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

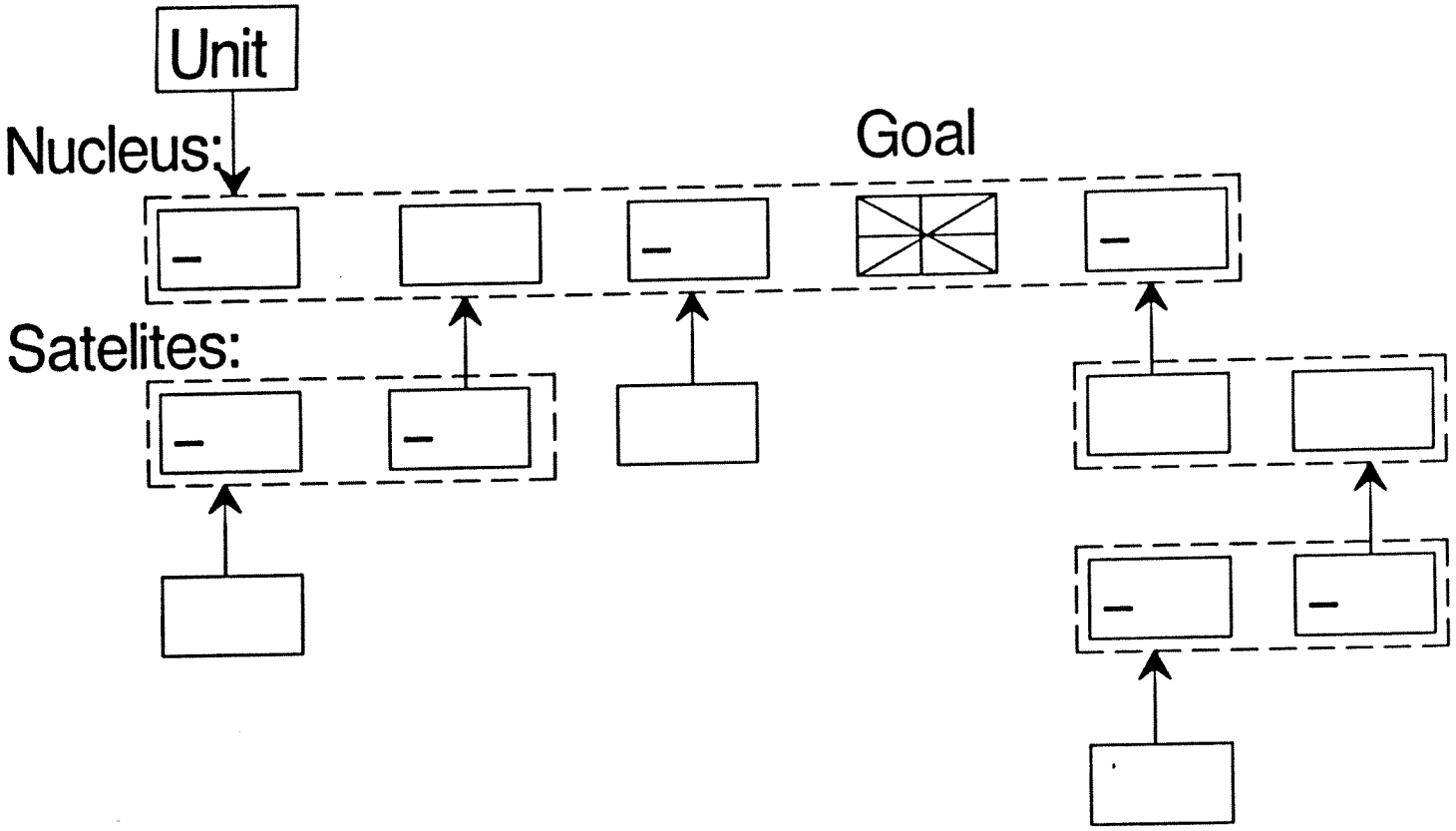
$F \quad -B \quad -C \quad E, \quad F \quad G \quad -C \quad E$

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.

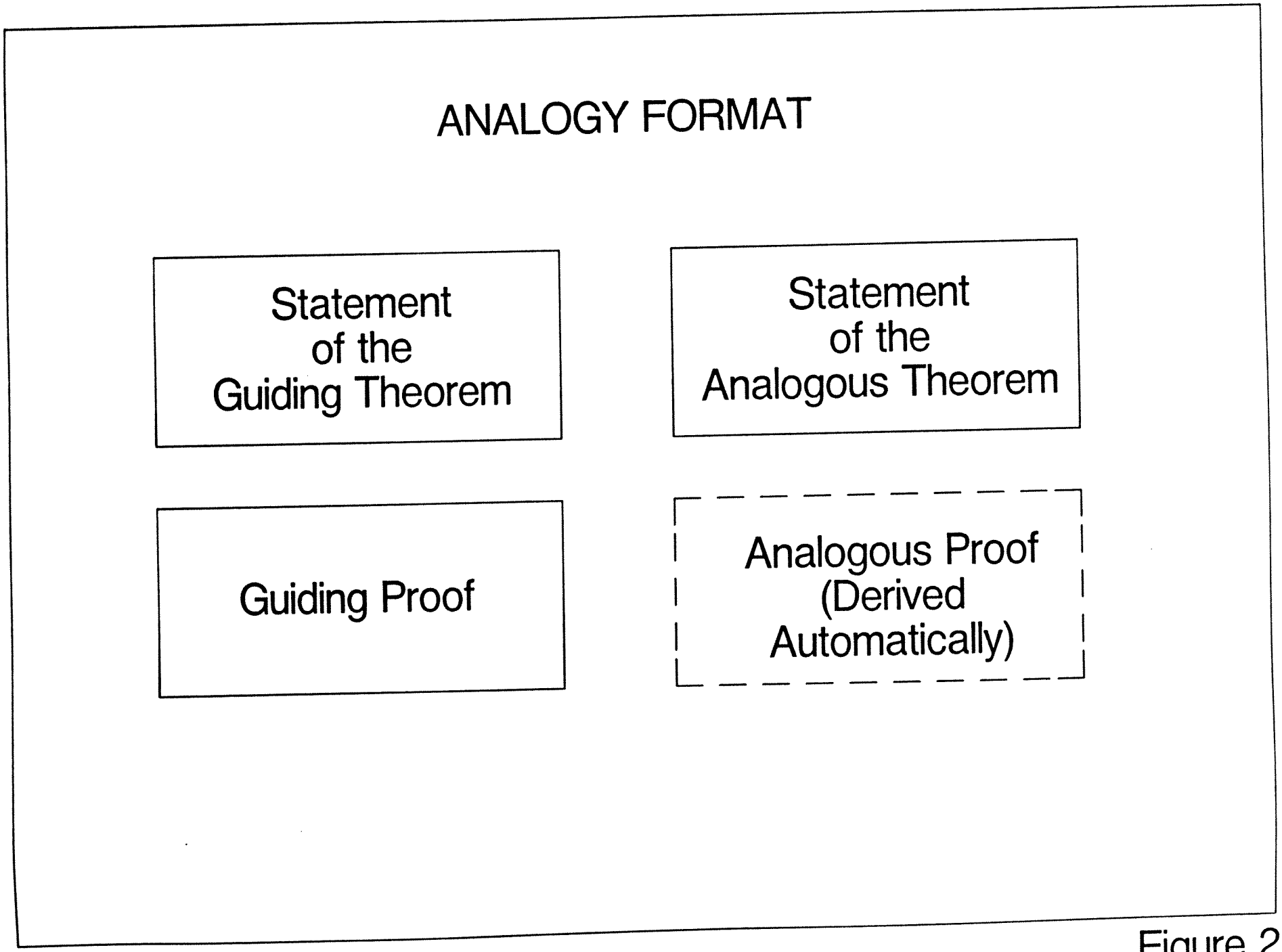

ANALOGY FORMAT
Statement of the Guiding Theorem
Statement of the Analogous Theorem
Guiding Proof
Analogous Proof (Derived Automatically)

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

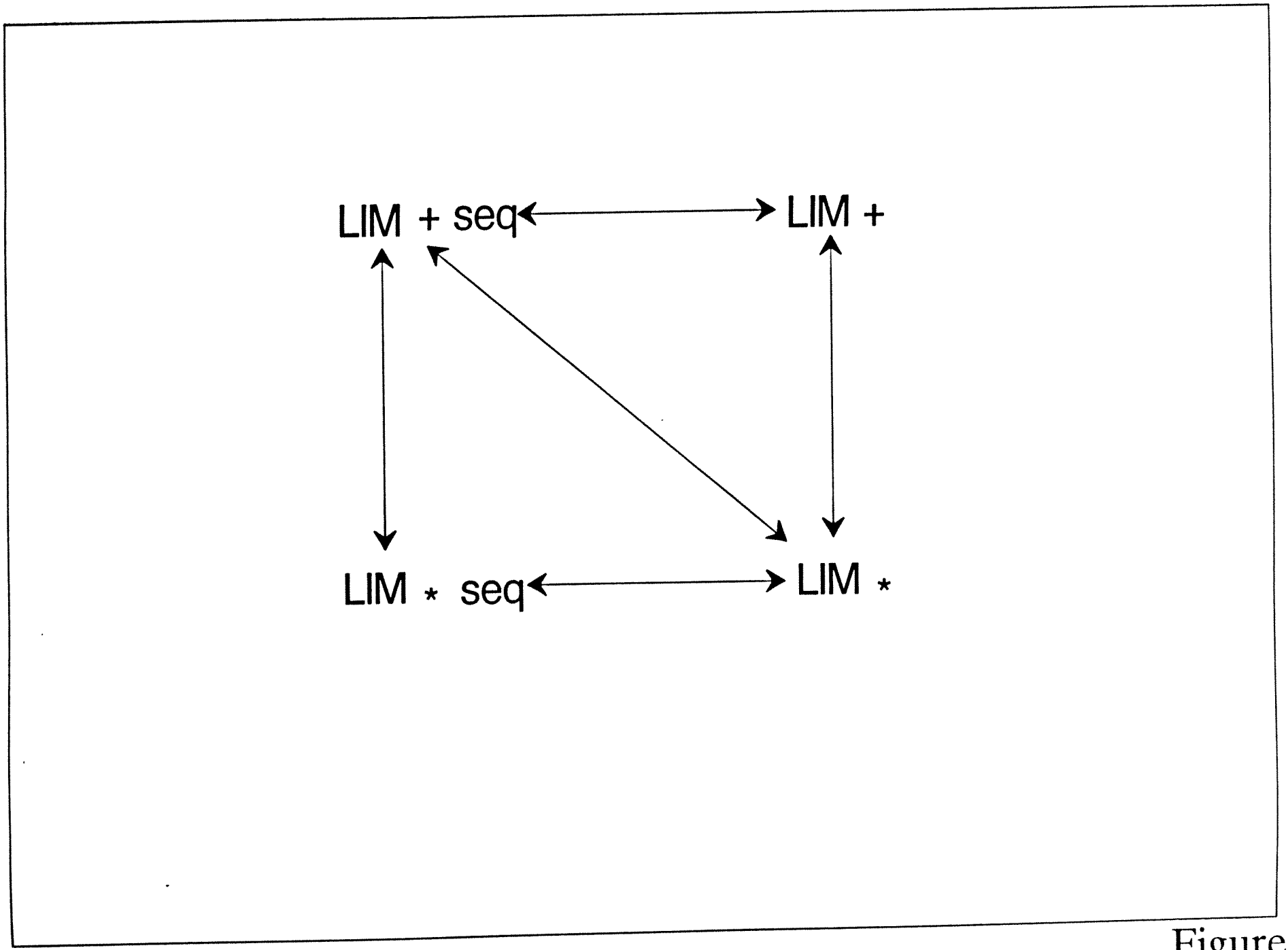


Figure 2

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)

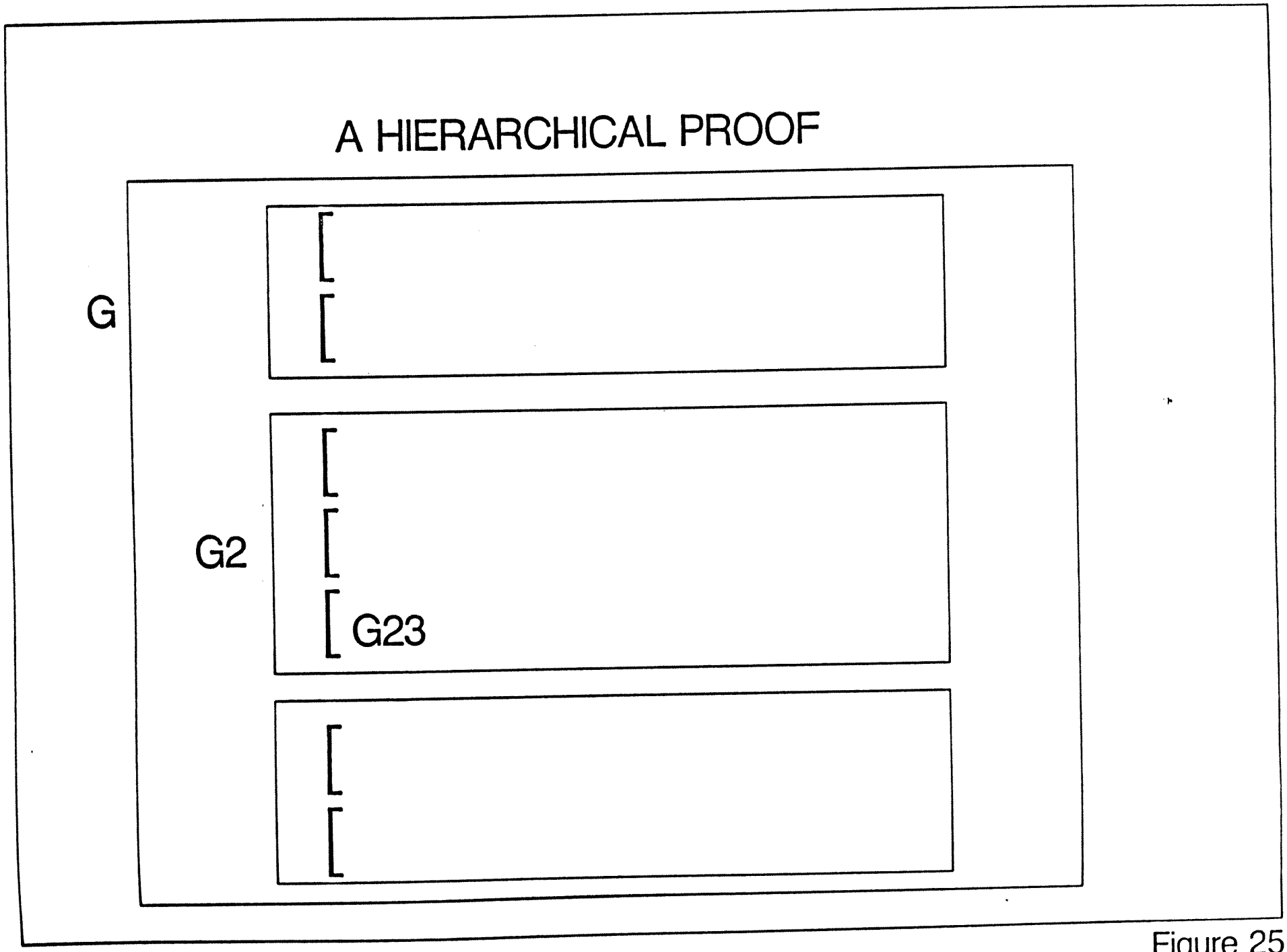


Figure 25

- 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

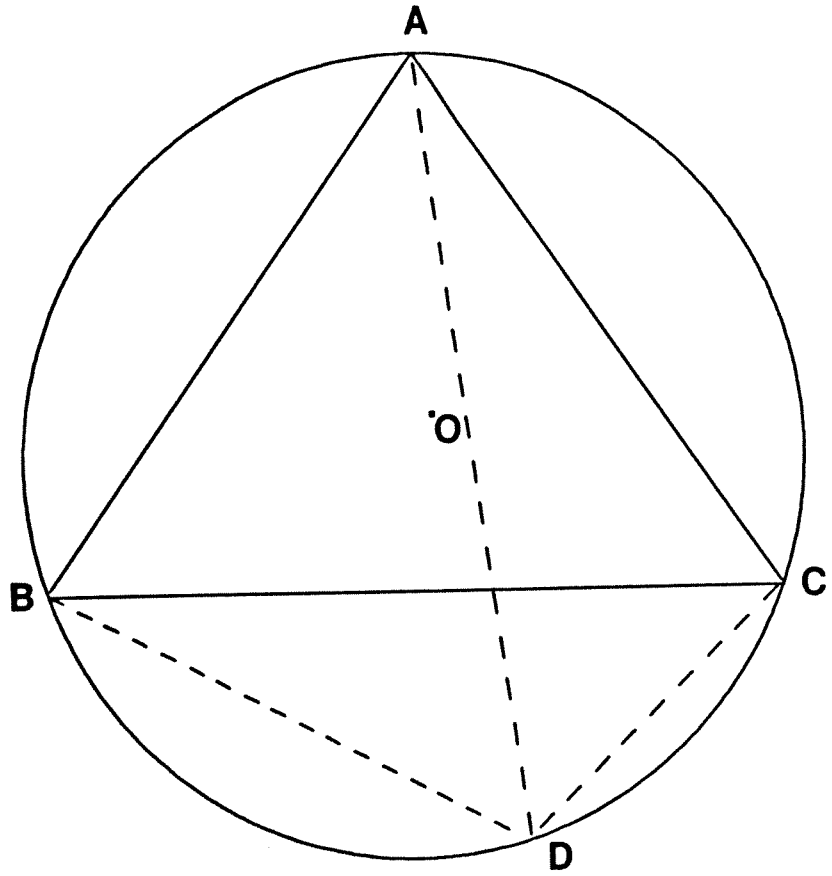


Figure A30–29

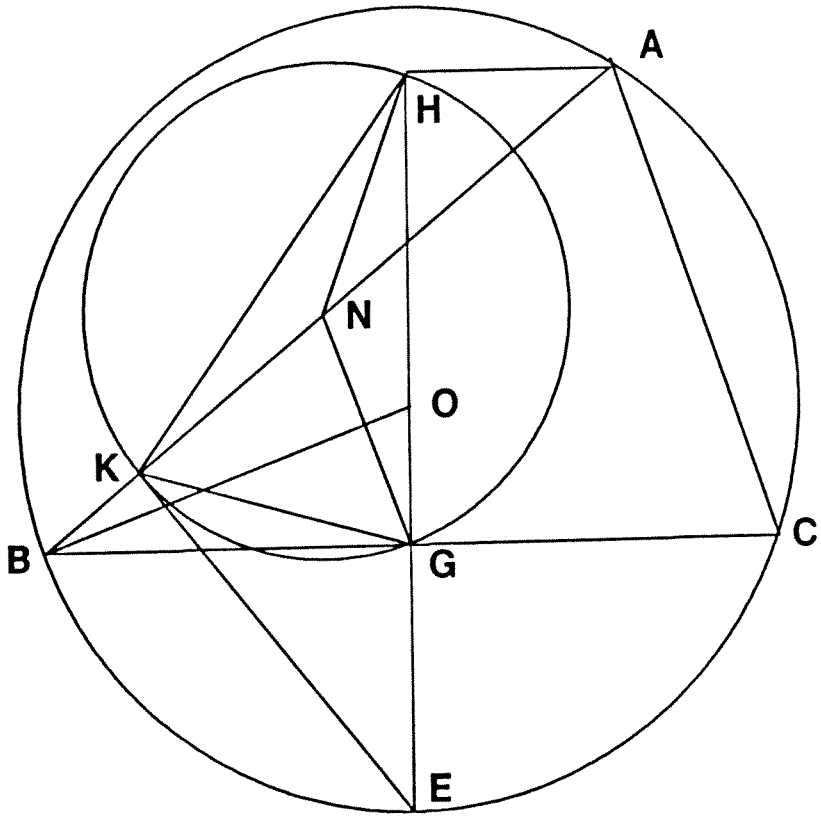


Figure A32–49

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]