A Survey of Automated Deduction

Woody Bledsoe
University of Texas
Computer Sciences Department
Taylor Hall 2.124
(512) 471-9568
ai.bledsoe@r20.utexas.edu

Richard Hodges

January 29, 1988

Abstract

DRAFT

This is an enlarged version of a survey talk given by Woody Bledsoe at the AAAI National Conference, Seattle, Washington, July 16, 1987 and will appear in a collection of survey talks published by Morgan Kaufmann, Los Altos, CA.

Contents

[1]now at Univ. British Columbia

1 INTRODUCTION

What is Automated Deduction?

It includes many things. A part of it involves *proving theorems by computer*, theorems like the Pythagorean theorem from Plane Geometry (Figure 1) or the theorem: If an equilateral triangle is inscribed in a circle, and lines are drawn from its corners to a point on the circumference, then the length of the longest such line is equal to the sum of the lengths of the others. (Figure 1.)

Figure 1 near here.

Or theorems from algebra such as:

> A group for which $x^2 = e$ for each of its elements x, is commutative.
>
> A ring for which $x^3 = x$ is commutative.

Or theorems from analysis such as the *maximum value theorem* and the *intermediate value theorems*, depicted in Figure 2:

Figure 2 near here.

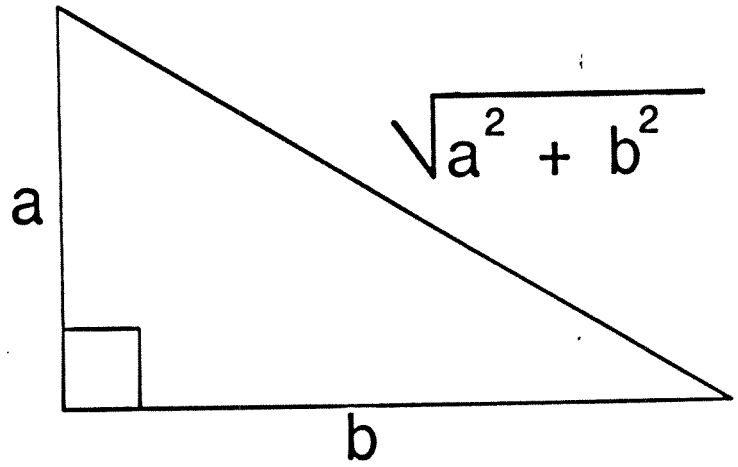
> A Continuous Function f defined on a closed interval $[a, b]$, attains its Maximum (and Minimum) on that interval.
>
> And if $f(a) < 0$ and $f(b) > 0$, then $f(x) = 0$ for some x in $[a, b]$.

Also puzzles such as the *truthtellers and liars* one, can be solved by theorem proving. See [LO85].

> On a certain island the inhabitants are partitioned into those who always tell the truth and those who always lie. I landed on

EXAMPLE THEOREMS FROM GEOMETRY

Pythagorean Theorem:



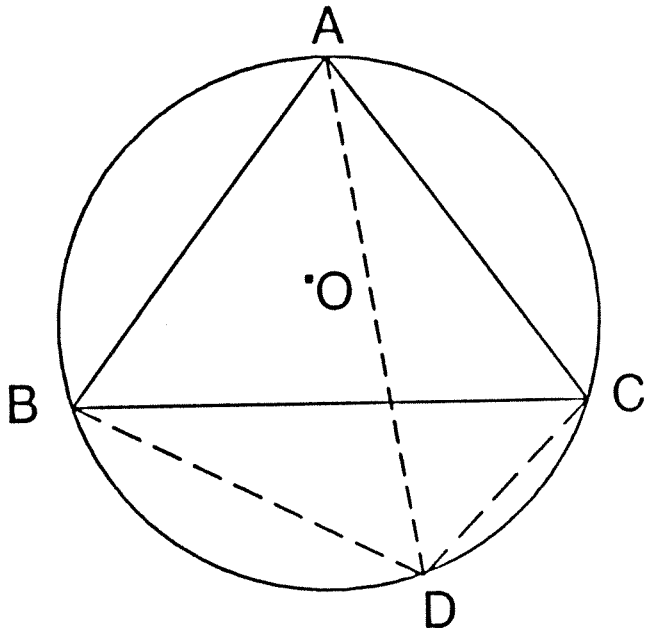
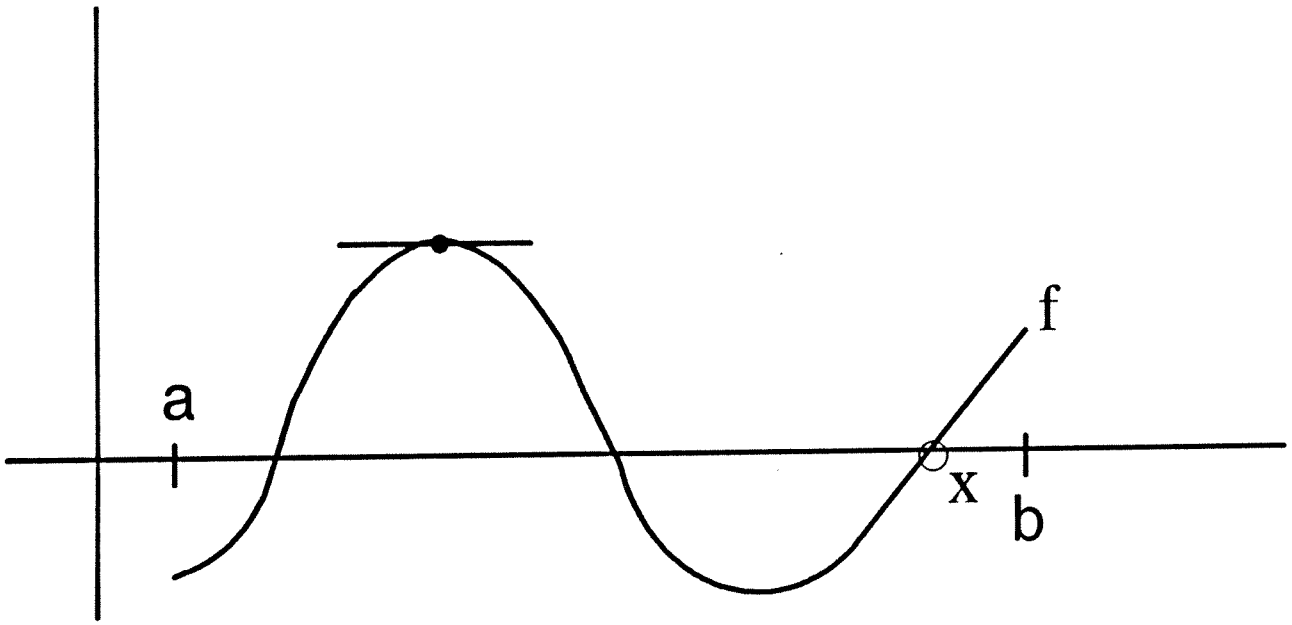


Figure 1

EXAMPLES FROM ANALYSIS



Maximum Value Theorem and Intermediate Value Theorem, for continuous functions.

Figure 2

the island and met three inhabitants A, B, and C. I asked A, 'Are you a truthteller or a liar?' He mumbled something which I couldn't make out. I asked B what A had said. B replied, 'A said he was a liar'. C then volunteered, 'Don't believe B, he's lying!'.

What can you tell about A, B, and C?

The halting problem theorem (figure 3) shows how complicated these theorems can get, and others more so.

Figure 3 near here.

1.1 Facets of Automated Deduction

What is Automated Deduction? It is a number of things. But in all cases one is *making deductions* by computer. It is often called *Automated Theorem Proving* (ATP), or *Automatic Reasoning (AR)*. We will use these terms interchangeably.

Let me list some of the *facets* and *applications* of Automated Deduction. See Figure 4.

Figure 4 near here.

We consider *proof discovery* to be the major component of ATP, because every application of ATP uses some amount of automatic proof discovery. We will tend to concentrate on it in this talk, since we are personally interested in it, and will discuss the others only briefly, if at all. There are a number of review papers and references for each of these areas. One might add to this list: *all non-numeric programming*, since some form of inferencing is involved in all of it.

Automatic proof checking is a very important part of AR (see, for example, [BM82, Con85, Hun85, We77]) but will be discussed only briefly here. The reader is referred to [MS84] for a report on using ATP in CAI.

HALTING PROBLEM IS UNSOLVABLE
(Burkholder)

(1) $(Ex)[Gx \,\&\, (Ay)(Py \rightarrow (Az)Dxyz)] \rightarrow (Ew)[Pw \,\&\, (Ay)(Py \rightarrow (Az)Dwyz)]$

(2) $(Aw)([Pw \,\&\, (Ay)(Py \rightarrow (Az)Dwyz)] \rightarrow (Ay)(Az)([Py \,\&\, H_2yz) \rightarrow (H_2wyz \,\&\, Owg)]$
$\&\, [(Py \,\&\, \neg H_2yz) \rightarrow (H_3wyz \,\&\, Owb)]))$

(3) $(Ew)[Pw \,\&\, (ay)([Py \,\&\, H_2yy) \rightarrow (H_2wyy \,\&\, Owg)] \,\&\, [(Py \,\&\, \neg H\ yy)$
$\rightarrow (H_2wyy \,\&\, Owb)])] \rightarrow (Ev)[Pv \,\&\, (Ay)([Py \,\&\, H_2yy) \rightarrow (H_2vy \,\&\, Ovg)]$
$\&\, [(Py \,\&\, \neg H_2yy) \rightarrow (H_2vy \,\&\, Ovb)])]$

(4) $(Ev)[Pv \,\&\, (Ay)([Py \,\&\, H_2yy) \rightarrow (H_2vy \,\&\, Ovg)] \,\&\, [(Py \,\&\, \neg H_2yy) \rightarrow (H_2vy \,\&\, Ovb)])]$
$\rightarrow (Eu)[Pu \,\&\, (Ay)([(Py \,\&\, H_2yy) \rightarrow \neg H_2uy] \rightarrow \neg H_2uy] \,\&\, [(Py \,\&\, \neg H_2yy)$
$\rightarrow (H\ yy) \rightarrow (H_2uy \,\&\, Oub)])].$

$\longrightarrow$

(5) $\neg(Ex)\,[G\,x \,\&\, (Ay)\,(Py \rightarrow (Az)\,DxyZ)]$

Figure 3

APPLICATIONS OF ATP

Proof Discovery

Proof Checking: Including Computer-Aided Instruction

Interactive Provers (Man-machine)

Logic Programming & Programming Languages

Deductive Data Bases

Program Verification & Automatic Programming

Expert-Systems Inferencing

Algebraic Manipulation (such as Macsyma)

Proof Representation & Manipulation

Figure 4

We will also not discuss *interactive provers*, but consider this to be one of the most important areas of ATP. See [BBr73, BM79,].

We will discuss *logic programming* shortly. Many efforts are underway to combine logic and functional programming languages such as PROLOG and LISP, and to join this with *rapid type inheritance*, to make it easier to write AI applications, and attain greater speed. See, for example, [AN85].

In the near future we expect to see an increased research effort on *deductive data bases*, especially for very large collections of *facts* and *rules*, written in logic, and requiring *a great deal of inferencing* to answer a query. See [GM78] for a review and also [HN84] for an example of *compiling DB* queries, to speed up retrieval.

Such a DB might contain the facts about a corporation and its operating "rules". Similarly for a political situation, such as the Middle East (will country X cut off the oil or go to war), and for military situations. We believe that a structured knowledge base of *general* (common-sense) knowledge, such as [Le86], will play a big role in these efforts.

Program verification (e.g., [Good85, BM79]) and automatic programming [MW85] continue to be significant application areas for ATP. Algebraic manipulation [Buch83], as represented by MACSYMA [MAC] and other systems, has grown to be a sizable part of AR.

Of most interest to the AI community is automatic inference associated with Expert Systems and related "intelligent" programs. In this conference alone there were 46 papers (out of 150) related to automatic reasoning. We expect that trend to continue, especially as AI programs are being based more on traditional logic and extensions of it. Here we could include *non-monotonic* reasoning (e.g., circumscription) [McC80] Truth Maintenance [Do79, deK84], common-sense reasoning [McC, Le86], qualitative reasoning, (see, for example, [deK84, Fo84, Ku86]), meta Reasoning [GGS83, GN87].

1.2 Proof Representation & Manipulation

Another branch of automated deduction studies methods of representing and transforming proofs. Human mathematicians seem to be able to understand a proof as a whole, wheras automated deduction systems tend to have a very narrow view, centered around a single clause or a small group of clauses at any one time.

One reason for wanting to be able to manipulate proofs is to facilitate higher-level strategies for proof discovery. The method of proof by analogy is an area which needs the ability to transform proofs, to extract the abstract content of a proof, and to annotate proofs with additional information such as the "motivation" for a given step. (See Section 5.2.1).

The internal representations used in automated deduction are often not very easy for people to understand. Many theorem provers use clausal resolution. But putting a theorem into clauses often introduces redundancy and obscures the logical structure of the theorem and its proof. Observing that it is often much easier to understand a proof in natural deduction format, Peter Andrews and Dale Miller have developed algorithms for transforming resolution proofs into an intermediate form called an "expansion tree" and then into a natural deduction proof [An81]. Amy Felty, a student of Miller, has recently developed a system to translate proofs into natural English. These systems use "higher order logic" (see section 3.3) and have automatically proven Cantor's theorem and a version of Russell's paradox.

A group of Systems [GMW82, Ne80, Card86, CoH85, Cons86, deB80] have been developed for representing and checking mathematical proofs using a higher order logic based on the Curry-Howard isomorphism between propositions and lambda-types (see section 3.4) These systems have also been used for verifying software and hardware [G087]. Proofs often can be written in a form much closer to that used by a human mathematician than by employing first-order predicate calculus and resolution. So far,

little work has been done on proof-discovery in these systems.

McAllester (MIT) has developed a theorem prover with set theory "built-in" and with a novel concept for proof guidance: the user specifies a "focus object" and the prover tries to forward chain from established facts to prove everything it can about the selected object. The prover can then search using patterns to see if anything useful has been proved. This seems potentially useful as a representation for motivation in proofs. His ONTIC has been used to proof-check the Stone Representation Theorem as well as others [McA87].

Weyrauch [We77, We82] has developed a system called FOL in which the syntax and reasoning rules of a deductive system can be formalized in First Order Logic. In paricular, FOL can formalize its own logic. It can conduct reasoning about proofs and about its own rules of inference. New rules can be verified using the deductive capabilities of FOL and can be added declaratively to the set of meta-theorems representing facts FOL knows about itself.

2 REFERENCES

There have been a number of excellent *review papers* of ATP during the last few years. Perhaps the review by Loveland [lo84] or [Bhe85] (in the first issue of the Journal of Automated Reasoning, 1985) would be the best for the beginner. In that same issue of JAR is an extended review of AR. Those interested in the prehistory and early history of ATP should see Martin Davis' [Da83]. Also see [WH83]. Bill Pase, of I.P. Sharp Associates, has recently revised his 70 page bibliography of Automated Deduction, which is very useful for those serious about this subject. [Pa87]

There are a number of books and collections of important papers which are introductory to the subject. For example, [CL73, Lo78, Bi82-87, Wo84, GN87, Ko79, Bu83, An86, IEEE-C25, Wo87, BM79, Sw83, BL84]. Also

there are chapters on ATP in various books on AI such as [Nil80, Rich83], and various Journals and Conference Proceedings (JAR, AAR Newsletter, CADE Reports, AI Journal, MI Series, AAAI, IJCAI, IEEE Transactions PAMI and SSC, etc.).

Other books of related interest include Konolige [Kon86] on representing the capabilities of intelligent agents with imperfect reason; and Smullyan's books of logic puzzles, especially [Smu85], a good source of challenge problems for ATP systems.

3 BRIEF HISTORY OF AUTOMATED DEDUCTION

Modern ATP was born in the middle 1950's with the "Logic Machine" of Newell, Simon, and Shaw [NSS56]. Gelernter's "Geometry Machine" [Ge59] followed in the late 50's, as well as other interesting work by Hao Wang [WaH60], Davis and Putnam [DP60] and many others (see [Da83]). But it was the advent of J. A. Robinson's RESOLUTION paper [Ro65] in 1965 that forever changed this field.

Also note that Maslov's *inverse method* [Mas68] stems from the mid 60's. (Vladimir Lifschitz has recently completed an excellent paper [Li87] simplifying the presentation of this powerful method.)

Other proof procedures, such as the so called "Natural Deduction" Provers [Wah60, Bl75, Lo78, Bl77, Pl82], Model Elimination, Connection and Mating Methods [An81, Bi82], Interconnectivity graphs [Ko75, Si76], Semantic Tableaux [Op1, Smu68], and the earlier "inverse Methods" of Maslow [Mas 68], have much in common with Resolution and also suffer many of its shortcomings.

Still, we believe that the introduction of resolution represents the single most important event in ATP so far. What is it?

3.1 Resolution

The basic idea of Resolution is simple and is very easy to learn. See, for example, the presentation in [CL73]. It is based upon the *modes ponens* rule, or more generally the *chain rule*. Referring to Figure 5, if the chain rule is converted to *clausal form* (by replacing an expression $x \rightarrow y$ by $(\neg x \vee y)$ then the rule is effected by cancelling the q and $\neg q$ in the upper clauses. Shown at the bottom of Figure 5, is the Resolvent Rule for first order logic, where *unification* is required; here the variable x is bound to the term a.

Figure 5 near here.

Figure 6 shows a resolution proof of a simple theorem. Note that the hypotheses are converted to clausal form and the conclusion is negated. Then clauses are resolved until a contraction, $\square$, is reached.

Figure 6 near here.

For Propositional Logic (where no variables are to be bound) Resolution is quite simple:

RESOLUTION RULE

1. Negate Theorem
2. Put in "Clausal Form"' (i.e., Conjunctive Normal Form, CNF)
3. Resolve until a contradiction, $\square$, is obtained

Now let us look at Resolution for First Order Logic (FOL). Figure 7 shows some expressions in FOL and a theorem. One is dealing here with quantifiers and variables. In order to prove this by resolution we must convert it to *clausal* form. (Figure 8) First each hypothesis is ***skolemized*** by removing the quantifiers.

RESOLVENT RULE

OD

MODES PONENS

$$\frac{p, \quad p \rightarrow q}{q}$$

CHAIN RULE

$$\frac{p \rightarrow q, \quad q \rightarrow r}{p \rightarrow r}$$

RESOLVENT RULE

$$\frac{\neg p \ v \ q, \quad \neg q \ v \ r}{\neg p \ v \ r}$$

$$\frac{\neg p(x) \ v \ q(x), \quad \neg q(a) \ v \ r}{\neg p(a) \ v \ r}$$

Figure 5

EXAMPLE Resolution Proof

Theorem: $[(p \rightarrow q) \,\&\, p] \rightarrow q$

Use CONTRADICTION. (Clauses)

1.	$\neg p \vee q$	4. q	1, 2
2.	p	5. $\square$	3, 4
3.	$\neg q$	"box"	

Actually:

$$\left\{\begin{array}{l}\neg p \vee q \\ p \\ \neg q\end{array}\right\} \Rightarrow \left\{\begin{array}{l}\neg p \vee q \\ p \\ \neg q \\ q\end{array}\right\} \Rightarrow \left\{\begin{array}{l}\neg p \vee q \\ p \\ \neg q \\ q \\ \square\end{array}\right\}$$

Figure 6

Figure 7 and 8 near here.

In the first hypothesis, the expression is true for all x and y, so we discard the quantifiers, and remember that we can replace x and y by any term we please in the proof. We also convert the implication as before. Similarly in the next hypothesis, except that we require a skolem function. For each p, there exist a z such that Mother(z, p). It is clear that z *depends* on p, so we show that dependence by replacing z my the expression $m(p)$. The conclusion is negated (since resolution uses Contradiction). The z remains a variable that also might be replaced with a term. Figure 9 shows the corresponding clauses and the derivation of $\square$ by resolution. There, x, y, p, and z are variables, and John and m are constants. The proof goes as before except that some of the variables are *bound* in the process. These bindings are called a *substitution*. The process of determining the substitution is called *unification*. Two formulas are UNIFIED (made one) in the process.

Figure 9 near here.

For example, the pair

$$P(g(x), x)$$

$$P(y, x0)$$

are unified by the substitution $[x \leftarrow x0,\ y \leftarrow g(x0)]$ (where x and y are variables and g and $x0$ are function symbols)

But the pair

$$P(g(x), x)$$

$$P(y, h(y))$$

has no unifier. Why?

The first step in trying to unify

$$P(g(x), x)$$

FIRST ORDER LOGIC

Girl (x), Female (x), Person (p)

THEOREM:

$\forall x \quad \forall y \; [\text{Mother } (x,y) \longrightarrow \text{Female } (x)] \;\&$

$\forall p \; [\text{Person } (p) \longrightarrow \exists z \; \text{Mother } (z,p)] \;\&$

Person (John)

$\longrightarrow \exists z \; \text{Female } (z)$

Figure 7

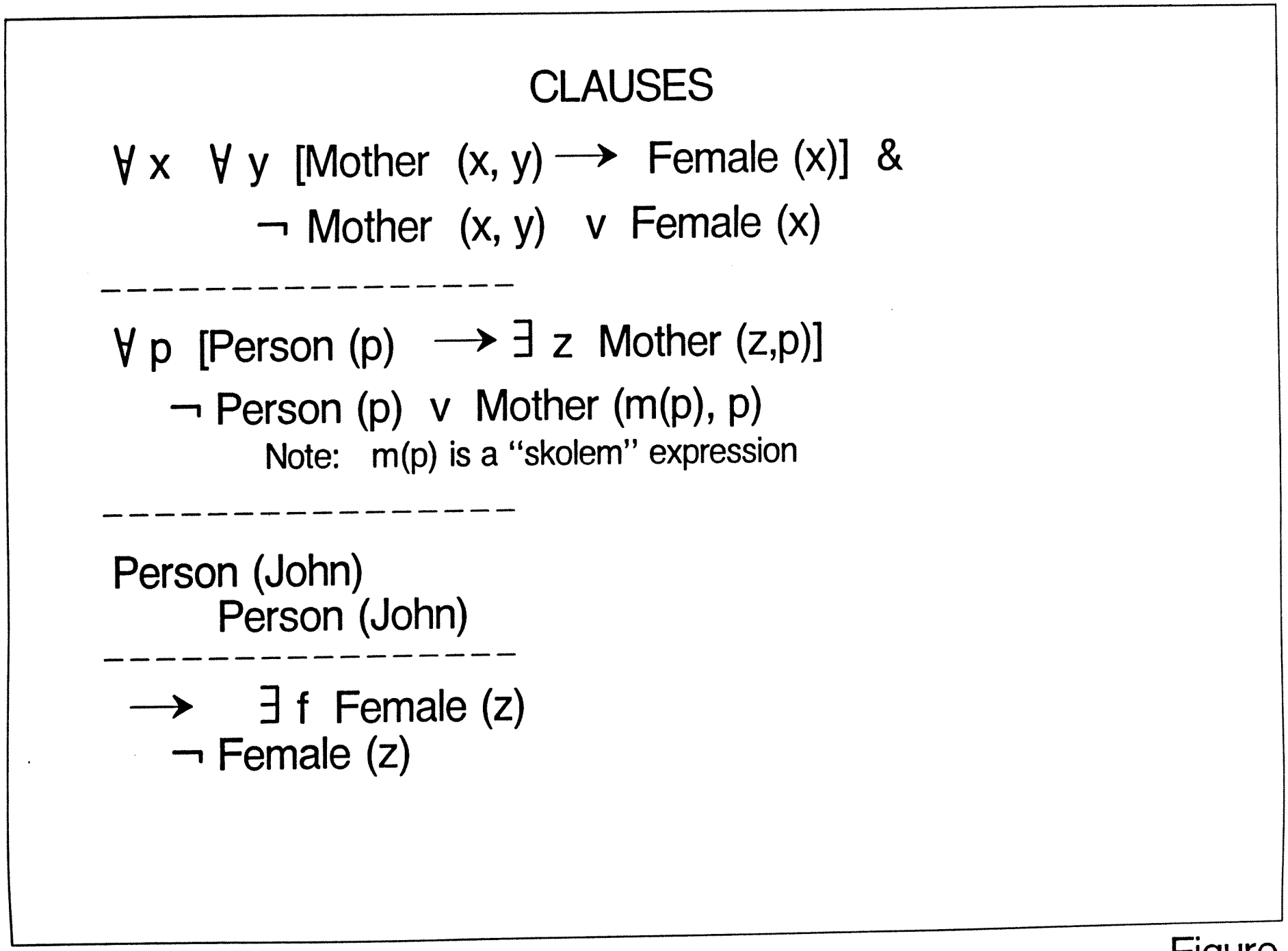

CLAUSES
∀ x ∀ y [Mother (x, y) → Female (x)] &
¬ Mother (x, y) v Female (x)
∀ p [Person (p) → ∃ z Mother (z,p)]
¬ Person (p) v Mother (m(p), p)
Note: m(p) is a "skolem" expression
Person (John)
Person (John)
→ ∃ f Female (z)
¬ Female (z)

Figure 8

PROOF

1. ¬Mother (x, y) v Female (x)
2. ¬Person (p) v Mother (m(p), p)
3. Person (John)
4. ¬Female (z)

5.	Mother (m(John), John)	3, 2, p ← John
6.	Female (m(John))	5, 1, y ← John, x ← m(John)
7.	□	4, 6 z ← m(John)

Figure 9

$$P(y, h(y))$$

yields

$$P(g(x), \quad x \quad)$$

$$P(g(x), h(g(x)))$$

But we cannot finish, because x occurs in $h(g(x))$. If we tried to continue by substituting $[x \leftarrow h\ (g(x))]$ we would get into an infinite loop. We prevent this kind of error by what is called the "occurs check" in the unification algorithm. If we don't use such occurs check, we could prove non-theorems, such as

$$\forall x\ \exists y\ P(y, x) \ \rightarrow\ \exists y\ \forall x\ P(y, x)$$

We will see more on the *occurs check* problem when we discuss *logic programming.*

Resolution is *complete* for first order logic; i.e., any theorem expressed in FOL can be proved by resolution. This is an important result since FOL includes much of mathematics (indeed, can include *all* of mathematics).

However, resolution is not a *decision procedure* for FOL, there is no guarantee that it will detect non-theorems in finite time; in fact FOL has no decision procedure. Higher Order Logic, which we will discuss shortly, has *no* complete proof procedure, let alone a decision procedure.

3.2 Completeness

Completeness is a desirable property of a proof procedure such as resolution; we want to know what it *can* and *cannot* do before we employ it. But completeness alone is not enough. We also need *speed* as well. But Resolution – as well as other proof procedure for FOL – tend to be slow when attempting the discovery of proofs of hard theorems.

We are faced with the classic *combinatorial explosion* problem when we automatically search a *proof tree*, such as the one depicted in Figure 10. The prover searches down along the branches looking for the *goal nodes*, depicted by the asterisks. Finding such a goal finishes the proof.

Figure 10 near here.

Actually, in standard resolution, the search space is not really a tree, since branches often rejoin other branches. Linear formats for organizing resolution search (such as SL-resolution, model elimination, problem reduction) make the search more tree-like. In any case, the "tree" metaphor in the following discussion is useful for intuition.

The number of branches in the tree increases at least exponentially with depth. When the solution nodes lie even moderately deep, brute-force search methods quickly exhaust available resources.

Professional mathematicians have an uncanny way of excluding much of the "brush" of the tree by heading directly toward one of these solution nodes. But the computer - though a million times faster - tends to hopelessly thrash around through all the branches (using depth first or breadth first search methods). The *challenge of this age* for this field is to *shorten* the search time. Attempts to do so can be classified into two categories.

1. Methods that speed up the *inherent reasoning process* by

   (a) Using faster *hardware*, or by

   (b) Clever programming *tricks*, such as *clause compiling*

2. Those that *prune* the search tree.

The effect of the first category is to push down a few layers in the search tree. See Figure 10A. The swath indicates how a *faster* prover might *push farther down* in the tree. This may or may not help, depending, of

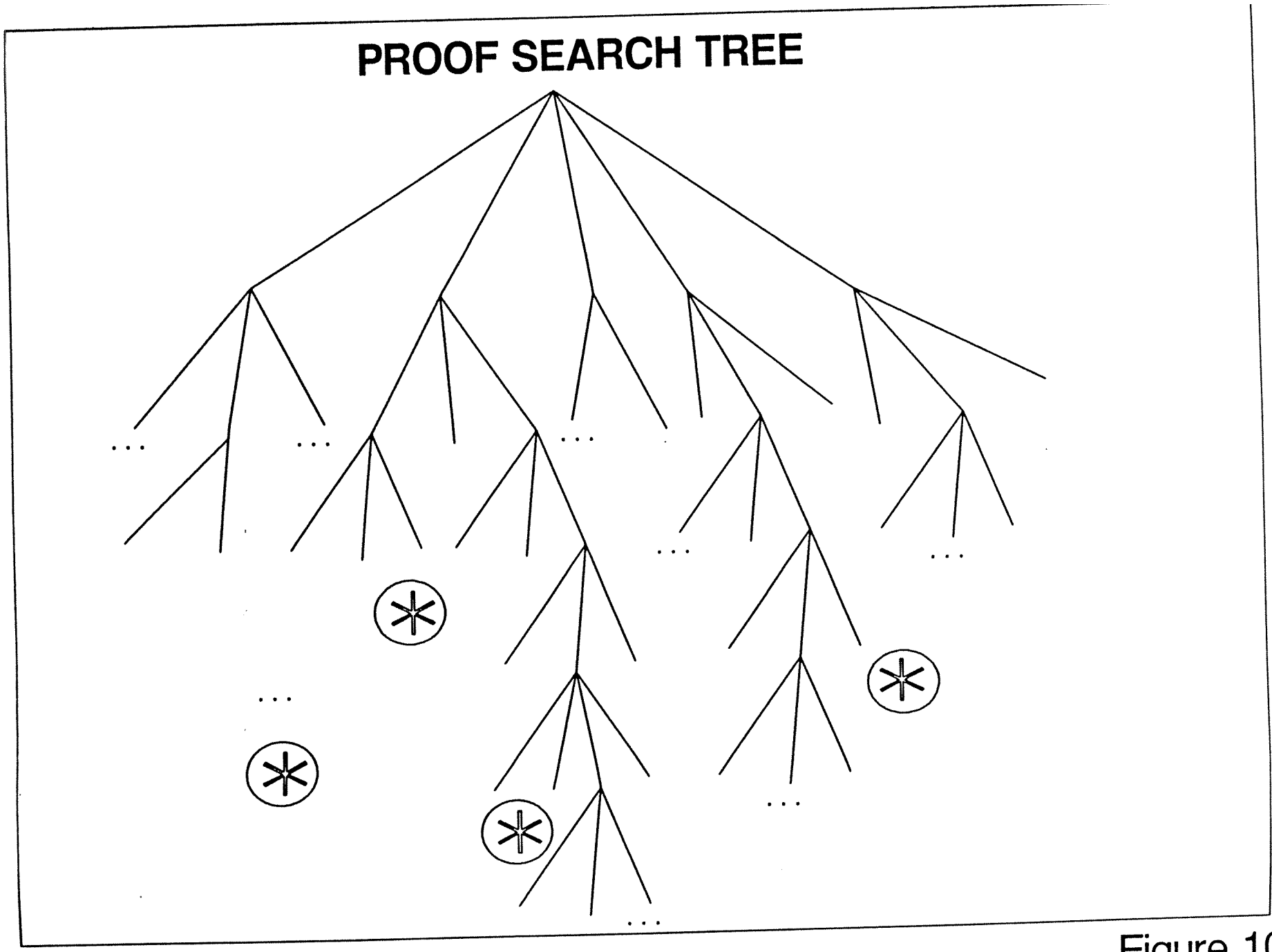


Figure 10

course, on the positions of the goal nodes in the tree. For *many applications* in AI and related fields, it *does* help. A speed up of *one* or *two orders of magnitude*, that seems to be attainable by the new *clause-compiling* techniques coming from the PROLOG community has made possible the proofs of many theorems previously unattainable by automatic methods. This is *good news* for many workers in AI who are beginning to use logic more extensively for *representing rules* for *expert systems* and for entries in *logic data bases*, etc.

Figure 10A near here.

This extended use of logic is placing a greater load on the *"inference engine"* of these systems, and these new *compiling techniques* will help greatly with that load. But it is through the *second category*, the *pruning* strategies, that we can expect satisfactory solutions for the long run. *speed alone* cannot replace the judicial use of *knowledge*. (See our recent paper, *Some Thoughts on Proof Discovery* [Bl86], for a further articulation of this argument.) There were many early attempts to *prune* the search tree. Most of these are *syntactic* in nature, applying equally well from one subfield to another. Some refinements of Resolution to speed up proof discovery are:

- Set-of-Support Resolution
- Hyper-resolution
- SL-resolution (=Model Elimination)
- Connection Method, Matings
- Interconnectivity Graphs
- Locking
- Dozens more.

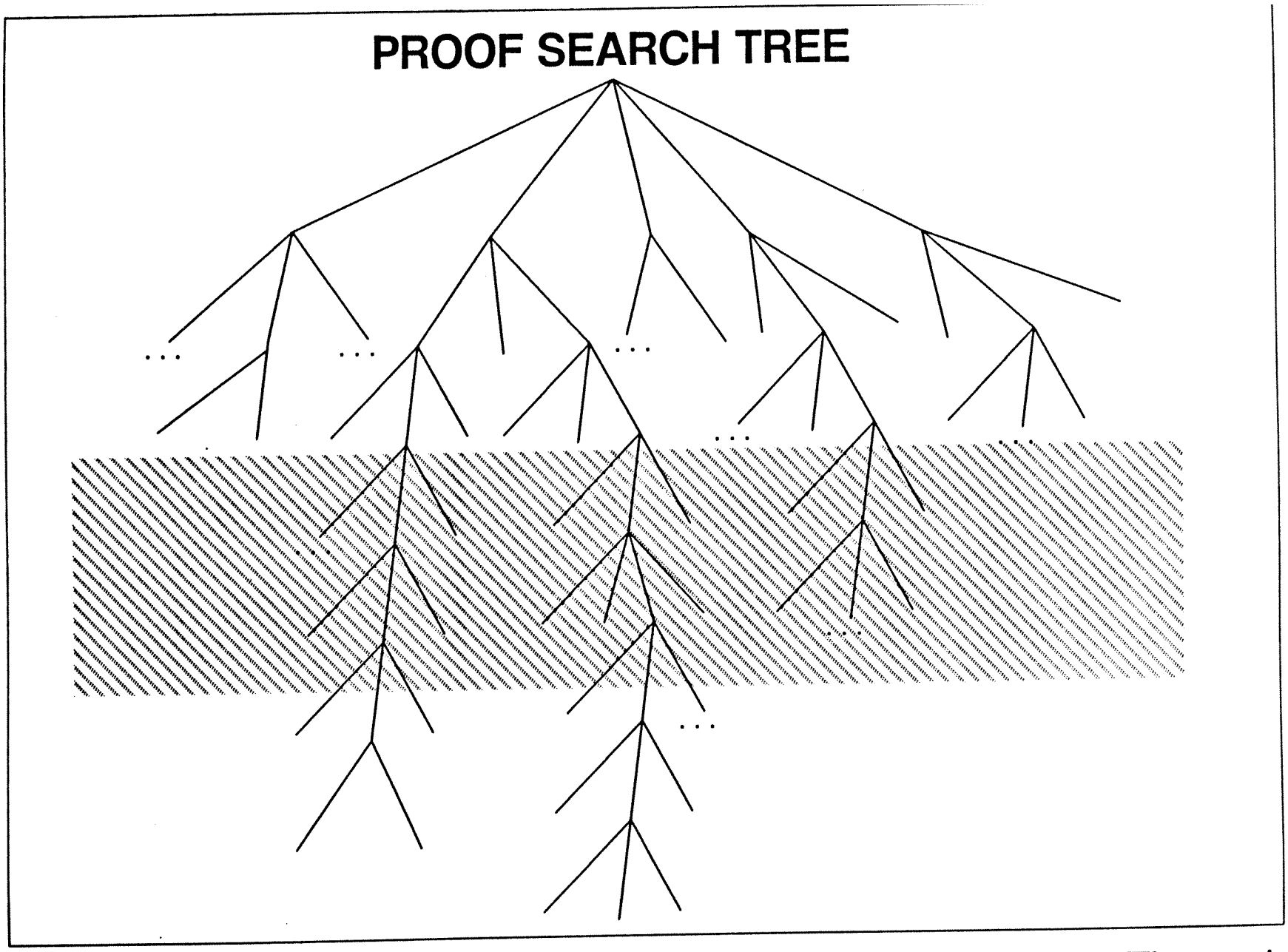


Figure ~~11~~ 10A

One such method, an important one, is the *set of support* strategy [We70], whereby the program works back from the desired goal, and avoids generating unmotivated lemmas that may or may not contribute to the final solution.

Another important one is called *hyperresolution* [Ro65A] wherein a number of resolution steps are combined into one *larger step*, with the program keeping *only* the final resolvent and discarding the intermediate resolvents ("fragments"). (See Section 5 below). This method has been especially powerful in the hands of the *Argonne Group* headed by Larry Wos. Many other pruning strategies have been tried, but these will not be reviewed here [Ko71, Lo68, An84, Bi82, Ko75, Si76, . . .].

It should be noted the *ground* proofs (proofs in which no binding of variables takes place) are hardly ever *difficult.* It is only when we allow the *binding* of *variables* (i.e., the replacement of variables by other terms), through the *unification* process, that we encounter the *combinatorial explosions* that so hamper our provers. There have been developed ground provers which are *enormously fast*, and it is questionable whether further progress in this area is necessary.

We will return to the problem of speeding-up proof discovery shortly, but we first briefly discuss other logics and equality.

3.3 Higher Order Logic

In first order logic we do *not* quantify *function* symbols, *predicate* symbols, or symbols representing *higher* order objects. For example, the formula

$$\forall a\ [\forall x\ P(x)\ \rightarrow\ P(a)] \tag{1}$$

is from the first order logic because only the a and x are quantified. But the formula

$$\forall a\ \exists Q[\forall x\ P(x)\ \rightarrow\ Q(a)] \tag{2}$$

is not, because the predicate symbol Q has been quantified. [2]

Actually (2) is an *easy theorem* for people or machines: we simply replace "Q" by "P", and "x" by "a", but it is part of Higher Order Logic (HOL), which is not even complete, let alone decidable. Inherently, HOL is *harder* than FOL. However, the methods of Unification and Resolution have been extended HOL [Hu73, An84] with a certain amount of success. For example, Andrew's Prover, based on the Huet Unification Algorithm has proved

> **Cantor's Theorem**: If N is the set of integers, and SN is the set of subsets of N, then there is no one-to-one function from N to SN.

More difficult theorems, such as

> **Intermediate Value Theorem**: If f is a continuous function on a non-empty closed interval $[a, b]$, $f(a) < 0$, and $f(b) > 0$, then $f(x) = 0$ for some x in $[a, b]$. (Using the Least Bound Axiom.)

have been proved by special purpose provers such as the one described in [Bl79], but that Prover has limited generality. General Purpose Provers tend to be SLOW, especially for HOL.

3.3.1 Propositions as Types

An interesting approach to HOL has been developed from the so-called Curry-Howard isomorphism. This is an elegant relationship between the typed lambda-calculus and intuitionistic logic. It has been championed, primarily by Martin-Lof [Ma84], as a basis for abstract computer science.

[2] The predicate symbol P is also universally quantifies (implicitly) in (1) and (2), it is only when "existential' type quantifiers are used, where the quantified predicate symbol is to be replaced (bound) in the proof process, that we enter true higher order logic.

Basically, the idea is that if a proposition is viewed as a type and the proof of a proposition is viewed as an object having that type, lambda conversion is formally the same as *modus ponens*. If A and B are propositions (types) and f is a term of type B, the expression

$$(\lambda\ (x : A)\ f)$$

is a function mapping the type A into the type B. The type of this function is symbolized as $A \rightarrow B$, which can be thought of as expressing the implication $A \rightarrow B$, with the meaning that given a proof p of A, we can get a proof $(\lambda(x)\ f)\ (p)$ for B. To prove $A \rightarrow B$ means to demonstrate an object of type $A \rightarrow B$, i.e., an effective procedure for obtaining a proof of B from a proof of A.

This calculus is a sufficient starting point to do mathematics. It is possible to construct definitions of all the usual logical connectives (and, or, not), quantifiers, and equality (using Leibniz' definition of substitutivity of equals). See [CoH85] for an example of how this is done in one system.

The resulting logic is intuitionistic; all objects purported to exist must be constructed, and there is no law of excluded middle. However, if desired, logical connectives and quantifiers obeying the usual non-intuitionistic rules can be constructed from the intuitionistic ones.

A branch of category theory, the theory of Topoi [Top79] leads naturally to the same intuitionistic logic and is a convenient abstract setting for foundational questions in this kind of logic.

Potential advantages of Curry-Howard systems for ATP include: higher order quantifiers are naturally available; we can get a lot of security in the logic from the strong typing; and there is a natural mapping between proofs and programs for constructing objects. So far the only provers using such representations are proof checkers, having very limited search capabilities.

3.4 Other Logics

Many sorted logic brings the idea of typed variables and terms into first-order logic. Walther [Wa83] (see section 6.9) has developed a complete many sorted extension of resolution. Mathematical problems can often be expressed more compactly in many-sorted logic than in standard FOL. There is a significant gain in efficiency of search for proofs, since the types attached to the terms place restrictions on permissible unifications.

An example which has been widely used as an ATP benchmark is "Schubert's Steamroller". (See below.) Fig. 11 shows how many sorted logic can improve the proof length and input sizes for this problem, and also includes data on further improvements which are possible using Cohn's LLAMA logic [Coh87].

Figure 11 near here.

Schubert's Steamroller Problem

> Wolves, foxes, birds, caterpillars, and snails are animals, and there are some of each of them. Also there are some grains, and grains are plants. Every animal either likes to eat all plants or all animals much smaller than itself that like to eat some plants. Caterpillars and snails are much smaller than birds, which are much smaller than foxes, which in turn are much smaller than wolves. Wolves do not like to eat foxes or grains, while birds like to eat caterpillars but not snails. Caterpillars and snails like to eat some plants. Therefore there is an animal that likes to eat a grain eating animal.

For reasoning about the common-sense world, for planning actions, and for communicating with agents (including people), it is necessary to express and reason about ideas like possiblity, belief, knowledge, successiveness

STEAMROLLER PROBLEM STATISTICS

	FOL	Walter's logic	LLAMA
No. of clauses initially	27	12	3
No. of possible inferences	102	12	7
Length of proof	33	10	5

Figure 11

(in time), etc. Modal logics and temporal logics have been developed for this purpose. Proof procedures based on connection methods [Wal86] and semantic tableaux [Smu68] have been developed. See [Tur84] for a review.

Others, particularly Kowalski [Ko79], have argued that modal and temporal logics are unnecessary and that the corresponding reasoning can be formulated and carried out entirely in FOL. The Situation Calculus [McC63, McC69] formulates actions and there effects on states in FOL. Green [Gr69] developed a large working system based on resolution for performing such reasoning.

For recent work in applications of these methods, see [Ko86, Ko86a, Ko86b, Ap82, Moo85]. An excellent textbook covering this area is [GN87].

3.5 Equality

An early problem, a persistent one, is that involving *equality*, the "substitution of equals". For example, the theorem

$$(a = b \ \wedge \ P(a)) \ \rightarrow \ P(b)$$

is rather easy, one simply substitutes a for b, or vice versa (assuming of course that "=" has its traditional meaning). But in more complex examples, like the following theorems,

A group for which $x^2 = e$, is commutative, (Hard)

A ring for which $x^3 = x$, is commutative, (Very Hard)

the proof discovery process is difficult for a computer program, because there are so many ways in which one term can be replaced by another.

The problem arises because, if $a = b$ is hypothesized, then we can replace either a by b, or b by a. This branching factor of 2, when invoked many times, leads to a serious combinatorial explosion. Paramodulation

[Wo70] and E-Resolution [Mo69], provided *complete* solutions to the equality problem, but brought very little to prevent the inherent explosion. Some ATP researchers have greatly tamed the problem by the use of *rewrite rules*. Called *demodulators* by Wos [Wo67] and *reductions* by Bledsoe [Bl71], these procedures rewrite a formula using a set of *reducers* or *rewrite rules*. For example, if we have the rewrite rules

$$x + 0 \rightarrow x$$

$$t \in (A \cap B) \rightarrow t \in A \ \& \ t \in B$$

...

we would rewrite the formula

$$f(t) \in (A(x) \cap B(x+0))$$

as

$$f(t) \in A(x) \ \& \ f(t) \in B(x).$$

The great advantage here is that the substitution on *one-way* only. We replace "$x + 0$" by "x", but do *not* replace "x" by "$x + 0$", as might be possible by paramodulation and E-resolution. Thus a branching factor of 2 is replaced by 1! However, the disadvantage is that such procedures are *incomplete*, some theorems cannot be proved by rewriting alone.

3.5.1 Term Rewriting Systems

An exciting advancement in this area was an attempt to enlarge these sets of rewrite rules to *complete* sets, the so called *complete sets of reductions*. A signal paper in this subarea was [KB70] by Knuth and Bendix that provided a set of ten rewrite rules which constitute a *complete set of reductions* for (non-commutative) group theory. See figure 12. These can be used, by rewriting alone, to prove a variety of theorems in group theory.

Figure 12 near here.

Knuth and Bendix also offered a procedure for *completing* an incomplete set, where that is possible.

This is part of a rapidly growing subfield of ATP called *Term Rewriting* Systems, which includes work on *narrowing* [Sl74] and *unification algorithms with built-in theories* [Fay79].

The first studies concerning the use of complete sets of reductions in resolution was conducted by Dallas Lankford [La75]. It brought together the notion of complete sets of reductions with that of "narrowing" introduced by Slagle [Sl74].

The connection between CSOR's and the study of unification algorithms became closer when independently, Peterson and Stickel [Pe81] and Lankford and Ballantyne [LB77] used the commutative associative unification algorithm [St81] to extend the Knuth-Bendix completion algorithm to handle commutative associative functions. Conversly, Fay [Fay79] used the narrowing algorithm to generate unification algorithms for theories which could be represented by CSORs. Fay's work was extended by Hullot [Hul80]. The study of unification algorithms is now being actively pursued by several research groups, at SRI [Sm87] and Kirchner [Kir86] in particular. See also [CREAS87].

A good survey of the field up to 1980 is found in [HO80]. A more up-to-date survey on completion can be found in [Der87A], and an equally recent survey on the termination of systems of reductions can be found in [Der87b].

COMPLETE SET OF REDUCTIONS

For a Group

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 $-(-x) \rightarrow x$

KB8 $-(x + y) \rightarrow (-y) + (-x)$

KB9 $x + ((-x) + y)) \rightarrow y$

KB10 $(-x) + (x + y) \rightarrow y$

Figure 12

4 LOGIC PROGRAMMING AND CLAUSE-COMPILING

Another giant subarea of ATP is represented by the PROLOG community, or more correctly *Logic Programming*. During the early 1970's Kowalski, Colmereauer, Roussel and others [Ko74, Rou75], discovered that one could use a theorem proving system as a programming language. This is in the spirit of earlier work by Green [Gr69], where an *answer-clause* was used to return the list of bindings of variables, resulting from the proof of a theorem. For example, if one asserts the facts

$$Father(Frank, Mary)$$

$$Mother(Mary, Ted)$$

$$Grandfather(x, z) \leftarrow father(x, y) \ \& \ Mother(y, z),$$

and proves the theorem

$$\exists x \ Grandfather(x, \ Ted),$$

he can obtain the binding

$$x \leftarrow Frank,$$

which gives an answer to the question, "Who is Ted's Grandfather?".

PROLOG is widely used as a programming language, especially in AI, and there are a number of implementations of it. The "standard" version employs ordinary resolution, but

1. allows only horn clauses[3]

2. does not do the "occurs check" during unification.

[3]A clause is horn if it has at most one positive literal. e.g., $\neg P(x) \vee Q(x) \vee \neg R(x, y)$

By restricting use to horn clauses, the implementation can employ a depth-first search, which greatly simplifies the storage allocation problem, and enables high performance via *clause-compiling* (which we will discuss shortly).

There is no apparent difficulty with ignoring the occurs check when PROLOG is used as programming language. But it is unsound as a Theorem Prover, because it would allow the unification of formulas such as

$$P(g(x), x) \text{ and } P(y, h(y)),$$

thereby (as we saw earlier), "proving" non-theorems such as

$$\forall\, x\ \exists y\ P(y, x)\ \rightarrow\ \exists y\ \forall x\ P(y, x)$$

It is also incomplete for FOL, because it employs a depth first search, and is restricted to horn clauses.[4] So why are we interested in PROLOG as a reasoning mechanism, since it is unsound and incomplete? The reason is that during the last few years David Warren (for DEC10 PROLOG) and others have used some compiling techniques (clause-compiling, or rule-compiling) to greatly speed up the process – by orders of magnitude.

Shortly we will (very) briefly describe how clause-compiling is done for PROLOG, and how that is extended to speed up proofs in full first order logic.

Our interest is in Automatic Deduction more than Programming, so we will not report on the enormous literature on Logic-Programming and PROLOG. Those with further interest should consult review papers such as those found in [CT82].

[4]Of course PROLOG, like any other programming language, can be used to implement a sound and complete theorem prover. What is more, Plaisted's SPRF [Pl87] (see Section 6.11) gains much of the speed of PROLOG for ATP.

4.1 Clause Compiling in PROLOG

Clause compiling is like ordinary compiling (of say LISP), in that it involves: structure sharing, clever use of the stack, open coding of unification, and much more. See papers by Warren [War87] and Stickel [St86].

A key to clause-compiling is to have an unchanging set of (original) clauses which will not be enlarged during the proof. So that these can be compiled *once and for all* at the beginning, in a way that makes their use extremely fast. Additionally, there will be one goal literal which continually changes (during the proof search). These original clauses are compiled by anticipating how unification might be accomplished with each of their literals, and constructing a computer program to carry out that unification and other tasks.

This program can be written in some computer language such as C, LISP, or an Abstract Machine Language such as Warren's [War87], and then compiled (ordinary compiling) into machine code. See [War87, St86] for details.

Suppose we have the following input clauses (and others)

1. $(P\ x\ 1) \leftarrow (Q\ x\ z)\ (S\ z)$

2. $(P\ (f z)\ y) \leftarrow (R\ y\ z)$

3. ...

The clause compiler will compile each of the predicates P, Q, S, R, ..., by constructing a LISP[5] Function for each of them, and other supporting functions (not shown here).

Shown here is the function, FUN-P, which has been constructed for the predicate P.

[5]or a C program, etc. We have used LISP here to simplify the presentation.

```
(DE FUN-P  (u v CONTINUATION) (GOAL)
    (PROG (z)
          (COND((UNBOUND-VARIABLE v) (ASSIGN V 1))
                ((NOT (= V 1)) (GO OUT)))
          ( . . .      Allocate, etc . . .)
          ( . . .      Alter CONTINUATION to include the further goal(S z))
          (Q u z CONTINUATION)
    OUT
          (COND((=(FCN-SYM u) 'f) (SETQ Z (ARG1 u)))
                (T (go OUT2)))
           (R v z CONTINUATION).
    OUT2  . . . .  ))
```

Much has been left out, but the main idea is that when a goal literal of the form (P u v) is encountered, to determine whether clause 1. will apply to it (i.e., whether (P x 1) will unify with (P u v)), we can ignore u since x is a variable and hence can be bound to any term; we need only check whether v is 1 or is a variable, and then accomplish the further goal (Q u z).

The *continuation* parameter refers to any additional goals that were carried over from a previous call; we must add to it the subgoal (S z) before proceeding to the goal (Q u z). If (Q u z *continuation*') succeeds, i.e., the goal (Q u z) is accomplished plus the goals of *continuation*', then the proof is finished; if not, then it attempts to apply clause 2. to the goal literal (P u v). This is done at the point OUT in the program.

Similar LISP functions are constructed by the clause-compiler for the other predicates Q, R, S, and any others that appear in the original clause set. All of these LISP functions are then compiled (traditional compiling) to C code or machine code. Of course, as mentioned earlier, the clause-

compiler could avoid LISP altogether. But LISP offers a convenient tool for the clause-compiler and a convenience to us for explaining how this part of clause-compiling works.

4.2 Clause-Compiling for First Order Logic

The phenomenal speeds gotten by clause-compiling in PROLOG were not lost on the rest of the ATP community – they wanted this performance too, but could not use the results from PROLOG unless three major difficulties with it were overcome:

1. the horn clause restriction
2. the depth-first search problem
3. the occurs-check problem

Work on these problems, to bring clause-compiling (and its inherent speeds) to all of first order logic, represents some of the most exciting work in ATP right now. Some systems which extend the PROLOG compiling techniques as follows:

- Stickel's "Prolog Technology Prover" [St86]
- Plaisted's "Simplified Problem Reduction Format" [Pl87]
- Loveland's "Near Prolog" [Lo87]
- Overbeek and Lusk's New Argonne Prover"
- Munich Group's "PROTHEQ" [BAY86]

There are probably a number of others. How do these systems overcome the restriction, 1–3? Let us consider them in order.

The *horn clause* restriction (1) was used in PROLOG to allow a linear search mechanism: once a proof-search is started it can proceed to success or failure without having to backtrack, as is necessary when using ordinary-clauses resolution. This linear format greatly simplifies the search mechanism; one only needs a "stack" and no auxiliary clause storage; only the original clauses are retained, and they can be compiled before the proof search starts.

The way that Stickel [St86] avoids the horn clause restriction for full resolution is to employ a variation of resolution called *model-elimination* (which is essentially *SL-resolution*)[6], which uses chains instead of clauses.

These chains act like clauses, with extra data in them which code the history of how they were constructed in the proof process. This allows a linear format similar to that used in PROLOG, but requires the addition of many contrapositives[7] of input clauses.

Plaisted avoids the horn-clause restriction by using a form of "Case-Splitting", which does not require contrapositives[Pl87].

Loveland uses "multiple-head horn clauses" e.g. $P,\ Q \leftarrow R$, with no contrapositives needed. His technique is similar to Model Elimination but it greatly reduces the amount of extra "history information" recorded with clauses [Lo87].

The *depth-first* search problem (2), is avoided by "iterative deepening", i.e., by repetitively searching to deeper and deeper levels of the search tree. The added cost for recomputing the top parts of the tree is minimal when the search tree is branchy, which is usually the case.

There have been two ways used for avoiding the *occurs check* problem (3):

---

[6]Model Elimination was discovered by Loveland [Lo68, Lo69]; it is equivalent to SL-Resolution, developed independently by Kowalski and Kuener [Ko71].

[7]e.g., for the clause $P \leftarrow Q \wedge R$, we would add the contrapositives $\neg Q \leftarrow (\neg P \wedge R)$ and $\neg R \leftarrow (\neg P \wedge Q)$

(i) by detecting at compile time which literals can possibly have an occurs-check problem e.g., $P(x,\ f(x))$, tagging them, and handling only them during the proof.

(ii) by examining the substitution resulting from any successful unification to determine if there was a problem, and rejecting substitutions with "cyclic" terms, like $x \leftarrow h(g(x))$. (Plaisted, Overbeek and Lusk)

Both methods cause a loss of speed, but not a severe one because such problems rarely occur. (e.g., it is necessary for a variable to occur twice in such a literal for it to present an occurs check problem.)

We believe that clause-compiling will be very important for the future of ATP. These great speeds cannot be ignored. Granted that the ultimate solution is not in speed, but in the better use of knowledge to prune the search tree. Nevertheless, fast reasoning components will be important parts of future technology.

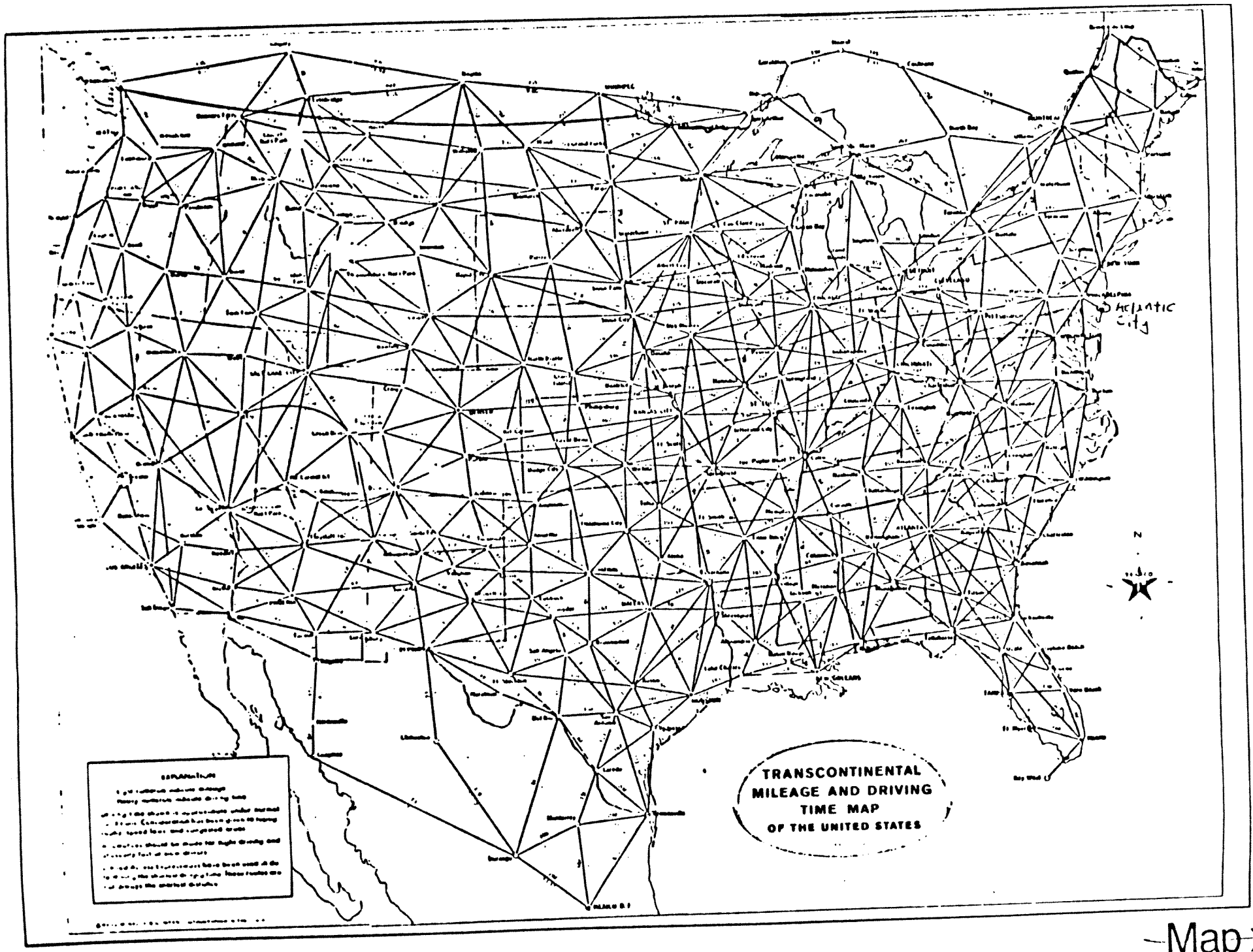
Also, compiling methods of the kind that we have described, are useful for other components of the reasoning process. For example, similar improvements in performance have been obtained for forward chaining [Fo80], rewriting or demodulation [Bo86], inheritance [AN85], and data-base indexing [But86].

5 OVERVIEW OF PROOF DISCOVERY

Now let us give an overview of (our version) of Automated Proof Discovery. How do we classify the research that is being done and should be done?

We feel that building a program for discovering proofs is like designing an autonomous vehicle to cross the USA, say from Atlantic City to Fresno. See Figure 13.

Figure 13 near here.



Map xx

Exhibit 13

To do so one needs:

1. Fast cars;

2. Tactics: For getting from city to city;

3. Strategy: An overall plan of action.

And one needs a map.

But note that speed alone is not enough; dashing off in more less the right direction will not lead to a distant goal without some guidance, no matter how fast the car.

One could liken this to the way that automated proof discovery is being attacked. See Figure 14. Here again we have "fast cars" (fast inference vehicles), tactics and strategy. Let us break this down into more detail.

Figure 14 near here.

Category 1 is easy to define, it consists of those efforts which produce *speed* of inferencing. They are *essential* to the success of ATP. Whatever else we do to prune the tree, it is absolutely necessary that we have great speeds for the "vehicle."

Examples of parallel processing in ATP, are the efforts of Overbeek et al, at Argonne National Lab [REF], the Munich Group [REF], and Waltz and Stanfield at Thinking Machines [REF].

But speed alone is not enough. Again we need overall guidance that comes from tactics and strategy.

It is not so clear what to put in Category 2, tactics, but we feel that those methods which employ "large inference" steps tend to have the "city to city" flavor, as do the special purpose provers. We will discuss these in more detail shortly.

But what do we put in Category 3, strategy? Is there any method being used, that takes an overall,*global* view, that provides and uses an overall

OVERVIEW OF AUTOMATED PROOF DISCOVERY

1. Fast Inference Vehicles:
   - Faster Hardware, Parallel Processing
   - Clause–Compiling (and Compiling Rewrite–rules, etc.)

2. Tactics:
   - Large Inference Steps
   - Semantic Methods
   - Special Purpose Provers

3. Strategy:
   - Analogy, Abstraction, etc.
   - "People" Methods

(and a "MAP": Knowledge Base)

Figure 14

strategy? Probably not. Perhaps *analogy* comes the closest to it; whereby, the (complete) proof of one theorem acts as an *overall guide* to finding the proof of another. *Abstraction* is surely another. All such methods that are used or appear to be helpful, can be classified under the heading of "people methods", methods *routinely* used by practicing mathematicians, but hardly used at all by existing programs. And it is quite clear that there is an *absolute requirement* for a *structural knowledge base* of mathematical knowledge (a "map" if you will), if we are to attain substantial success at this field.

5.1 Tactics

5.1.1 Large Inference Steps

Under tactics, we have listed *large inference steps* (or multi-steps), where the prover tries to accomplish its goal (discover the proof) by a few large steps rather than a whole bunch of small ones.

The *key* here is to *discard* the intermediate results. Many current provers "choke" from retaining unneeded proof fragments, such as intermediate clauses.

Another key point is to identify for each such large step, the *objective* of that step. The prover then sets out to achieve that objective, and if it succeeds it retains only the objective and discards all intermediate results. In fact it discards the intermediate results even if it *fails* to achieve the desired objective. Thus it keeps only a few powerful results for further use. These results act as a kind of *subsummers* to those discarded.

Some examples of systems using large inference steps are:

- Hyper-Resolution (J. A. Robinson)
- Linked-UR-Resolution (Wos, et al)
- Terminator (Antoniou & Ohlbach, Kaiserslautern, Germany)

- Variable Elimination (Bledsoe & Hines)
- Hyper-chaining (Hines)
- Theory Resolution (Stickel)
- Complete Sets of Resolutions [KB70, etc., See Section 3]

Hyper-Resolution [Ro65A]

As mentioned earlier, *hyper-resolution* has been extensively used for a number of years. Figure 15 gives an example of its use, showing also the *objective*, and the discarded intermediate clause. The example shown is from propositional logic, but the method works equally well for full FOL, using unification.

Figure 15 near here.

Linked-UR-Resolution [Wo84]

Linked-UR-Resolution is somewhat like hyper resolution. The idea is depicted in Figure 15A, where a *nucleus* is given which contains a *goal literal.* The *objective* is to obtain a *unit clause* by a set of resolutions, which eliminates all literals except (possibly) one, the goal literal. A variation allows the goal literal to occur in one of the satellite clauses. Also an *initiating* satellite (a unit clause) might be used to start the process. The goal literal can also be required to satisfy a given predicate P. "This allows the use of semantic criteria for guiding the proof discovery."

Figure 15A near here.

Terminator [AO83]

The *objective* is to try for a *unit* proof of $\square$, at various points in the proof.

Variable Elimination [BH80]

HYPER–RESOLUTION

Example

Nucleus Clause:

$-A \quad -B \quad -C \quad E$

Satelite Clauses:

$A \quad F$
$B \quad G$
$C \quad H$

Hyper–Resolvent:

$F \quad G \quad H \quad E$

Three Resolution steps in one.

Discard intermediate Resolvents

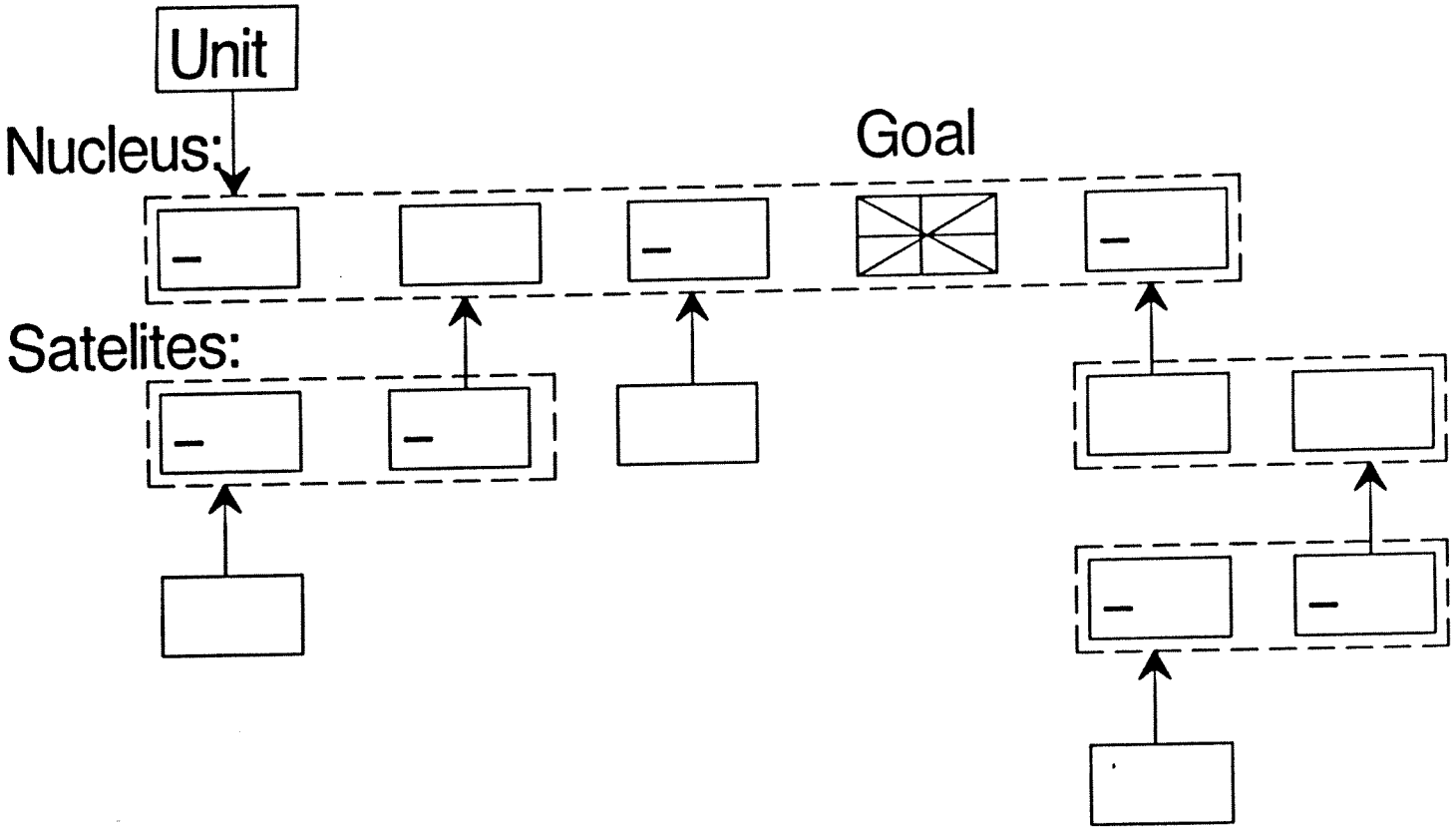
$F \quad -B \quad -C \quad E, \quad F \quad G \quad -C \quad E$

OBJECTIVE: Remove all negative literals from a clause

Figure 15

LINKED UR-RESOLUTION [Wo7]

Initiating Satelite:



OBJECTIVE: A unit clause

Allows the use of Semantic Criteria for guiding proof discovery.
It is related to other Connection Methods.

Figure 15a

This procedure is designed for the field of real analysis, where the inequality predicates $\leq$ and $<$ are used.

Figure 16 shows an example where the variable x is eliminated from the target clause 1. to obtain the VE-Resolvent 2. The *objective* is to remove an eligible[8] variable from a target clause.

Figure 16 near here.

In this example, the one large step is equivalent to six resolution steps. The method implicitly uses the axioms of real inequality theory, including those for transitivity and interpolation.

This method has greatly helped with proofs in intermediate analysis. For example, the proof of *lim+*, a limit theorem for sums,

$$\lim_{x \to a} f(x) = \ell \;\&\; \lim_{x \to a} g(x) = k \rightarrow \lim_{x \to a} [f(x) + g(x)] = \ell + k$$

took only 13 steps instead of an estimated 100,000 or more by resolution.

Hyper-Chaining [Hi87]

Hyper-Chaining is an extension of variable elimination, wherein the variable x being removed does *not* need to be *eligible* in the target clause. The hyper-chain rule works to *make* the variable eligible (using other hyper-rules) and then eliminates it.

Figure 17 shows hyper-chaining on a simple example. A much harder example, the limit of a sum theorem, lt, shown above, is proved in three steps. See Figure 18. The *objective* is to remove the variable δ from the target clause 10, which is done in one step to obtain Clause 11. This large step also utilizes Clauses 2, 3, 5, 6, 8, 9, and is equivalent to at least 22 resolution steps. Two more uses of Hyper-chaining yields $\square$. Figure 19 shows a few of the intermediate steps which were discarded.

[8] A variable clause x is eligible in a clause C if it does not occur within the scope of any uninterrupted function of predicate symbol.

VARIABLE ELIMINATION

Target Clause:

1. $a < x \quad x < b \quad Q$

   x is a variable not occurring in Q

VE-Resolvent:

2. $a < b \quad Q$ 1, VE x

Six Resolution steps in one.

Implicitly uses the axioms of Real Inequality theory, including:

Transivity: $x < y \wedge y \leq z \longrightarrow x < z$, etc.

Interpolation: $u < v \longrightarrow \exists\, w\ (u < w < v)$

OBJECTIVE: Remove a variable from a clause (if eligible)

Figure 16

Figure 17, 18 and 19 near here.

Theory Resolution [St85]

Stickel's *Theory Resolution* encompasses many of the ideas from the other large inference steps methods discussed above. It incorporates a *theory* (or theories) into a Resolution Theorem Prover, thereby making it unnecessary to resolve directly upon the axioms of that theory. Two or more clauses are resolved *with respect* to that theory. Intermediate results are discarded.

Figure 20 shows two simple examples from taxonomic theory and inequality theory. See [ST85] for other examples, especially for useful applications in AI. Figure 21 lists some of the other work that resembles theory resolution.

Figure 20 and 21 near here.

Complete Sets of Reductions [KB70, etc]

See Section 3. The *objective* is to reduce a target formula (e.g., clause) as far as possible by applying to it a (complete) set of rewrite rules.

5.1.2 Semantic Methods

One of the most characteristic methods employed by people is to use semantics to guide proof; a mathematician knows what his symbols mean (for example, he knows that x is a real number when doing analysis). He also knows many examples of predicatively defined structures. (such as groups, continuous function, etc.) He uses this knowledge in at least two ways: 1) by extending known examples (closely related to analogy; see section 5.2.1 below); and 2) by not attempting to prove intermediate goals for which he has a counterexample.

Method 2, checking for reasonableness seems to be extremely powerful—it probably accounts for a major portion of the mental effort used by human

HYPER–CHAINING
Example

Target Clause:

1. $a < x \quad x < b \quad f(x) \leq c$

   x is a variable, not occuring in a, b, or c

Supporting Clauses:

2. $d \leq f(y)$

Hyper–Chain Resolvent:

3. $a < b \quad d \leq c \qquad\qquad 1, 2 \qquad\qquad x$

OBJECTIVE: Remove a variable from a clause.

USES: Variable Elimination, Chaining, . . .

Figure 17

Proving Sum-of-Limits Theorem

1.	$0 < \delta'_{\epsilon'}$	$\epsilon' \leq 0$		
2.	$\delta'_{\epsilon'} + x' < x_0$	$\delta'_{\epsilon'} + x_0 < x'$	$(f\ x_0) \leq (f\ x') + \epsilon'$	$\epsilon' \leq 0$
3.	$\delta'_{\epsilon'} + x' < x_0$	$\delta'_{\epsilon'} + x_0 < x'$	$(f\ x') \leq (f\ x_0) + \epsilon'$	$\epsilon' \leq 0$
4.	$0 < \delta''_{\epsilon''}$	$\epsilon'' \leq 0$		
5.	$\delta''_{\epsilon''} + x'' < x_0$	$\delta''_{\epsilon''} + x_0 < x''$	$(g\ x'') \leq (g\ x_0) + \epsilon''$	$\epsilon'' \leq 0$
6.	$\delta''_{\epsilon''} + x'' < x_0$	$\delta''_{\epsilon''} + x_0 < x''$	$(g\ x_0) \leq (g\ x'') + \epsilon''$	$\epsilon'' \leq 0$
7.	$0 < \epsilon_0$			
8.	$x_0 \leq x_\delta + \delta$	$\delta \leq 0$		
9.	$x_\delta \leq x_0 + \delta$	$\delta \leq 0$		
10.	$\epsilon_0 + (f\ x_0) + (g\ x_0) < (f\ x_\delta) + (g\ x_\delta)$ $(f\ x_\delta) + (g\ x_\delta) + \epsilon_0 < (f\ x_0) + (g\ x_0)$ $\delta \leq 0$			

Target Clause.
δ is the Target variable.

(Hyper-Chain 10 [δ]: 3, 2, 6, 5, 9, 9, 8, 8)

11.	$\epsilon_0 < \epsilon' + \epsilon''$	$\delta'_{\epsilon'} \leq 0$	$\epsilon' \leq 0$	$\delta''_{\epsilon''} \leq 0$	$\epsilon'' \leq 0$

(Hyper-Chain 11 [ϵ']: 1)

12.	$\epsilon_0 < \epsilon''$	$\delta''_{\epsilon''} \leq 0$	$\epsilon'' \leq 0$

(Hyper-Chain 12 [ϵ'']: 4)

13. □.

Figure 18

(Chaining ...)
$(g\,x_0) + \epsilon_0 < (g\,x_\delta) + \epsilon'$ $(f\,x_\delta) + (g\,x_\delta) + \epsilon_0 < (f\,x_0) + (g\,x_0)$ ...
$\delta'_{\epsilon''} + x_\delta < x_0$ $\delta'_{\epsilon''} + x_0 < x_\delta$ $\epsilon' \leq 0$

(Chaining ... 2)
$(g\,x_0) + \epsilon_0 < (g\,x_\delta) + \epsilon'$ $(g\,x_\delta) + \epsilon_0 < (g\,x_0) + \epsilon'$ $\delta \leq 0$
$\delta'_{\epsilon''} + x_\delta < x_0$ $\delta'_{\epsilon''} + x_0 < x_\delta$ $\epsilon' \leq 0$

(Chaining ... 6)
$(g\,x_0) + \epsilon_0 < (g\,x_\delta) + \epsilon'$ $\epsilon_0 < \epsilon' + \epsilon''$ $\delta \leq 0$
$\delta'_{\epsilon''} + x_\delta < x_0$ $\delta'_{\epsilon''} + x_0 < x_\delta$ $\epsilon' \leq 0$ $\delta''_{\epsilon''} + x_\delta < x_0$ $\delta''_{\epsilon''} + x_0 < x_\delta$ $\epsilon'' \leq 0$

(Chaining ... 5)
$\epsilon_0 < \epsilon' + \epsilon''$ $\delta \leq 0$
$\delta'_{\epsilon''} + x_\delta < x_0$ $\delta'_{\epsilon''} + x_0 < x_\delta$ $\epsilon' \leq 0$ $\delta''_{\epsilon''} + x_\delta < x_0$ $\delta''_{\epsilon''} + x_0 < x_\delta$ $\epsilon'' \leq 0$

discarded

(Chaining ... 9)
$\epsilon_0 < \epsilon' + \epsilon''$ $\delta \leq 0$
$\delta'_{\epsilon''} + x_\delta < x_0$ $\delta'_{\epsilon''} < \delta$ $\epsilon' \leq 0$ $\delta''_{\epsilon''} + x_\delta < x_0$ $\delta''_{\epsilon''} + x_0 < x_\delta$ $\epsilon'' \leq 0$

(Chaining ... 9)
$\epsilon_0 < \epsilon' + \epsilon''$ $\delta \leq 0$
$\delta'_{\epsilon''} + x_\delta < x_0$ $\delta'_{\epsilon''} < \delta$ $\epsilon' \leq 0$ $\delta''_{\epsilon''} + x_\delta < x_0$ $\delta''_{\epsilon''} < \delta$ $\epsilon'' \leq 0$

(Chaining ... 8)
$\epsilon_0 < \epsilon' + \epsilon''$ $\delta \leq 0$
$\delta'_{\epsilon''} < \delta$ $\epsilon' \leq 0$ $\delta''_{\epsilon''} + x_\delta < x_0$ $\delta''_{\epsilon''} < \delta$ $\epsilon'' \leq 0$

(Chaining ... 8) [before variable elimination of δ]
$\epsilon_0 < \epsilon' + \epsilon''$ $\delta \leq 0$
$\delta'_{\epsilon''} < \delta$ $\epsilon' \leq 0$ $\delta''_{\epsilon''} < \delta$ $\epsilon'' \leq 0$

11. $\epsilon_0 < \epsilon' + \epsilon''$ $\delta'_{\epsilon''} \leq 0$ $\epsilon' \leq 0$ $\delta''_{\epsilon''} \leq 0$ $\epsilon'' \leq 0$

Figure 19

THEORY RESOLUTION EXAMPLES

1. Taxonomic Theory:

   1. $Boy(x) \longrightarrow Person(x)$
   6. $NoDaughter(x) \ \& \ Child(x,y) \longrightarrow Boy(y)$

   Resolve: 11. Child(Chris, sk2) with
   10. NoDaughter(Chris) to get

   ---

   13. Boy(sk2) in one step.

2. Inequality Theory:

   1. $\neg(x < x)$
   2. $x < y \ \& \ y < z \longrightarrow x < z$
   ....

   Resolve: 6. $a < b$, 7. $b < c$, & 8. $\neg(a < c)$
   to get 9. $\square$

Figure 20

OTHER WORK RESEMBLING THEORY RESOLUTION

Hyperresolution (J. A. Robinson) [Ro65A]

Z-resolution (Dixon) [Dix73]

U-generalized resolution (Harrison and Rubin) [HR]

E-resolution (J. Morris) [Mo69]

Linked inference Principle (Wos, et al) [Wo84]

General Inequality Prover (Bledsoe and Hines) [BH80]

- Variable Elimination
- Shielding Term Removal
- Attached ground Prover

Figure 21

mathematicians. Several researchers have attempted to apply this principle with varying success (Gelernter [Ge59], Bledsoe [BB82], Bl83], Wang [Wan85 and Section 6.3 below]).

It appears to be quite challenging both to represent and to access the large variety of examples the human has available.

5.1.3 Special Purpose Provers

We list here areas for which a few special purpose provers have been developed, and which are classified under "tactics."

- Inequalities
  - Ground [NO78?, Sho77, Sho79, Bu83, ?AAAI paper?]
  - General [Bh80, BKS85, Hod72]
- Geometry (Wu and Chou) (See Section 6)
- Non-standard Analysis (Ballantyne) [BB88]
- Algebraic Manipulation (Macsyma, etc.)
- Equality Subsystems (Richard ?)

5.2 Strategy

5.2.1 Analogy

Analogy is the heart and soul of intelligent behavior. We do very little that is absolutely new. Somehow intelligent machines (including reasoners) must make use of analogy, but success with it has been limited, so far. It is closely related to the field of *Machine Learning* [ML1, ML2].

There have been a number of AI researchers working on Analogy, including Winston, Carbonell, Greiner, Russell, and others. I will not review

all of that literature here. Much of it is reviewed by Dedre Gentner's survey paper in this conference. (There are also a number of other papers in this conference on analogy). Another review, with an extensive set of references, is given by Hall [Ha85].

There are many aspects to *analogy*, but we are concerned here only with the situation where the *solution of one problem* is used as *guide* to the solution of another, or the proof of one theorem the guide to the proof of another.

A signal paper of this sort, is that of Bob Kling [Kli71], wherein he used the proof of a theorem in Group Theory to guide the search for an analogous proof in Ring Theory.

Figure 22 depicts this idea. The *guiding* proof proposes actions to the prover. If the proposed action fails, then the prover must somehow recover, to get the process back on track. Also a *fetching* mechanism is needed to automatically select, from a database, proofs that might be used as a guide to the current endeavor.

Figure 22 near here.

As an example of this, three University of Texas graduate students working at MCC have developed an analogy prover based on Resolution and Chaining [BCP86], which has used the proof of $lim+$ as a guide for proving $lim*$. See Figure 23. Since the proof of $lim*$ makes some major detours from that of $lim+$, it was necessary to rely on its "expert system" component for recovery from failed actions, and to also rely on its stand-alone proving capability. See [BCP86] for details. This same prover handled other pairs of theorems, including those depicted in Figure 24, and has been extended and converted to a *natural deduction format* [BCP87], which we feel will be better able to handle more complex proofs, especially where *parts of proofs* are needed as guides.

Figure 23 and 24 near here.

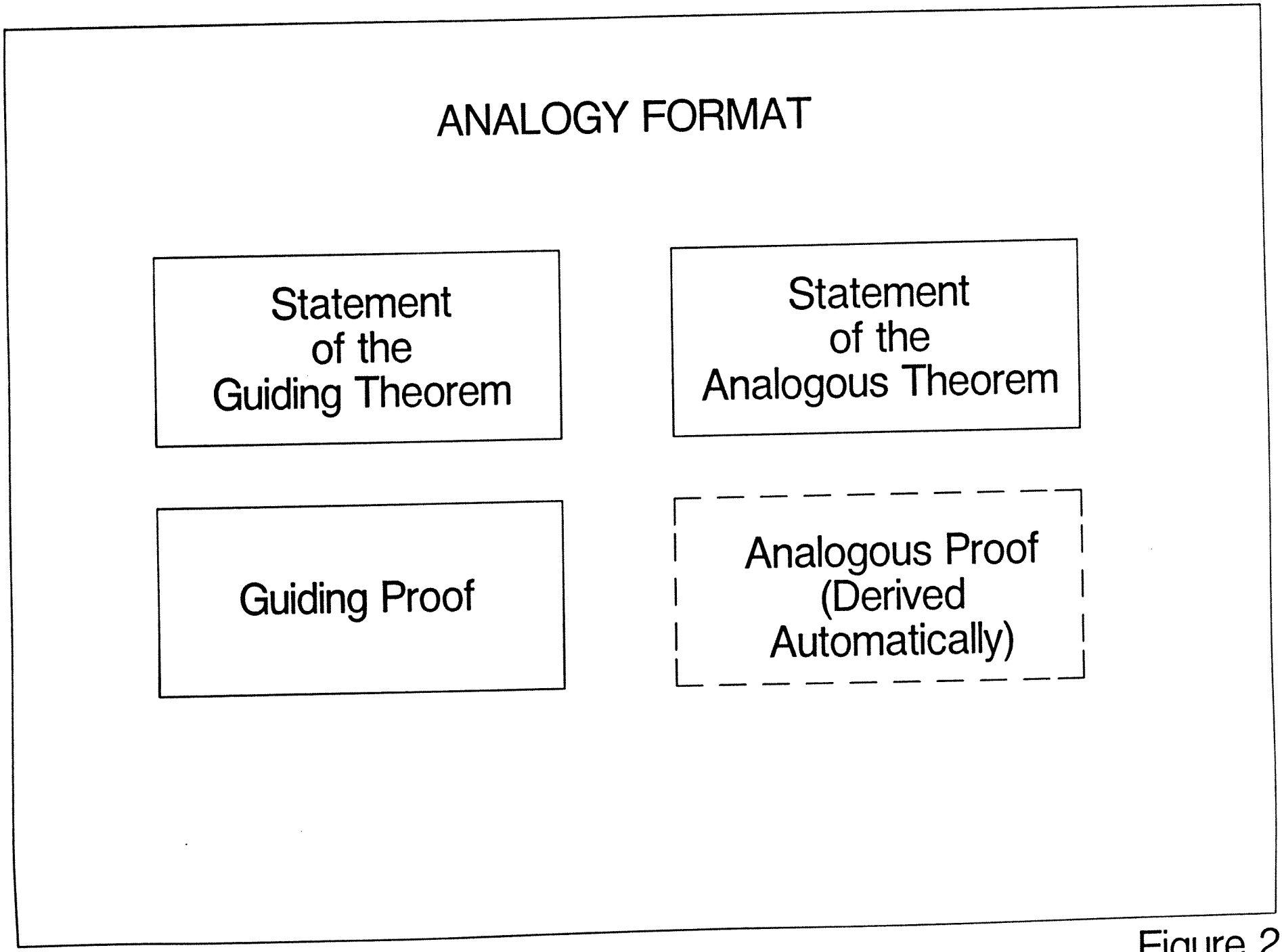

ANALOGY FORMAT
Statement of the Guiding Theorem
Statement of the Analogous Theorem
Guiding Proof
Analogous Proof (Derived Automatically)

Figure 22

AN EXAMPLE

LIM +

$$\lim_{x \to a} f(x) = l \ \wedge \lim_{x \to a} g(x) = k \longrightarrow \lim_{x \to a} [f(x) + g(x)] = l + k$$

LIM*

$$\lim_{x \to a} f(x) = l \ \wedge \lim_{x \to a} g(x) = k \longrightarrow \lim_{x \to a} [f(x) \cdot g(x)] = l \cdot k$$

Figure 23

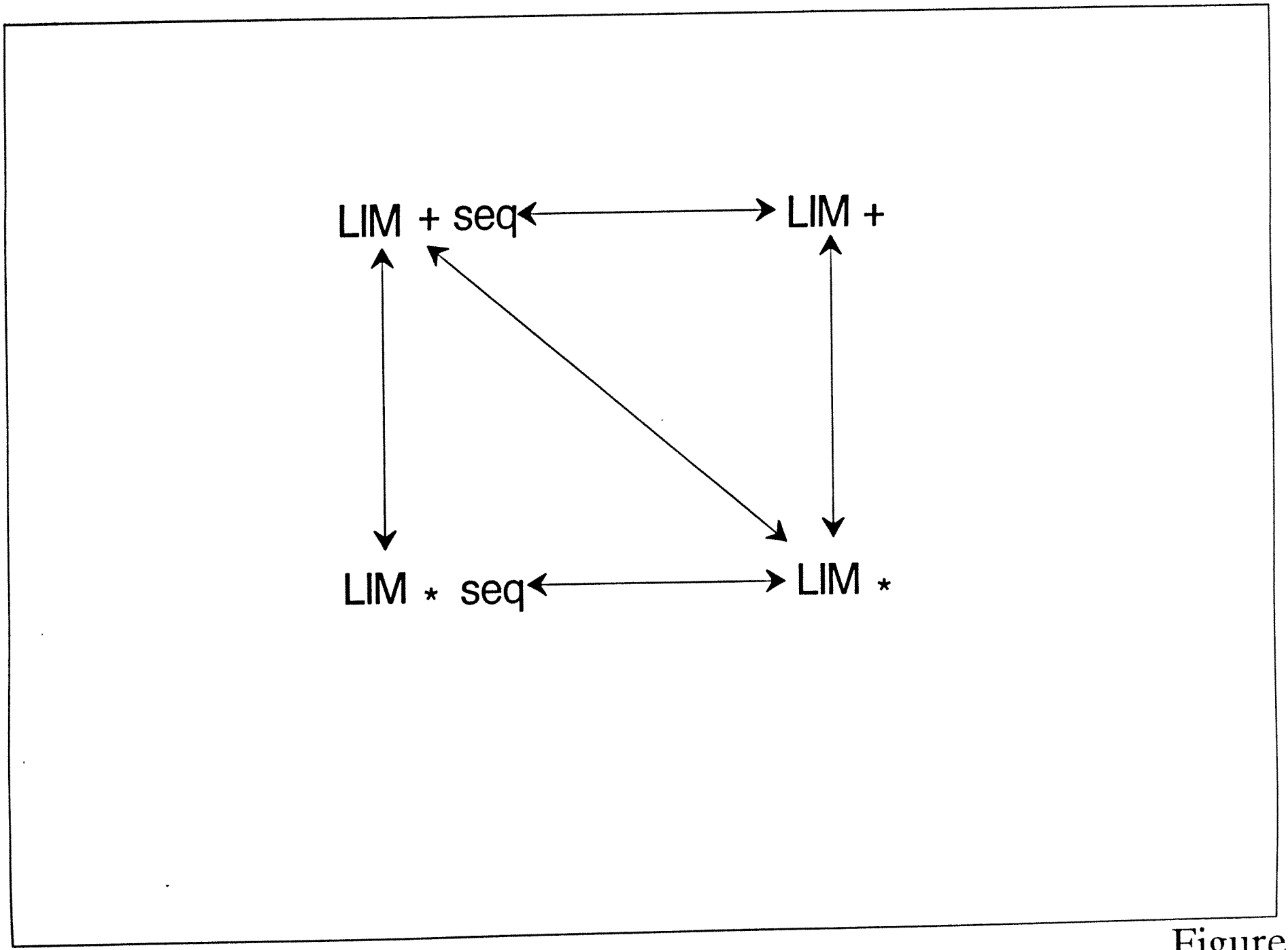


Figure 2

As was pointed out by Carbonell [Car83], the *derivational history* of a problem solution is very important when that solution is used as a guide to solving an analogous problem. The reason for this is that when an analogous action fails, the problem solver needs to "know" what was the *intended goal* of the action, so that it can try to attain that goal by another action (through analogy, or by stand-alone methods). Such a derivations history provides for *annotating* a proof, with *motivational* information.

Another reason for the natural deduction format, is that subgoals of the proof can be treated in a hierarchical way. Thus, in Figure 25, suppose the hierarchical structure represents the *proposed* proof of a new theorem (as proposed by a guiding proof). Now if, for example, goal G23 fails, then the prover can execute the following strategy:

1. Fetch another guiding proof and try to apply it to G23.
2. If step 1 fails, try to prove G23 by a stand-alone prover.
3. If step 2 fails, fail the goal, backtrack and try steps 1-2 on goal G2.

Figure 25 near here.

Such a hierarchical structure helps make use of the derivational history (annotated proof) that is needed. (Other useful information could also be included in the derivational history.)

A problem with this is that one must collect and store this additional information (i.e., not just proofs, but annotated proofs) if it is to be used to guide new proof searches.

Some possible mechanisms for these annotated proofs are:

- Expansion Trees (Andrews, Miller) (Section 1)
- Proof Parsers (Simon) (Section 6.9.2)

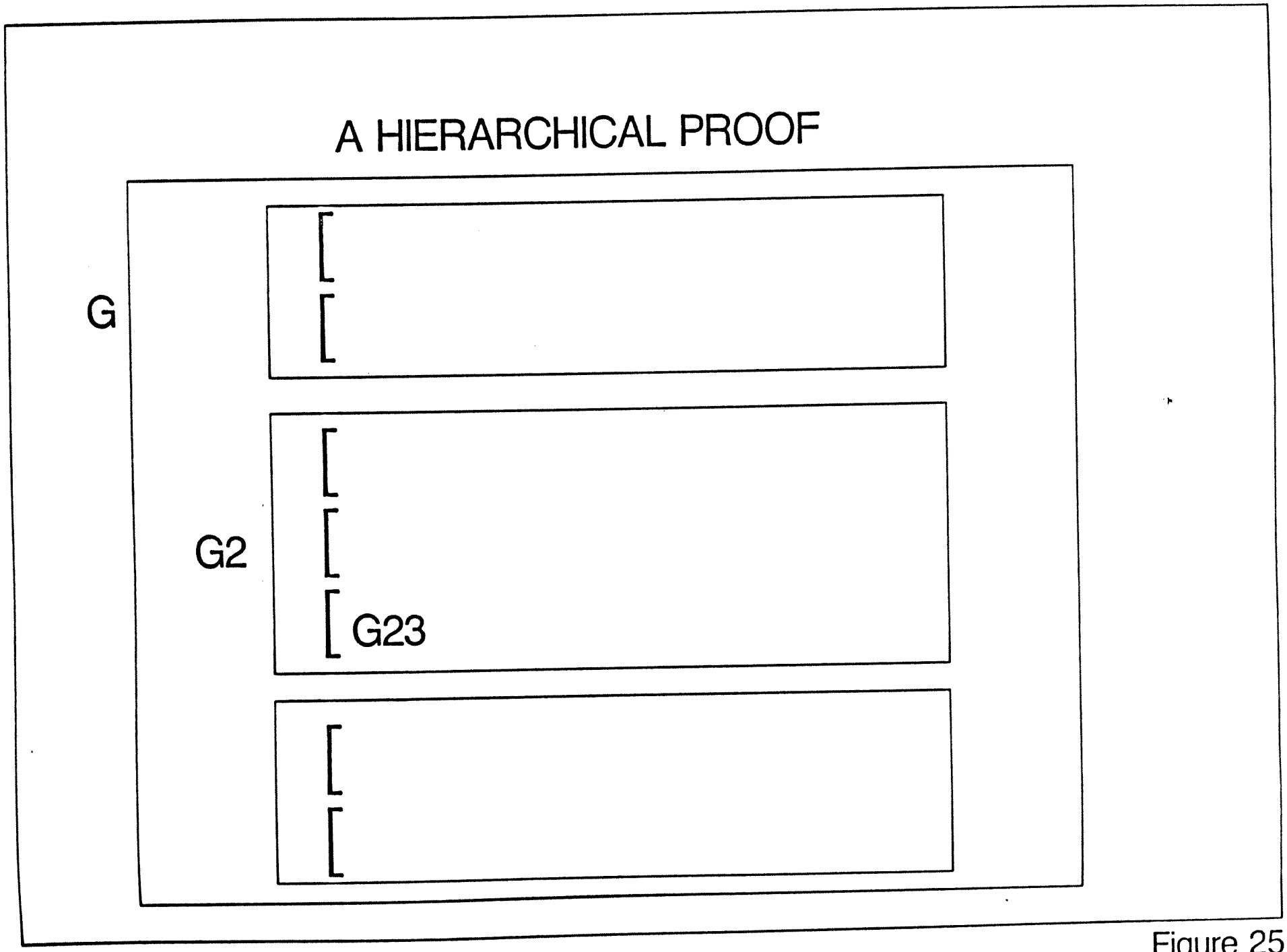


Figure 25

- Requirement Graphs (Bledsoe) [Bl86]
- Multi-Step Rules (Hines) (Section 5.1.1)
- Other formal representations (Section 1)

We believe that in the long term a large structured knowledge base will be needed, such as CYC, the commonsense knowledge base being built by Doug Lenat and his team at MCC [Le86, LF87]. See also [Hob85, Bor?]. Indeed, analogy plays a central role in the building and use of CYC.

5.2.2 Abstraction

The idea here is to prove an abstraction of a theorem, as a subgoal, and use that proof as a guide for proving the theorem itself. For example, one could abstract a formula $P(x, y)$ by suppressing the second argument and retaining only $P(x)$.

Such an idea was first introduced by Newell, Simon, and Show [NSS56]. But the best work in this area is by Plaisted [Pl81, Pl82], wherein he suggests and uses a number of kinds of abstraction, and uses a number of layers of abstraction.

5.2.3 Other "People" Methods

We list here some other methods in addition to analogy and abstraction, that are extensively used by professional mathematicians, with some reference to machine implementation:

- Generating and using Examples in proof discovery [BB82, Bl83]
- Using Counter-examples to prune search trees [Ge59, BB82] (See Section 5.1.2)
- Automatic Conjecturing of lemmas and subgoals [Le76, Le82]

- Automatic Fetching of useful lemmas and definitions from a large Knowledge Base
- Agenda mechanisms for controlling the proof search [Ty81]
- Higher-level reasoning, meta-reasoning [Ges83], higher-order logic [An84]

6 CONTEMPORY PROVERS, CENTERS, PEOPLE

We describe here the work of a few groups and individuals conducting ATP research. Some of the efforts of others are described in other parts of this survey. This list is by no means complete, nor is it ordered by importance. For example, much of the work in Expert Systems is not included as well as the work in PROLOG and common-sense reasoning. See also [Pas87].

6.1 Argonne Laboratory Theorem Provers, L. Wos, E. Lusk, R. Overbeek, et al. [Wo84, Wo87]

Argonne is one of the most prolific center for ATP research in the world. They have implemented a series of systems including AURA [Wo81] and ITP [Lu84]. Currently, [But86] they are implementing a new system aimed largely at getting an increase of speed ($>$ 100 times) compared to ITP. This system will use implementation techniques from Prolog (e.g. clause compiling), multiprocessors, associative-commutative unification, and database indexing techniques (for clause retrieval). McCune also has implemented an interactive resolution proof checker. With Boyer, this system was used to prove some basic mathematical theorems from Godel's axiomatization of set theory[Bo86a].

The Argonne group has used ITP extensively in ATP research, proving many theorems, verifying software and hardware, solving word problems

using ATP methods [AAR newsletter often reports examples of this work], and solving open questions in mathematics. They have distributed ITP to over 200 sites (it is written in Pascal for portability).

The basic technique is clausal resolution with set-of-support, paramodulation, demodulation, and subsumption (all optional). Elaborate data structures are used to permit full structure-sharing for terms and literals (only one copy of each unique object is kept). Indexing techniques allow efficient access to terms which might unify with a given term. A complex evaluation function is used for prioritizing the next resolution step. A "user friendly" interface is provided for interactive or batch use.

6.2 KLAUS Automated Deduction System (originally called CG5): Mark Stickel (SRI) [St85, St86, St86a]

This large system implements a number of techniques of ATP. The basis is a connection graph encoding possible resolution steps between non-clausal first-order formulae. Special techniques include:

1. Control of inference direction (a formula may be restricted to forward or backward chaining);

2. Theory resolution [St85] which increases efficiency by allowing a single resolution step to incorporate a whole "theory" such as rewriting (demodulation), associative-commutative unification, many-sorted unification, taxonomic hierarchies, etc. (See Section 5.11);

3. A Knuth-Bendix algorithm is provided for completion of sets of rewrite rules;

4. a Priority control mechanism employing evaluation function;

5. A Prolog Technology Theorem Prover (PTTP) component. Using Loveland's Model Elimination style of Prolog-like linear search, PTTP compiles each clause into LISP functions which carry out the search corresponding to that clause. Iterative deepening is the search strategy. Occurs check is used except in cases where it can be determined that it is necessary. (See Section 4.2)

Stickel has proved a good collection of standard ATP test theorems and theorems from mathematicians.

6.3 Kaiserslautern: N. Eisenger, H. J. Ohlbach, J. Siekmann, Universitat Kaiserslautern

The Margraf Karl Refutation Prover (MKRP) [Karl84] is a powerful system developed over many years at Kaiserslautern and Karlsruhe. It uses connection graphs, due originally to Kowalski [Ko75]. Each possible inference step (resolution, paramodulation, factoring) in the clause set is represented as a link in a graph. After performing a chain of inference steps, it is often possible to "reduce" the graph, removing irrelevant and redundant links [Ohl87]. This is the source of efficiency of the algorithm, but it is also the source of a problem: there is no completeness theorem for connection-graph resolution with inference restriction strategies typically used (In practice, this does not seem to be a problem).

Unification in MKRP is many-sorted [Wa83] (see section 3.4). Further research on unification theory promises to add the capabilities for handling equational theories and structured sorts.

An important technique in MKRP is the "terminator module" [AO83] which quickly detects situations where the refutation of a set of clauses can be completed immediately.

Extensive input and output translation facilities are provided.

The Kaiserslautern group is currently working on a successor system

called HADES (Highly Automated Deduction System). Among other features, it attempts to incorporate higher level links as atomic inference steps in the connection graph.

They aim to encode and prove all theorems in a standard textbook on semigroups and automata.

6.4 Munich: W. Bibel[9], S. Bayer, et al.

The Munich group has implemented as a project within ESPRIT, a PROLOG-like theorem prover called PROTHEO based on Bibel's connection method [Bi83]. Special hardware including associative memory for accessing connections and highly parallel multiprocessing is under development.

Available input preprocessing includes translation to clausal form. Lemmas are generated and retained. Depth bound search is used. The system is complete for first order logic.

Special reductions of the clause set [Bi87] are used for efficiency; for example, Schubert's steamroller is proved in 7 steps.

6.5 University of North Carolina: David Plaisted

Plaisted's Simplified Problem Reduction Format prover (SPRF) [Pl82, Pl87] is written in PROLOG and obtains efficiency by encoding first-order formulae as PROLOG clauses. A special splitting rule is used for non-Horn clauses for completeness. Contrapositives of the input clauses are not required, but help in some cases. Rewrite rules can also be given and Knuth-Bendix completion is available.

The search strategy is depth-limited with iterative deepening. Solutions to subgoals are cached.

The code is noteworthy for its conciseness, about 15 pages of PROLOG. Speed is competitive with major resolution based provers such as ITP,

[9]now at Univ. British Columbia

Stickel, etc.

6.5.1 Greenbaum

The Illinois Prover was written by S. Greenbaum [Grb86, GP86] as a test-bed for Plaisted's abstraction methods [Pl81]. It became a general purpose prover of considerable power, employing many interesting implementation techniques.

A special refinement of locking resolution and unit preference is used which simulates backwards and forward chaining. Complex data structures are used for structure sharing and indexing speed.

The aim in uniformly good performance with minimal user guidance. Schubert's Steamroller is obtained in about 1 minute on a VAX.

6.6 Edinburgh: A. J. Milner, M. J. Gordan, et al.

Logic for Computable Functions (LCF) [GMW82] is a large system for verifying properties of computable functions defined in typed lambda calculus. It is efficiently implemented in ML [Card82].

LCF has been used to verify thousands of standard mathematical theorems. It has recently been enhanced by Larry Paulson to include higher-order deduction [Pau86].

6.7 Boyer-Moore Prover: University of Texas [BM79]

This is a large system for verifying properties of recursive functions defined by lambda expressions in "pure LISP". Structural Induction on the size of the input is used, with many heuristics available.

The prover has been implemented in several dialects of LISP and is widely distributed, referenced and used by others. Applications, some of commercial importance, have included program verification [BM81], hardware verification [Hun86, Borr87], verification of compilers, and verification

of the proofs of many theorems in mathematics and metamathematics including the uniqueness of prime factorization for natural numbers, Wilson's Theorem [Ru85], The Church-Rosser theorem for pure lambda calculus, and Goedel's Incompleteness Theorem [Sh86, Sh87].

One of the commendable features of this Prover is its ability to automatically carry out the proof of a theorem when given the necessary lemmas by the user. Another is its ability to automatically construct a *generalized* induction hypothesis when the obvious one does not suffice.

Boyer has also done important work on compiling rewrite rules [Bo86].

6.8 The Wu-Chou Geometry Provers

An interesting Proof Procedure for Theorems in Geometry has been given by the Chinese mathematician, Wen-Tsun Wu [Wu78, 84]. Shang-Ching Chou (University of Texas) has extended and refined that work and used his implementation to prove a number of difficult theorems in Plane Geometry (about 2000 Theorems), some of which are new. [Cho85, Cho86, Cho87, CS86]

The procedure is as follows:

> Transform the Hypotheses and Conclusion of a theorem in Geometry to sets of Algebraic equations. Show that the conclusion follow from the hypotheses by performing a series of "divisions" (somewhat like Matrix operations). This requires factoring of polynomials over algebraic extensions of fields of rational functions (very difficult in some cases).

The method does not apply to all parts of Plane Geometry (only cases where hypotheses and conclusions can be expressed as equalities, not inequalities). The general method can be applied to other areas, such as Differential Geometry. Figure 26 shows drawings from two examples from [Cho87], the first of which was given in Section 1 of this survey.

Figure 26 near here.

6.9 Bledsoe, et al (University of Texas & MCC)

Figure 27 shows some provers from this group. See also [Bl84, Bl86].

Figure 27 near here.

6.9.1 Wang's SHD (Semantically-guided Hierarchical Prover) [WaT85, WaT87]

An interesting aspect of SHD is the hierarchical format. This is similar to SL-resolution, recording extra information along with each clause to record the history of subgoals which led to the clause. Wang has implemented a number of completeness-preserving refinements (restrictions on resolution) allowed by this annotation. For example, redundant subgoals can be avoided, certain forms of subsumption can be checked quickly, etc.

A number of heuristic methods for assigning priority to subgoals are available, and a user interface allows control of parameters affecting these heuristics.

Another goal of Wang's prover was to provide a base for semantic guidance to the proof process. A partial model of the axioms of the input theorem may be provided by the user. The user specifies a finite set of (ground) terms from the Herbrand universe and provides effective procedures for evaluating predicates built on these terms. Candidate subgoals are only attempted if they are acceptable in the model.

Several difficult theorems have been proved, such as IMV (a first-order form of the intermediate value theorem).

EXAMPLES USING THE WU-CHOU PROVER

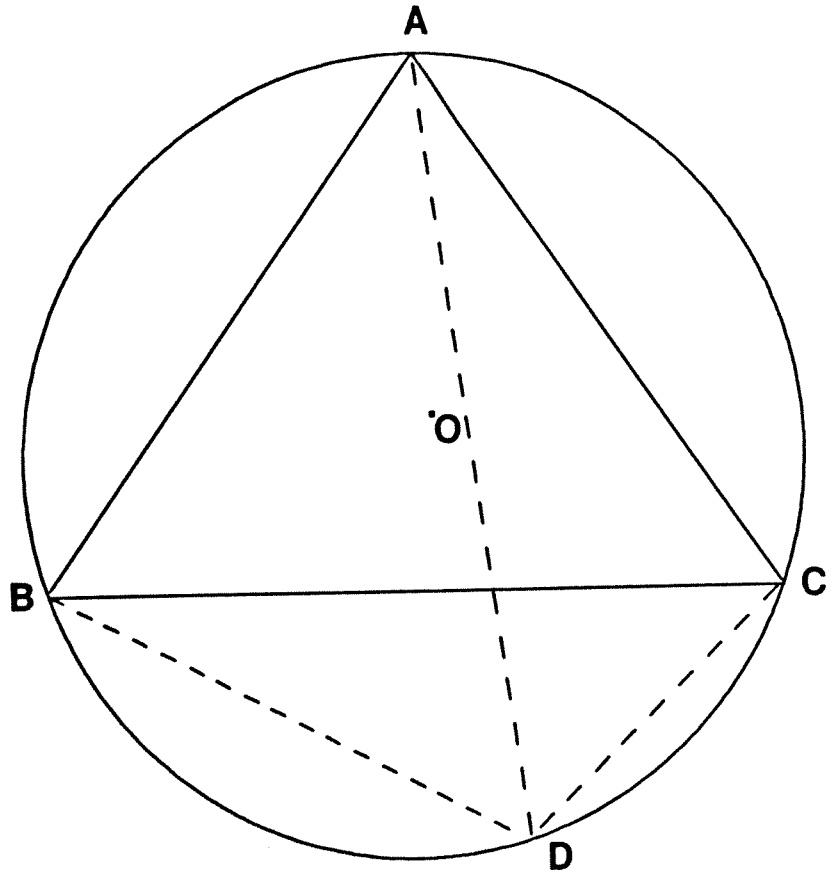


Figure A30–29

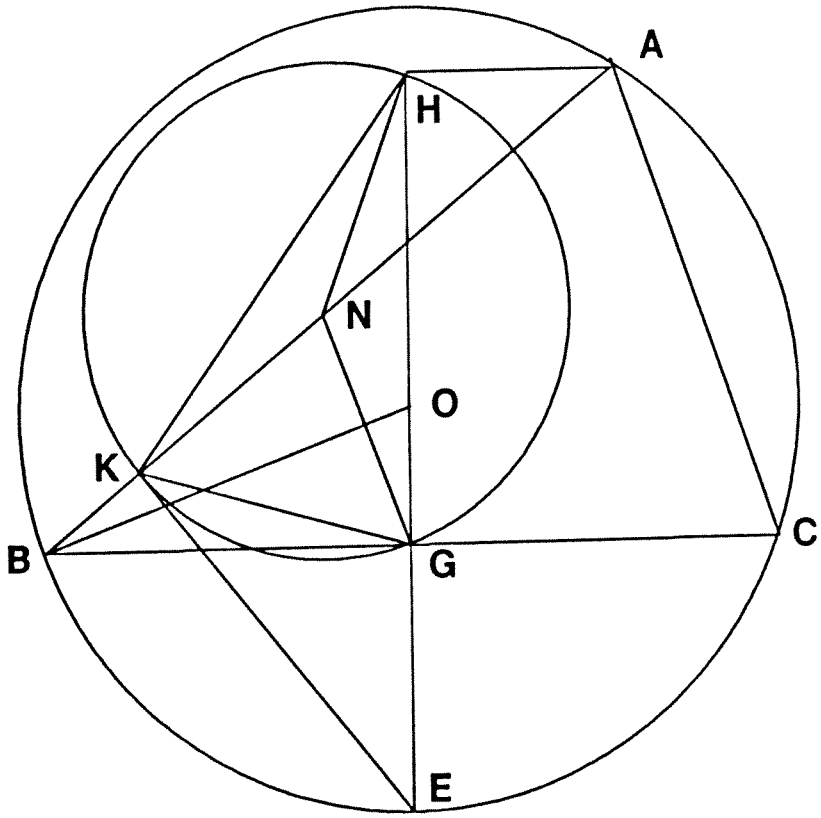


Figure A32–49

Figure 26

BLEDSOE, et al (UTexas and MCC)

IMPLY – Natural Deduction Style Prover [Bl75]
– Regular and Interactive Versions

General Inequality Prover [Bl84]
– Proofs in Analysis

Wang's Hierarchical Prover [WaT87]

Building–in Multistep Axiom Rules – Larry Hines

Gazing – Plummer (U. Edinburgh)

Proof Checking in Number Theory – Don Simon

Analogy Prover – Brock, Cooper, and Pierce

Figure 27

6.9.2 Proof Checking Number Theory: Don Simon

This system accepts a proof in its Natural Language form (Figure 28) exactly as it is written by the mathematician.[10]. The proof is then parsed: first the sentences are parsed, then the whole proof (See Figure 29, 30), using a proof grammar. This enables the deduction component to verify the statements in the proof. A powerful reducer for number theory [Sim84] is used.

Figure 28, 29 and 30 near here.

All proofs in Chapter's 1 and 2 of LeVeque's book were proof checked [Sim88].

6.9.3 Building-In Multistep Axiom Rules: Hines [Hi86, Hi87]

This system compiles multistep actions into a single rule, thereby attaining higher-level objectives. Interim results are discarded.

Examples of these are the VE rule and Hyper-Chaining rules described in 5.1 above. Each rules has restricted entry points, and other restrictions on their use. Most rules will not apply, but when one does, it can give sizable results. They are somewhat like expert systems rules in that respect.

The rules are built up in a hierarchical way, some rules are subparts of others.

6.9.4 GAZING: Dave Plummer [Plu87]

His system, VOYER, is a natural-deduction style prover, which uses the concept of *gazing* to control the use of rewrite rules. Abstractions of rules

[10]The system is currently working on proofs from LeVeque's book on Numbered Theory [LeV62]

ELEMENTARY THEORY OF NUMBERS – W. J. Leveque

THEOREM 1–1. If a is positive and b is arbitrary, there is exactly one pair of integers q, r such that the conditions

$$b = qa + r, \quad 0 \leq r < a, \tag{6}$$

hold.

Proof: First, we show that (6) has at least one solution.

OMITTED

To show the uniqueness of q and r, assume that q' and r' also are integers such that

$$b = q'a + r', \quad 0 \leq r' < a.$$

Then if $q' < q$, we have

$$b - q'a = r' \geq b - (q-1)a = r + a \geq a,$$

and this contradicts the inequality $r' < a$. Hence $q' \geq q$.

Similarly, we show that $q \geq q'$. Therefore $q = q'$, and consequently $r = r'$. ▲

Figure 28

```
|-TO
|-SHOW
|-UNIQUENESS
|-IMPLICITLY-SUPPOSE (B = Q*A+R & 0 <= R & R < A)
|-SUPPOSE (B = Q1*A+R1 & 0 <= R1 & R1 < A)
| |-ASSUME
| |-THAT
| |-(FORMULA (B = Q1*A+R1 & 0 <= R1 & R1 < A))
| |-BREAK
|-PROVE (Q = Q1 & R = R1)
  |-PROVE Q = Q1
  | |-PROVE (Q1 >= Q & Q >= Q1)
| | |-PROVE Q1 >= Q
  | | | |-SUPPOSE Q1 < Q
  | | | | |-THEN
  | | | | |-IF
  | | | | |-(FORMULA (Q1 < Q))
  | | | |-CONTRADICTION
  | | | | |-PROVE R1 >= A
  | | | | | |-WE
  | | | | | |-HAVE
  | | | | | |-(FORMULA (B-Q1*A = R1 & R1 >= B-(Q-1)*A
  | | | | | |    & B-(Q-1)*A = R+A & R+A >= A))
  | | | | | |-DEDUCE B-Q1*A = R1
  | | | | | |-DEDUCE B-Q1*A >= B-(Q-1)*A
```

Figure 29

```
| | | | | |-DEDUCE R+A >= A
| | | | |-AND
| | | | |-THIS
| | | | |-CONTRADICTS
| | | | |-(FORMULA (R1 < A))
| | | | |-DEDUCE R1 < A
| | | | |-DEDUCE R1 >= A <=> NOT(R1 < A)
| | | | |-BREAK
| | | |-HENCE
| | | |-(FORMULA (Q1 >= Q))
| | | |-DEDUCE Q1 < Q <=> NOT(Q1 >= Q)
| | |-BREAK
| | |-PROVE Q >= Q1
| |  |-SIMILARLY
| |  |-WE
| |  |-SHOW
| |  |-(FORMULA (Q >= Q1))
| |-BREAK
| |-THEREFORE
| |-(FORMULA (Q = Q1))
| |-DEDUCE (Q1 >= Q & Q >= Q1) => Q = Q1
|-AND
|-CONSEQUENTLY
|-(FORMULA (R = R1))
|-BREAK
|-DEDUCE Q = Q1 => R = R1
```

Figure 30

are used, stored in a concept hierarchy graph, to facilitate the proper acquisition and use. (Plummer, a visitor at the University of Texas, is finishing his PhD Thesis under Bundy at Edinburgh).

7 CONCLUDING REMARKS

Logic is emerging as a foundation for AI and all of Computer Science. The consequence of this is that some form of *automatic reasoning* is a *requirement* for most AI programs. Much of the research in ATP over the last thirty years is applicable to this need.

As these programs grow more complex, the corresponding inference problems will become more difficult, comparable in difficulty to the proof substantial theorems in mathematics.

We have reviewed the current research on automated reasoning and given a proposed classification of that work.

We note that some research areas, such as *clause-compiling* and *parallel processing*, are very exciting, and this is rightly so. But we wonder whether these efforts on *fast implementation*, which are very important in their own right, might divert us from the even more important areas (in the long run) of *tactics* and *strategy*.

Under tactics, we are especially hopeful about the work on *larger-inference-steps*, and the work on special purpose systems such as those for the use of rewrite rules.

We believe that more *large scale experiments* are needed, wherein researchers exercise their provers on worthwhile examples, rather than play with toy problems and/or a couple of harder problems (such as the Steamroller problem or the Intermediate Value Theorem).

What about Strategy? Are we to soon attain "over all" strategies for our provers? There has been some promising work on Analogy and Machine Learning; a little on Conjecturing, Abstractions, and using Examples to

guide proof discovery, but not much else.

We feel that fundamental progress will require advances in representing and accessing the knowledge used by human mathematicians. This knowledge includes examples, rules, heuristics, and motivations, in addition to the more commonly recognized declarative facts represented by axioms and lemmas. The experiments we have reported on demonstrate simplified approaches to representing one or more forms of mathematical knowledge, but the realization of an integrated truly powerful system remains for the future.

References

References

[AN85] Ait-Kaci, H. and R. Nasr, LOGIN: A Logic Programming Language with Built-in Inheritance. Technical Report MCC-AI-068-85. Microelectronics and Computer Technology Corp., Austin, TX., July 1985. (To appear in Journal of Logic Programming.)

[An86] Andrews, P. B., "An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof", Academic Press, NY, 1986.

[An80] Andrews, P.B., Transforming Matings into Natural Deduction Proofs, in 5th Conference on Automated Deduction, Les Arcs, France, (Bibel and Kowalski, Eds), Lecture Notes in Computer Science 87, Springer Verlaga, 1980, 281-292.

[An81] Andrews, P.B., Theorem Proving via general matings, JACM 28 (1981) 193-214.

[An84] Andrews, P.B., et al, Automating Higher Order Logic, In:[BL84] 169-192.

[AO83] Antoniou, G. and H. J. Ohlback, TERMINATOR; Proc of the 8th Int'l Joint Conf. on AI, Karlsruhe, West Germany, August 1983, pp 916-919.

[Ap82] Appelt, D.E., Planning Natural-Language Utterances to Satisfy Multiple Goals, SRI International Tech Note 259, 1982.

[Bay86] Bayerl, S., E. Eder, F. Kurfess, R. Letz, J. Schumann, An implementation of a PROLOG-like theorem prover based

on the Connection method; In: AIMSA'86 (P.Jorrand, ed.) North Holland, Amsterdam.

[BB77] Ballantyne M. and W. W. Bledsoe, Automatic Proofs of Theorems in Analysis Using Non-Standard Techniques, JACM 24 (1977) 353-374.

[BB82] Ballantyne, M. and W. W. Bledsoe, On Generating and Using Examples in Proof Discovery, Machine Intelligence 10 (Harwood, Chichester, 1982), pp. 3-39.

[BBr73] Bledsoe, W.W. and P. Bruell, A Man-machine Theorem Proving System, Proc 3rd IJCAI, Stanford U., 1973; also in AI Jour 5 (1974) 51-72.

[BCP86] Brock B., S. Cooper, W. Pierce, Some Experiments with Analogy in Proof Discovery, MCC Technical Report AI-347-86, October 1986.

[BCP87] Brock B., S. Cooper, W. Pierce, Analogical Reasoning and Proof Discovery, submitted to CADE-9.

[BH80] Bledsoe, W.W. and L. Hines, Variable Elimination and Chaining in a Resolution-Based Prover for Inequalities, Proc 5th Conf on Automated Deduction, Les Arcs, France, (Bibel and Kowalski, Eds), Springer Verlag, 1980, 281-292.

[BHe85] Bledsoe, W.W., and L. Henschen, What is Automated Theorem Proving? In: Journal Automated Reasoning 1 (1985) 23-28.

[Bi82,87] Bibel, W., Automated Theorem Proving. Vieweg Verlag, Braunschweig, 1982. 2nd Edition, 1987.

[Bi87] Bibel, W., R. Letz, J. Schumann, Bottom-up enhancements of deductive systems; In: AI and Robotics (I. Plander, ed), North Holland (to appear).

[Biu86] Biundo, S., et al, THE KARLSRUHE INDUCTION THEOREM PROVING SYSTEM, Proc. CADE-8, Oxford (1986).

[BKS85] Bledsoe, W.W., K. Kunen, and R. Shostak, Completeness Proofs for Inequality Provers, Artificial Intelligence 27 (1985) 225-288.

[BL84] Bledsoe, W.W. and D. Loveland (Eds), Automated Theorem Proving: After 25 Years, American Math Society, Contemporary Mathematics Series, 19, 1984.

[Bl71] Bledsoe, W.W., Splitting and Reduction Heuristics in Automatic Theorem Proving, Artificial Intelligence 2 (1971) 55-77.

[Bl75] Bledsoe, W.W. and Mabry Tyson, The UT Interactive Prover. Memos ATP17A & ATP17B, Math Dep, Univ Texas, 1975, 1983.

[Bl77] Non-Resolution Theorem Proving, Artificial Intelligence 9 (1977) 55-77. In Reading in Artificial Intelligence (Webber, Nilsson, Eds), Tioga, Palo Alto, 1981, pp 91-108.

[Bl79] Bledsoe, W.W., A Maximal Method for Set Variables in Automatic Theorem Proving, Machine Intelligence 9 (1979) 53-100.

[Bl83] Bledsoe, W.W., Using Examples to Generate Instantiations of Set Variables, Proc IJCAI-83, Karlsruhe, Ger., Aug 1983.

[Bl84] Bledsoe, W.W., Some Automatic Proofs in Analysis, In [BL84], 89-118.

[Bl86] Bledsoe, W.W. Some Thoughts on Proof Discovery. Proc. 1986 Sym. on Logic Programming, Salt Lake City, Ut., 1986, pp 2-10. MCC Tech Report AI-208-86, June 1986.

[Bl86A] Bledsoe, W.W., The use of Analogy in Proof Discovery. MCC Technical Report AI-158-86.

[Bla81] Blasjus, K., N. Eisinger, J. Siekmann, G. Smolka, A. Herold, and C. Walther, The Markgraf Karl Refutation Proceure, Proc. 7th Int'l Joint ;Conf. on AI, Vancouver, B.C., Canada, August 1981, pp 511-518.

[BM79] Boyer, R.S. and J S. Moore, A COMPUTATIONAL LOGIC. Academic Press, New York, 1979.

[BM81] Boyer, R.S. and J S. Moore, A verification condition generator for FORTRAN. In: The Correctness Problem in Computer Science (Boyer and Moore, Eds.), Academic Press, London, 1981.

[BM82] Boyer, R.S. and J S. Moore, Proof Checking the RSA Public Key Encryption Algorithm, ICSCA-CMP-37, U. of Texas, Sept 1982.

[Bo71] Boyer, R.S., Locking: a Restriction on Resolution. PhD dissertation, University of Texas, Austin, 1971.

[Bo86] Boyer, R.S., Rewrite Rule Compilation, MCC Technical Report AI-194-86, June 1986.

[Bo86a] Boyer, R. S., et al, Set Theory for First Order Logic: Clauses for Goedel's Axioms, JAR 2 (1986) p. 287.

[Bor ?] (WWB get reference) Boring

[BorrS7] *From HDA Description to Guaranteed Correct Circuit Designs*, D. Borrione (Ed.), North Holland, IFIP, 1987.

[BuS3] Bundy, Alan, The Computer Modelling of Mathematical Reasoning, Academic Press, 1983.

[BuchS3] Buchberger, B., G. E. Collins, R. Loos, "Computer Algebra, Symbolic and algebraic Manipulation," Springer Verlag 1983.

[CarS3] Carbonell, J. G., Learning by Analogy: Formulating and Generalizing Plans from Past Experience, Machine Learning, Michalski, Carbonell, Mitchell (eds.), Tioga Publishers, 1983, pp. 137-161.

[CardS2] Cardelli, L., "ML under Unix', Bell Laboratories, Murray Hill, New Jersey (1982).

[CardS6] Cardelli, Luca, "A Polymorphic Lambda-calculus with Type:Type", DEC SCR Report, Digital Equipment Corp., Palo Alto Calif.

[ChoS5] Chou, Shang-Ching, "Proving and Discovering Theorems in Elementary Geometries Using Wu's Method", Department of Mathematics, University of Texas, Austin (1985).

[ChoS6] Chou, Shang-Ching, "Proving Geometry Theorems Using Wu's Method: A Collection of Geometry Theorems Proved Mechanically", Technical Report 50, Institute for Computing Science, University of Texas at Austin, July 1986. (366 theorems)

[Cho87] Chou, Shang-Ching, "Mechanical Geometry Theorem Proving", to be published by Reidel Publishing company, 1987.

[CS86] Chou, S. C. and W. F. Schelter, "Proving Geometry Theorems with Rewrite Rules", J. of Automated Reasoning, 2(4) 1986, 253-273.

[CT82] Clark, K., and S.A. Tarnlund (Eds.) Logic Programming, 1982, Academic Press.

[CL73] Chang, C.C. and R.C.T. Lee, Symbolic Logic and Mechanical Theorem Proving, Academic Press, 1973.

[Coh87] Cohn, A.G., A More Expressive Formulation of Many Sorted Logic, JAR 3(1987) 113-200.

[Col] Colmerauer, a. [1973].

[Cons85] Constable, R.L., Constructive Mathematics as a Programming Logic I: Some Principles of Theory, Annals of Discrete Mathematics 24 (1985), 21-38.

[Cons86] Constable, R.L., et.al., Implementing Mathematics with the Nuprl Proof Development System, Prentice-Hall, 1986.

[CoH85] Coquand, Th. and G. Huet, Constructions: A Higher Order Proof System for Mechanizing Mathematics, EUROCAL85, Linz, Springer-Verlag LNCS 203 (1985).

[CREAS] Preliminary Proceedings of CREAS Workshop, Lakeway, Texas, May 4-7, 1987. Forthcoming Book: *Resolution of Equations in Algebraic Structures*, (H. Ait-Kaci, M. Nivat, Eds.), Academic Press, 1988.

[Da83] Davis, M., The Prehistory and Early History of Automated Deduction, in [SW83].

[deB80] de Bruijn, N.G., A Survey of Project Automath, in: To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism, Eds. J. P. Seldin and J. R. Hindley, Academic Press 1980

[deK84] de Kleer, J. and J.S. Brown, A Qualitative Physics Based on Confluences. Artificial Intelligence, 24 (1984) 7-83.

[deK84A] deKleer, J., Choices without backtracking. AAAI-84, pp. 79-85.

[Der87A] Derschowitz, N., "Completion and its Applications", [CREAS87].

[Der87B] Derschowitz, N. "Termination of rewriting"", Journal of Symbolic Computation, (in press) 1987.

[Do79] Doyle, J., A Truth Maintenance System. Artificial Intelligence 12 (1979), 231-272.

[Dix73] Dixon, J.K., Z-resolution: theorem-proving with compiled axioms, Jour ACM 20, (1973), 127-147.

[Eis86] Eisenger, N., "What you always wanted to know about Clause Graph Resolution", CADE-8, 1986.

[Ev51] Evans, T. " On multiplicative systems defined by generators and relations, I,", Proceedings of the Cambridge Philosophical Society 47, pp 637-649, 1951.

[Fay79] Fay, M., First order unification in equational theories, in Proceedings, 4th Conference on Automated Deduction, pp

161-167, Springer-Verlag, Lecture Notes in Computer Science, Volume 87, 1979

[Fo80] Forgy, C.L.Forgy, RETE: A fast Algorithm for the many pattern/many object pattern match problem, Tech Rep 309, CMU, 1980.

[Forb84] Forbus, K.D., Qualitative Process Theory, Artificial Intelligence 24 (1984) 65-168.

[Ge59] Gelernter, H., Realization of a Geometry-theorem proving machine, Proc Inter Conf on Information, UNESCO House, Paris, 1979. Also in Computers and Thought (Feigenbaum, Feldman, eds.) McGraw-Hill, 1963, 134-152

[GGS83] Genesereth, M.R., R. Greiner, and D.E. Smith, A Meta-Level Representation system, Stanford Memo HPP-83-28, May 1983. See also [MN87].

[GM78] Gallaire, H. and J. Minker, Logic and Data Bases, Plenum, 1978.

[GN87] Genesereth, M.R. and N.J.Nilsson, Logical Foundations of Artificial Intelligence, Morgan Kaufmann, 1987.

[GMW82] Gordon M., A. Milner, C.Wadsworth, Edinburgh LCF: A Mechanized Logic of Computation, Springer-Verlag, LNCS 78, 1982.

[Go80] Goguen, J., How to Prove Algebraic Induction Hypotheses without Induction, CADE-5, 1980.

[Go87] Gordon, Michael, "HOL, A Proof Generating System for Higher-Order Logic," to appear in "VLSI Specification, Verification, and Synthesis"

[Good85] Mechanical Proof about Computer Programs, in *Mathematical Logic and Programming Languages*, Prentice Hall International Series in Computer Science, 1985, (Eds: T. Hoare and J. C. Shepardson).

[Gr69] Green, C., Theorem proving by resolution as a basis for question-answering systems, Machine Intelligence 4 (American Elsevier, NY 1969), pp 183-205.

[Grb86] Greenbaum, S., Input transformations and resolution implementation techniques for theorem proving in first order logic, Ph.D. thesis, University of Illinois at Urbana-Champaign, September 1986.

[GP86] Greenbaum, S., and D. Plaisted, The Illinois prover: a general purpose resolution theorem prover, extended abstract, 8th Conf on Automated Deduction, July 1986.

[Gre85] Greiner, R., Learning By Understanding Analogies, Ph.D. Thesis, Stanford University, September 1985. Technical Report STAN-CS-1071.

[Ha85] Hall, R.P. Analogical Reasoning in Artificial Intelligence and Related Disciplines, Irvine Computational Intelligence Project, University of California, Irvine.

[HM86] R. Harper and K. Mitchell, Introduction to Standard ML, Laboratory for Foundations of Computer Science, University of Edinburgh (1986)

[HR78] Harrison,M.C. and N. Rubin, Another Generalization of Resolution, JACM 25 (1978) 341-351.

[Hi 87] Hines, L.M., Hyper-chaining and Knowledge-based Theorem Proving, submitted to CADE-9.

[Hi88] Hines, L.M., Building-in Knowledge of Axioms, PhD Dissertation. Univiversity of Texas.

[HN84] Henschen, L.J. and S.A.Naqvi, On Compiling Queries in Recursive First Order Databases, JACM 31 (1984) 47-84.

[Hob85] Hobbs, J.R., and R.C.Moore, Formal Theories of the Commonsense World, Ablex Publishing Corp, Norwood, NJ, 1985.

[Hod72] Hodes, L., Solving Problems by formula manipulation in logic and linear inequalities, Artificial Intelligence 3 (1972) 165-174.

[Hu73] Huet, G.P., A Mechanization of Type Theory, Proc. IJCAI-73, 139-146.

[HO80] Huet, G. and D. C. Oppen, Equations and rewrite rules: a survey, in R. Book (ed.) Formal Languages: Perspectives and Open Problems, Academic Press 1980, 348-405

[Hul80] Hullot, J.-M., Canonical Forms and Unification, Proc. 5th International Conference on Automated Deduction, 1980, Springer LNCS 87, 318-334

[Hun86] Hunt, Warren A., Jr., The Mechanical Verification of Microprocessor design. In [Borr87], 89-129.

[IEEE-C25] IEEE Transactions on Computers, V C-25 no. 8, 1976 (Special issue on ATP).

[Karl84] Karl Mark G. Raph (Siekmann et al), The Markgraf Karl Refutation Procedure, Seiki-84-08-K1, FB Inf., Univ. Kaiserslautern.

[Kir87] Kirchner, C. "Computing unification algorithms", Proc. of the 1st IEEE Symposium on Logic in Computer Science, 206-216, 1986

[Kli71] Kling, R. E., A Paradigm for Reasoning by Analogy, Artificial Intelligence, Vol.2, 1971, pp. 147-178.

[KC86] Knoblock, T.B., and R.L.Constable, Formalized Metareasoning in Type Theory, LICS 86 (ed. by A.K.Chandra and A. R. Meyer), 237-248.

[KB70] Knuth, D. and P. Bendix, Simple word problems in universal algebras Computational Problems in Abstract Algebra (J. Leech, ed.) Pergamon Press, Oxford, 1970, pp 263-297.

[Kon86] Konolige, Kurt, A Deduction Model of Belief, Morgan-Kaufmann Pub, 1956.

[Kon86A] Konolige, Kurt, Resolution and Quantified Epistemic Logics, CADE-8, Oxford, England, 1986.

[Kon86B] Geissler, C. and K. Konolige, A Resolution Method for Quantified Modal Logics of Knowledge and Belief, Proc Conf on Theoretical Aspects of Reasoning about Knowledge, 309-324, 1986.

[Ko71] Kowalski, R. and D. Keuhner, Linear resolution with selected functions. Artificial Intelligence 2, 1971, pp 227-260.

[Ko74] Kowalski, R., Predicate Logic as a Programming Language, Information Processing,

[Ko75] Kowalski, R., A Proof Procedure using Connection Graphs, JACM, Vol.22 No.4(1975), 424-436.

[Ko79] Kowalski, R. "Logic for Problem Solving", North-Holland NY, 1979.

[La75] Lankford, D. S., Canonical Inference, Memo ATP-32, Automatic Theorem Proving Project, University of Texas, Austin, TX, December 1975.

[LB77] Lankford, D.S. and A. M. Ballantyne, Decision procedures for simple equational theories with commutative-associative axioms: Complete sets of commutative-associative reductions, memo ATP-39, Automatic Theorem Proving Project, University of Texas, Austin, TX, August 1977.

[LPS86] Lenat, D., M.Prakash, M.Shepherd, CYC: Using common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks, AI Magazine, VI, Winter 1986.

[LF87] Lenat, D. and E.A. Feigenbaum, On the Thresholds of Knowledge, IJCAI-87.

[LeV62] LeVeque, W. J., Elememtary Theory of Numbers, Addison-Wesley, Reading, MA., 1962.

[Lif87] Lifschitz, V., What is the Inverse Method?, Memo, Stanford U. CS Dept, June 1987.

[Lo68] Loveland, D.W., Mechanical theorem proving by model elimination, Jour ACM 15, 1968, pp 236-251.

[Lo69] Loveland, D.W., A simplified format for the model elimination procedure Jour ACM 16 (1969) pp 349-363.

[Lo78] Loveland, D.W., Automated Theorem Proving: A Logical Basis, North-Holland, Amsterdam, 1978, xiii + 405 pp.

[Lo84] Loveland, D.W., Automated Theorem-Proving: A Quarter-Century Review. In [BL84].

[Lo86] Loveland, D.W., Automated Theorem proving: mapping logic into AI, invited paper, Int'l Symposium on Methodologies for Intelligent Systems, Z. Ras and M. Zemankova, Eds., Knoxville, TN October 1986, pp 214-229.

[Lo87] Loveland, Near-Horn PROLOG, CS-1987-14, Duke University, April, 1987.

[LO82] Lusk, E.L. and R.A. Overbeek, An LMA-based theorem prover, ANL-82-75, Argonne Natl Lab, 1982.

[LMO86] Lusk, E.L., W. McCune, and R.A.Overbeek, ITP at Argonne National Laboratory, CADE-8, Oxford, 1986, 697-698.

[LO85] Lusk, E. and R. Overbeek, Non-Horn problems, Jour of Automated Reasoning 1 (1985)

[LMO86] Lusk, E., W.McCune, and R.Overbeek, ITP at Argonne National Laboratory, 8th Int'l Conf on Automated Deduction, Oxford England, 1986, pp 697-698.

[Macsyma] MACSYMA Reference Manual, Lab. for Computer Science, MIT, 545 Technology Square, Cambridge Mass 02139 (1983)

[Martin-Lof 84] Martin-Lof, P., Intuitionistic Type Theory, Studies in Proof Theory Lecture Notes, Bibliopolis, 1984.

[McC63] McCarthy, J., "Situations, Actions, and Causal Laws", AI Memo 2, Stanford University AI Porject, 1963.

[Mas68] Maslov, S.J., The inverse Method for establishing deducibility for logical calculi, Proc Steklov Inst Math, 98, 1968.

[McC69] McCarthy, J., P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence", in Machine Intelligence 4, Melzer & Michie eds., Edinburgh Univ. Press 1969 pp 463-502.

[McA87] McAllister, D.A., ONTIC: A Knowledge Representation System for Mathematics, MIT AI Lab Tech Report 979, July 1987. PhD Thesis.

[McC68] McCarthy, J., Programs with Common Sense, in Semantic Information Processing (Minsky, ed) MIT Press, 1968.

[McC80] McCarthy, J., Circumscription – A form of Non-monotonic Reasoning, Artificial Intelligence 13 (1980) 27-39.

[McD79] McDermott, J., Learning to use analogies, IJCAI-79, 568-576.

[MS84] McDonald, James, and Patrick Suppes. Student use of an Interactive Theorem Prover. In [BL84], 315-360, 1984.

[Mi83] R. Milner, A Proposal for Standard ML, Report CSR-157-83, Computer Science Dept., Univ Edinburgh (1983); Also pub. in Conference Record of 1984 ACM Symposium on LISP and Functional Programming, Austin, Texas (Aug 1984)

[ML1] Michalski, R.S., J.C.Carbonell, and T.M.Mitchell, Machine Learning (Vol 1), Tioga Press, Palo Alto, 1983.

[ML2] Michalski, R.S., J.C.Carbonell, and T.M.Mitchell, Machine Learning (Vol 2), Tioga Press, Palo Alto, 1986.

[ML86] The ML Handbook, Internal Document, Project Formel, INRIA (July 1985)

[Mo69] Morris, J.B., E-resolution: extension of resolution to include the equality relation, Proc of the Int'l Joint Conf on AI, Washington, D. C. May 1969, pp 287-294.

[Mor85] Moore, R.C., A Formal Theory of Knowledge and Action, in [Hob85].

[Ne80] Nederpelt, R. P., "An approach to Theorem Proving on the basis of a typed lambda-calculus" CADE-5 pp182-194

[NSS56] Newell, A., J.C.Shaw, and H.A.Simon, The Logic Theory Machine, IRE Trans Information Theory IT-2, 1956; also in Computers and Thought, McGraw-Hill, 1963.

[Nil80] Nilsson, N.J., Principles of Artificial Intelligence, Tioga Press, Palo Alto, 1980.

[NO78] Nelson, G. and D.C.Oppen, A Simplifier Based on Efficient Decision Algorithms, 5th ACM Sym on Prin Prog Lan (1978) 141-150.

[Ohl87] Ohlback, H.J., Link Inheritance in Abstract Clause Graphs, JAR 3 (1987), 1-34.

[Op1] Oppacher, F., and E. Suen, Controlling deduction with proof condensation and heuristics. School of CS, Carleton U., Ottawa, Ontario.

[Pa87] Pace, Bill, A bibligraphy of Automated Deduction, TR-87-5400-08, I.P.Sharp Associated Limited, Ottawa, Ontario, Canada, Jan 1987.

[Pas87] Pastre, Dominique, MUSCADET: An Atutomatic Theorem Proving System using Knowledge and Metaknowledge in Mathematics, PhD Thesis, U. Paris.

[Pau86] Paulson, L., Natural Deduction as Higher-Order Resolution, Journal of Logic Programming, 3(3), 1986, 237-258.

[PRS86] Petrie, Charles J., David M. Russinoff, Donald D. Steiner, PROTEUS: A Default Reasoning Perspective. MCC-AI-352-86.

[Pla81] Plaisted, D., Theorem Proving with Abstraction, Artificial Intelligence 16, 1981, pp. 47-108.

[Pla82] Plaisted, D.A., A simplified problem reduction format, Artificial Intelligence 18 (1982) 227-261.

[Pla87] Plaisted, D.A., Non-Horn Clause Logic Programming without Contrapositives, Memo. Dept CS, U. North Carolina, 1987.

[Plu86] Plummer, D., Gazing: A Technique for Controlling the Use of Rewrite Rules in a Natural Deduction Theorem Prover, Department of Artificial Intelligence, University of Edinburgh, Ph.D. Thesis, Summer 1986

[Ric83] Rich, Elaine, Artificial Intelligence, McGraw-Hill, 1983.

[Rob65] Robinson, J.A., "A machine oriented logic based on the resolution principle" J. ACM 12 (1965) pp 23-41.

[Rob65A] Robinson, J.A., Automatic deduction with hyper-resolution, Interat.J. of Computer Math 1 (1965) 227-234.

[Ross86] Ross, K., Elementary Analysis: The theory of Calculus, Springer- Verlag, New York, 1986.

[Rou75] Roussel, P., PROLOG: Manuel de Reference et d'utilisation, Groupe d'Intelligence Artificielle, Universite d'Aux-Marseille, Luminy, Sept 1975.

[Ru85] Russinoff, David. An Experiment with the Boyer-Moore Theorem Porver: A proof of Wilson's Theorem, JAR 1 (1985) 121-139.

[Sh86] Shankar, N., Proof-checking Metamathematics, Dept of CS, Univ. of Texas, 1986.

[Sh87] hankar, N., A Machine-checked proof of Godel's Incompleteness Theorem, In: Proc. of Eighth Int'l Conf. on Logic, Methodology and Philosophy Computer Science Congress, to Appear.

[Sho77] Shostak, R.E., On the SUP-INF Method for proving Presburger Formulas, JACM 24 (1977) 520-543.

[Sho79] Shostak, R.E., A Practical decision Procedure for Arithmetic with function symbols, JACM 26 (1979) 351-360.

[Si76] Sickel, S., Interconnectivity Graphs, IEEE Trans on Computers, C-25 (1976).

[Sim84] Simon, D., A Linear Time Algorithm for a subcase of Second-order Instantiation, 7th Conf on Automated Deduction, Napa, Ca, May 1984.

[Sim88] Simon, D., Checking National Proofs, submitted to 9th Conf on Automated Deduction, 1988.

[Smu85] Smullyan, R., "To Mock a Mockingbird", Alfred Knopf, NY, 1985.

[SW83] Siekmann, J. and G. Wrightson, The Automation of Reasoning I,II, Springer Verlag, 1983.

[SW79] Siekmann, J., Wrightson, G., Paramodulated Connection Graphs, Acta Informatica (1979)

[Sla74] Slagle, J. "Automated Theorem Proving with Simplifiers, Commutativity, Associativity", J.ACM, 21, 622-642,(1974)

[Sm87] Smolka, G., W. Nutt, J. Meseguer, J.A. Goguen, "Order Sorted Equational Computation", [CREAS87]

[Smu68] Smullyan, Raymond, First Order Logic, Springer-Verlag, Berlin, 1968.

[SW86] Stanfield C., Waltz D., Toward Memory-Based Reasoning, Thinking Machines Report.

[St81] Stickel, M. E., A unification algorithm for associative-commutative functions, J.ACM 28(3), pp423-434, 1981

[St84] Stickel, M.E., A Case Study of Theorem Proving by the Knuth-Bendix Method: Discovering That $x^3 = x$ Implies Ring Commutativity, 7th International Conference on Automated Deduction, Napa, CA., May 14-16, 1984.

[St85] Stickel, M.E., Automatic Deduction by Theory Resolution, Proceeding IJCAI-85, 1985, pp. 1181-1186.

[St86] Stickel, M.E., A Prolog Technology Theorem Prover: implementation by an extended Prolog compiler, Eight Int'l Conf.

on Auto. Deduction, Lecture Notes in C.S. 230, Springer-Verlag, Berlin, July 1986, pp 573-587.

[Top79] Goldblatt, Robert, "Topoi: The categorial Analysis of Logic," North Holland, New York, 1979.

[Ty81] Tyson, Mabry, APRVR: A Priority-ordered Agenda Theorem Prover, PhD Dissertation, U. Texas CS Dept, Aug 1981. See also Proc. AAAI-82.

[Wal86] Wallen, Lincoln, Chapter on Modal Logic in Artificial Intelligence and its applications, Eds: A.G. Cohn and J.R. Thomas, John Wiley and Son, 1986.

[Wa83] Walther, C., A many-sorted calculus based on resolution and paramodulation, Proc 8th Int'l Joint Conference on AI, Karlsruhe, West Germany, August 1983, pp 882-891.

[Wa84] Walther, C., A mechanical solution of Schubert's steamroller by many-sorted resolution, Proceedings of the AAAI-84 Nat'l Conf on AI, Austin, TX August 1984, pp 330-334.

[WaT85] Wang, T.C., Designing examples for semantically guided hierarchical deduction, IJCAI-85, pp. 1201-1207.

[WBS7] Wang, T.C. and W.W. Bledsoe, Hierarchical Deduction, JAR 3 (1987) 35-77.

[WaH60] Wang, Hao, Toward Mechanical Mathematics, IBM J. of Research and Development 4 (1960) 2-22. Also see [SWS3] 244-264.

[War87] Warren, D.H.D., Implementing Prolog - compiling predicate logic programs, DAI Research Reports 39, 40, University of Edinburgh, May 1987.

[We77] Weyhrauch, R., FOL: A proof checker for first order logic, Stanford AI Memo AIM-235.1, 1977.

[WHS3] Wos, L. and L. Henschen, Automated Theorem Proving 19865-1970, In [SW83].

[Wink76] Winker, S., An evaluation of an implementation of qualified hyperresolution, IEEE Transactions on Computers C-25, 8 (1976) 835-843.

[Win80] Winston, P.H., Learning and Reasoning by Analogy, CACM 23(12), 689-703, 1980.

[Wo65] Wos, L., G. Robinson, D.F. Carson, Efficiency and completeness of the set of support strategy in theorem proving, JACM 12 (1965) 536-541.

[Wo67] Wos, L., G. Robinson, D. Carson, L. Shalla, The concept of demodulation in theorem proving JACM 14 (1967) 698-709.

[Wo70] Wos, L., G. Robinson, Paramodulation and set of support, Symp. on Automatic Demonstration, Lecture Notes in Math 125, Spring Verlag, 1970, 276-310.

[Wo84] Wos, L., R.Overbeek, E.Lusk, and J.Boyle, Automated Reasoning: Introduction and Application, Prentice-Hall, Inc., Englewood Cliffs, NJ., 1984.

[Wo84A] Wos, L., R. Veroff, B. Smith, and W. McCune, The linked inference principle, II: the user's viewpoint, Proc. of the 7th Int'l Conf on Automated Deduction, Napa, CA May 1984, pp 316-332.

[Wo87] Wos, L., Automated Reasoning: 33 Basic Research Problems, Prentice Hall, August 1987.

[Wu78] Wu, "On the Decision Problem and the Mechanization of Theorem Proving in Elementary Geometry" Scientia Sinica Vol. 21, 1978 pp 157-179.

[Wu84] Wu, "Basic Principles of Mechanical Theorem Proving in Geometries", J. of Sys. Sci. and Math. Sci 4(3) 1984, pp 207-235, republished in Journal of Automated Reasoning 2(4) 1986, 221-252.