these. This may not remain to be the case as automatic provers get more sophisticated, and as these languages get more powerful and efficient. Many of their features are ready-made for provers, and we should move toward adopting them, with needed modifications, for our use.

4. Comments

The reader should not get the idea that we have found the secret to automatic theorem proving. We believe in these concepts but are certain that others will evolve.

We have *talked* a lot and proved very *few* hard theorems (by computer) during the last several years. It is time to *do*, to show that our concepts are good. It is time to get a lot more *experience* with our provers. This will allow us to eliminate some of our "good" ideas.

It is *not* the time to give up on automatic theorem proving. How can that be advisable at a time when so little has been done to develop and apply the ideas we already have? For example, why doesn't someone else use analogy in automatic proofs?

One thing that would help push this field ahead, would be for authors to follow the practice of publishing the proof of at least one *hard theorem* in each new methods paper. We do not believe this field will remain vital unless we develop truly powerful provers, and not just theories.

Completeness in itself is not a bad concept, if handled correctly. For example, a complete unification system with built-in associativity and commutativity, such as [77], needs to be reworked in a way that will make it a useful part of a practical prover. It is believed that a properly constructed *overdirector* (see II.7) can so direct the search that one can have both efficiency and (essential) completeness. At least we can try for this.

"Trapping" remains a serious problem, whereby a substitution a/x that satisfies a goal $P(x)$ may fail on $Q(x)$, and hence on

$$(P(x) \wedge Q(x)).$$

Backing-up theoretically solves this but can be very time consuming. Huet's "delaying" as used for matching in higher order logic [36] might be a good idea here.

Another worry is the "learning" problem. During the last decade most researchers in A.I. have avoided machine learning, because of such poor results from earlier experiments, and have favored the use of Man's "knowledge" in A.I. programs or human imposed learning such as used by Winston [85]. However, eventually that barrier must be removed if the automatic prover is to be very effective. Figs 35 and 33 and accompanying comments provide an example of the kind of controlled learning that might be useful.

Other work such as studies on induction (by Meltzer [53] and others) might be important to our efforts.

One should also not ignore proof checking as a potential use for automatic theorem proving [1, 11, 51], and also computer aided teaching of mathematics [47].

Another recent effort which may have a far reaching effect on automatic theorem proving is the work of Lenat [42A] which automatically discovers concepts in mathematics.

5. Challenges

Let me close by suggesting a few theorems, from various fields of mathematics, whose proofs by automatic means would be impressive at this time or in the near future. See Table 1.

In our efforts to mold our experience into an effective theorem prover, we are reminded of a 1918 statement by Albert Einstein [62]:

> Man tries to make for himself in the fashion that suits him best a simplified and intelligible picture of the world. He then tries to some extent to substitute this cosmos of his for the world of experience, and thus to overcome it . . . .
> The supreme task . . . is to arrive at those universal elementary laws from which the cosmos can be built up by pure deduction. There is no logical path to these laws; only intuition, resting on sympathetic understanding of experience, can reach them . . . .

TABLE 1. Some challenging theorems for automatic provers

Field	Theorems Proved	Challenge
Set theory	Elementary set theory $\omega = \omega \cap \text{Subsets}(\omega)$ $\omega = \bigcup_{\alpha \in \omega} \alpha$	Schoeder-Bernstein Theorem
Calculus	Limit Theorems (with Limit Heuristic)	Limit Theorems (ω/o Limit Heuristic) Rolle's Theorem $\int f\,dx$ exists for f continuous
Analysis	Bolzano Weierstrass Theorem Cont. *fcn* on a Compact set is Uniformly Cont. (Using Non-Standard Anal.)	Bolzano Weierstrass Theorem (ω/o Non-Standard Analysis) Cont. *fcn* on a Compact Set is Uniformly Cont. (ω/o Non-Standard Analysis) Heine Borel Theorem Hahn Banach Theorem
Geometry	Gelernter's	Pythagorean Theorem
Topology	Open $(A) \wedge$ Open $(B) \rightarrow$ Open $(A \cup B)$	A separable, normal space is metrizable Tichenoff Theorem
Algebra	Group Right Identity $x+(x+x)=0 \rightarrow a+b+(-a)+b+a+(-b)+(-a)+(-b)=0$	Ring $x^3 = x$ for $x \neq 0 \rightarrow a \cdot b = b \cdot a$

REFERENCES

1. Abrahams, P. W., in: *The Programming Language* LISP*: Its Operation and Applications*, Application of LISP to checking mathematical proofs (The M.I.T. Press, Cambridge, MA, 1966) 137–160.
2. Allen, J. and Luckham, D., An interactive theorem-proving program, *Machine Intelligence* 5 (1970) 321–336.
3. Ballantyne, M., Computer generation of counterexamples in topology, The University of Texas at Austin Math. Dept. Memo ATP-24 (1975).
4. Ballantyne, M. and Bennett, W., Graphing methods for topological proofs, The University of Texas at Austin Math. Dept. Memo ATP-7 (1973).
5. Ballantyne, A. M. and Bledsoe, W. W., Automatic proofs of theorems in analysis using non-standard techniques, The University of Texas at Austin Math. Dept. Memo ATP-23 (July 1975); *J. ACM*, to appear, July 1977.

5A. Ballantyne, A. M. and Lankford, D.S., An implementation of derived reduction, Local Memo ATP Project, Dept. of Math. and Comput. Sci., University of Texas, Austin, TX (August 1975).

6. Bibel, W. and Schreiber, J., Proof search in a Gentzen-like system of first order logic, Bericht Nr. 7412, Technische Universität (1974).
7. Bledsoe, W. W., Splitting and reduction heuristics in automatic theorem proving, *Artificial Intelligence* 2 (1971) 55–77.
8. Bledsoe, W. W., The sup-inf method in Presburger arithmetic, Dept. of Math., The University of Texas at Austin, Memo ATP-18 (December 1974); essentially the same as: A new method for proving certain Presburger formulas, *Fourth Int. Joint Conf. Artificial Intelligence*, Tbilisi, U.S.S.R., September 3–8 (1975).
9. Bledsoe, W. W., Boyer, R. S. and Henneman, W. H., Computer proofs of limit theorems, *Artificial Intelligence* 3 (1972) 27–60.
10. Bledsoe, W. W. and Bruell, P., A man–machine theorem-proving system, in: *Adv. Papers 3rd Int. Joint Conf. Artificial Intelligence* (1973) 55–65; also *Artificial Intelligence* 5 (1974) 51–72.
11. Bledsoe, W. W. and Gilbert, E. J., Automatic theorem proof-checking in set theory, Sandia Corp. Research Report, SC-RR-67-525 (July 1967).
12. Bledsoe, W. W. and Tyson, M. The UT interactive theorem prover, The University of Texas at Austin Math. Dept. Memo ATP-17 (May 1975).
13. Bledsoe, W. W. and Tyson, M., Typing and proof by cases in program verification, The University of Texas at Austin Math. Dept. Memo ATP-15 (May 1975). In MI 8.
14. Boyer, R. S. and Moore, J. S., Proving theorems about Lisp functions, *J. ACM*, 22 (1975) 129–144.
15. Brown, F., Unfinished Ph.D. thesis on automatic theorem proving, University of Edinburgh (1975).
16. Bruell, P., A description of the functions of the man–machine topology theorem prover, The University of Texas at Austin Math. Dept. Memo ATP-8 (1973).
17. Bundy, A., Doing arithmetic with diagrams, *Adv. Papers 3rd Int. Joint Conf. Artificial Intelligence* (1973) 130–138.
18. de Carvalho, R. L., Some results in automatic theorem-proving with applications in elementary set theory and topology, Ph.D. Thesis, Dept. of C.S., University of Toronto, Canada; Tech. Report No. 71 (November 1974).
19. Chang, C., and Lee, R. C., *Symbolic Logic and Mechanical Theorem Proving* (Academic Press, New York, 1973).
20. Cooper, D. C., Theorem proving in computers, in: Fox, L. (Ed.), *Advances in Programming and Non-numeric Computation* (Pergamon, NY, 1966) 155–182.
21. Darlington, J. L., Automatic theorem proving with equality substitution and mathematical induction, *Machine Intelligence* 3 (1968) 113–127.
22. Deutsch, L. P., An interactive program verifier, Ph.D. Thesis, University of California, Berkeley (1973); also Xerox Palo Alto Research Center Report CSL-73-1 (May 1973).
23. Ernst, G. W., The utility of independent subgoals in theorem proving, *Information and Control* (April, 1971); A definition-driven theorem prover, *Int. Joint Conf. Artificial Intelligence*, Stanford, CA (August 1973) 51–55.
24. Fishman, D. H., Experiments with a resolution-based deductive question-answering system and a proposed clause representation for parallel search, Ph.D. Thesis, Dept. of Comp. Sci., University of Maryland (1973).
25. Fronig, X., private communication, Institut fur informatik, Universität Bonn (1975).
26. Gelernter, H., Realization of a geometry theorem-proving machine, *Proc. Int. Conf. Information Processing*, Paris UNESCO House (1959) 273–282.
27. Gentzen, G., Untersuchungen über das logische Schliessen I, *Math. Zeitschrift* 39 (1935) 176–210.
28. Goldstein, I., Elementary geometry theorem proving, M.I.T.-AI Lab Memo 280 (April 1973).
29. Good, D. I., London, R. L. and Bledsoe, W. W., An interactive verification system, *Proc. Int. Conf. Reliable Software*, Los Angeles (April 1975) 482–492; *IEEE Trans. on Software Engineering* 1 (1975) 59–67.
30. Guard, J. R., Oglesby, F. C., Bennett, J. H. and Settle, L. G., Semi-automated mathematics, *J. ACM* 16 (1969) 49–62.

31. Hayes, P., Forthcoming book on automatic theorem proving, University of Essex.
32. Hearn, A. C., Reduce 2: A system and language for algebraic manipulation, *Proc. ACM 2nd Symp. Symbolic and Algebraic Manipulation* (1971) 128–133; also Reduce 2 User's Manual, 2nd ed. (University of Utah, Salt Lake City, UCP-19, 1974).
33. Henschen, L. J., Semantic resolution of horn sets, *Adv. Papers Int. Joint Conf. Artificial Intelligence*, Tbilisi, U.S.S.R. (September 1975).
34. Hewitt, C., Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot, Ph.D. Thesis (June, 1971); AI-TR-258 M.I.T.–AI-Lab. (April 1972).
35. Hewitt, C., How to use what you know, M.I.T.-AI Lab. working paper 93, (May 1975).
36. Huet, G. P., Constrained resolution: a complete method for higher order logic, Ph.D. thesis, Report 1117, Jennings Computing Center, Case Western Reserve University.
37. Huet, G. P., Experiments with an interactive prover for logic with equality, Report 1106, Jennings Computing Center, Case Western Reserve University.
38. King, J. C., A program verifier, Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, PA (1969).
39. Kling, R. E., A paradigm for reasoning by analogy, *Artificial Intelligence* **2** (1971) 147–178.
39A. Knuth, D. E., Notes on central groupoids, *J. Combinatorial Theory* **8** (1970) 376–390.
40. Knuth, D. E. and Bendix, P. B., Simple word problems in universal algebras, in: Leech, J. (Ed.), *Computational Problems in Abstract Algebra* (Pergamon Press, Oxford, 1970) 263–297.
41. Lankford, D. S., Complete sets of reductions for computational logic, The University of Texas at Austin Math. Dept. Memo ATP-21 (January 1975).
42. Lankford, D. S., Canonical algebraic simplification in computational logic, The University of Texas at Austin Math. Dept. Memo ATP-25 (1975).
42A. Lenat, D. B., AM: An artificial intelligence approach to discovery in mathematics as heuristic search, Stanford AI Lab. Memo AIM-286 (July 1976).
43. Lifsic, V. A., Specialization of the form of deduction in the predicate calculus with equality and function symbols, *Proc.* steklov *Inst. Math.* **98** (1968) 1–23.
44. Loveland, D. W., *Automatic Theorem Proving: A Logical Basis* (North-Holland, Amsterdam, 1977).
45. Loveland, D. W. and Stickel, M. E., A hole in goal trees: some guidance from resolution theory, *Proc. Third Int. Joint Conf. Artificial Intelligence*, Stanford, CA (1973) 153–161.
46. Luya, B., Un systeme complet de deduction naturelle, Thesis, University of Paris VII (January 1975).
47. Marinov, V., An interactive system for teaching set theory by computer at IMSSS, Ventura Hall, Stanford, private communication.
48. Ju Maslov, S., Proof-search strategies for methods of the resolution type, *Machine Intelligence* **6** (1971) 77–90.
49. Maslov, Ju S., An inverse method of establishing deducibility in classical predicate calculus, *Dokl. Nauk SSSR* **159** (1964) 17–20.
49A. Matijasevic, J. V., Simple examples of undecidable associative calculi, *Soviet Math. Dokl.* **8** (1967) 555–557.
50. McCarthy, J., Programs with common sense. (The advice taker), in: Minsky, M. (Ed.), *Semantic Information Processing*, 403–418.
51. McCarthy, J., Computer programs for checking mathematical proofs, *Proc. Amer. Math. Soc. Recursive Function Theory*, New York (April 1961).
52. McDermott, D. V. and Sussman, G. J., The CONNIVER reference manual, AI Memo 259, M.I.T.-AI-Lab (May 1972); revised (July 1973).
53. Meltzer, B., The programming of deduction and induction, University of Edinburgh, Dept. of Art. Int., DCL Memo 45 (1971); also *Artificial Intelligence* **1** (1970) 189–192.
54. Minker, J., Fishman, D. H. and McSkimin, J. R., The Q* algorithm—A search strategy for a deductive question-answering system, *Artificial Intelligence* **4** (1973) 225–243.
55. Minsky, M., A framework for representing knowledge, in: Winston, P. (Ed.), *The Psychology of Computer Vision* (McGraw-Hill, New York, 1974).
55A. Moore, R. C., Reasoning from incomplete knowledge in a procedural deduction system, M.I.T.-AI Lab Memo AI-TR-347 (December 1975).
56. Nevins, A. J., A human oriented logic for automatic theorem proving, M.I.T.-AI-Lab Memo 268 (October 1972); *J. ACM* **21** (1974) 606–621.
57. Nevins, A. J., A relaxation approach to splitting in an automatic theorem prover, M.I.T.-AI-Lab. Memo 302 (January 1974); *Artificial Intelligence*, **6** (1975) 25–39.
58. Nevins, A. J., Plane geometry theorem proving using forward chaining, M.I.T.-AI-Lab. Memo 303 (January 1974). *Artificial Intelligence*, **6** (1975) 1–23.
59. Newell, A., Shaw, J. C. and Simon, H. A., Empirical explorations of the logic theory machine: a case in heuristics, RAND Corp. Memo P-951 (28 February 1957); *Proc. Western Joint Computer Conf.* (1956) 218–239; also in: Feigenbaum and Feldman (Eds.) *Computers and Thought*, 134–152.
60. Newell, A., Shaw, J. C. and Simon, H. A., Report on a general problem-solving program, RAND Corp. Memo P-1584 (30 December 1958).
61. Nilsson, N., Artificial Intelligence (including a review of automatic theorem proving) IFIP, Stockholm, Sweden (1974).
61A. Overbeek, R. A., A new class of automatic theorem proving algorithms, *J. ACM* **21** (1974) 191–200.
62. Pirsig, R. M., *Zen and the Art of Motorcycle Maintenance*, 106–107.
63. Plotkin, G. D., Building equational theories, *Machine Intelligence* **7** (1972) 73–89.
64. Prawitz, D., An improved proof procedure, *Theoria* **25** (1960) 102–139.
65. Reboh, R. and Sacerdoti, E., A preliminary QLISP manual, Stanford Research Inst., AI Center Tech. Note 81 (August 1973).
66. Reiter, R., A semantically guided deductive system for automatic theorem proving, *Proc. Third Int. Joint Conf. Artificial Intelligence* (1973) 41–46; *IEEE Trans. on Elec. Computing* C-25 (1976) 328–334.
67. Reiter, R., A paradigm for automated formal inference, *IEEE Theorem Proving Workshop*, Argonne Nat. Lab., IL (3-5 June 1975).
67A. Robinson, G. and Wos, L., Paramodulation and theorem-proving in first-order theories with equality, *Proc. IRIA Symposium on Automatic Demonstration*, Versailles, France (1968); (Springer-Verlag, Berlin, 1970) 276–310.
68. Rulifson, J. R., Derksen, J. A. and Waldinger, R. J., QA4: a procedural calculus for intuitive reasoning, Stanford Research Inst. AI Center, Stanford, CA Tech. Note 13 (November 1972).
69. Shostak, R. S., On the completeness of the sup-inf method, Stanford Research Institute, Report (1975).
70. Siklossy, L. and Roach, J., Proving the impossible is impossible is possible: disproofs based on hereditary partitions, *Int. Joint Conf. Artificial Intelligence* (1973) 383–387.
71. Siklossy, L., Rich, A. and Marinov, V., Breadth-first search: some surprising results, *Artificial Intelligence* **4** (1973) 1–28.
72. Slagle, J. R., Automatic theorem proving with renamable and semantic resolution, *J. ACM* **14** (1967) 687–697.
73. Slagle, J. R., Automated theorem-proving for theories with simplifiers, commutativity and associativity, *J. ACM* **21** (1974) 622–642.
74. Slagle, J. R., Automatic theorem proving with built-in theories of equality, partial order and sets, *J. ACM* **19** (1972) 120–135.
75. Slagle, J. R. and Norton, L., Experiments with an automatic theorem prover having partial ordering rules, *Commun. ACM* **16** (1973) 682–688.
76. Norton, L. M., Experiments with a heuristic theorem-proving program for the predicate calculus with equality, *Artificial Intelligence* **2** (1971) 261–284.

77. Stickel, M., A complete unification algorithm for associative-commutative functions, *Adv. Papers, Int. Joint Conf. Artificial Intelligence*, Tbilisi, U.S.S.R (September 1975) 71–76.
78. Sussman, G. J., Winograd, T. and Charniak, E., Micro-planner manual, M.I.T.-AI Lab. Memo 203A (December 1971).
79. Suzuki, N., Verifying programs by algebraic and logical reduction, *Proc. Int. Conf. Reliable Software* (1975) 473–481.
80. Ullman, S., Model-Driven Geometry Theorem Prover, M.I.T.-AI Lab Memo 321 (May 1975).
81. Waldinger, R. J. and Levitt, K. N., Reasoning about programs, *Artificial Intelligence* **5** (1974) 235–316.
82. Wang, H., Toward mechanical mathematics, *IBM J. Res. Dev.* **4** (1960) 224–268.
83. Winker, S. K., Complete demodulations in automatic theorem proving, University of Northern Illinois, Computer Science Department (July 1975).
84. Winograd, T., Procedures as a representation for data in a computer program for understanding natural language, Ph.D. Thesis, M.I.T. MAC-TR-84 (February 1971).
85. Winston, P. H., Learning structural descriptions from examples, in: Winston, P. H. (Ed.), *The Psychology of Computer Vision* (McGraw-Hill, New York, 1974).