1986

Prototype for a high school geometry tutorial

Timothy Spangler
The University of Montana

A PROTOTYPE FOR A HIGH SCHOOL GEOMETRY TUTORIAL

By

Timothy Spangler

B.A., Whitman College, 1967

Presented in partial fulfillment of the requirements

for the degree of

Master of Science

University of Montana

1986

Approved by

Alden H. Wright

Chairman, Board of Examiners

[illegible]

Dean, Graduate School

Date July 28, 1986

Spangler, Timothy, M.S., August 1986 Computer Science

A Prototype for a High School Geometry Tutorial (208 pp.)

Director: Alden H. Wright [illegible]

This thesis examines requirements of an Intelligent Tutoring System (ITS) for tutoring proof learning in geometry. Computerized systems for tutoring high school geometry can be designed and implemented with current technology using tools of Artificial Intelligence, high resolution graphics, and knowledge gained from pioneering ITSs. This thesis describes the components of a typical ITS and compares it to a geometry tutor currently under development at Carnegie Mellon University by a group led by John Anderson.

Using a method known as rapid prototyping, the programming part of this this thesis implemented a user interface for a geometry tutor on a Macintosh microcomputer for demonstration to local high school teachers and geometry students. The lesson implemented with this prototype dealt with development of skills needed to apply both geometric and heuristic rules to problems encountered in the construction of a proof. The learning transition considered by the project was based on skill levels described by the van-Hiele model of geometric learning.

The thesis also begins the top down development of a second prototype and points out requirements for expert components of an ITS that must be developed. It was concluded that an ITS for geometry would require several additional software and hardware tools currently not available at this school. Development of an effective ITS for proof in geometry would also require several individuals possessing different fields of expertise including: geometry, geometry education, and system development skills.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This thesis is an investigation into the feasibility of building a computerized Intelligent Tutoring System (ITS) for high school geometry. Geometry proof, often considered critical to the development of a student's reasoning skills, is difficult for the average student to learn in a typical classroom. Statistical studies have shown that up to 85% of high school students cannot do proofs, a "skill that underlies the structure of a standard geometry course (Senk, 1985 / p. 85).

Individualized instruction, such as one-on-one tutoring makes geometry proof much easier to learn. Good students are able to learn much faster and slower students are able to understand concepts they fail to grasp in the classroom environment. A study by John Anderson a psychologist at at Carnegie Mellon University indicates that private tutoring can speed up geometry learning by as much as four times (Anderson, 1985).

The advent of the computer in the 1950s offered much hope to mathematics educators who saw the machines as new educational tools. Systems to "instigate and control learning" called Computer Aided Instruction (CAI) were developed to perform individual instruction. Most of these programs were only a form of "drill and practice". They lacked the individualized analysis of a student's performance which is necessary to correct misconceptions. The CAI systems of the 1960s lacked most of the resources to build a truly intelligent educational system and were considered failures by many (Barr, 1982 / p. 226).

Today, major constraints upon the development of intelligent systems are beginning to loosen. Both computer speed and memory size have increased while costs for these more powerful machines have dropped significantly. Machines are now available whose hardware is designed to quickly implement LISP, a primary language of intelligent systems. A recent article in Mini-Micro Systems pointed out that:

> Last year, Xerox Corp. introduced two low-cost AI workstations, the Xerox 1185 and 1186. At $9,995, the 1186 may be the cheapest LISP machine on the market (Tucker, 1986 / p. 73).

Machines like the 1186 are much faster, have more memory, and are more versatile than the PDP-11, a computer used to build experimental intelligent tutors over the past 15 years. Thirty years of research into artificial intelligence is also available to today's developer of intelligent systems. Today's developers have access to highly interactive graphic interfaces and often can use software tools that are designed for expert systems and other AI applications.

In order to develop an Intelligent Tutoring System for geometry, it is necessary to compile much knowledge about geometry, the process of tutoring geometry, and requirements needed to create a system. This thesis points out several of those requirements. It also attempts to determine the scope of a feasible system.

An important constraint that influenced this study was the need to limit both the size and scope of the attempted system. In Intelligent Tutoring Systems Sleeman and Brown point out:

> ITS has clearly abandoned one of CAI's early objectives, namely that of providing total courses, and has concentrated on building systems which provide supportive environments for more limited topics (Sleeman, 1982 / p. 8).

They go on to say that the existing intelligent systems for education concentrate on bottlenecks in the development of a subject. In those systems each topic "represents an educational 'watershed' in that if any of these skills are not acquired, further progress is greatly inhibited" (Sleeman, 1982 / p. 8).

In an attempt to limit the educational scope of a small system, this study has concentrated on the problem geometry students have in making a formal inference from a rule. The problem corresponds to what mathematicians call modus ponens. It also corresponds to the transition between two "levels" for geometry learning described by the van-Hiele model (Shaughnessy, 1985) discussed later in this paper.

To begin the development of system requirements, several ideas are borrowed from existing experimental ITS systems. This paper also investigates research of mathematics educators and attempts to apply some of their ideas to the creation of an adequate system.

The bulk of this project has been the development of a prototype geometry tutorial. To get a feeling for what the tutoring system could be, an environment was created on the Macintosh microcomputer. A short geometry lesson was implemented in this environment that involved a three-step proof. Tutoring of high school geometry students and informal conversations with mathematics teachers were held concurrent with the development of this prototype. Minimal requirements for the prototype were based on these sessions.

After describing the prototype, this paper reports the reactions of some mathematics teachers and geometry students who worked through the prototype's lesson. It describes changes that were made to accommodate the criticism, and other changes that could be made easily. A potential second prototype is then described.

CHAPTER 2

THE INTELLIGENT TUTORING SYSTEM

This chapter describes the basic components found in most Intelligent Tutoring Systems (ITS). It examines some experimental systems which are classified as intelligent tutorials and establishes a context for a geometry tutorial. Included is a description of an ITS for geometry currently under development by John Anderson at Carnegie Mellon University.

A primary goal of an ITS system is to establish student driven educational environments by simulating a human tutor. Traditional CAI systems attempt to simulate a human tutor by guiding a student through the learning of a particular subject. However, CAI systems tend only to lecture and drill students while

> a good human tutor does not merely traverse a predetermined network of knowledge in selecting material to present. Rather it is the process of

> the tutor's ferreting out student misconceptions that drives the dialogue. (Barr, 1982 / p. 231)

Attempting to emulate a human tutor, most experimental ITS systems include three basic modules: 1) an domain expert, 2) a student model, and 3) a tutor (Barr, 1982 / p. 229).

The domain expert of an Intelligent Tutoring System has the function of generating problems and evaluating the correct solutions to those problems. Usually the domain expert is driven by an inference engine operating on domain knowledge. This knowledge often has a declarative representation in the form of if-then production rules.

The student model of an ITS maintains a record of selected student knowledge and skills. As Sleeman and Brown point out:

> If a conversational system is to manage realistic dialogues, it must have some representation of the user's conceptualization of the domain. Without such a model, the system may provide comment at the wrong level of detail or mistake the user's current focus of attention (Sleeman, 1982 / p. 5).

By creating a student model, an ITS is able to evaluate a student's performance and determine misconceptions. This is often done with an "overlay model" which compares the student's knowledge with knowledge in the domain expert (Barr, 1982 / p. 231)

The third necessary ITS module, the tutor, has two basic functions. Using the student model, the tutor evaluates student performance and determines possible misconceptions. Also the tutor handles all communications from the system that help a student realize errors.

2.1. The Domain Expert

Ideally an ITS domain expert can solve all the problems presented by the system during a tutorial session. For example WEST, a type of ITS called a "computer coach", employs an expert to solve problems in arithmetic and game playing. WEST is based on a game similar to a popular board game called "Chutes and Ladders". In the computer game, "How The West Was Won", players move their playing piece along a path toward a goal. With three numbers obtained from spinners, players can use any combination derivable with arithmetic operators and parentheses as their step count. There are different game playing strategies that players can use to their advantage such as jumping to the next town when they land on a town, or sending their opponent back when they land on the opponent's square. (Burton, 1982)

WEST helps students learn skills in mathematics and gamesmanship. Each time a player spins, WEST's domain expert ranks the different correct answers in order of optimality. It also identifies the skills, which it calls "issues", necessary to come up with those answers. WEST's expert divides these skills into three levels: 1) math skills, 2) WEST playing skills, and 3) general game playing skills (Barr, 1982 / p. 257). With this information, WEST is able to identify what an expert game player would do in a given situation.

Like WEST, other intelligent tutorials should be able to evaluate any possible action by the student, identify skills needed for expert performance, and discriminate among the types of skills needed to perform a task.

Another domain expert used in an ITS is MYCIN, a well-known expert system containing domain-specific knowledge needed to diagnose bacterial infections. Using patient data obtained through interactions with doctors and built-in production rules, MYCIN "provide(s) consultative advice ... and therapy for infectious diseases" (Barr, 1982 / p. 184). MYCIN is the domain expert for GUIDON, a system designed to help medical students learn both the domain specific knowledge of MYCIN and the logic used in its diagnostic process. Within the framework of an AND-OR search tree, MYCIN solves a "case" and provides GUIDON with the solution.

The student and GUIDON start at the beginning stages of the diagnosis and proceed through MYCIN's solution of the case. GUIDON walks a student through the logical chain created by MYCIN thus demonstrating MYCIN's obscure rules and methods. The important contribution GUIDON made to ITS was its demonstration that:

> In addition to the domain knowledge ... a tutorial ... requires teaching expertise such as the ability to tailor presentation of domain knowledge to the students competence and interests (Clancey, 1982 / p. 204).

GUIDON serves as an important contrast to a geometry tutorial. A proof tree created for a geometry problem is similar to that created by MYCIN. However, the theorem prover for a geometry tutorial should create solutions that are more intuitive and understandable by students.

In contrast to WEST and GUIDON, BUGGY is an ITS system whose expert does more than generate correct solutions to problems. It also generates incorrect solutions. It is designed to discover why students make errors in the application of basic arithmetic algorithms. BUGGY's expert

> represents a skill, such as addition , as a collection of subskills, ... The subprocedures in BUGGY that correspond to human subskills are linked into a procedural net, which is BUGGY's representation of the entire human skill (Barr, 1982 / p. 280).

BUGGY will solve problems correctly when it is using the proper subskills. It also has "buggy rules" which if substituted for the correct subskills produce errors. Matching performance of buggy solutions with a student's wrong answers help determine what the student is doing wrong. This feature of BUGGY'S expert module performs functions found in the student model of other systems.

The principles of BUGGY's domain expert have been adopted in the development of an intelligent geometry tutorial. Anderson has created a geometry tutorial based on a memory model called ACT. ACT is a production system where: "Essentially, every production in the system encodes a meaningful step of cognition." ACT serves as the system expert and solves geometry problems using production rules for geometry in a human like way. The expertise module of Anderson's tutorial uses a set of ideal and buggy rules rules called an IBR, and works in a way similar to BUGGY. The ideal rules represent subskills necessary to solve geometry problems correctly, and the buggy rules represent misconceptions which, when applied, will produce incorrect results. This extensive knowledge is stored in a production system. Anderson thinks that with these rules "it is possible to outperform human tutors ..." (Anderson, 1985 / p. 1-2)

Based on the IBR, the expertise module of Anderson's tutorial can solve geometry problems using the ideal rules of the knowledge base. Like a human problem solver, ACT activates both forward chaining from the given information and backward chaining from the goal to create a proof tree that solves the problem (Anderson, 1983). Like BUGGY, ACT uses buggy rules in the knowledge base to generate incorrect solution paths that match the activity of students with misconceptions. Anderson is the definitive reference for any computerized geometry tutor. More will be said about his memory model and its attempt to emulate a human problem solver.

2.2. The Student Model

The goal of a student model is to record what the student knows while working problems within the system. Burton and Brown, in their description of WEST give a strong justification for the student model component of an ITS.

> Apart from outright errors, the main window a computer based coach has to a student's misconceptions is through a 'differential' modeling technique that compares what the student is doing with what the expert would do in his place. (Burton, 1982 / p. 81)

In WEST the differential model has two tasks: 1)it must evaluate the current move of the student and 2) it must determine the skills that were necessary to make that move. Potential skills can be found by "looking at the expert's problem-solving trace for generating a given move ..." (Burton, 1982 / p. 82). By comparing this information, the system can discover skills the student knows and skills to be learned.

Each skill or "issue" necessary for good moves has two procedures related to it: 1) an issue recognizer that "watches student's behavior for evidence that a student does or does not use its particular concept or skill " and 2) an issue evaluator that is used by the tutor module to decide the student's weaknesses (Burton, 1982 / p. 83).

Like arithmetic, geometry problem solving requires various skills. A tutoring system must be able to determine a user's skills and to discover missing skills. Furthermore it would be beneficial to maintain a record of this information to use later in the tutoring session.

GUIDON also creates a student model. The record GUIDON keeps of student performance helps choose "knowledge to present to a student based on his competence and interests." GUIDON goes a step further than WEST in maintaining this record. The GUIDON system "acts as an agent that keeps track of the knowledge that has been presented to the student in previous sessions". (Clancey, 1982 / pp. 201-205)

GUIDON maintains a history of a student's knowledge by means of a three-part, iterative "USE" cycle. To do this, GUIDON borrows the concept of "certainty or confidence factors" from MYCIN that give truth value to beliefs held by the system. One of the components of the USE cycle is a

> cumulative record of which rules the student knows ... called the USE-HISTORY ... [which] is represented by a certainty factor that combines the background evidence with the implicit evidence stemming from needs for assistance and verbalized partial solutions, as well as explicit evidence stemming from a direct question that tests knowledge of the rule (Barr, 1982 / p. 271).

GUIDON updates the student model at critical times using active components of the model. Another part of the USE cycle is updated each time the system receives input concerning the student's understanding. In the other component, whenever MYCIN fires a rule in its solution of the case, GUIDON records the system's belief that the student could use the newly fired rule.

GUIDON maintains this complex student model in coordination with its parent, MYCIN. GUIDON's design is important to a geometry tutorial because it indicates the complexity involved in maintaining information about student knowledge and misconceptions over time. GUIDON makes an attempt to simulate a human tutors who know their students' capabilities from a familiarity that builds up over several sessions.

Rather than maintain an explicit student model, both BUGGY and Anderson's geometry tutorial have the student model built into buggy rules. These systems use a process of searching through the buggy subskills to find a combination of skills, both good and buggy, that will reproduce the incorrect answers given by a student. In certain situations this may be very effective. However, it presents some problems. First, it assumes the system knows all of the subskills needed to solve a problem. Second, it does not appear to form an image of individual students over time. Tutors are able to effectively help students because they know them personally and can help them on individual problems.

2.3. The Tutor

Both the expert and the student model become operational by means of the tutoring component of an ITS. This module acts as the communication link between the user and the expert. It maintains the educational dialogue of an interactive session.

The WEST tutor or "coach" takes a "constructivist position" when helping a student. It distinguishes between what it calls constructive and non-constructive bugs.

> If the student has enough information to determine what caused the error and can correct it, the bug is referred to as constructive. If, however, the student does not have sufficient information to change his behavior as a result of the perceived error, the bug is termed non-constructive. (Burton, 1982 / p. 80)

Following this philosophy the WEST tutor is very careful about interrupting the fun of the game to help the student. As the student plays the game, a student model is created as described above. If a student makes a less than optimal move, the student model is evaluated to determine those issues in which a student is weak. The tutor may respond by:

> providing both the description of a generic issue (a concept) and a concrete example of its use increasing the chance that the student will integrate this piece of tutorial commentary into his knowledge. (Burton, 1982 / p. 90)

WEST's tutor uses goal-directed "Issue evaluators" that find the student weaknesses and tutoring principles to determine how and if it should interrupt (Barr, 1982 / p. 257). If it decides to interrupt, it does so through a special procedure attached to each issue called a speaker.

WEST's tutor reveals three things that help determine requirements for a geometry tutor: it gives only constructive help, its use of "issues and examples" to teach abstract concepts using concrete examples could be copied by a geometry tutorial, and it uses a set of tutoring principles to determine how and when the student should be interrupted.

The GUIDON tutor observes a student's progress through a case diagnosis. As Barr and Feigenbaum point out:

> (The) record of what the expert (i.e., MYCIN) 'knows' at any time during the student-run consultation forms, the basis for evaluating a student's partial solutions and providing assistance. ...

> Referring to the rules that MYCIN uses to solve subproblems, ... GUIDON decides which of these rules, if any, might have been used by the student. That is, what inference chains are consistent with the student behavior? (Barr, 1982 / p. 270-274)

GUIDON's tutor gets information about the student from a communication model. This model is made up of the student overlay described above, a case syllabus which includes information to be learned from the specific case, and a focus record that indicates those interests the student is pursuing at the time. This information allows GUIDON to present students with material that is appropriate to their abilities and interests.(Burton, 1982)

Using the information of the communication model, the tutor selectively activates a "discourse procedure" which communicates with the student. These procedures conduct a "goal directed dialogue" in which; the system is geared to teach the student a particular phase of the MYCIN solution, and the student is given a degree of flexibility in choosing the depth of detail. (Burton, 1982)

GUIDON-WATCH, another system which is an extension of GUIDON, gives students even more access to information. Using a graphic interface, it provides users with mouse controlled access to multiple views of the system. They can choose to look at MYCIN's solution tree as it dynamically processes the diagnosis. They can also watch the changing stacks of subgoals which MYCIN must examine to come up with a solution. (Richer, 1985)

The original GUIDON tutor followed a set of tutoring rules that function as productions within the system. These rules were not the productions of MYCIN which encode the medical knowledge of the system. Instead they had access to 1) knowledge about dialogue patterns, 2) forms of domain knowledge for carrying on dialogues and 3) knowledge of the communication situation (Clancey, 1982). The tutoring rules worked together with the discourse procedures to create interaction with the student.

> A discourse procedure step specifies in a schematic form WHEN a type of remark might be appropriate. WHETHER to take the option ... and WHAT to say exactly ... will be be dynamically determined by tutoring rules ... whose preconditions refer to the student model, case syllabus, and focus record. (Clancey, 1982 / p. 209)

GUIDON provided a concrete example of a method for presenting information from a diverse array of help procedures. The GUIDON tutor presented necessary material based on student knowledge and interest. The tutor component of a geometry tutorial can use several of the features which GUIDON provides including: 1) tutoring rules to drive the interaction and 2) discourse procedures that are tailored to a variety of situations and abilities. GUIDON-WATCH demonstrates the potential of a graphic interface that reveals inner workings of an expert system which might be incorporated into an expert geometry solver.

The tutor of Anderson's geometry tutorial controls communication between the user interface and the system expert, the IBR. The two methods of interaction with the IBR are:

(a)It can look at which ideal and buggy rules are currently instantiated in the IBR and use these to interpret the student's behavior.

> (b)It can request of the ideal model whether a statement can be proven, subject to certain constraints. This will cause the IBR to attempt a proof and report back information such as whether such a proof exists, how long it is, how optimal it is, what rules it involves, etc. (Anderson, 1985 / p. 2)

The first method allows the tutor to determine correct progress and misconceptions of the student. The second can be used to determine whether a new assertion made by a student is both logical and constructive.

The tutor also communicates with the interface, the third component of Anderson's tutorial. The purpose of the interface is to "communicate to the student the logical structure of a proof and the structure of the problem-solving process by which the proof is generated."(Anderson, 1985 / p. 4) Essentially the interface provides students with a means to build a graph of the proof. They can either forward chain from the givens or backward chain from the conclusions of the problem. As Anderson describes it:

> The student grows the graph by a combination of pointing to statements on the screen and typing in

> information. Each step of the inference involves a set of premises, a reason, and a conclusion. Reasoning forward, the student points to the premises, types in the reason, and points to the conclusion or types it. ... The student is finished when there is a set of logical inferences connecting the givens to the statements to be proven. (Anderson, 1985 / p. 4)

This interface is very effective. It is supported with syntax checks and help windows that provide statements of applicable rules. Anderson has implemented a "minimal tutor" which

> can be described with respect to three steps a student must go through to complete an inference: selecting a set of statements from which to make an inference, specifying the rule of inference that will apply to these statements and then specifying the statements that result from applying this rule of inference. (Anderson, 1985 / p. 6)

Anderson's tutorial is an ambitious project in the development of Intelligent Tutoring Systems. The human-like expert and stimulating interface should be copied or used if possible. However, the present minimal tutor has some serious gaps that need to be filled. Anderson admits that the system falls short in its ability to help confused students who do not know what to do. Also, he points out that the system lacks "remedial problems tailored to student weaknesses"(Anderson, 1985 / p. 7). Anderson's geometry tutorial could use features like the student model and discourse procedures of GUIDON. These features could work together with tutoring rules and an IBR to control the system interface.

2.4. Problem Solution as Heuristic Search

In the development of a geometry tutorial, Anderson's group has found that students use a mixture of forward chaining and backward chaining in the development of a geometry proof. He points out that:

> This mixture along with various search heuristics they acquire, enables students to deal with search demands of proof problems in high school geometry texts. (Anderson, 1983 / p. 194)

Heuristics, the ancient study of "the methods and rules of invention and discovery" (Polya, 1945 / p. 112) can well serve as part of the knowledge base of an intelligent tutor. These methods can be encoded as rules used by the expert to solve problems in a humanlike way. Also problem solving heuristics can be incorporated in the framework of an overlay student model.

George Polya in How To Solve It describes several problem solving heuristics that could be used by a geometry tutor. Polya established a list of heuristic questions which builds a foundation for interaction between a teacher and a student who is learning to solve problems. Using the explicit questions listed in this book, a teacher can attempt to instill a heuristic method of problem solving in students. In his translation of Pappus, an ancient Greek mathematician, Polya describes the problem solving process which uses heuristics.

> "In analysis, we start from what is required, we take it for granted, and we draw consequences from it, and consequences form the consequences, till we reach a point that we can use as starting point in synthesis. For in analysis we assume what is required to be done as already done (what is sought is already found, what we have to prove as true). We inquire from what antecedent the desired result could be derived; then we inquire again ... This procedure we call analysis, or solution backwards.

"But in synthesis,... we start from the point which we reached last of all in the analysis, from the thing already known or admittedly true. We derive from it what preceded it in the analysis, and go on making derivations .. we finally succeed in arriving at what is required. (Polya, 1945 / p. 142)

Polya's rendering of Pappus is an ancient description of what computer scientists now call backward chaining (analysis) and forward chaining (synthesis). It points out a very powerful problem solving technique that good problem solvers use all of the time. The questions used to drive Polya's method could become a part of the student model that records the strengths and weaknesses of a user's problem solving ability while progressing through an interactive session.

Polya's method also describes a method that can be used by the expert module of an ITS for human like problem solving. Anderson's tutorial appears to be doing this with ACT.

2.5. ACT (Problem Solving Mind Model)

Anderson's mind model, ACT, is based on production rules and their application to appropriate situations. These production rules, which are stored in a declarative representation, must undergo a transformation that Anderson calls "knowledge compilation" before they can be applied. As Anderson describes it:

> Knowledge compilation is the process by which subjects go from the declarative representation of a skill to a procedural representation. The declarative representation is applied to the task by means of general interpretive productions. ... After achieving a procedural form, in contrast, the knowledge applies directly because it is encoded in production form. (Anderson, 1983 / p. 202)

Anderson claims that the ACT model simulates processes used by students doing inferences in a geometry proof. This includes a simulation of the human tendency to compose several productions into a single production. However, this is not the whole of Anderson's model. He also includes the elements of heuristic search.

> Having operators proceduralized is not enough to guarantee successful proof generation. There is still a potentially very large search space of forward and backward inferences. Finding the proof tree in this net would often be infeasible without some search heuristics that cause the system to try the right inferences first. (Anderson, 1983 / p. 209)

Anderson uses heuristics extensively in the development of his ACT model. In his description of heuristic search he points out that "at the general level, expertise does not develop by becoming more restrictive in search, rather it develops by becoming more appropriately restrictive." He recognizes several heuristics used by problem solvers including analogy, generalization, discrimination, and composition. (Anderson, 1983 / p. 209)

ACT has been implemented to solve geometry problems of a level found in high school textbooks. It appears able to build a proof tree using backward and forward chaining. However, its use of other heuristics and the extent of the implementation appears somewhat unclear. Regardless, ACT does form a framework for the development of a geometry tutorial's expert module.

2.6. Learning by Doing (The Open System)

The work of Jean Piaget and Seymour Papert has greatly influenced computer education. In learning situations, they have found "evidence that (a) child's activity is the key -- learning must take place by 'doing'" (Barr/ 1982 / p. 291) This has led to the idea of the open system, an environment that allows unstructured learning.

Papert, the most vocal advocate of unstructured learning, promotes a learning environment of "'Piagetian learning' or learning without being taught" (Papert, 1980 / p. 7). By "Piagetian leaning" he means:

> the natural spontaneous learning of people in interaction with their environment, .. contrasted with the curriculum-driven learning characteristic of traditional schools. (Papert, 1980 / p. 156)

The computer language, Logo, is Papert's implementation of this environment. Papert's theory of cognition is based on Piaget's "notion of assimilation". In Mindstorms, Papert observes:

> Anything is easy if you can assimilate it to your collection of models. ... The understanding of learning must be genetic. It must refer to the genesis of knowledge. What an individual can learn, and how he learns it, depends on what models he has available. (Papert, 1980 / p. ii)

These ideas are very powerful. Papert has many followers and Logo has spawned a generation of educational systems attempting to be open. As pointed out by the developers of GUIDON-WATCH:

> Papert offers a provocative view of AI and computers in education; he influenced us to consider how we can provide students with conceptual software tools to explore computational models. (Richer, 1985 / p. 61)

Papert bases his model on the creation of "body syntonic" and "ego syntonic" objects. The turtle, a "computer-controlled cybernetic animal" (Papert, 1980) found in the Logo computing environment, is "body syntonic" in the sense that it can be related "to children's sense and knowledge about their own bodies." Also it is "ego syntonic in that it is coherent with children's sense of themselves as people with intentions, goals. ... Turtle geometry is learnable because it is syntonic" (Papert, 1980 / p. 63). The trees established in both Anderson's tutorial and GUIDON-WATCH might be viewed as ego syntonic objects. Use of these objects is a very powerful means of teaching the concepts in those systems.

Papert's "conjecture is that the computer can concretize (and personalize) the formal" (Papert, 1980 / p. 21). Optimistically he perceives an environment where children become epistemologists via the process of computer programming. He sees the turtle and LOGO "as a vehicle for Piagetian learning, which (to him) is learning without curriculum" (Papert, 1980) p. 31).

AI developers of ITS recognize the importance of Papert's position and attempt to incorporate many of his ideas into their systems. However Papert holds an extreme position in education and most ITS developers temper his perspective. They feel there are many other things that AI can contribute to education besides providing a strictly open environment. As Barr and Feigenbaum point out: "A tutor is a learning resource, and 'hybrids' between (ITS) ideas and learning-by-doing systems are possible." (Barr, 1982 / p. 293)

The implementation of experts in intelligent systems such as WEST, GUIDON, and Anderson's geometry tutorial allow educational systems which are much more open than traditional CAI systems. As simulators of human tutors the experts of these systems achieve flexibility in their presentation of material and responsiveness to student needs.

2.7. Geometry Education Today

Traditionally geometry has had a place in the curriculum because it is an excellent environment to learn skills of reasoning and logic. As Polya points out in *How To Solve It* :

> If the student failed to get acquainted with this or that particular geometric fact, he did not miss so much; he may have little use for such facts in later life. But if he failed to get acquainted with geometric proofs, he missed the best and simplest examples of true evidence and missed the best opportunity to acquire the idea of strict reasoning. Without this idea, he lacks a true

> standard with which to compare alleged evidence of all sorts aimed at him in modern life. (Polya, 1945 / p. 216)

For this reason proof in geometry is still considered a very important part of the high school curriculum. An ITS for geometry should develop skills that promote the learning of proof. It should be rigid enough to direct users toward learning needed skills, but open enough to allow them the freedom necessary to develop confidence in the mathematical structures they build in their minds.

CHAPTER 3

PROTOTYPE DEVELOPMENT

The prototype created in this project is a user interface for an intelligent geometry tutorial. The system it anticipates would allow students to learn construction of simple proofs in geometry and help them develop skills necessary to write those proofs. The projected system would be an expert system similar to those described in chapter 2. This prototype, while not an expert system, is attempting to lay groundwork for an expert geometry tutorial. In the development of an expert system:

> You cannot interrogate the experts, then go off and code up what they said. Instead, you must enter into a longer-term relationship, in which you keep coming back to them for criticism and extensions of the program, until it begins to solve a significant fraction of problems in the domain. A program of this kind must be open-ended; it is never finished, and must be easy to modify at all times. (Charniak, 1983 / p. 438)

In order to implement this kind of program, the development method known as "rapid prototyping" was used. Rapid prototyping

> means building or designing with a new technology or for a new application while the feasibility of the design remains in question. (it is) basically a feasibility study that serves to demonstrate system aspects critical to the user. (Ramamoorthy, 1984 / p. 193)

Also described as "incremental development", rapid prototyping builds a system from the bottom up by creating a visible coded system. This allows requirements to be discovered in the process of development. The method contrasts with traditional Software Engineering methods which follow lengthy steps of analysis and design before implementation in actual code.

This paper describes three iterations through the development cycle of the rapid prototype. In the first iteration, a simple interface was developed and criticized. Int the second iteration, an analysis was made of the existing system and priorities were made of feasible enhancements to the system. Using incremental development, some of these enhancements were constructed and a second evaluation was held which included reactions from high school students. The third iteration begins with the fourth chapter of this paper in a top down approach to a second prototype.

3.1. Initial Analysis

The initial purpose of this prototype is to establish an interactive dialogue between system developers at the university and our counterparts in the high schools. Those high school counterparts include geometry teachers who are the experts and high school geometry students who are the users. By establishing this dialogue, the process of accumulating necessary knowledge for an expert geometry tutorial can proceed.

A dialogue was established with two teachers and two students at a Missoula high school, an urban setting of about 50,000 people, to develop the first prototype. Several meetings were held with one of the teachers in the high school and about 30 tutorial sessions were conducted with the students to get a feeling for the process of learning geometry.

3.2. The van Hiele Model for Geometry Learning

Geometry education at the school is in a state of controversy. Traditionally, proof has been introduced early in a geometry course. For example the text used was Geometry published by Scott Foresman and Company. which immediately introduces formal proof. (Hirsch, 1984) In contrast to this, the two teachers involved with this development spent the first semester teaching informal geometry. For one semester, the students informally applied several rules used in geometry and did no proofs until the second semester. These teachers are beginning to implement a change in geometry education which is best understood by looking at a learning model developed by the Dutch couple, Pierre van Hiele and Dina van Hiele-Geldof (Shaughnessey, 1985). It describes levels of geometry learning. The five van Hiele levels of geometry learning are described by Shaughnessey and Burger in The Mathematics Teacher where they number them level 0 to level 4.

Level 0 students according to the van Hiele model have a simple visual understanding of geometry. For example when asked why a picture is a rectangle, a student at this level will give reasons like: "Because it looks like a rectangle".

Level 1 students are able to analyze objects and think of them as a collection of properties. When asked the above question about a rectangle, a student at this level would list properties: "Opposite sides are parallel, opposite sides are congruent, opposite angles are equal .. etc."

Level 2 students are able to do are able to do informal deduction. They can select sufficient conditions of objects such as rectangles. They understand definitions and can make informal inferences. Students at this level of understanding can infer things like: "squares are rectangles".

Level 3 students are able to exercise formal deduction. With skills needed for this level, students understand the role of axioms and theorems and they can construct proofs. Problems like proving that the diagonals of a rectangle have equal length since two triangles are congruent is possible at this level.

Level 4 students can make comparisons between different axiomatic systems. For example level 4 students can consider questions like: "What happens to geometry if we do not assume the parallel postulate". This level of rigor is rarely found in high school students. (Shaughnessy, 1985)

Shaughnessy and Burger point out that most geometry classes in the United States are taught at level three while most students are reasoning at level one (Shaughnessy, 1985). Senk in another study makes similar observations (Senk, 1985).

Hoffer also puts the American geometry experience in the light of the van Hiele model. He feels that more attention should be given to non-proof skills such as the verbal, drawing and modeling skills which can be learned in a geometry class (Hoffer, 1981).

The consensus seems to be that many informal skills must be developed before proof is introduced. As Hoffer puts it:

> By beginning formal proofs too early in a geometry course, we may not account for those students who have not yet reached a sufficiently high level of mental development to enable them to function adequately at the formal level. (Hoffer, 1981 / p. 14)

One of the recommendations made by Shaughnessy and Burger was for educators to "develop activities that will move students through the (van Hiele) levels" (Shaughnessy, 1985 / p. 426). By considering the van Hiele model, bottlenecks in geometry learning can be made explicit and addressed.

Using the van Hiele model as a reference, development of this prototype attempts to address problems encountered by students in their transition from level 2, informal deduction, to level 3, formal deduction. Specific differences in these two skill levels are pointed out by Hoffer in the <u>Mathematics Teacher.</u>

> Level 2 students have the visual ability to "recognize interelationships between different types of figures" and "recognize common properties of different types of figures". The level 3 student "uses information about a figure to deduce information."
>
> Verbal skills known by level 2 students include the ability to "formulate sentences showing interrelationships between figures". The level 3 student can "recognize what is given in a problem and what is required to find or do".
>
> At level 2, students' logical skills include the ability to "understand qualities of a good definition" while a level 3 student "uses rules of logic to develop proofs" and "is able to deduce consequences from given information". (Hoffer, 1981 / p. 15)

The present prototype system builds toward an environment that accommodates students possessing the level 2 skills described above. The geometry exercise implemented within this system is a problem that tries to help a user achieve level 3 skills.

3.3. Preliminary Implementation

The prototype was developed on a Macintosh under Exper-LISP and the VAX-750 under Franz LISP. The current system runs on a Macintosh with 512 kilobytes. The

Quickdraw Library which is built into the Macintosh was used extensively to provide the graphic interface.

Early development concentrated on establishing windowed Input/Output (I/O) and use of the menu facility of Quickdraw. Many of the low level Input/Output routines such as text input, text scrolling and character rubout had to be written with LISP functions. Exper-LISP presented problems because it does not allow an application developer access to the event queue. This made it difficult for the system to accept either text input or mouse input at the same time. Initially the system utilized three windows, one for the drawing and problem description, another for proof construction, and a blackboard for user interaction. It also gave the user access to one help menu.

After a limited I/O was established, a simple parser was built to allow text input by the user. It accepts keyboard input of geometry assertions, but recognizes only a subset of the assertions used in geometry proofs. The parser uses a simple set of four parsing rules (see

Appendix B) and does limited error response. It was developed on the VAX and ported to the Macintosh.

Tutoring sessions with the students revealed a few basic requirements for the tutor. The early prototype system included code that attempts to address the following requirements.

* Help on the syntax of geometry assertions must be available.
* Explicit statements of rules needed to solve the problems and illustrations to clarify their meaning must be available.
* The problem statement and drawing must always be visible.
* Students who jumped steps in the proof should be notified of this fact.

A simple problem interaction was coded into the system to experiment on the environment. Given that B is the midpoint of segment AC, the problem requires the user to prove that segment AB is congruent to segment BC (Hirsch, 1984 / p. 65). To solve this problem, a user must write three geometric assertions and give reasons for those assertions. The reasons are input by a choice from a list of numbered rules.

The interface is an open system which allows the user to write an assertion in the proof. When the user is incorrect the system indicates a mistake. If the user makes a correct assertion, the system asks for the reason. As the user progresses, the system writes the proof in the proof window until the problem is proved.

In terms of van Hiele level 2 skills as described by Hoffer above, a student working the problem must be able to: 1) recognize relationships within a provided drawing of a midpoint on a segment. 2) formulate sentences showing the relationships of equal length and congruence, and 3) understand how to apply the definitions of midpoint and congruent segments. The level 3 skills which the lesson hopes to encourage include: 1) the inference of knowledge about equal segment length and segment congruence from the figure provided, 2) exposure to a problem with given information and consequents to be proved, 3) an exercise in logical chaining with more than one inference.

A primitive help facility was established. It shows syntax needed to write assertions and gives explicit examples of definitions that can be applied in the proof.

With a running program, prototyping of the geometry tutorial began. Since the system is quite small and must grow to become effective, only those improvements that can be considered within the constraints on the prototype are discussed in this chapter. Comparison of the prototype with a complete ITS will be made in Chapter 4.

3.4. Initial Criticism

At this stage in the development, the system was viewed and criticized by university faculty members and one of the high school teachers. This simple system was found to be weak in its interaction with the user. It allowed a user to solve the problem correctly but it failed to prompt and respond effectively. There were too many windows and the help was hard to use. System response to user errors was minimal and often uninformative.

After this initial round of criticism, the system was viewed in terms of five feasible problem areas that could be worked on.

1. The parser: Instances of error response could be handled by more extensive work on the parser.
2. The screen: Window management was a difficult problem due to the small size of the screen and use of different windows for different interactions.
3. Verbal responses: There are problems of how the system communicates with the user and what it should say in different circumstances.
4. Tutoring: The system needs the ability to guide a student through the problem, show him examples and point out heuristic rules that will help him solve the problem.
5. Help: Help must be easy to access and automatically provided when needed.

3.5. Incremental Extensions to the Prototype

The first round of criticism completed the first iteration in the prototyping cycle. A second iteration of the cycle began with an analysis of the criticisms of the first cycle and design of feasible enhancements that would expand the system.

A complete correction of the above problems was beyond the scope of this project. The parser for example was hardly touched since it appeared that an extension at this point would provide limited new information about the requirements for a more complete system..

The screen needed some immediate work since it was confusing. The blackboard was cleared at appropriate times and the help window was expanded for clarity.

Attempts were made to clarify verbal responses by the system. Help procedures were created which asked questions of the user and gave him verbal information that might help him solve the problem.

The most extensive improvement was the addition of a tutoring feature which gives automatic help. With this feature the system responds when a user gives an incorrect reason for an assertion. It attempts to show the user why an incorrect reason is not valid. By showing a picture with the incorrect rule instantiated, the system points out why the rule can not be used.

The automatic help is designed to draw on a user's van Hiele level 2 skills. It requires that the student recognize the relationships in the figures in both the help window and proof window. It also requires that the user understand the demonstrated definition. It then points toward the correct reason by mentioning a general heuristic. System response to the mistake will promote the van Hiele level 3 skills needed to deduce information from a figure and to infer consequences from given information.

User selected access to help was simplified slightly but no major changes were made due to lack of the necessary facilities in Exper-LISP. Constraints on both the development time frame and the limited computer memory halted any further extensions to the current system.

3.6. User Reactions

After the first extension was written, the prototype was demonstrated and tried out by 10 users. They included a university professor, two high school geometry teachers, a college student in mathematics education, and seven high school geometry students. They all helped uncover flaws in the interface and provided suggestions for an improved system.

Some of the problems with the interface can be fixed with minor changes. These include:

BUG	REMEDY
*Confusing Menu Bar	Remove development menus and simplify.
*Cluttered Blackboard.	Clear screen more often.
*Where is action?	Provide flashing prompt.
*Rule window is redundant.	Eliminate rule window.
*Hard words. "Assertion" "Syntax"	Change words to "Statement" "Statement form".

*User does not know wrong reason was given.	Tell user reason is wrong.
*Proof window too far from blackboard.	Have student write assertions in proof window.
*Confusion with meaning of prompt.	Provide simple prompt explanations.

There exist some problems with Exper-LISP and the Macintosh system which must be solved. This could be facilitated with better documentation from ExperTelligence. The bugs include:

BUG	POSSIBLE REMEDY
*Menu crashes.	Access to Quickdraw event queue.
*More than one method to ask help.	Create single help method using mouse.
*Students do not use help.	Simplify user requested help.

There are potential improvements to the system in modifications to existing modules. These include changes to screen management functions and the parser.

PROBLEM	POTENTIAL IMPROVEMENT
*Parser crash on unrecognized assertions.	Strengthen parser with error recovery routines and error messages.
*Space limited. Screen busy and hard to read.	Use fewer windows(3), overlapping windows, 10 point type, or use a computer with a larger higher resolution screen.
*Students don't know difference between equal and congruent.	Call help routines from the parser.
*Student difficulty in writing assertions.	Provide automatic help on syntax from the parser.

Some shortcomings of the interface could be improved by the addition of new modules. These modules include:

PROBLEM	NEW MODULE
*No initial instruction on use of system.	Introduction module.
*Student failure to proceed with proof.	Help modules that create interaction.

*Rules difficult to find and read	Functions to find and scroll rules.
*Too short.	More problem modules.
*Problem too simple.	Interesting problems.

Demonstration of the prototype not only provided suggestions for changing the system but also stimulated an analysis of the educational goals of the tutorial.

3.7. <u>Teaching Objectives and Methods</u>

Prototype demonstration revealed different objectives and methods teachers deal with in teaching geometry. It was an opportunity to learn from teachers what they would do, if they could program such a tutorial system.

Development of the prototype system required a detailed analysis of the geometry problem to be solved by the student. Several observers felt that the main geometry lesson to be learned from this problem was the misconception many students have about the difference between congruence and equal distance. Known misconceptions such as this and others uncovered as a result of implementation should be exploited by the tutor.

Educational methods suggested by observers included: active marking of congruent segments as the proof progressed, activity by the student to choose which part of an if and only if rule to apply, and techniques to teach the student the skill of applying a rule.

The prototype let observers point out places in the proof process where problem solving heuristics could be taught. Suggestions were given as to which heuristic rules to teach, when to teach them, and how to present them.

Finally the prototype includes specific places in the code that a ITS student model could be used. One place to build on the student model is the condition arising when a student writes an incorrect assertion. A good example of this is when a user calls line segments equal rather than congruent. Another time is the condition when a student chooses the wrong rule as a reason.

CHAPTER 4

FURTHER DEVELOPMENT

The objective of this project was to initiate development of an ITS for high school geometry. The method used involved bottom up incremental development of a user interface while maintaining a continuous interaction with geometry teachers and high school geometry students. To limit the scope of the project, it was decided to concentrate on a transition that is needed by these students between skill levels and is described by a geometry learning model known as the van Hiele model. The prototype attempts to help students possessing skills of informal deduction progress to a level of formal deduction. Transformation between these two van Hiele skill levels could well form the educational foundation for another prototype.

This chapter will consider constraints on the development of a second prototype and present alternatives for dealing with those constraints. It will consider the results of the first prototype and begin another iteration in the prototyping cycle. It will also describe areas of potential system development some of which were neglected by the first prototype.

4.1. Human Constraints

Development of an effective ITS for geometry would require several man-years of work. An individual could continue a study of the requirements for such a project, but substantial development would require a team of programmer-analysts working with educators and students.

Expert geometry teachers are needed to provide expertise for establishing the student model and the tutor modules. Finding an expert with the time to spend is often one of the most difficult problems in developing an expert system. High school geometry students must also become involved since they are the ultimate users. Involving these students can be difficult since they must see an immediate and obvious benefit before they participate.

4.2. System Constraints and Alternatives

The first prototype was developed on the Macintosh computer using Exper-LISP. The Macintosh offered portability, due to its size, and was transportable to the high school for demonstrations and feedback. Exper-LISP provided the project with a potential AI language, a necessity in the expert stage of development.

However, the 512 kilobyte Macintosh with Exper-LISP offers a very poor development environment. Exper-LISP needs much improvement in its access to Quickdraw, and the Macintosh has limitations due to its small screen size and slow speed. The memory requirements and complexity of a comprehensive ITS make development on the small system provided by the Macintosh and Exper-LISP difficult.

A much better development environment would exist using a terminal emulator for the Macintosh that had extensive access to Quickdraw. With such an emulator, the Macintosh could act as a smart graphics terminal for a mainframe or minicomputer. This would allow for an environment that included a VAX computer, the Unix operating system, Franz LISP, and Macintosh graphics. If written properly, a system developed on the VAX could be ported to the Macintosh and use the Exper-LISP compiler to create a self-contained system on the Macintosh. In the development of the first prototype, the parser was easily ported from the VAX 750 to the Macintosh. Alternatively, the emulator itself could be used in the schools. An emulator like this is currently being developed at the University of Montana.

LISP machines such as the Xerox 1186 mentioned earlier, or the Sun-3/160M would be ideal environments to develop an ITS for geometry. For example features of the Sun machine include: a screen with four times the screen resolution of a Macintosh, 2 Mbytes of memory, 256 Mbytes of virtual memory, a Unix programming environment, and Common LISP with full graphics capabilities (Sun, 1986). A machine similar to these, the Xerox Dandetiger, is being used to develop and test Anderson's geometry tutorial (Anderson, 1985). The environment of a LISP machine would be much better than the Macintosh used for the first prototype. With a software environment such as ART (Automated Reasoning Tool), which is a shell specifically designed for the development of expert systems, very effective prototyping of an ITS could proceed. ART provides system tools such as an inference engine that does both forward and backward chaining on a knowledge base of rules and facts (Inference, 1985). However, the cost of LISP machines and expert system shells is quite expensive.

The development of a second prototype could be undertaken using the Macintosh and Exper-LISP. The present tutorial system is reaching the limits of 512 kilobytes, but the Macintosh is potentially expandable using commercially available tools such as a hard disk and memory expansion modules. Also ExperTelligence has been improving the Exper-LISP environment. They are currently developing a more sophisticated object-oriented environment which might be helpful in the development of an ITS. (Ritz, 1985 / p. 12). The description of a second prototype which follows will assume further development with an upgraded Macintosh system since they appear to be the best tools locally available.

4.3. The Second Prototype

The structure of the first prototype can be used to begin top-down development of a second prototype (see Appendix C). Functions of the existing system could be reorganized and incorporated as increments of a second prototype. The first prototype could provide I/O primitives (Appendix E), discourse procedures (Appendix E), and perhaps some data structures (Appendix D) to be used for development of a production system for the tutor module.

To simplify further development of an ITS for geometry, the system has been factored into modules. A complete ITS needs to address each of these modules but substantial effort would be required to implement any one of them.

4.3.1. Improve Input/Output

The parser in the existing system is a minimal implementation of four parsing rules (see Appendix B). It parses the assertions necessary for the lesson in the first prototype but does not adequately handle errors. It must be extended to recognize more assertions and respond better to errors. Enhanced error handling should include access to help procedures of the tutor module.

If access to the event queue of Quickdraw can be established, system control of I/O should be restructured to accept both mouse and keyboard input. Alternatively, mouse interaction that does not access Quickdraw should be implemented in new LISP utility functions. In future prototypes, the system should not return control to ExperLISP as it does in the first prototype. (see Appendix D)

A system where keyboard input was used only to input geometric assertions would be sensible. Such a system would allow all other user inputs with a mouse. For example a rule choice given as a reason for an assertion could be made from a scrolling rule list.

Window management should be treated as a separate submodule of I/O. One way to create a more useful and simplified screen is to use only three windows; one for the problem, one for the proof, and a help window. Reasons could be scrolled through the proof window and selected with a mouse. Alternatively, due to the small screen of the Macintosh, windows could overlap and be activated as needed using the mouse.

4.3.2. Implement a Domain Expert as Part of the System

The domain expert of the existing system is hard coded for the sample problem. However, an ITS for geometry needs a domain expert with the ability to construct proofs for given problems and check the truth of assertions made by users. To solve the simple problems used in the present implementation, an expert as sophisticated as ACT would not be necessary.

To begin implementation of the domain expert, a inference engine to solve geometry problems needs to be constructed. A theorem prover such as Gelernter's "Geometry Machine" (Bundy, 1983 / p. 134) could exist as a separate module more independent than the parser. With the theorem prover and a geometry knowledge base, the tutorial could ask the domain expert about the correctness of assertions and for proofs of simple theorems.

4.3.3. Design a Student Model

A student model could be implemented as a data object in the system. Geometric and problem solving subskills must be determined by talks with teachers and work with students. Demonstration of the prototype versions can serve to uncover subskills necessary to solve the problems. Once subskills have been determined, an object representing the student can be constructed.

A simple frame-like implementation of a student model could involve the use of property lists in LISP. The property could be the subskill name and the value could be the student's level of competence with that subskill. As a global object, the student model could be updated anywhere in the system where skill competence is determinable. Also the tutor could access this object when it needed information about the students skill history. A more complex student model could be developed using the method of prototyping.

A simple student model would begin to organize and define the subskills needed to solve the problems. As a set of subskills was developed, buggy rules could be written into the knowledge base and experimented with using the inference engine to find matches for incorrect assertions. Information about the student determined by use of the buggy rules could be used to update the student model. The student model could be used to heuristically direct the system on which buggy rules to use for generation of incorrect assertions. By having both methods of determining student misconceptions in the same system, other ways might be found where the two methods would complement each other.

4.3.4. Upgrade the Tutor and the User Interface

High level control of the system rests with the tutor module. A major problem encountered by the existing procedural system involves the different modes of interaction that exist in the process of solving a geometry problem. For example, an assertion by a student related to the proof must be handled by the system differently than an assertion by a student during a help session. To handle this problem, there must be different functions to evaluate the assertion corresponding to the different situations the user is in.

In the first prototype the system is procedurally controlled by various functions and global data structures. In a complete ITS, the system might be driven by data from student input and a production system similar to that of GUIDON. Using existing data structures that represent the state of the system and a student model, a component called "working memory" could be established. By building two other components, a "production memory" made up of a set of tutoring rules in if-then form, and a "rule interpreter" that applys the the tutoring rules to the working memory (Charniak, 1983 / p. 438), the tutorial could function as a production system.

If a rule interpreter was available with capabilities of matching declarative rules to the state of the system, programming the tutor could change from procedural to declarative prototyping of the system. This would allow the prototyping of a geometry tutor to function at a level of development now practiced by expert system developers. The tutoring system could be extended using knowledge engineering techniques of writing declarative rules that drive the system rather than writing procedural functions in LISP. With rule based control established in the tutor, the encoding of tutoring rules with information gained from expert teachers could establish rapid progress in the prototyping of an ITS for geometry.

Development of AI tools for a rule driven tutoring module might delay implementation of another prototype. Alternatively, it might be beneficial to implement a new procedural control module that did not let the system return to Exper-LISP. This would allow the creation of more discourse procedures for the tutor and further development of other modules in the system such as design of the student model. With an expert system shell such as ART this would not be as big a problem, since a rule interpreter and a means to write tutoring rules could easily be made available.

4.4. Conclusions

The environment needed to develop an effective ITS for geometry does not currently exist on the University of

Montana campus. Software, like the existing expert system shells, that could implement a knowledge based system, and the hardware necessary to run that software using a graphic interface is necessary to create a state-of the-art ITS for geometry.

Development of an ITS on the Macintosh could continue but would require more hardware to extend both primary and secondary memory. Possibly expert system tools from Exper-Telligence such as ExperOps5 or ExperFacts (ExperTelligence, 1986) could be used to develop the expert components of the Domain Expert and Tutor.

Development on the VAX using the Macintosh as a graphic interface would be a preferable alternative. This would require a sophisticated terminal emulator for the Macintosh but would open up several available tools that exist on the VAX. Software tools on the VAX such as MRS, a logic based AI system, and Franz LISP would be available to facilitate development of an ITS. The VAX would also offer higher speeds and larger memories.

Lisp Machines such as the Xerox 1186 or the Sun machine coupled with a shell like ART offer a new dimension to development of an ITS for geometry. Components for controlled screen management, graphic display routines, an inference engine, and other useful AI tools are provided with ART. Configurations possible with these tools could set in motion development of state-of-the art expert systems such as GUIDON-WATCH and Anderson's Geometry Tutorial.

Tools needed for the development of effective ITS systems are now commercially available. Creation of expert systems in those fields which are commercially profitable is progressing at a rapid pace. Probably, an effective ITS for geometry could be made available in the near future. Experiments like the one currently being conducted by Anderson's group, which puts the Geometry Tutor using Dandetiger LISP machines in Pittsburg classrooms (Anderson, 1985), should help establish the effectiveness of these systems with school-children.

The problem of technology transfer may be the biggest hurdle for the implementation and distribution of

Intelligent Tutoring Systems. Computers have been around since the 1940s but they did not appear in classrooms until this decade. Due to the current high costs for expert systems, ITS systems may not reach the classrooms for several years. ITS systems are now becoming technologically feasible but they must also be affordable, which they currently are not. When they exists and are affordable they must be sellable or somehow acceptable to users. Past experience of technology transfer (TI, 1986) shows that many products which are both feasible and affordable are never used because the public does not buy them.

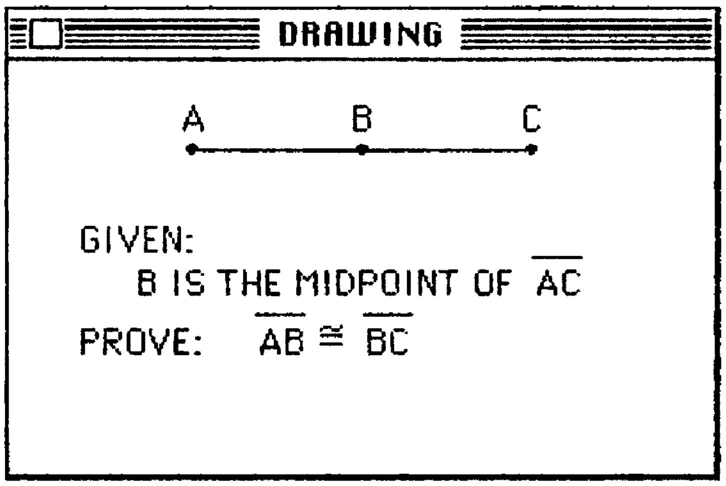
The most important thing that can be learned from the development of this prototype is the need to maintain contact with potential users of the system. Any further development beyond creation of a rule interpreter and parser needs the continual interaction with the geometry teachers and high school students. The experience of tutoring geometry is a good means of developing this necessary relationship.

By creating an ITS that helps the student develop simpler skill levels of geometry, the system has the potential of growing into an effective educational tool. Since the higher skill levels build on the lower skill levels, a system which instructs the higher skill levels could draw knowledge from the system that instructs the lower skill levels.

The method of rapid prototyping can continue in the development of an ITS for geometry. Feedback from demonstrations of new versions can help avoid serious mistakes in the overall system. The exposure of prototype demonstrations in high schools will also help overcome the problems of technology transfer which is encountered by many newly emerging technologies.

APPENDIX A

SCREENS

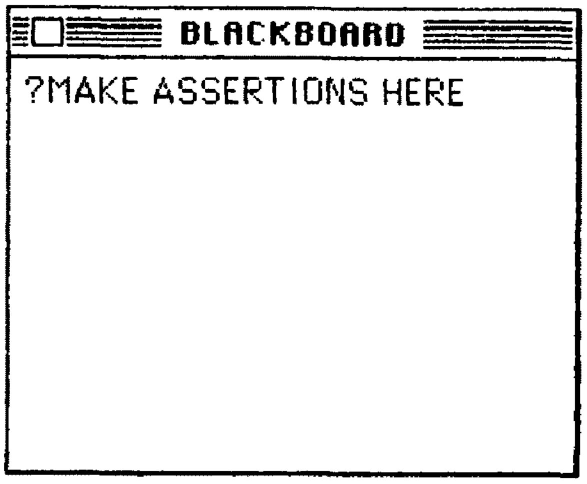


GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$

PROVE: $\overline{AB} \cong \overline{BC}$

DRAWING WINDOW

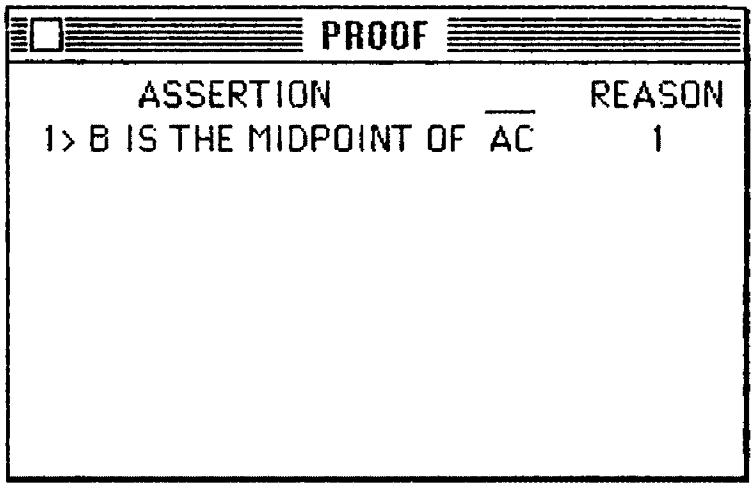
This window is used to show the problem to the user. It has a drawing, the given and the to prove for a proof problem.



?MAKE ASSERTIONS HERE

BLACKBOARD WINDOW

This window is used for all user keyboard input. It is used to make assertions, give reason choices , make selections for help.

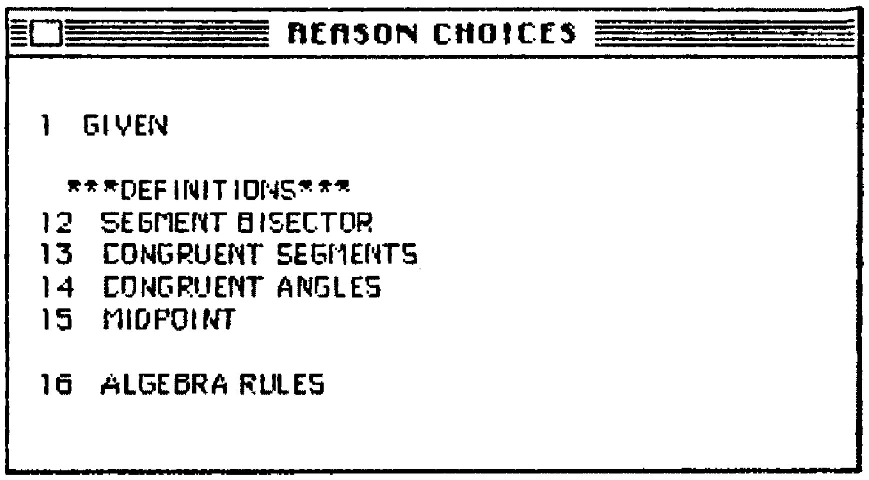

PROOF

ASSERTION REASON

1> B IS THE MIDPOINT OF $\overline{AC}$ 1

PROOF WINDOW

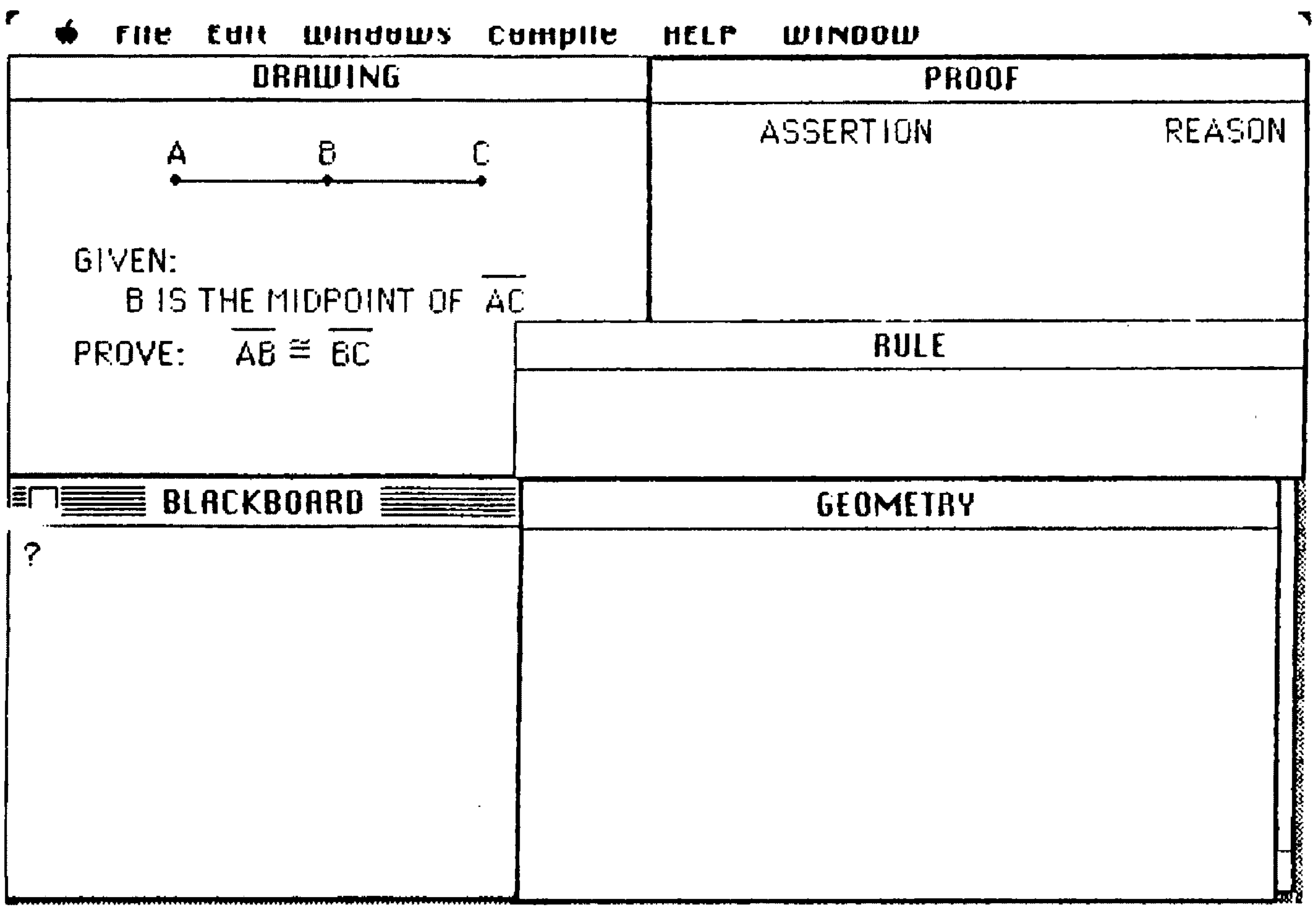
This window is used to record the progress of the proof. When an assertion is correctly made or a correct reason is given, it is written in the proof window.



```
1  GIVEN

  ***DEFINITIONS***
12  SEGMENT BISECTOR
13  CONGRUENT SEGMENTS
14  CONGRUENT ANGLES
15  MIDPOINT

16  ALGEBRA RULES
```

HELP AND REASON WINDOW

This window is used for several messages to the user including: reason choices, rule selection choices, and several kinds of help.



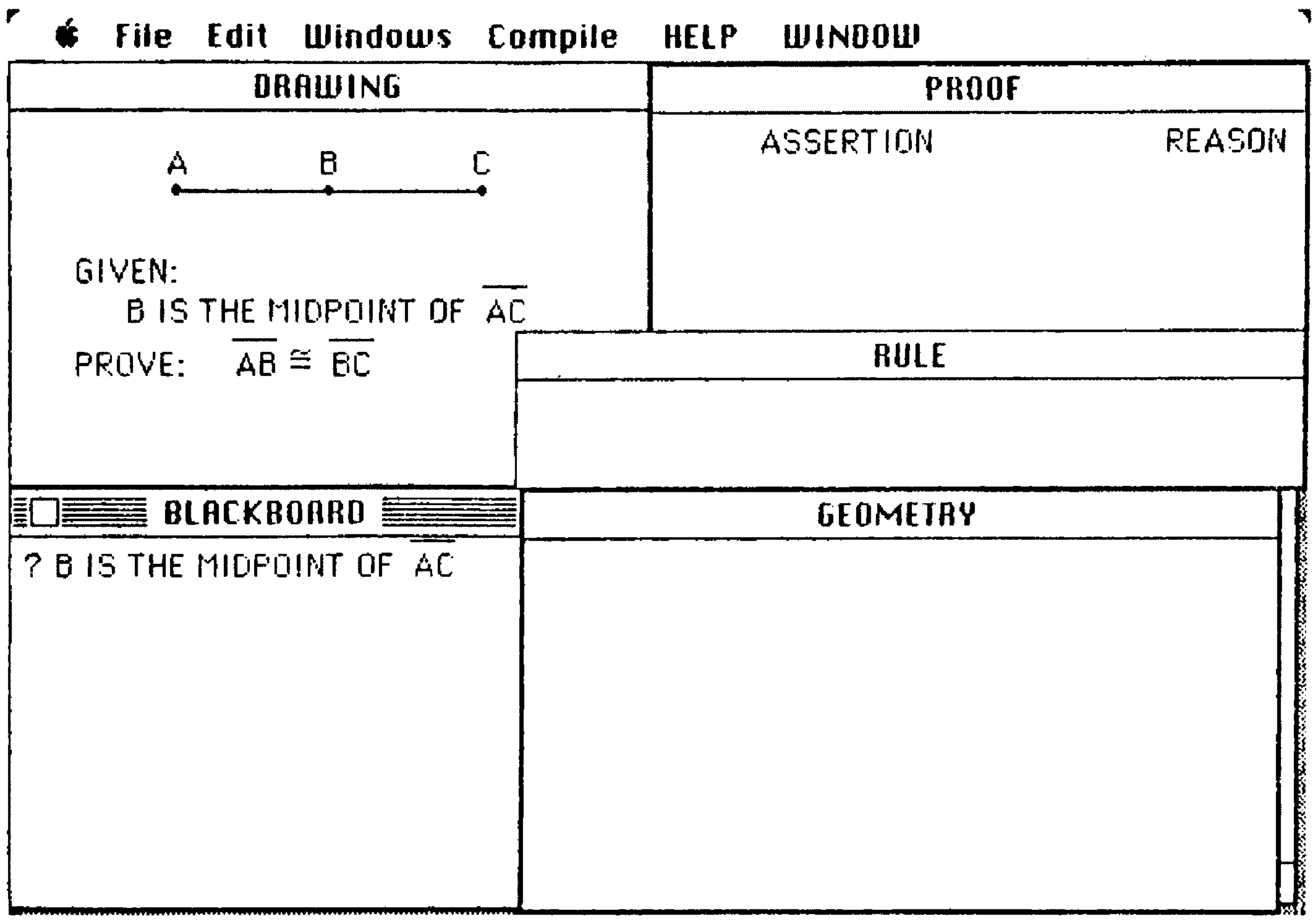
GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$

PROVE: $\overline{AB} \cong \overline{BC}$

?

STARTING SCREEN

This screen is the initial screen shown after try is called from ExperLISP. At this level, the user can type an assertion in the Blackboard or ask for help with the mouse.

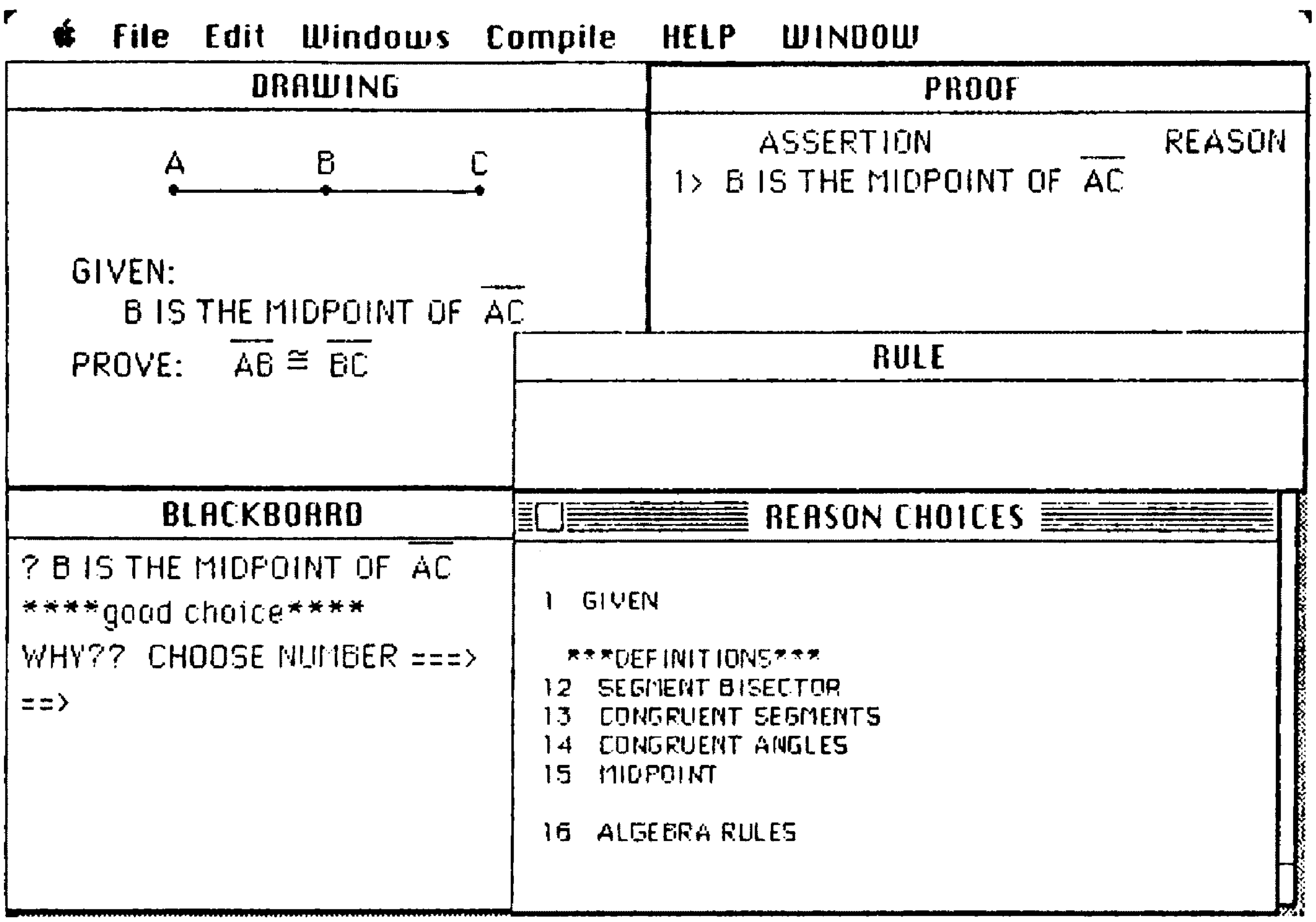


GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$
PROVE: $\overline{AB} \cong \overline{BC}$

? B IS THE MIDPOINT OF $\overline{AC}$

FIRST ASSERTION

This screen shows the first correct assertion made by the user for this problem.



GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$
PROVE: $\overline{AB} \cong \overline{BC}$

ASSERTION REASON
1> B IS THE MIDPOINT OF $\overline{AC}$

? B IS THE MIDPOINT OF $\overline{AC}$
****good choice****
WHY?? CHOOSE NUMBER ===>
==>

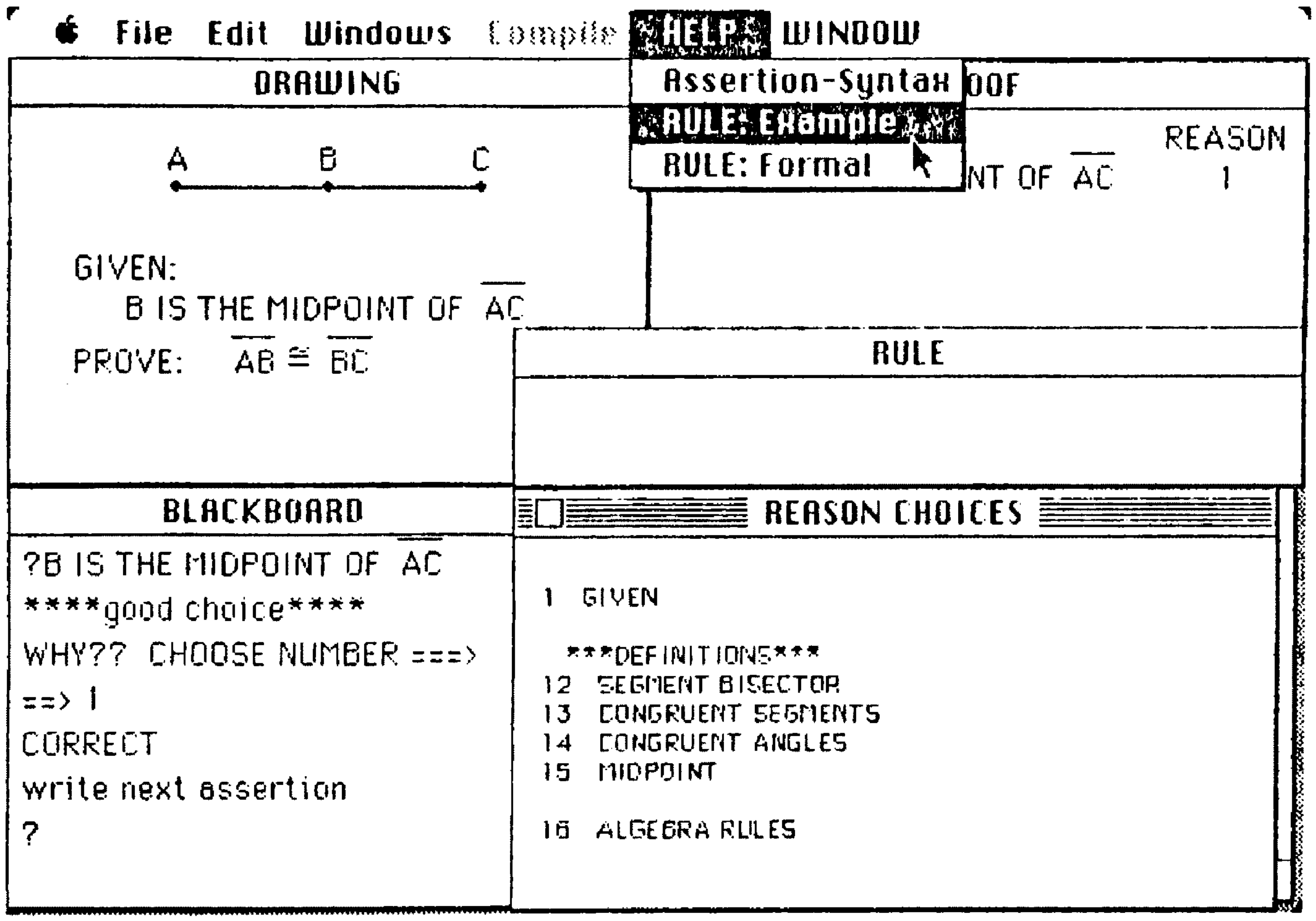
1 GIVEN

DEFINITIONS
12 SEGMENT BISECTOR
13 CONGRUENT SEGMENTS
14 CONGRUENT ANGLES
15 MIDPOINT

16 ALGEBRA RULES

REASON CHOICE

This screen shows the situation after a correct assertion has been made. The user is given choices to make from the keyboard for the reason.



A B C

GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$
PROVE: $\overline{AB} \cong \overline{BC}$

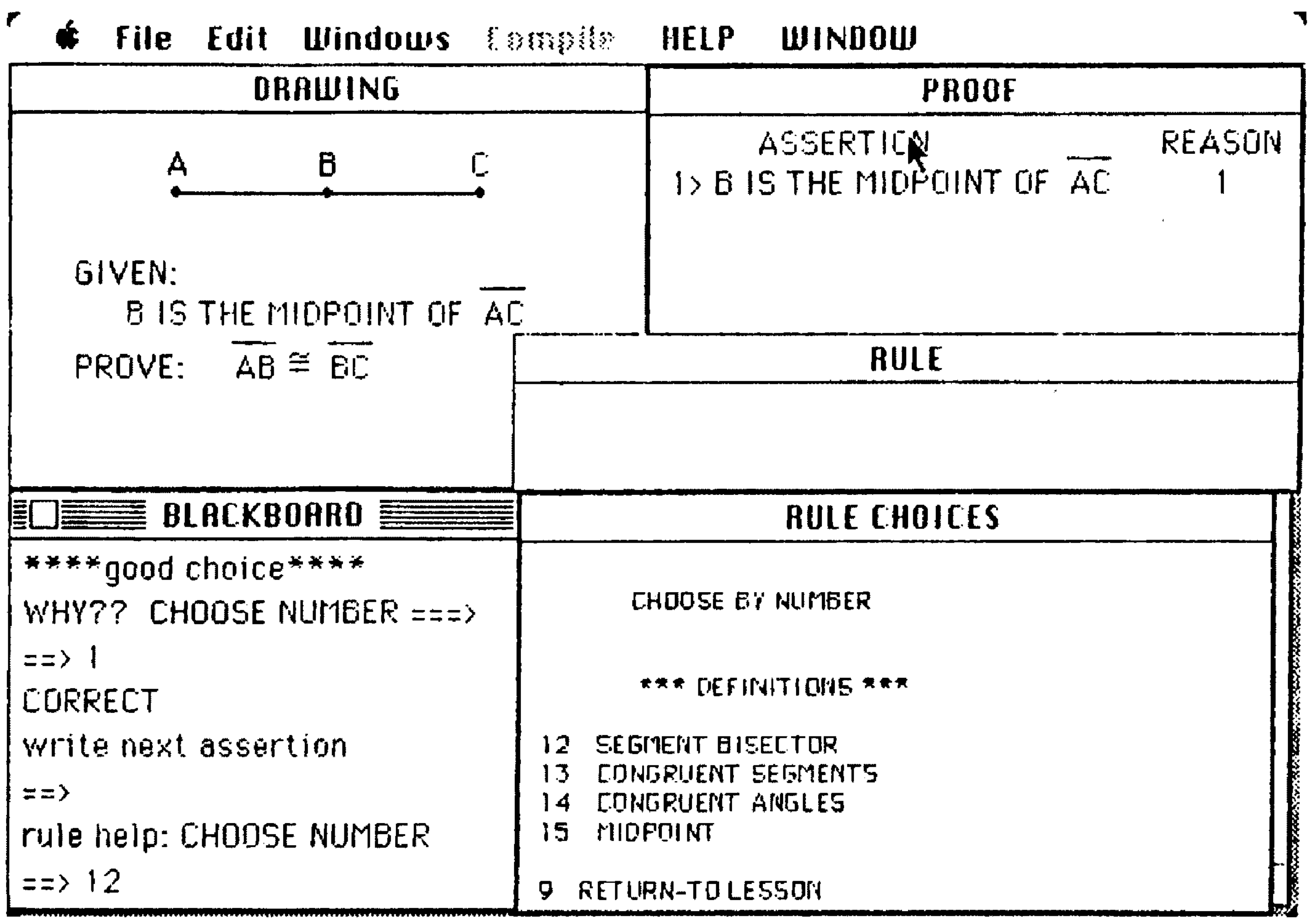
NT OF $\overline{AC}$ 1

```
?B IS THE MIDPOINT OF AC
****good choice****
WHY?? CHOOSE NUMBER ===>
==> 1
CORRECT
write next assertion
?
```

```
1  GIVEN

 ***DEFINITIONS***
12  SEGMENT BISECTOR
13  CONGRUENT SEGMENTS
14  CONGRUENT ANGLES
15  MIDPOINT

16  ALGEBRA RULES
```

CALL FOR HELP

This screen shows a call for help using the mouse. This example calls for help on rules and the user wants to see an example of a rule.



GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$
PROVE: $\overline{AB} \cong \overline{BC}$

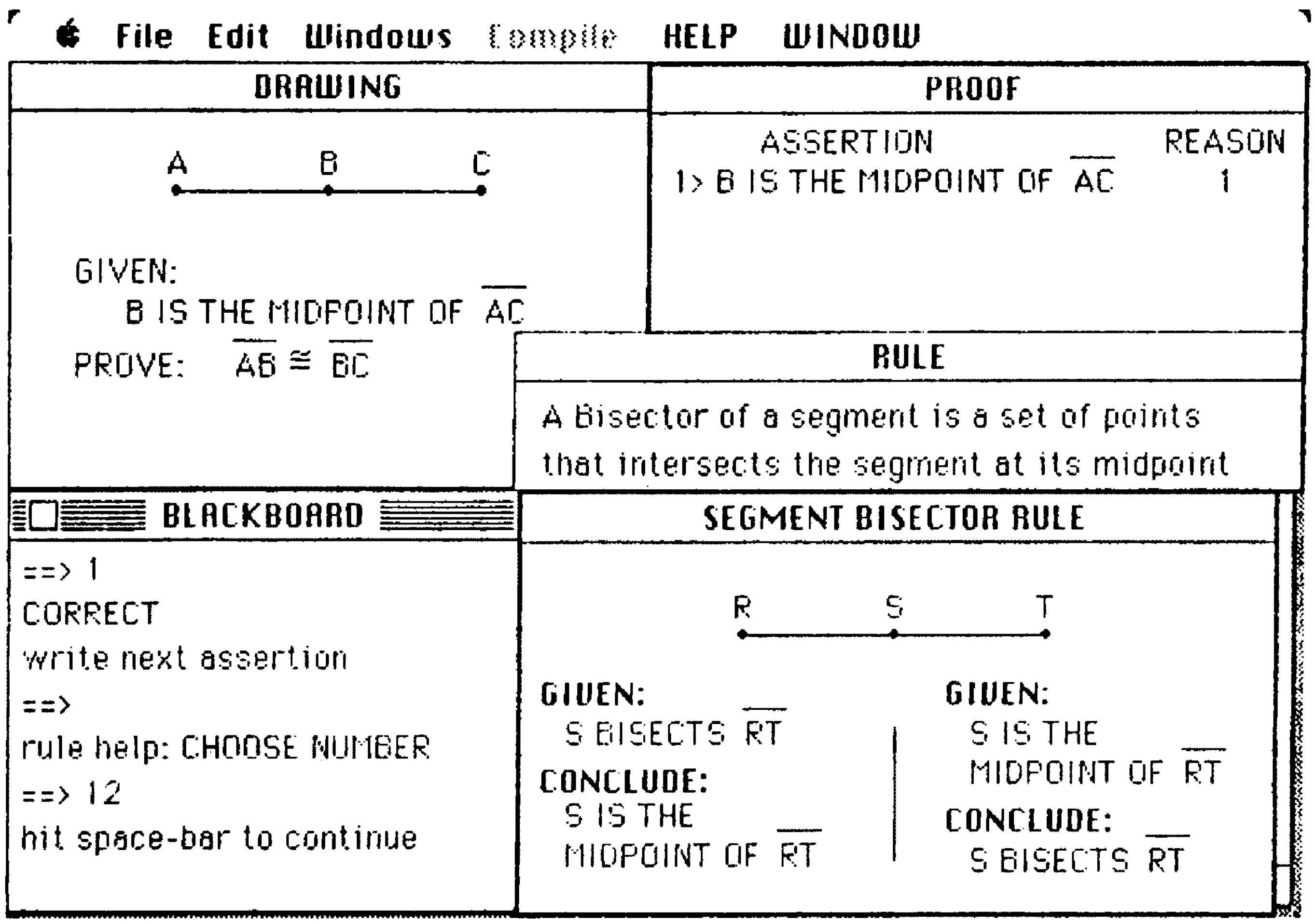
ASSERTION	REASON
1> B IS THE MIDPOINT OF $\overline{AC}$	1

```
****good choice****
WHY?? CHOOSE NUMBER ===>
==> 1
CORRECT
write next assertion
==>
rule help: CHOOSE NUMBER
==> 12
```

```
CHOOSE BY NUMBER

 *** DEFINITIONS ***

12  SEGMENT BISECTOR
13  CONGRUENT SEGMENTS
14  CONGRUENT ANGLES
15  MIDPOINT

9  RETURN-TO LESSON
```

HELP CHOICE

This screen shows the menu after a call for help on rule examples. The user chooses 12 which will give the definition of segment bisector.



GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$
PROVE: $\overline{AB} \cong \overline{BC}$

ASSERTION — REASON
1> B IS THE MIDPOINT OF $\overline{AC}$ — 1

A Bisector of a segment is a set of points that intersects the segment at its midpoint

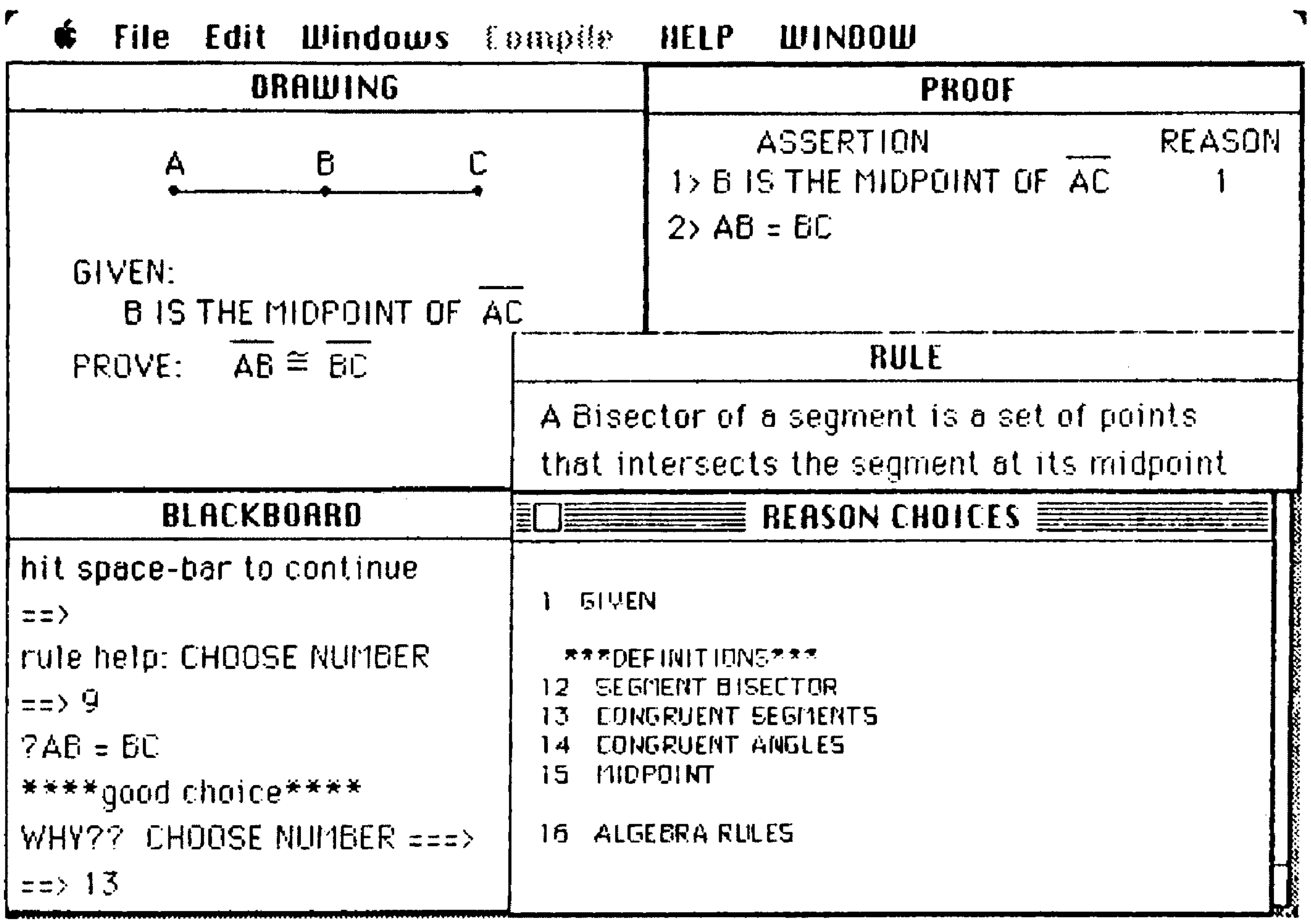
==> 1
CORRECT
write next assertion
==>
rule help: CHOOSE NUMBER
==> 12
hit space-bar to continue

GIVEN:
S BISECTS $\overline{RT}$
CONCLUDE:
S IS THE MIDPOINT OF $\overline{RT}$

GIVEN:
S IS THE MIDPOINT OF $\overline{RT}$
CONCLUDE:
S BISECTS $\overline{RT}$

SEGMENT BISECTOR HELP

This screen gives the user help about the meaning of the segment bisector definition.



GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$
PROVE: $\overline{AB} \cong \overline{BC}$

ASSERTION	REASON
1> B IS THE MIDPOINT OF $\overline{AC}$	1
2> AB = BC	

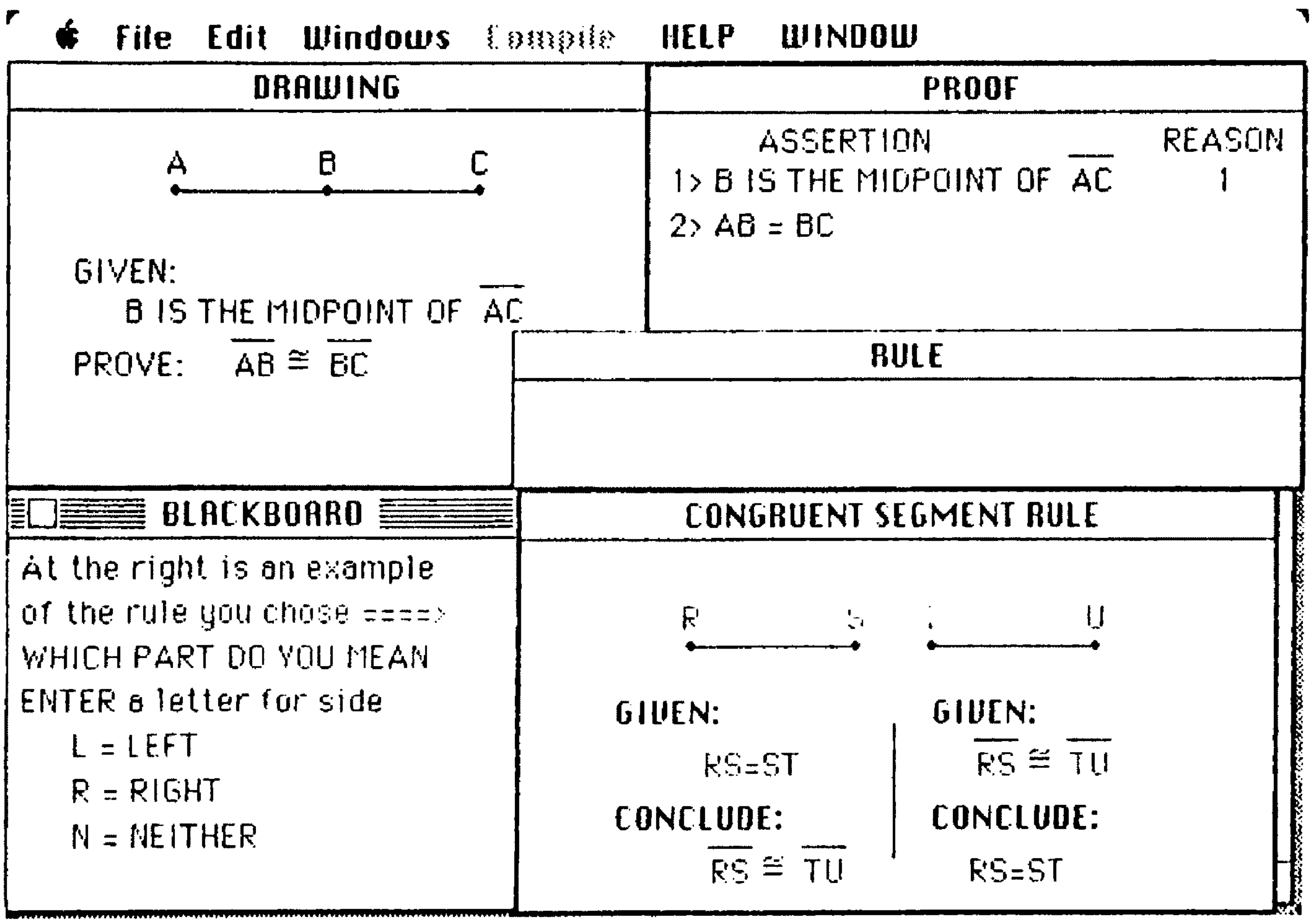
A Bisector of a segment is a set of points that intersects the segment at its midpoint

```
hit space-bar to continue
==>
rule help: CHOOSE NUMBER
==> 9
?AB = BC
****good choice****
WHY?? CHOOSE NUMBER ===>
==> 13
```

```
1  GIVEN

 ***DEFINITIONS***
12  SEGMENT BISECTOR
13  CONGRUENT SEGMENTS
14  CONGRUENT ANGLES
15  MIDPOINT

16  ALGEBRA RULES
```

INCORRECT REASON

This screen shows an incorrect reason choice. It creates a situation for automatic help from the system.



DRAWING

A B C

GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$

PROVE: $\overline{AB} \cong \overline{BC}$

PROOF

ASSERTION	REASON
1> B IS THE MIDPOINT OF $\overline{AC}$	1
2> AB = BC	

RULE

BLACKBOARD

At the right is an example
of the rule you chose ====>
WHICH PART DO YOU MEAN
ENTER a letter for side
L = LEFT
R = RIGHT
N = NEITHER

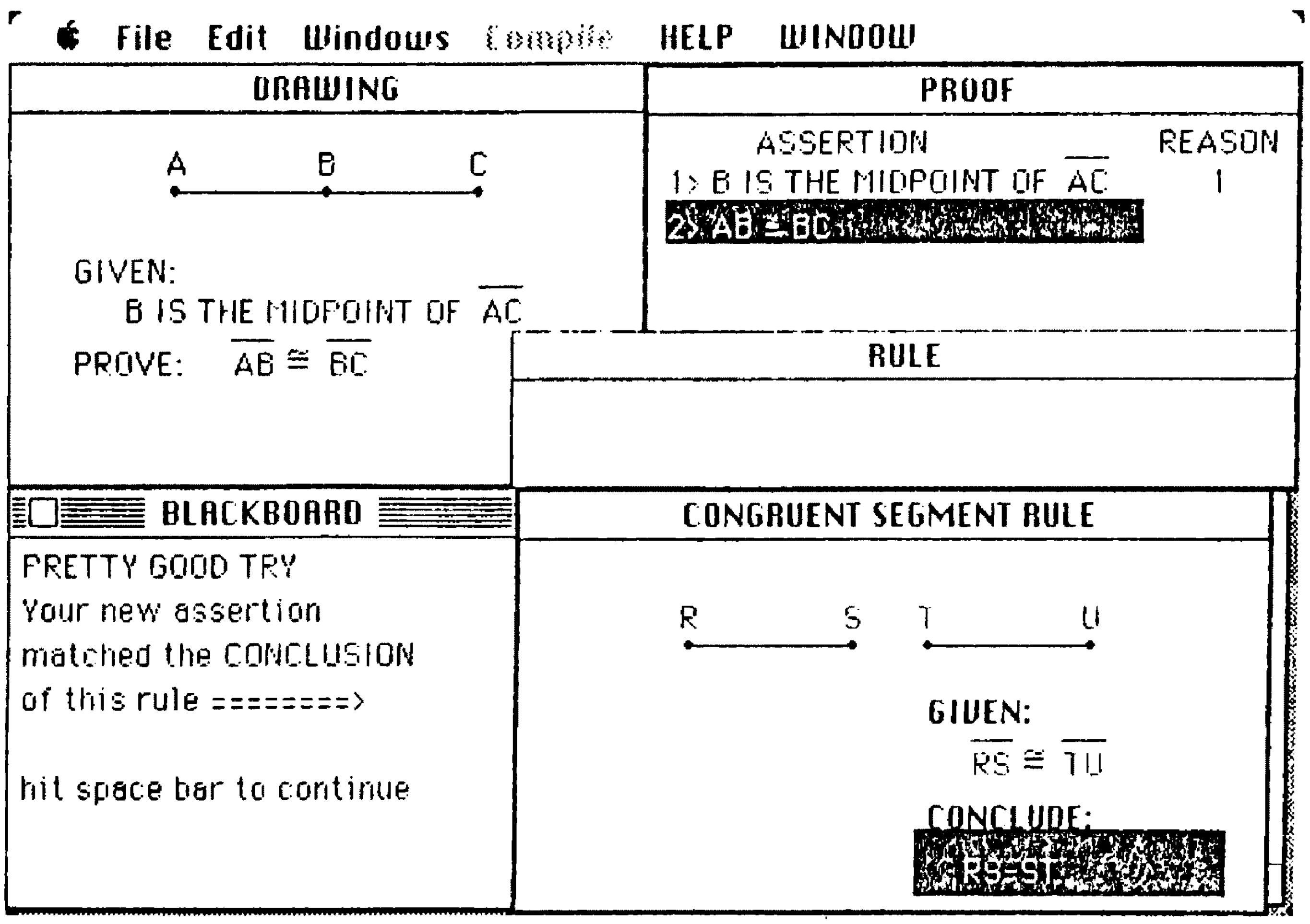
CONGRUENT SEGMENT RULE

R S T U

GIVEN:	GIVEN:
RS=ST	$\overline{RS} \cong \overline{TU}$
CONCLUDE:	CONCLUDE:
$\overline{RS} \cong \overline{TU}$	RS=ST

AUTOMATIC HELP - 1

This screen shows the system's response to the incorrect reason choice of page 83. It asks the user to clarify use of the misused rule.



GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$
PROVE: $\overline{AB} \cong \overline{BC}$

ASSERTION	REASON
1> B IS THE MIDPOINT OF $\overline{AC}$	1
2> $AB \cong BC$	

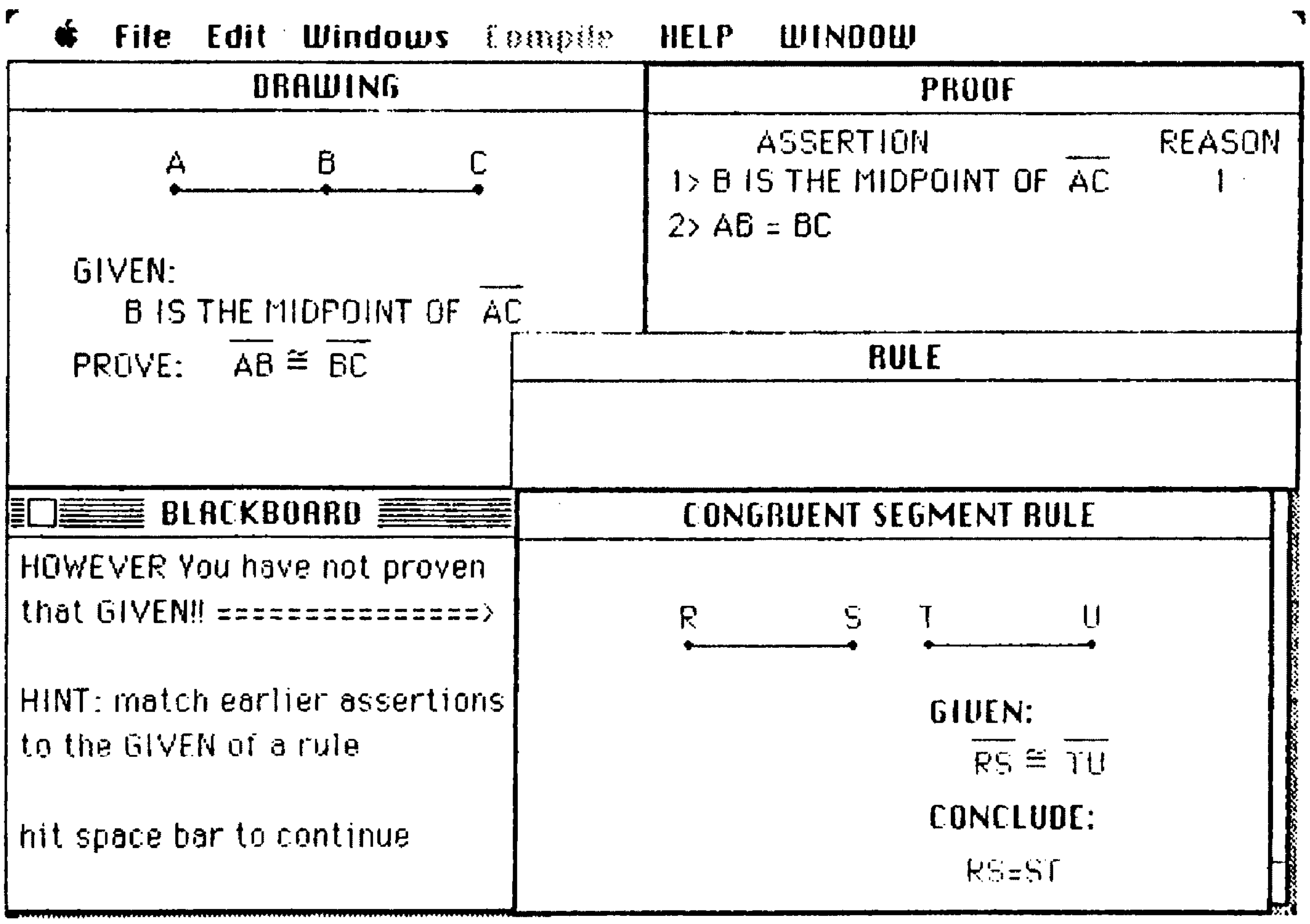
PRETTY GOOD TRY
Your new assertion
matched the CONCLUSION
of this rule ========>

hit space bar to continue

GIVEN:
$\overline{RS} \cong \overline{TU}$
CONCLUDE:
$RS \cong ST$

AUTOMATIC HELP - 2

This screen shows the system's response to the user's choice on page 84. The system points out the misuse of backward chaining by the user.



GIVEN:
B IS THE MIDPOINT OF $\overline{AC}$

PROVE: $\overline{AB} \cong \overline{BC}$

ASSERTION	REASON
1> B IS THE MIDPOINT OF $\overline{AC}$	1
2> AB = BC	

HOWEVER You have not proven that GIVEN!! ===============>

HINT: match earlier assertions to the GIVEN of a rule

hit space bar to continue

GIVEN:
$\overline{RS} \cong \overline{TU}$

CONCLUDE:
RS=ST

AUTOMATIC HELP - 3

This screen is a continuation of the help for an incorrect reason choice shown on the previous two pages. The system trys to help the user by pointing at a heuristic rule.

APPENDIX B

PARSING RULES

It was necessary in this system to create a simple parser that would accept geometric assertions users made while solving proofs. The following grammer was written to establish a design for the parser. It is based on traditional methods for language design which employ Backus-Naur form. ([Hayes85] p. 49) The Syntax rules and Lexical rules of the parser design were then translated into code for the parser. The assertions are the form used in the Scott Foresman GEOMETRY text [Hirsch84].

B.1 Key to Parsing Operators

::=	rewrites as
()	optional element
|	or

B.2 Key to Geometric Operators

symbol	char	operator
=	"="	equal
+	"+"	plus
−	"-"	minus
∡	","	angle
△	"."	triangle
$\overline{}$	"["	segment
⟷	"]"	line
≅	"/"	congruent
⊥	";"	perpendicular
//	"'"	parallel

B.3 Syntax Rules

Assertion ::= Expression Relop Expression

Expression ::= Simple-Expression (Logop Simple-Expression)

Simple-Expression ::= Object (Combop Object)

Object ::= (M | m) Geo-Entity Identifier

| Identifier

| number

B.4. Lexical Rules

Identifier ::= Letter (Letter)

Number ::= digit {digit}

Logop ::= "and"

Relop ::= = | ≅ | ⊥ | // | "bisects"

| "is the midpoint of"

Combop ::= + | -

Geo-Entity ::= $\measuredangle$

| $\overline{\quad}$

| $\overleftrightarrow{\quad}$

| Δ

Letter ::= "a" .. "z" | "A" .. "Z"

APPENDIX C

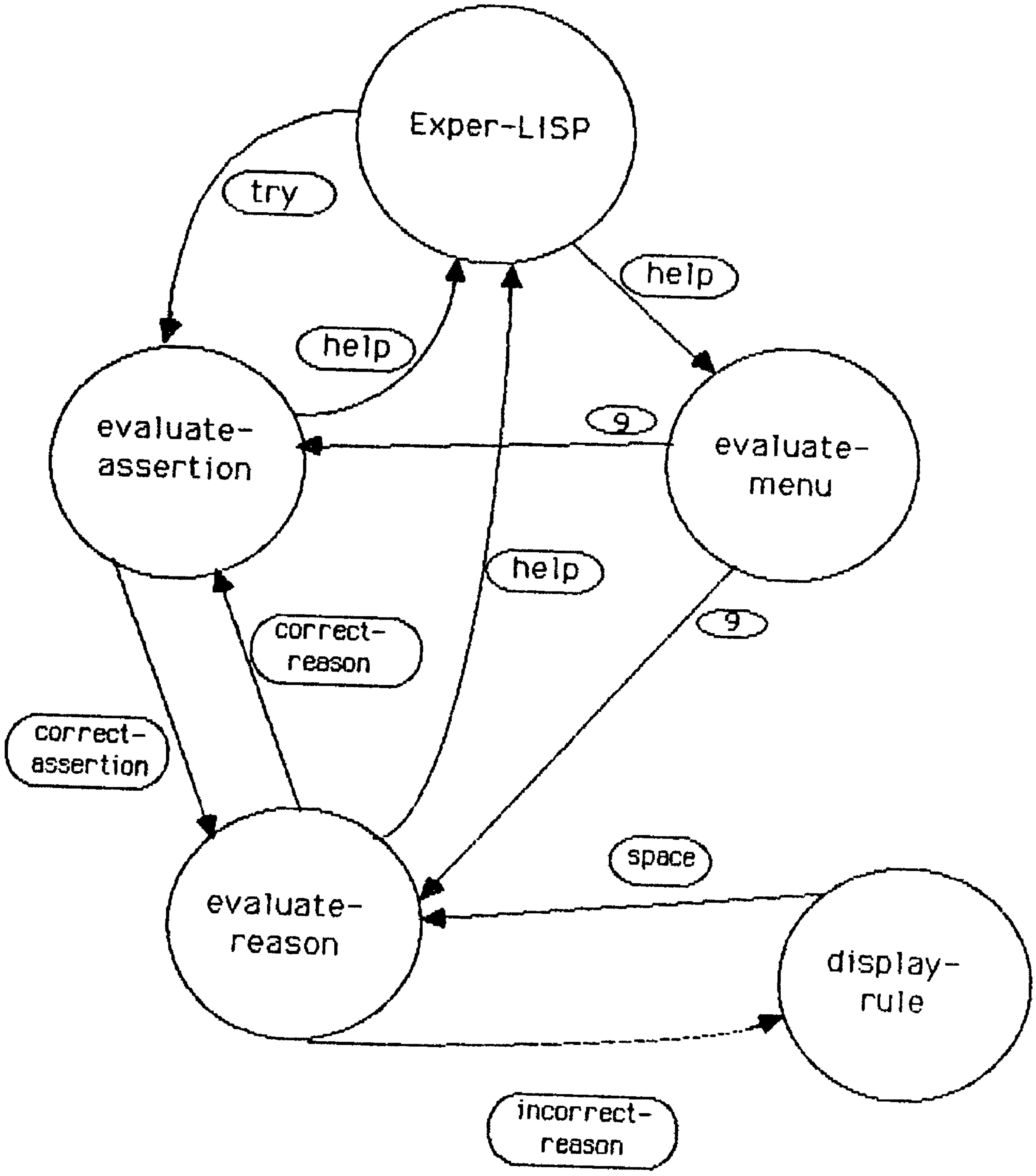
STRUCTURE CHARTS

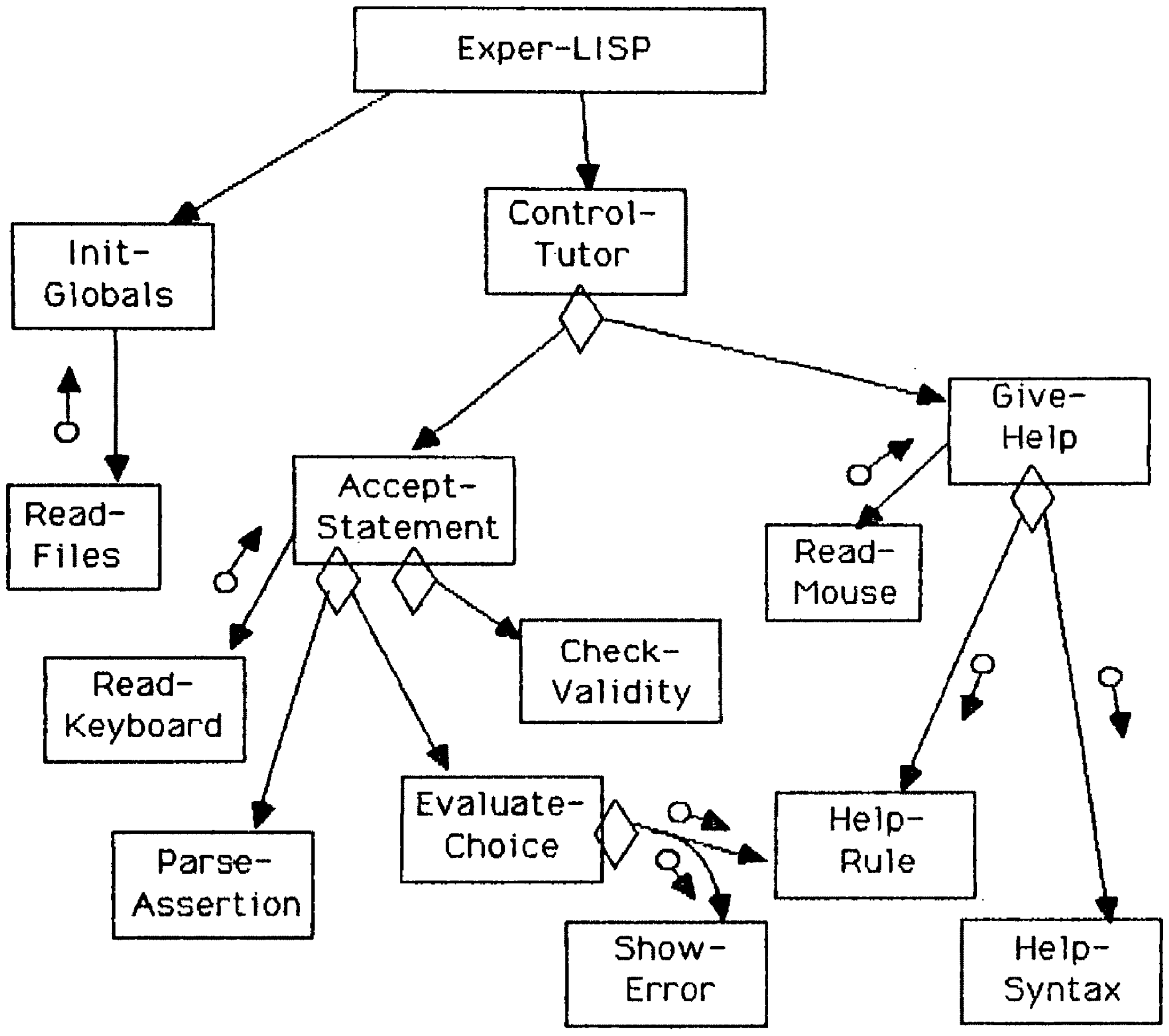
C.1. STATE TRANSITION DIAGRAM

The chart on page 93 is the state transition diagram that shows the flow of control within the system

C.2. HIGH LEVEL STRUCTURE

The chart on page 94 shows the high level logic of the tutorial system. This system was built using bottom up methods and incremental development, so the high level chart is the structure established after development. The high level chart is not a rigorous reflection of the actual system but gives a clearer description of the actual process. A couple of modules are implied by the system rather than implemented in actual code. This chart could be the precursor of the top down development of a second prototype. Descriptions of each module ar on the pages that follow the chart.


Exper-LISP
try
help
help
evaluate-assertion
9
evaluate-menu
help
9
correct-reason
correct-assertion
evaluate-reason
space
display-rule
incorrect-reason


Exper-LISP
Init-Globals
Control-Tutor
Read-Files
Accept-Statement
Give-Help
Read-Mouse
Read-Keyboard
Check-Validity
Evaluate-Choice
Parse-Assertion
Help-Rule
Show-Error
Help-Syntax

C.2.1. Init-Globals

This module is not implemented as a function inside the system. Rather, the globals are declared throughout the system in different files. The initialization is done when the source files are compiled or the compiled files loaded. The initialization establishes the environment for the system including: windows, menus, the problems description static mode values, the problem solution, and other environmental necessities.

C.2.2. Control-Tutor

This module is implemented in event-control and various globals used to maintain control. It is a polling loop that accepts both mouse and keyboard input.

C.2.3. Accept-Statement

This module is implemented by the read-line function. It collects input characters into statements and evaluates them based on the current state of the system. (i.e. which mode global is active)

C.2.4. Parse-Assertion

This module parses geometric assertions. It is implemented in the subtree of functions shown on 105 and 106. The language is described in Appendix B.

C.2.5. Show-Error

Show-Error works in conjunction with Help-Rule described below. It shows a user the possible error made when making a reason choice to justify an assertion. It is implemented by the prototype function choose-rule-direction.

C.2.6. Check-Validity

Check-Validity in this system checks to see if a predicate calculus expression successfully returned from the parser matches the next required assertion of the proof. In a more substantial system, Check-Validity would call an inference engine to determine whether the assertion logically follows from the knowledge base.

C.2.7. Give-Help

Give Help is the help subsystem activated by user requests with the mouse. It gives help on both rules and syntax. Currently it is controlled from Exper-LISP with mode globals that activate the evaluate-menu and display-rule functions

C.2.8. Evaluate-Choice

This module represents the function evaluate-reason in the prototype. If the reason is incorrect, it activates a routine from the Help-Rule array of functions together with a routine that points out the error made in choosing the reason.

C.2.9. Help-Rule and Help-Syntax

These modules represent arrays of lambda functions that display information in the help window.

C.3. FUNCTION STRUCTURE

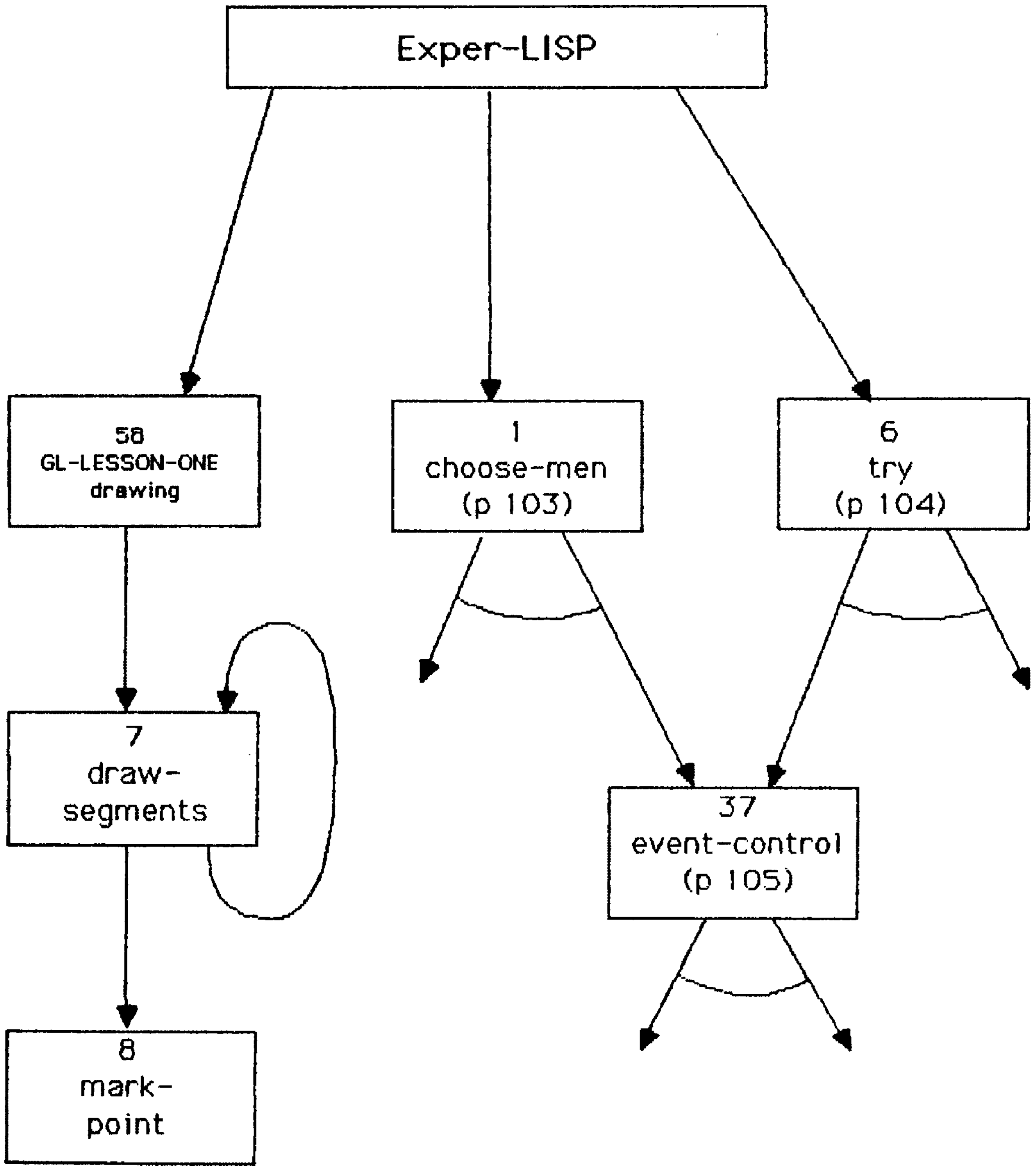
Pages 102 to 110 show the hierarchical structure chart of the prototypes defined functions. These function charts better reflect the actual structure of the system than the high level chart.

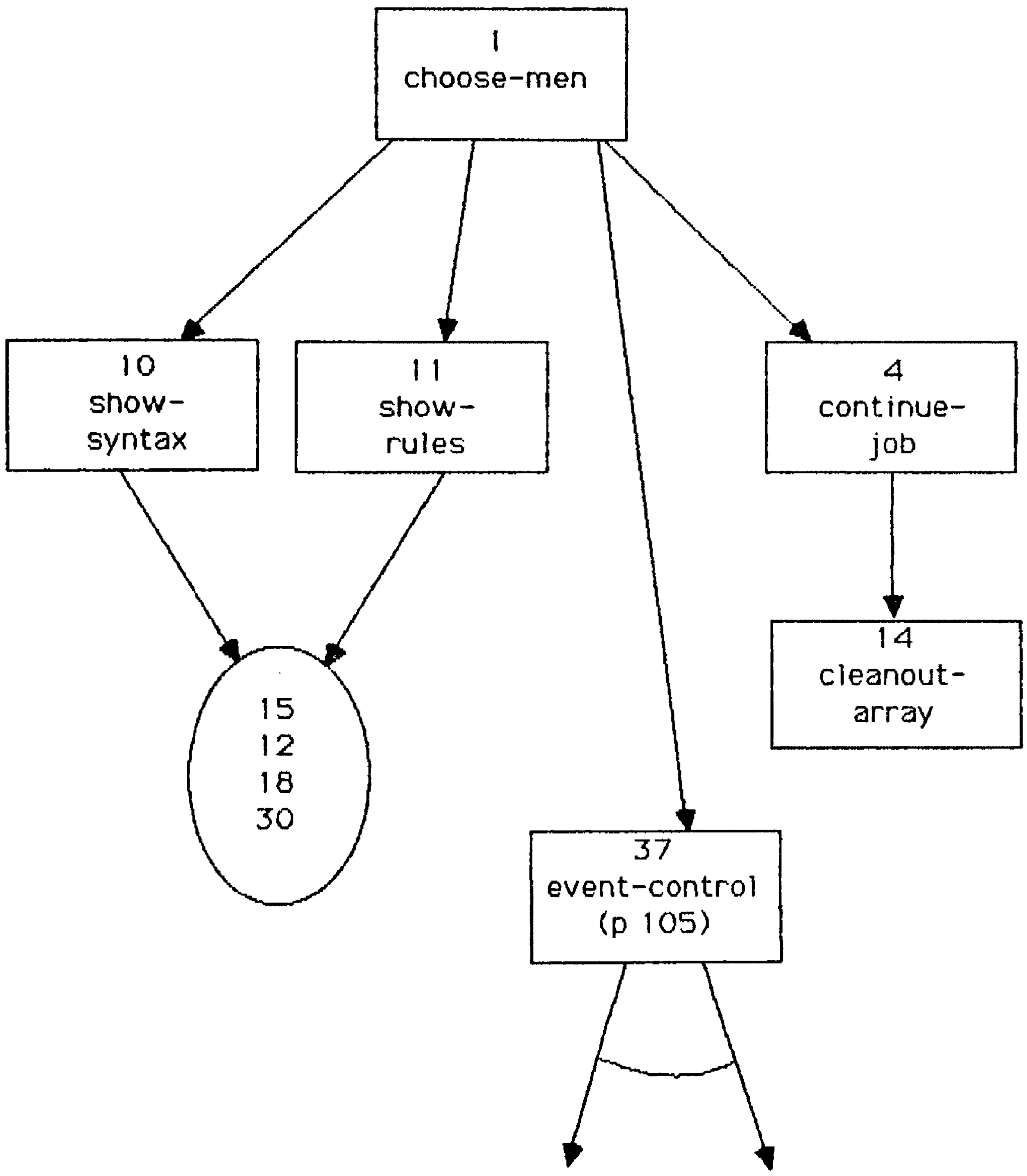
A cross reference for the function charts is included on page 99.

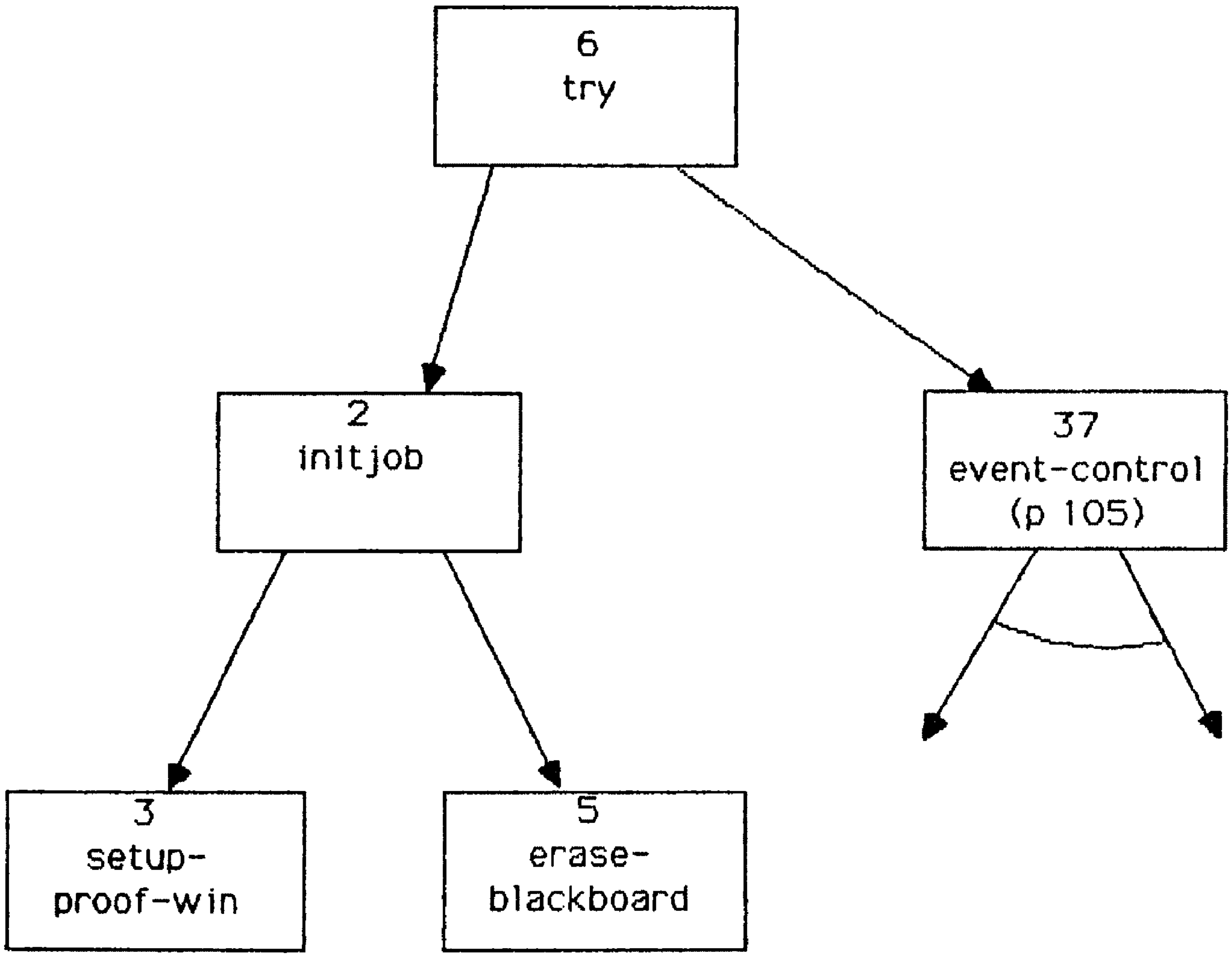
Function Chart Cross Reference

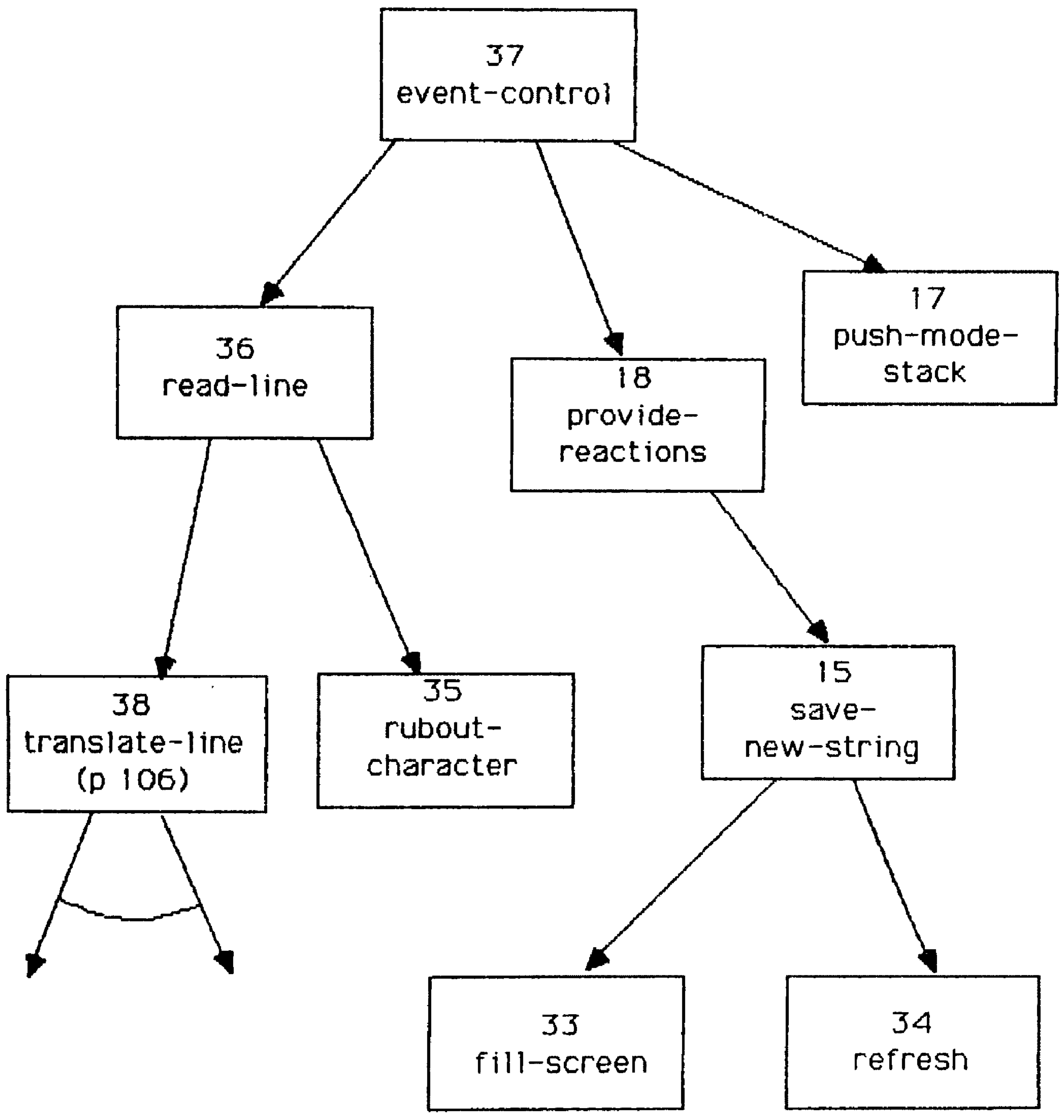
#	FUNCTION NAME	PAGE OF ORIGIN	OTHER PAGE CALLED ON
1.	choose-men	103	102
2.	initjob	104	
3.	setup-proof-win	104	
4.	continue-job	103	
5.	erase-blackboard	104	108
6.	try	104	102
7.	draw-segments	102	108
8.	mark-point	102	
9.	draw-text-lines	107	107
10.	show-syntax	103	107
11.	show-rules	103	107
12.	arr-to-string	106	103
13.	arr-to-list	106	
14.	cleanout-array	103	106
15.	save-new-string	105	103,106
16.	pop-mode-stack	107	107
17.	push-mode-stack	105	107
18.	provide-reactions	105	103,107 108,109
19.	evaluate-assertion	108	106

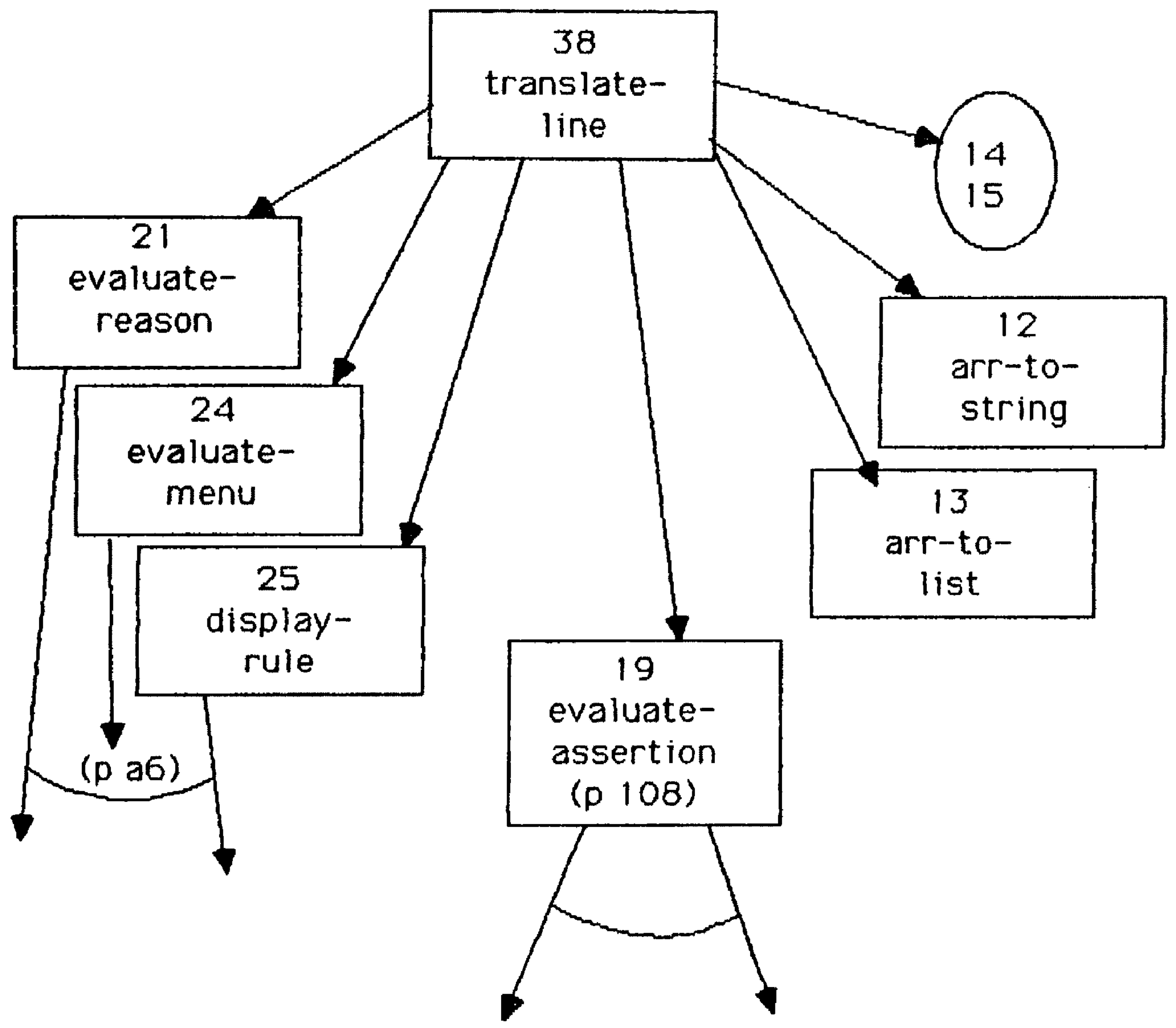
20.	assertion-to-proof	108	
21.	evaluate-reason	107	106
22.	choose-rule-direction	107	
23.	reason-to-proof	107	
24.	evaluate-menus	107	106
25.	display-rule	107	106
26.	check-validity	108	
27.	provide-reasons	108	
28.	is-reason-correct	not used	
29.	restart-reasons	107	107
30.	GL-OPTION-ARRAY	107	103,108
31.	GL-HSYNTAX-ARRAY	107	
32.	GL-EXRULES-ARRAY	107	
33.	fill-screen	105	
34.	refresh	105	
35.	rubout-character	105	
36.	read-line	105	
37.	event-control	105	102,103 104
38.	translate-line	106	105
39.	get-token	109	
40.	peek-token	109	
41.	c-membs	110	


Exper-LISP
58
GL-LESSON-ONE
drawing
1
choose-men
(p 103)
6
try
(p 104)
7
draw-
segments
37
event-control
(p 105)
8
mark-
point

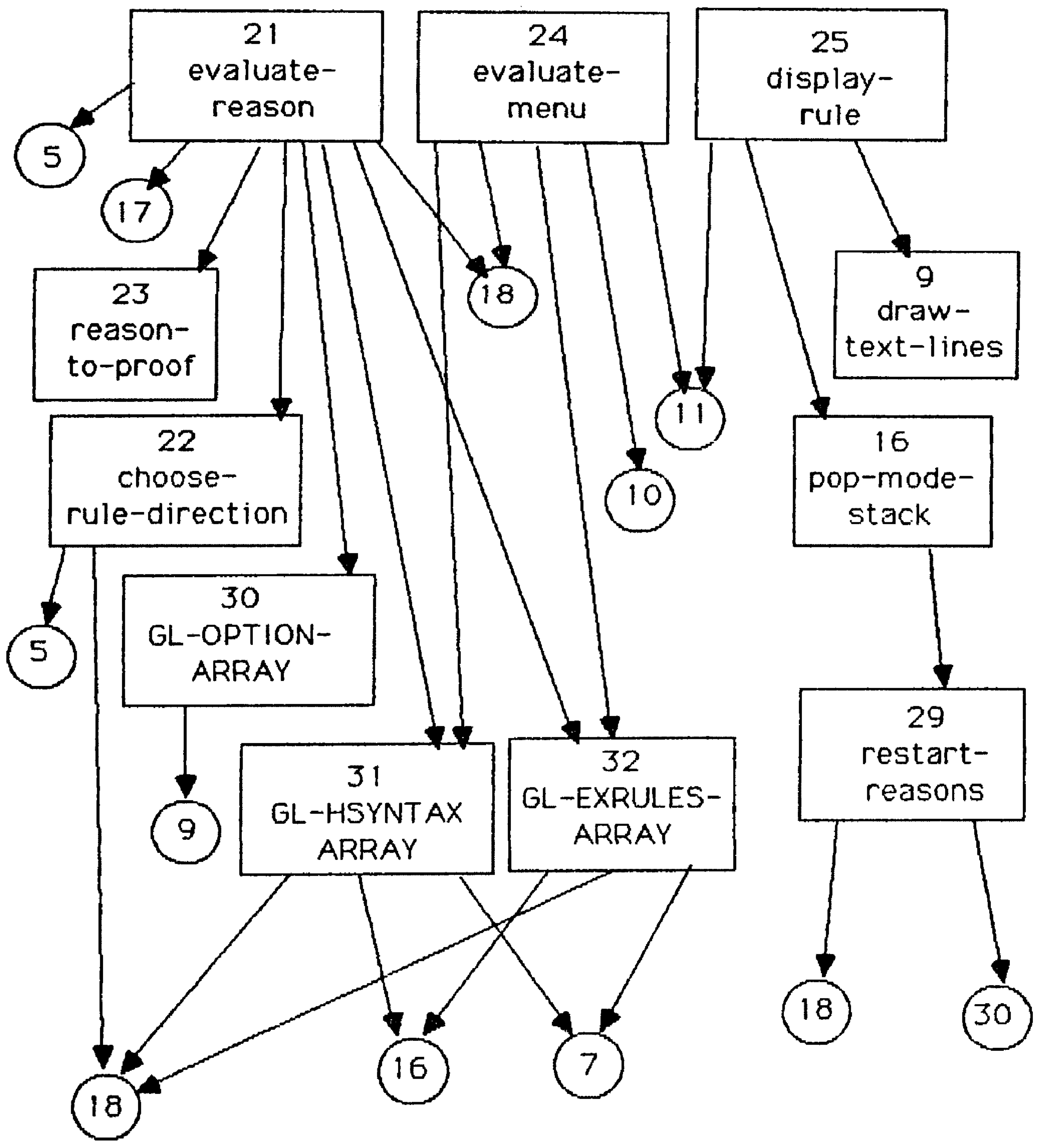

1
choose-men
10
show-
syntax
11
show-
rules
4
continue-
job
14
cleanout-
array
15
12
18
30
37
event-control
(p 105)

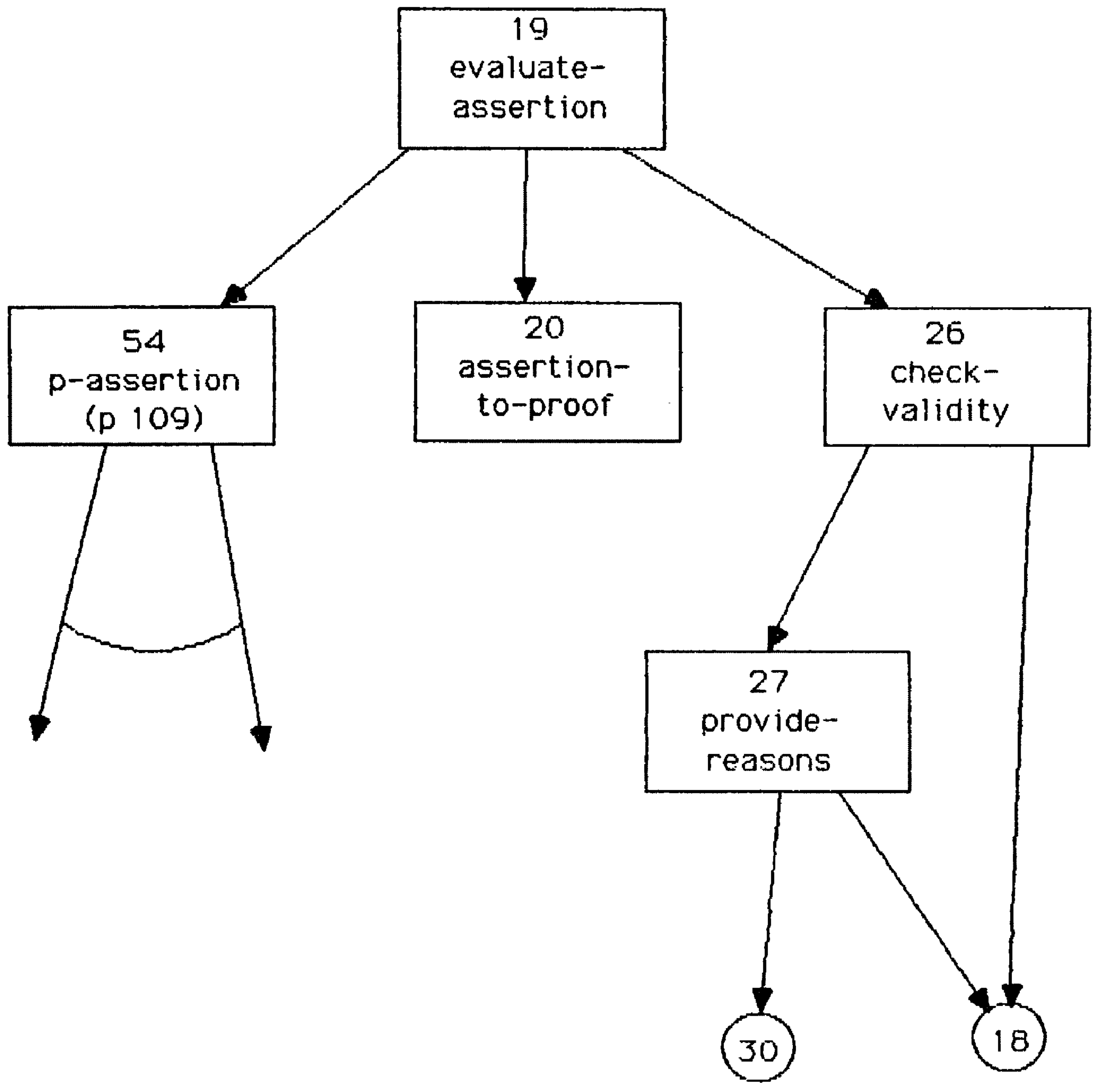

6
try
2
initjob
37
event-control
(p 105)
3
setup-
proof-win
5
erase-
blackboard

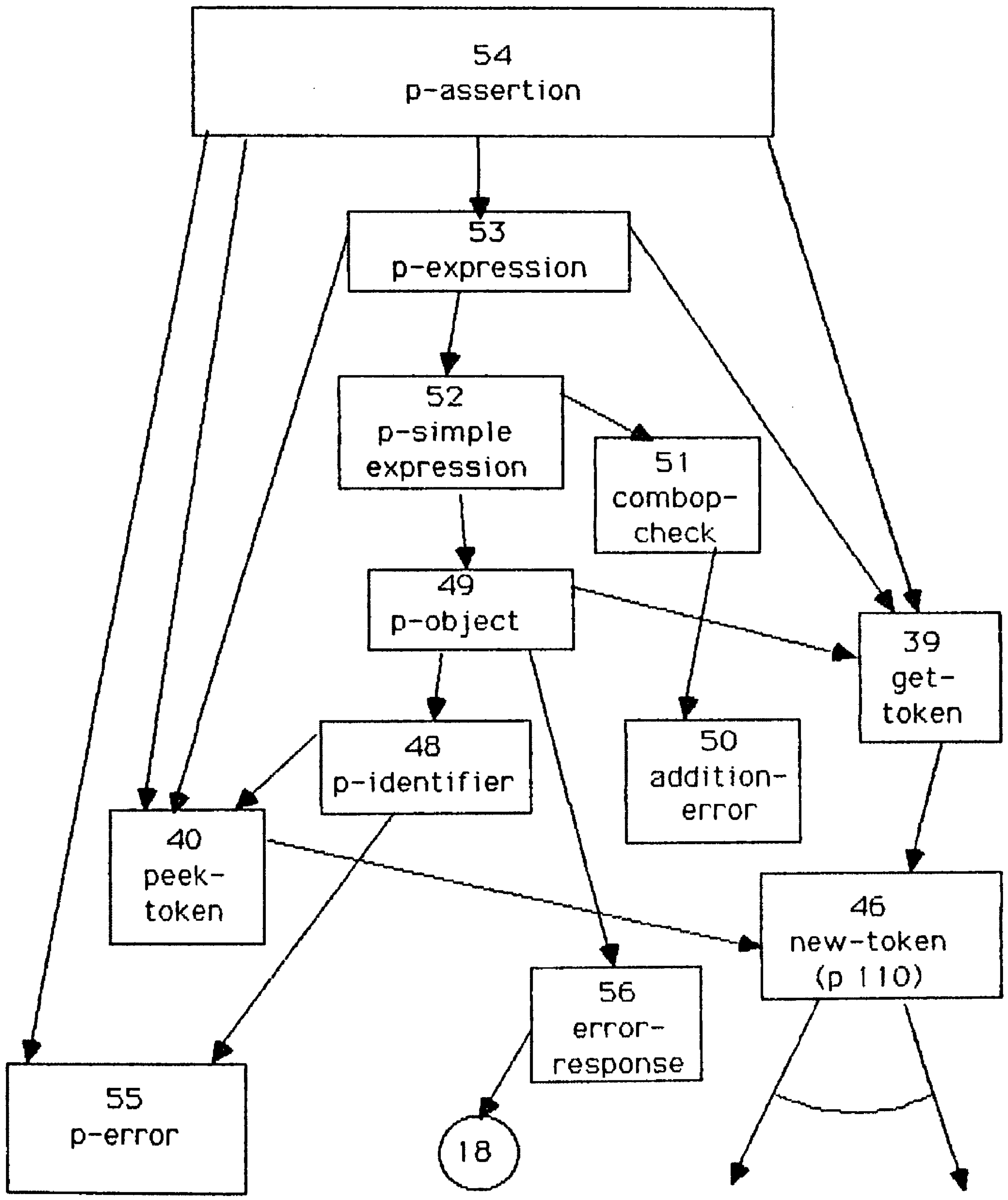

37
event-control
36
read-line
18
provide-
reactions
17
push-mode-
stack
38
translate-line
(p 106)
35
rubout-
character
15
save-
new-string
33
fill-screen
34
refresh

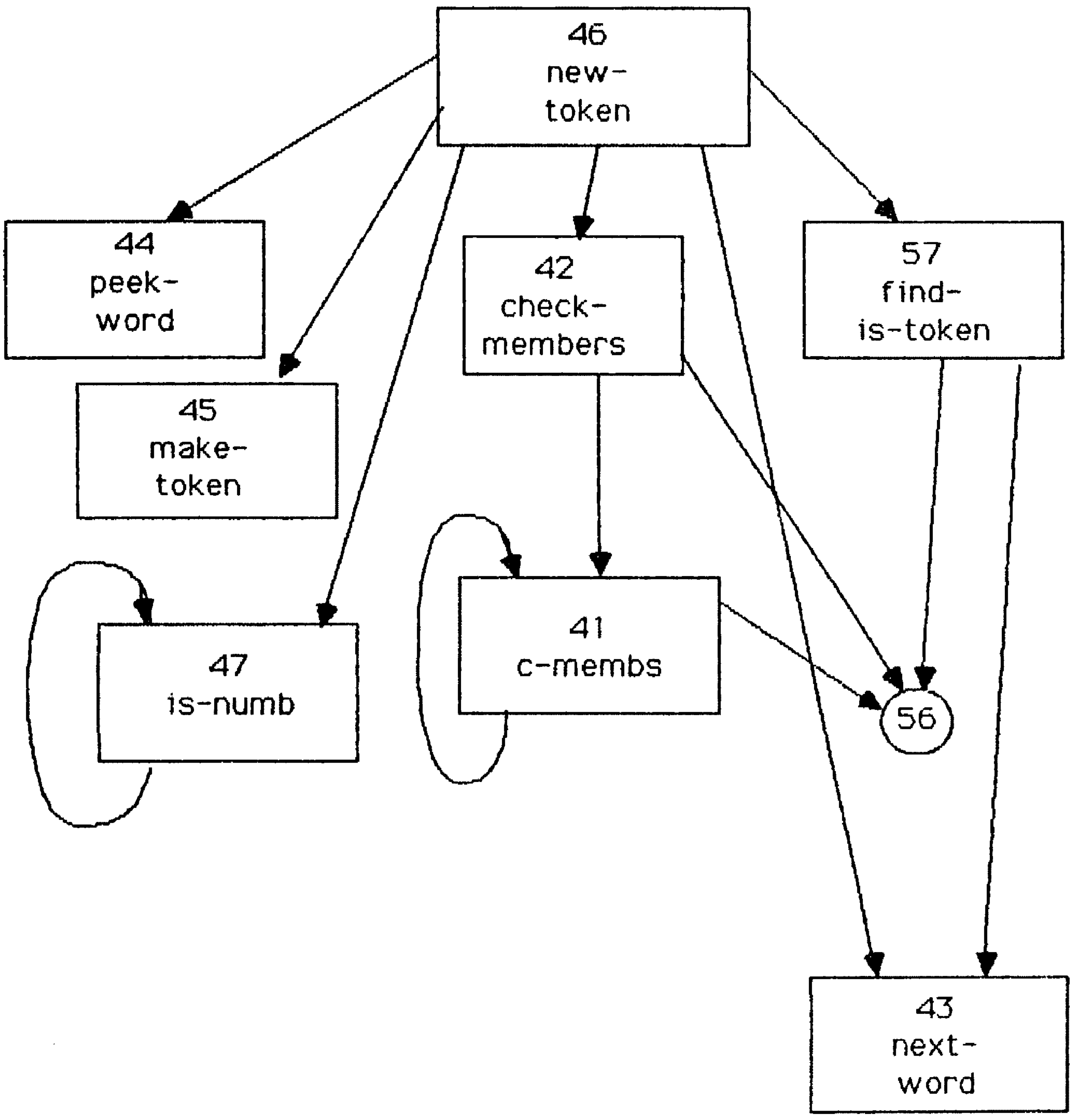

38
translate-
line
14
15
21
evaluate-
reason
24
evaluate-
menu
25
display-
rule
(p a6)
19
evaluate-
assertion
(p 108)
12
arr-to-
string
13
arr-to-
list

SPECIAL EVALUATORS




19
evaluate-
assertion
54
p-assertion
(p 109)
20
assertion-
to-proof
26
check-
validity
27
provide-
reasons
30
18


54
p-assertion
53
p-expression
52
p-simple
expression
51
combop-
check
49
p-object
39
get-
token
48
p-identifier
50
addition-
error
40
peek-
token
46
new-token
(p 110)
56
error-
response
55
p-error
18


46
new-
token
44
peek-
word
45
make-
token
42
check-
members
57
find-
is-token
47
is-numb
41
c-membs
56
43
next-
word

APPENDIX D

GLOBALS

The nature of Exper-LISP made it necessary to create several globals in the implementation of this system. It was both necessary and convenient to make global objects visible from several places in the system that were maintained as static values throughout the lesson. More specific reasons for the globals will be given for each type of global mentioned.

D.1. WINDOWS

These variables were based on Exper-LISP's implementation of a window class and were defined as global using system functions. Objects of this class were instantiated in the following windows and window specific properties were maintained for each window.

```
gl-awin
        STRUCTURE: window.
        DECLARED IN: A-START.
        PURPOSE: Drawing.
```

gl-bwin
    STRUCTURE: window.
    DECLARED IN: A-START.
    PROPERTIES: x-start, y-start
    PURPOSE: Proof.

gl-cwin
    STRUCTURE: window.
    DECLARED IN: A-START.
    PROPERTIES: lines, size, input-line,
     input-xval, in-loc, array-loc, x-start,
     y-start, upkeep
    PURPOSE: Blackboard.

gl-helpwin
    STRUCTURE: window.
    DECLARED IN: A-START.
    PROPERTIES: x-start, y-start
    PURPOSE: Geometry help.

gl-rulewin
    STRUCTURE: window.
    DECLARED IN: A-START.
    PROPERTIES: x-start, y-start
    PURPOSE: Rule help.

D.2. MENUS

These globals were used to implement menus in the ExperLisp system.

```
gl-ch-menu
        STRUCTURE: menu.
        DECLARED IN: A-START.
        PURPOSE: Window control.

gl-w-menu
        STRUCTURE: menu.
        DECLARED IN: A-START.
        PURPOSE: Help control.
```

D.3. CONTROL

The current Exper-Lisp system did not allow access to the event queue of Quickdraw. This complicated a system which required both keyboard and mouse input during the same time period. In order to access menu procedures of Quickdraw, the system dropped down to the Exper-LISP control level to allow mouse input. To allow this, globals were created to maintain information about current states of the system. Depending upon which of the states was active at the help call, a mode variable with records in a property list was pushed onto a stack and was later used to recreate the system. Some of the control globals are simply flags to maintain control of input.

```
gl-curwin
        STRUCTURE: atom.
        DECLARED IN: A-START.
        PURPOSE: I/O control.

gl-first-click
        STRUCTURE: boolean.
        DECLARED IN: A-START.
        PURPOSE: Input control.

gl-help-avail
        STRUCTURE: list.
        DECLARED IN: A-START.
        PURPOSE: Input control.

gl-hrules1-mode
        STRUCTURE: property list.
        DECLARED IN: A-START.
        PROPERTIES: prompt, available-options,
         help-array-name, evaluator, help-choice.
        PURPOSE: Text evaluation and output control.

gl-hrules2-mode
        STRUCTURE: property list.
        DECLARED IN: A-START.
        PROPERTIES: prompt, available-options,
         help-array-name, evaluator, help-choice
        PURPOSE: Text evaluation and output control.
```

gl-hsyntax-mode
STRUCTURE: property list.
DECLARED IN: A-START.
PROPERTIES: prompt, available-options, help-array-name, evaluator, help-choice
PURPOSE: Text evaluation and output control.

gl-lesson-one
STRUCTURE: property list.
DECLARED IN: F-LESSON.
PROPERTIES: prompt, rules-avail, assertion-list, solution-size, solution-step, mode-stack, reason-choice, evaluator, latest-as-list, save-line, drawing.
PURPOSE: Text evaluation and output control.

gl-m-pos
STRUCTURE: list.
DECLARED IN: D-INPUT/OUTPUT.
PURPOSE: Input control.

gl-menu-call
STRUCTURE: boolean.
DECLARED IN: D-INPUT/OUTPUT.
PURPOSE: Input control.

gl-present-lesson
STRUCTURE: property list.
DECLARED IN: A-START and F-LESSON.
PROPERTIES: solution-step, mode-stack, assertion-list.
PURPOSE: Text evaluation and output control.

gl-present-mode
STRUCTURE: atom.
DECLARED IN: A-START.
PURPOSE: Evaluation control.

gl-reason-mode
STRUCTURE: property list.
DECLARED IN: A-START.
PROPERTIES: prompt, restart, correct-response, evaluator.
PURPOSE: Text evaluation and output control.

gl-test
STRUCTURE list.
DECLARED IN: B-UTILITIES.
PURPOSE: trace.

gl-tst
STRUCTURE list.
DECLARED IN: B-UTILITIES.
PURPOSE: trace.

D.4. FUNCTION ARRAYS

These arrays were established to provide help in an organized way. There is an array of functions for each type of help available An element of each array is used for the corresponding rule for which help is asked.

gl-exrules-array
STRUCTURE: array(20) of lambda functions.
DECLARED IN C-SYSAR.
PURPOSE: To give help on application of rules of geometry.

gl-hsyntax-array
STRUCTURE: array(10) of lambda functions.
DECLARED IN C-SYSAR.
PURPOSE: To give help on the syntax of geometric assertions.

gl-option-array
STRUCTURE: array(5) of lambda functions.
DECLARED IN C-SYSAR.
PURPOSE: To output menus for keyboard choices.

D.5. STRING ARRAYS

These arrays were established to organize message strings. They are accessible from functions that need to print them.

gl-error-message
STRUCTURE: array(20) of strings.
DECLARED IN: E-PARSER.
PURPOSE: Error messages for parser.

gl-formal-rule
STRUCTURE: array(20) of string lists.
DECLARED IN C-SYSAR.
PURPOSE: Text for output on formal rules of geometry.

gl-wrong-reason
STRUCTURE: array(7,7,2) of integer indices.
DECLARED IN C-SYSAR.
PURPOSE: Decision table of indices to access textual responses of heuristic rules.

gl-wrong-response
STRUCTURE: array(20) of string lists.
DECLARED IN C-SYSAR.
PURPOSE: Text for output on heuristic rules.

D.6. PARSING GLOBALS

These variables were established as global objects accessible from any place in the parser. They are lists of symbols needed to parse the assertions made in the system.

```
gl-all-symbols
        STRUCTURE: list.
        DECLARED IN: E-PARSER.
        PURPOSE: List of all special characters used
         for geometric symbols to be parsed.

gl-combinationals
        STRUCTURE: list.
        DECLARED IN: E-PARSER.
        PURPOSE: List of characters that are
         combinational operators to parse.

gl-figures
        STRUCTURE: list.
        DECLARED IN: E-PARSER.
        PURPOSE: List of characters
         used for geometric figures.

gl-p-list
        STRUCTURE: list.
        DECLARED IN: E-PARSER.
        PURPOSE: Token list to be parsed.

gl-points
        STRUCTURE: list.
        DECLARED IN: E-PARSER.
        PURPOSE: List of points in figure of problem.
```

gl-properties
STRUCTURE: list.
DECLARED IN: E-PARSER.
PURPOSE: List of strings that are geometric properties.

gl-relationals
STRUCTURE: list.
DECLARED IN: E-PARSER.
PURPOSE: List of characters that represent geometric relationships to be parsed.

gl-t-list
STRUCTURE: list.
DECLARED IN: E-PARSER.
PURPOSE: Token list to be parsed.

APPENDIX E

CODE

The following list corresponds to the code on the following pages. It includes numbered functions which map from the functional chart of Appendix C to the coded functions of this chapter. It also lists those functions called by each function.

E.1. A-START file

1. choose-men
    a. show-syntax 10
    b. show-rules 11
    c. continue-job 4
    d. event-control 37

2. initjob
    a. erase-blackboard 5
    b. setup-proof-win 3

3. setup-proof-win

4. continue-job
    a. cleanout-array 14

5. erase-blackboard

6. try
    a. initjob 2
    b. event-control 37

E.2. B-UTIL file

7. draw-segments
    a. mark-point 8
    b. draw-segments 7

8. mark-point

9. draw-text-lines

10. show-syntax
    a. save-new-string 15
    b. arr-to-string 12
    c. provide-reactions 18
    d. GL-OPTION-ARRAY(1) 30

11. show-rules
    a. save-new-string 15
    b. arr-to-string 12
    c. provide-reactions 18
    d. GL-OPTION-ARRAY(4) 30

12. arr-to-string

13. arr-to-list

14. cleanout-array

15. save-new-string
    a. upkeep props
        i) fill-screen 33
        ii) refresh 34

16. pop-mode-stack
    a. restart props
        i) restart-reasons 29

17. push-mode-stack

18. provide-reactions
    a. save-new-string 15

19. evaluate-assertion
    a. p-assertion 54
    b. check-validity 26
    c. assertion-to-proof 20

20. assertion-to-proof

21. evaluate-reason
    a. reason-to-proof 23
    b. provide-reactions 18
    c. push-mode-stack 17
    d. erase-blackboard 5
    e. help-array-name prop
       i) GL-HSYNTAX-ARRAY 31
       ii) GL-EXRULES-ARRAY 32
    f. choose-rule-direction 22
    g. GL-OPTION-ARRAY() 30

22. choose-rule-direction
    a. erase-blackboard 5
    b. provide-reactions 18

23. reason-to-proof

24. evaluate-menu
    a. help-array-name prop
       i) GL-HSYNTAX-ARRAY 31
       ii) GL-EXRULES-ARRAY 32
    b. pop-mode-stack 16
    c. draw-text-lines 9
    d. provide-reactions 18
    e. help-choice prop
       i) show-syntax 10
       ii) show-rules 11

25. display-rule
    a. provide reactions
    b. pop-mode-stack 16
    c. draw-text-lines 9
    d. show-rules 11

26. check-validity
    a. provide-reasons 27
    b. provide-reactions 18

27. provide-reasons
    a. provide-reactions 18
    b. GL-OPTION-ARRAY 30

28. is-reason-correct
    a. provide-reactions 18

29. restart-reasons
    a. provide-reactions 18
    b. GL-OPTION-ARRAY 30

E.3. C-SYSTEM-ARRAYS file

30. GL-OPTION-ARRAY
    #1
      a. draw-text-lines 9
    #4
      a. draw-text-lines 9

31. GL-HSYNTAX-ARRAY
    #0
      a. provide-reactions 18
    #1
      a. draw-segments 7
    #2
      a. draw-segments 7
    #3
      a. draw-segments 7
    #4
      a. draw-segments 7
    #5
      a. draw-segments 7
    #9
      a. pop-mode-stack 16

32. GL-EXRULES-ARRAY
    #0
      a. provide-reactions 18
    #1

    #9
      a. pop-mode-stack 16
    #12
      a. draw-segments 7
    #13
      a. draw-segments 7
    #14
      a. draw-segments 7
    #15
      a. draw-segments 7
    #16

E.4. D-INPUT/OUTPUT file

33. fill-screen

34. refresh

35. rubout-character

36. read-line
    a. translate-line 38
    b. rubout-character 35

37. event-control
    a. provide-reactions 18
    b. push-mode-stack 17
    c. read-line 36

```
38. translate-line
    a. arr-to-list 13
    b. arr-to-string 12
    c. save-new-string 15
    d. cleanout-array 14
    e. evaluator prop
       i) evaluate-assertion 19
       ii)evaluate-reason 21
       iii) evaluate-menu 24
       iv) display-rule 25
```

E.5. E-PARSER

```
39. get-token
    a. new-token 46

40. peek-token
    a. new-token 46

41. c-membs
    a. c-membs 41
    b. error-response 56

42. check-members
    a. error-response 56
    b. c-membs 41

43. next-word 43

44. peek-word

45. make-token
```

46. new-token 46
    a. next-word 43
    b. is-numb 47
    c. make-token 45
    d. find-is-token 57
    e. peek-word 44
    f. check-members 42

47. is-numb
    a. is-numb 47

48. p-identifier
    a. peek-token 40
    b. p-error 55

49. p-object
    a. get-token 39
    b. p-identifier 48
    c. error-response 56

50. addition-error

51. combop-check
    a. addition-error 50

52. p-simple-expression
    a. p-object 49
    b. combop-check 51
    c. peek-token 40
    d. get-token 39

53. p-expression
    a. p-simple-expression 52
    b. get-token 39
    c. peek-token 40

54. p-assertion
    a. peek-token 40
    b. p-expression 53
    c. get-token 39
    d. p-error 55

55. p-error

56. error-response
    a. provide-reactions 18

57. find-is-token
    a. next-word 43
    b. error-response 56

E.6. F-LESSON file

58. GL-LESSON-ONE drawing
    a. draw-segments 7

59. GL-OPTION-ARRAY
    #2
      a. draw-text-lines 9

```
                    ;              //    A-START    //
;------------------------------------------------------------

;THIS FILE IS THE FIRST FILE COMPILED
;IT SETS UP THE WINDOWS AND MENUS
;IT ALSO HAS THE STARTUP FUNCTIONS

;  IT INCLUDES THE FOLLOWING AREAS

;  1.  windows
;      A. window creation
;      B. window properties
;  2.  menu creation
;      [1] choose menu ()
;  3.  MODE CREATION
;         1. GL-help-avail ( which modes available with help)
;      A. GL-reason-mode
;      B. GL-hsyntax-mode
;      C. GL-hrules1-mode
;      D. GL-hrules2-mode
;  4.  begin and continue
;      [2] initjob()
;      [3] setup-proof-win
;      [4] continue-job
;      [5] erase-blackboard
;      [6] try
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;WINDOWS
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;WINDOW CREATION

;this SECTION is for establishing a windowed enviornment for
;the current system

;the following assertions set up the menu enviornment


;(1) this sets up three instances of the class window for
;interaction
(setq gl-awin (newgrafwindow '(39 0 180 250)))
(setq gl-bwin (newgrafwindow '(39 250 180 510)))
(setq gl-cwin (newgrafwindow '(200 0 340 200)))

;this sets up a window for help pictures

(setq gl-helpwin (newgrafwindow '(200 200 340 500)))
(setq gl-rulewin (newgrafwindow '(140 200 180 510)))

(gl-awin 'setwtitle "DRAWING")
(gl-bwin 'setwtitle "PROOF")
(gl-cwin 'setwtitle "BLACKBOARD")
(gl-helpwin 'setwtitle "GEOMETRY")
(gl-rulewin 'setwtitle "RULE")

;THE FOLLOWING SET THE FONTS FOR EACH WINDOW

(textfont 17 gl-cwin)
(textfont 17 gl-bwin)
(textfont 17 gl-awin)
(textfont 17 gl-helpwin)
(textfont 17 gl-rulewin)
```

```
;*************************************************************
*************
;WINDOW PROPERTIES

;THE FOLLOWING AREA ESTABLISHES CERTAIN GLOBALS FOR THE
SYSTEM
; THEY ARE ESTABLISHED WHEN THIS FILE IS COMPILED

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;THESE ARE GL-CWIN PROPERTIES

(putprop gl-cwin (make-array 7) 'lines)
(putprop gl-cwin  6 'size)
(putprop gl-cwin (make-array 40) 'input-line)  ;input buffer
(putprop gl-cwin (make-array 40) 'input-xval)  ;input point array


;THESE ARE GL-HELPWIN PROPERTIES


(putprop gl-helpwin -140  'x-start)
(putprop gl-helpwin -55 'y-start)

;;;;;;;;;
;THESE ARE GL-RULEWIN PROPERTIES

(putprop gl-rulewin -145 'x-start)
(putprop gl-rulewin -3 'y-start)




;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;MENU CREATION
;;;;;;;;;;;;;;;;;;;;;;;;
;(2) this creates the new menus called WINDOW and HELP

(setq gl-w-menu (newmenu 6 " HELP "))
(appendmenu gl-w-menu "Assertion-Syntax;RULE: Example;RULE:
Formal")
(insertmenu gl-w-menu 0)

(setq gl-ch-menu (newmenu 7 "WINDOW"))
(appendmenu gl-ch-menu "Drawing;Proof;Blackboard;Help")
(insertmenu gl-ch-menu 0)

(drawmenubar)
;(3)
;[1] ---------------------->
;This function is called when a menu selection of HELP-SYN or
; BUILD is   chosen
;a menu selection function
(defun choose-men (themenu theitem)
 (cond ((= themenu 6)
      (cond((= theitem 1)
           (setq gl-present-mode 'gl-hsyntax-mode)
           (show-syntax)
          )
          ((= theitem 2)
           (setq gl-present-mode 'gl-hrules1-mode)
           (show-rules)
           )
          ((= theitem 3)
           (setq gl-present-mode 'gl-hrules2-mode)
           (show-rules)
           )
          )
      )
```

```
    ((= themenu 7)
     (cond((= theitem 1)
          (gl-awin 'selectwindow)
          )
         ((= theitem 2)
          (gl-bwin 'selectwindow)
          )
         ((= theitem 3)
          (gl-cwin 'selectwindow)
          )
         ((= theitem 4)
          (gl-helpwin 'selectwindow)
          )

        )
     )
    )
 (setq gl-first-click t)
 (continue-job gl-cwin)
 (event-control)
 )
;(4) this sets the hook for menu calls to try-menu

(setq ªmenuhook choose-men)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;                   // MODE-OBJECTS //
;-----------------------------------------------------------------

;THE FOLLOWING PART CONTAINS ESTABLISHMENT OF MODE OBJECTS
; USED TO CONTROL THE OPERATION OF THE SYSTEM

;       1. GL-help-avail
;   A. GL-reason-mode
;   B. GL-hsyntax-mode
;   C. GL-hrules1-mode
;   D. GL-hrules2-mode

(setq GL-help-avail '(gl-reason-mode)) ;have restart prop
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GL-REASON-MODE
;;;;;;;;;;;;;;;;;;;;;;;;

;THE FOLLOWING CALLS ESTABLISH A NEW MODE GL-REASON-MODE
;WHICH IS USED TO GET REASONS FOR ASSERTIONS IN A PROOF

;this frame includes
;  evaluator
;  prompt
;  restart
;  correct-response  ;initialized in initjob
;  save-line    ;this will be string to compare to correct-response
;  latest-as-list  REDUNDANT - used in translate-line
(putprop 'GL-reason-mode "==> " 'prompt)
(putprop 'GL-reason-mode 'restart-reasons 'restart)
(putprop 'GL-reason-mode nil 'correct-response)

(putprop 'GL-reason-mode 'evaluate-reason 'evaluator)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GL-HSYNTAX-MODE
;;;;;;;;;;;;;;;;;;;;;;

;THE FOLLOWING IS FOR SETTING UP THE OBJECT GL-HSYSNTAX-MODE
; it includes the following slots

;    prompt
;    save-line       ;entered in translate-line
;    available-options
;    help-array-name
;    evaluator


(putprop 'GL-hsyntax-mode "==> " 'prompt)
(putprop 'GL-hsyntax-mode
      '("1" "2" "3" "4" "5" "9") 'available-options)

(putprop 'GL-hsyntax-mode 'GL-hsyntax-array 'help-array-name)
(putprop 'GL-hsyntax-mode 'evaluate-menu 'evaluator)
(putprop 'GL-hsyntax-mode 'show-syntax 'help-choice)=
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;FOLLOWING IS THE BEGINNING OF HELP-RULES SECTION OF THE SYSTEM

;;;;;;;;;;;;;;;;;;;;
;FOLLOWING ARE SOME MORE MODES STARTING WITH GL-HRULES1-MODE
;it has following slots
;  prompt
;  available-options
;  help-array-name
;  evaluator
;  save-line   --inserted in translate line
;  rule-choice  -- inserted in save-rule-choice
```

```
(putprop 'GL-hrules1-mode "==> " 'prompt)
(putprop 'GL-hrules1-mode
      '("12" "13" "14" "15" )
      'available-options)
(putprop 'GL-hrules1-mode 'GL-EXrules-array 'help-array-name)


(putprop 'GL-hrules1-mode 'evaluate-menu  'evaluator)
(putprop 'GL-hrules1-mode 'show-rules 'help-choice)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;FOLLOWING ARE SOME MORE MODES STARTING WITH GL-HRULES2-MODE
;it has following slots
;  prompt
;  available-options
;  help-array-name
;  evaluator
;  save-line   --inserted in translate line
;  rule-choice  -- inserted in save-rule-choice

(putprop 'GL-hrules2-mode "==> " 'prompt)
(putprop 'GL-hrules2-mode
      '("12" "13" "14" "15" )
      'available-options)
(putprop 'GL-hrules2-mode 'GL-formal-rule 'help-array-name)


(putprop 'GL-hrules2-mode 'display-rule  'evaluator)
(putprop 'GL-hrules2-mode 'show-rules 'help-choice)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;THE FOLLOWING ARE FUNCTIONS TO MAINTAIN THE GLOBALS FOR
GL-CWIN
; WHEN THE SYSTEM GOES DOWN BY A MENU CALL

;[2] ---------------------->

;this sets up the blackboard properties for refresh
; IT IS CALLED EACH TIME A NEW SESSION IS STARTED
(defun initjob ()
 (erase-blackboard)
 (setup-proof-win)

 (setq gl-present-mode GL-present-lesson)
 (putprop GL-present-lesson 1 'solution-step)
 (putprop GL-present-lesson (list gl-present-lesson)
        'mode-stack)
 (putprop 'GL-reason-mode
        (cadr (get gl-present-lesson 'assertion-list))
        'correct-response)
 (cs gl-rulewin)
 (cs gl-helpwin)
 (setq gl-first-click t)

)
```

```
;[3]-------------------->
;THE FOLLOWING FUNCTION SETS UP THE PROOF WINDOW FOR A NEW
PROOF


(defun setup-proof-win ()
        (norecord gl-bwin)
        (cs gl-bwin)
        (record gl-bwin)
        (moveto -85 -55 gl-bwin)
        (drawstring  "ASSERTION" gl-bwin)
        (moveto 75 -55 gl-bwin)
        (drawstring "REASON" gl-bwin)
        (putprop gl-bwin -120  'x-start)
        (putprop gl-bwin -40 'y-start)
        )


;[4]-------------------->
;FOLLOWING IS A FUNCTION FOR REESTABLISHING THE INPUT BUFFER
; AFTER EACH SYSTEM SHUTDOWN WITH A MENU CALL

(defun continue-job (screen)
 (cleanout-array screen)
 (putprop gl-cwin 0 'in-loc))    ;array index for input buffer
```

```
;[5] -------------------->
;THE FOLLOWING FUNCTION CLEARS THE BLACKBOARD WINDOWN.

(defun erase-blackboard ()
  ;this gives us a setup for our cwin
  (cs gl-cwin)
  (putprop gl-cwin 0 'array-loc)
  (putprop gl-cwin -95  'x-start)
  (putprop gl-cwin -55 'y-start)
  (putprop gl-cwin 'fill-screen 'upkeep) ;refreshing fct for screen
  (putprop gl-cwin 0 'in-loc) ;array index for input buffer
  ;first we move to the input start for the blackboard
  (moveto (get gl-cwin 'x-start) (get gl-cwin 'y-start) gl-cwin)
  (gl-cwin 'selectwindow)
  (setq gl-curwin gl-cwin)
  )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;[6] ----------------->
;THIS FUNCTION IS JUST FOR STARTUP
(defun try ()
 (initjob)
 (event-control)
 )
```

```
                    ; //     B-UTILITIES     //
          ;-----------------------------------------------

;THIS FILE CONTAINS SEVERAL UTILITIY FUNCTIONS USED IN THE
SYSTEM
; IT IS BROKEN INTO 2 PARTS:: UTILITIES AND EVALUATION UTILITIES
;
; I.  UTILITIES
;     A. Drawing utilities
;       [7] draw-segments
;       [8] mark-point
;       [9] draw-text-lines

;    B. Prompting utilities
;       [10] show-sytax
;       [11] show-rules

;    C. General utilities
;       [12] arr-to-string
;       [13] arr-to-list
;       [14] cleanout-array
;       [15] save-new-string
;       [16] pop-mode-stack
;       [17] push-mode-stack
;       [18] provide reactions
; II. EVALUATION UTILITIES
;     A. Assertions
;        [19] evalutate-assertion
;        [20] assertion-to-proof
;     B. Reasons
;        [21] evaluate-reason
;        [22] choose-rule-direction
;        [23] reason-to-proof
;     C. Menus
;       [24] evaluate-menu
;       [25] display rule

;    D. Lesson utilities
;       [26] check-validity
;       [27] provide-reasons
;       [28] is-reason-correct(scratched)
;       [29] restart-reasons

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;UTILITIES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;DRAWING UTILITIES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;FOLLOWING ARE FUNCTIONS THAT WERE FROM THE OLD HELP FILE
;
;[7] ------------------->
;THIS FUNCTION DRAWS SEGMENTS IN A WINDOW AND CALLS A FCT TO
; DRAW POINTS AND LABEL


(defun draw-segments (segment-list screen)
 (cond ((null segment-list) nil)
      (t (let* ((segment (car segment-list))
                (firstpt (car segment))
                (secondpt (cadr segment)))
               (moveto (car firstpt)
                       (cadr firstpt)
                       screen)
               (mark-point firstpt screen)
               (lineto (car secondpt)
                  (cadr secondpt)
                  screen)
              (mark-point secondpt screen))
           (draw-segments (cdr segment-list) screen))
         )
  )

;[8] ----------------------->
;THIS FUNCTION MAKES A BLACK CIRCLE FOR A POINT.
(defun mark-point (new-point screen)
  (let ((x (car new-point))
        (y (cadr new-point)))
       (paintoval (list (isub y 2)
                        (isub x 2)
                        (iadd y 2)
                        (iadd x 2))
                  screen)
       (moveto (isub x 4) (isub y 6) screen)
       (drawstring (caddr new-point) screen)
       (moveto x y screen))
  )
;
;[9] ------------------------->
;THIS FUNCTION DRAWS A LIST OF LINES ON A SCREEN Y-DIST APART.

(defun draw-text-lines (s-list screen y-dist)
  (do  ((x-loc (get screen 'x-start))
        (y-loc (get screen 'y-start) (iadd y-dist y-loc))
        (next-string (car s-list) (car rem-list))
        (rem-list (cdr s-list) (cdr rem-list)))
       ((null next-string))
       (moveto x-loc y-loc screen)
       (drawstring next-string screen))
  )
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;PROMPTING UTILITIES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;FOLLOWING ARE FUNCTIONS FOR PROMPTING FROM THE OLD HELP FILES
;
;[10] -------------------->
;this is the controller for showing syntax help based on an index
; choice from a query

(defun show-syntax ()
 (let* ((query "syntax help: CHOOSE NUMBER" )
       (old-prompt (get GL-present-mode 'prompt))
       (help-index 0))
      (setq gl-present-mode 'gl-hsyntax-mode)
      (gl-helpwin 'selectwindow)

      ;THE FOLLOWING SAVES WHAT WAS LEFT WHEN MOUSE CLICKED
      (save-new-string  gl-cwin
                 (arr-to-string gl-cwin)
                 old-prompt )
      (provide-reactions (list query
                    ))
      (apply (GL-option-array 1) nil)
      )
 )
```

```
;[11]--------------->
;;;;;
;the following is the function called from the menu
; to implement the rule-help routines
(defun show-rules ()
 (let* ((query "rule help: CHOOSE NUMBER")
       (old-prompt (get GL-present-mode 'prompt))
       (help-index 0))
      (gl-helpwin 'selectwindow)
      ;THE FOLLOWING SAVES WHAT WAS LEFT WHEN MOUSE CLICKED
      (save-new-string  gl-cwin
                 (arr-to-string gl-cwin)
                 old-prompt )
      (provide-reactions (list query
                         ))
      (apply (GL-option-array 4) nil)
      )
 )
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GENERAL UTILITIES
;;;;;;;;;;;;;;;;;;;;;;
;
;[12]---------------------------->
;***********************************************************
************
;this function changes an array to a string .It is called by
;translate-line
;the array changed is the current input-line of the screen
(defun arr-to-string (screen )
 (let* ((i 0)
      (ch-arr (get screen 'input-line))
      (character (ch-arr i))
      (new-string "")
      )
     (while (not (null character))
          (setq new-string (string-append new-string
                                          character))
          (setq i (add1 i))
          (setq character (ch-arr i))
          )
      new-string)
  )
;
;[13] ------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;THE FOLLOWING FUNCTION CONVERTS THE ARRAY TO A LIST OF
STRINGS
; THIS IS NEEDED TO PARSE THE ASSERTION. called from translate-line

(defun arr-to-list (screen)

  (do ((ch-arr (get screen 'input-line))
      (new-list nil)
      (ind 0 (add1 ind)))
     ((equal (ch-arr ind) nil) new-list)
     (do ((next-char (ch-arr ind) (ch-arr ind))
         (new-string "")
         (new-ell nil)
         (stop nil))
        ((equal stop t) (setq new-list
                      (append  new-list new-ell)))
        (cond ((null next-char)
              (setq stop t)
              (cond ((equal new-string ""))
                   (t (setq new-ell (list new-string)))
                   ))
             ((or
               (equal next-char " ")
               (equal next-char (char 9)))
              (cond ((equal new-string "")
                (setq ind (add1 ind)))
               (t (setq stop t)
                 (setq new-ell (list new-string)))
               ))
           ((member next-char
                GL-all-symbols)
            (cond ((equal new-string "")
                (setq new-ell (list next-char)))
               (t (setq new-list
                      (append new-list
                            (list new-string)))
                 (setq new-ell (list next-char))))
            (setq stop t) )

           (t (setq ind (add1 ind))
             (setq new-string (string-append
                          new-string
                          next-char)))
           )
        )
     )
 )
;
;[14] ----------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;THIS FUNCTION CLEANS OUT THE INPUT ARRAY AND REPLACES WITH
NILS

(defun cleanout-array (screen)
  (let ((cur-loc (get screen 'in-loc))
      (cur-line (get screen 'input-line)))
     (dotimes (index cur-loc)
           (cur-line index nil))
     )
 )
```

```
;
;[15] ------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;THIS FUNCTION SAVES THE NEW STRING IN ITS BUFFER

(defun save-new-string (screen n-string prompt)

 (funcall (get screen 'upkeep)
        screen
        (string-append prompt n-string))
 )
;
;[16] ------------------------------->

; THIS FUNCTION TAKES A MODE AND POPS ITS FIRST STACK ELEMENT
; FROM THE PROPERTY mode-stack
(defun pop-mode-stack (con-mode)
 (let* ((stack-list (get con-mode 'mode-stack))
       (ell (car stack-list))
       (restarter (get ell 'restart)))
      (cond (restarter (setq inside ell)
            (apply restarter nil)))
      (cond ((equal con-mode ell)
           ell)
          (t (putprop con-mode (cdr stack-list) 'mode-stack)
            ell)
          )
      )
 )
 ;
```

```
;[17] ---------------------------->
;;;;;;;;;
; THIS FUNCTION SAME AS ABOVE EXCEPT IS A PUSH
(defun push-mode-stack (new-mode con-mode)
 (let ((stack-list (get con-mode 'mode-stack)))
     (putprop con-mode
             (cons new-mode stack-list)
             'mode-stack)
          )
     )
;
;[18] ------------------------>
;BELOW IS A UTILITY USED BY error-response and provide reason
; to output strings to the gl-cwin
;;;;
(defun provide-reactions (string-list)
 (do ((next-string (car string-list) (car rem-list))
     (rem-list (cdr string-list)(cdr rem-list)))
    ((null next-string))
    (drawstring next-string gl-cwin)
    (save-new-string  gl-cwin next-string "")
    )
 )
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; EVALUATION UTILITIES


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
;[19] -------------------------->
;THE FOLLOWING FUNCTION IS CALLED FROM translate-line
;AFTER AN ASSERTION
;HAS BEEN ENTERED INTO THE SYSTEM
(defun evaluate-assertion (cur-mode)
 (let ((parse-list (get cur-mode 'latest-as-list))
      (new-pce nil)
      (reason-loc nil)) (setq gl-tst parse-list)
     ;;;;;;the following 2 lines set up globals for the parser
     (setq gl-t-list nil)     ;these are two globals used by
                           ;; parser.
     (setq gl-p-list parse-list)
     (setq new-pce (p-assertion))
     (setq gl-test new-pce)
     (cond ((null new-pce)
            nil)
           (t (setq reason-loc
                   (check-validity new-pce))
              (cond ((numberp reason-loc) ;if a number returns
                    ;then it  it was a correct assertion.
                    (cond ((equal gl-present-mode
                                  gl-present-lesson)
                           (assertion-to-proof
                             gl-present-lesson))
                          )
                    (setq gl-present-mode 'GL-reason-mode))
                    ))
           )
     )
 )
;
```

```
;[20] ----------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;THE FOLLOWING FUNCTION ADDS A NEW ASSERTION TO THE PROOF
(defun assertion-to-proof (cur-lesson)
 (let ((pre-string
       (printrep (sub1 (get gl-present-lesson 'solution-step)))))
    (moveto (get gl-bwin 'x-start)
          (get gl-bwin 'y-start)
          gl-bwin)
    (drawstring (string-append
              pre-string
              "> "
              (get gl-present-lesson 'save-line))
             gl-bwin)
    (moveto 95 (get gl-bwin 'y-start) gl-bwin)
    (putprop gl-bwin (iadd (get gl-bwin 'y-start) 17) 'y-start)
    )
 )
;[21] ---------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; the following function is the evaluator for gl-reason-mode
(defun evaluate-reason  (cur-mode)
 (let ((correct-choice (get cur-mode 'correct-response))
     (cur-choice (get cur-mode 'save-line))
     (solution-length (length
                 (car
                   (get gl-present-lesson
                      'assertion-list)))))
    (cond ((equal correct-choice cur-choice)
         (reason-to-proof cur-mode)
         (cond ((< solution-length
                (get gl-present-lesson 'solution-step))
              (provide-reactions (list "EXCELLENT"
                             "YOU SOLVED THE PROBLEM")))
          (t
           (provide-reactions (list "CORRECT"
                                    "write next assertion"))))
       (setq gl-present-mode gl-present-lesson))
      ((equal "O" cur-choice)
       (provide-reactions
         (list "if you don't know the correct answer"
               "    then study the rules in the HELP menu"))
       (push-mode-stack gl-present-mode gl-present-lesson))
      ((member cur-choice
               (get gl-present-lesson 'rules-avail))
       (erase-blackboard)
       (provide-reactions
         (list "At the right is an example"
               "of the rule you chose ====>"
               "WHICH PART DO YOU MEAN"
               "ENTER a letter for side"
               "     L = LEFT"
               "     R = RIGHT"
               "     N = NEITHER"))
       (cs gl-rulewin)
       (apply ((get 'GL-hrules1-mode 'help-array-name)
               (stringtonum cur-choice)) nil)
       (send gl-cwin 'selectwindow)
       ;following will respond to choice of rule used
       (choose-rule-direction cur-mode)
       (apply (GL-option-array 2) nil)
       (provide-reactions '("WHAT IS THE REASON"
                            "FOR YOUR LAST ASSERTION?"))
       )
      (t (provide-reactions '("NOT AN OPTION"
                              "CHOOSE AGAIN")))
      )
    )
)
```

```
;[22] ---------------------->
;the following finds which side of the rule is being used
(defun choose-rule-direction (cur-mode)
 (let ((choice (printrep (readchar)))
      (correct-rule (stringtonum (get cur-mode 'correct-response)))
      (rule-chosen (stringtonum (get cur-mode 'save-line)))
      (side nil)
      (right-inv (cdr GL-INV-RECT))
      (left-inv (car GL-INV-RECT))
      (chosen-inv nil)(left nil)(right nil)
      (pr-high (- (get gl-bwin 'y-start) 32))
      (pr-low (- (get gl-bwin 'y-start) 15))
      (response-out nil))
     (while (not (member choice '("L" "l" "R" "r" "N" "n")))
          (erase-blackboard)
          (provide-reactions
            (list choice
              "is not an option"
              "ENTER a letter for side "
              "     L = LEFT"
              "     R = RIGHT"
              "     N = NEITHER"))
          (setq choice (printrep (readchar)))
          )
     (cond ((member choice '("L" "l" "R" "r"))
          (penpat white gl-helpwin)
          (cond ((member choice '("L" "l"))
                (setq chosen-inv left-inv)
                (setq left -135)(setq right -10)
                (paintrect '(-20 0 60 150) gl-helpwin)
                (setq side 0)
                )
               (t
                (setq chosen-inv right-inv)
          (setq left 10)(setq right 135)
          (paintrect '(-20 -150 60 1) gl-helpwin)
          (setq side 1))
         )
     (penpat black gl-helpwin)
     (erase-blackboard)
     (cond ((= correct-rule 1))
         (t (setq correct-rule (- correct-rule 10))))
     (cond ((= rule-chosen 1))
         (t (setq rule-chosen (- rule-chosen 10))))
     (invertrect (list pr-high -120 pr-low 70) gl-bwin)
     (invertrect (list
                (cadr chosen-inv) left
                (cddr chosen-inv) right) gl-helpwin)
     (setq response-out (GL-WRONG-REASON
                    correct-rule
                     rule-chosen
                     side))
     (provide-reactions (GL-WRONG-RESPONSE
                    response-out)
                     )
     (provide-reactions '("hit space bar to continue"))
     (while (not (keyp))) (readchar)
     (setq response-out (iadd response-out 10))
     (erase-blackboard)
     (invertrect (list pr-high -120 pr-low 70) gl-bwin)
     (invertrect (list
                (cadr chosen-inv) left
                (cddr chosen-inv) right) gl-helpwin)
     (provide-reactions (GL-WRONG-RESPONSE
                    response-out)
                     )
     (provide-reactions '("hit space bar to continue"))
```

```
     (while (not (keyp))) (readchar) ))
    (erase-blackboard))
 )
;
;[23] ------------------------------->
;THE FOLLOWING FCT IS LIKE ASSERTION TO PROOF BUT USED FOR THE
; REASON

(defun reason-to-proof (cur-mode)
 (drawstring (get cur-mode 'correct-response)
        gl-bwin
        )
 )
;
;[24] ----------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;FOLLOWING IS ANOTHER EVALUATION FUNCTION WHICH EVALUATES
; MENU CHOICES
(defun evaluate-menu (cur-mode)
 (let* ((choice (get cur-mode 'save-line)))
    (cond ((equal choice "0")
           (apply ((get GL-present-mode 'help-array-name) 0) nil))
          ((equal choice "9")
           (cs gl-helpwin)
           (gl-helpwin 'setwtitle "GEOMETRY")
            (setq gl-present-mode
                (pop-mode-stack GL-present-lesson)))
          ((member choice
                 (get GL-present-mode 'available-options))
           (send gl-rulewin 'selectwindow)
           (cs gl-rulewin)
           (draw-text-lines
             ((get 'GL-hrules2-mode 'help-array-name)
              (stringtonum choice))
             gl-rulewin 18)
          (send gl-helpwin 'selectwindow)
          (apply ((get GL-present-mode 'help-array-name)
               (stringtonum choice)) nil)
          (provide-reactions '("hit space-bar to continue"))
          (send gl-cwin 'selectwindow)
          (while (not (keyp)))(readchar)
           (apply (get GL-present-mode 'help-choice)
                 nil)

           )
          (t (provide-reactions (list "NOT AN OPTION"
                              "try again =======>")))
          )
     )
 )
;
;[25] ------------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;This FUNCTION EVALUATES TO DISPLAY THE PROPER FORMAL RULE

(defun display-rule (cur-mode)
 (let ((rule-string (get cur-mode 'save-line)))
     (send gl-rulewin 'selectwindow)
     (norecord gl-rulewin)
     (cs gl-rulewin)
     (record gl-rulewin)
     (cond ((equal rule-string "0")
          (provide-reactions '("Statements of Rules"
                         "in the RULE window"
                         "CHOOSE NUMBER ===>")))
          ((equal rule-string "9")
           (cs gl-helpwin)
           (gl-helpwin 'setwtitle "GEOMETRY")
           (setq gl-present-mode
               (pop-mode-stack GL-present-lesson)) )
```

```
        ((member rule-string (get cur-mode 'available-options))
         (draw-text-lines
          ((get cur-mode 'help-array-name)
           (stringtonum rule-string))
          gl-rulewin
          18)
         (cs gl-helpwin)
         (gl-helpwin 'setwtitle "GEOMETRY")
         (provide-reactions '("hit space-bar to continue"))
         (moveto -70 5 gl-helpwin)
         (drawstring "LOOK AT RULE WINDOW" GL-helpwin)
         (send gl-cwin 'selectwindow)
         (while (not (keyp)))(readchar)
         (show-rules)
         )
        (t (provide-reactions '("NOT AND OPTION "
                                "choose another number"
                                )))
        )
     )
 )
;
;[26] ---------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;THIS FUNCTION IS FOR CHECKING THE VALIDITY OF A PCE IN LESSON
ONE
(defun check-validity (new-pce)
  (let* ((mem-list (member new-pce
                          (car (get
                                GL-present-mode
                                'assertion-list))))
        (list-len (length mem-list))
        (ssize-ref  (get GL-present-mode 'solution-size))
        (new-step (isub ssize-ref list-len)))
        (cond ((= new-step (get GL-present-mode 'solution-step))
              (putprop 'GL-reason-mode
                     (nth
                       (sub1 new-step)
                       (cdr (get
                            GL-present-mode
                            'assertion-list)))
                     'correct-response)
              (putprop GL-present-mode
                     (add1 new-step)
                     'solution-step)
              (provide-reasons GL-present-mode)
              new-step)
             ((= new-step ssize-ref)
              (provide-reactions (list "this does not follow"
                                       "TRY AGAIN"))
              nil)
             (t (provide-reactions (list "you jumped a step"
                                         "TRY AGAIN"))
               nil)
             )
        )
    )
;
;[27] ------------------------->
;FOLLOWING IS THE BEGINNING OF THE REASON PROVISION STEP

(defun provide-reasons (lesson)
  (provide-reactions (list "****good choice****"
                    "WHY??  CHOOSE NUMBER ===>"))
  (funcall (GL-OPTION-ARRAY (get lesson 'reason-choice)))

 )
;
;[28] (scratched)
```

```
;FOLLOWING IS THE FUNCTION TO CHECK IF A GIVEN REASON IS
CORRECT

(defun is-reason-correct (selection)
 (cond ((= 1 selection)
      (provide-reactions (list "very good"
                            "what is the next assertion")))
     )

 )
;
;[29] ----------------------->
;THIS FUNCTION IS USED AFTER A HELP SESSION TO  PUT THE USER
; BACK IN THE MODE TO INPUT A REASON FOR AN ASSERTION.
(defun restart-reasons ()
 (provide-reactions (list "REASON FOR LAST ASSERTION?"
                  "CHOOSE NUMBER ======>"))
 (funcall (GL-OPTION-ARRAY (get gl-present-lesson 'reason-choice)))

 )
```

```
                        ;  // SYSTEM-ARRAYS //
  ;--------------------------------------------------------------
;THE FOLLOWING FILE CONTAINS SEVERAL ARRAYS USED THRU THE
SYSTEM
;

;I. HELP FUNCTION ARRAYS
;  [30] GL-option-array
;  [31] GL-hysntax-array
;
;  [32] GL-EXrules-array

;II STRING-ARRAYS
;  A. GL-rule-name
;  B. GL-formal-rule
;
;[30] -------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; GL-OPTION-ARRAY
;THE FOLLOWING CALLS SET UP HELP GLOBALS FOR PARTICULAR RULES
; THAT HELP WITH SYNTAX
(setq GL-OPTION-ARRAY (make-array 5))

(GL-OPTION-ARRAY 1
          (lambda ()
               (cs gl-helpwin)
               (gl-helpwin 'setwtitle "SYNTAX CHOICES")
               (textsize 10 gl-helpwin)
               (draw-text-lines
                '(
                 "         CHOOSE BY NUMBER"
                 ""
                 ""
                 "1  CONGRUENT SEGMENTS"
                 "2  CONGRUENT ANGLES"
```

```
                  "3  MIDPOINT"
                  "4  = SEGMENT MEASURE"
                  "5  = ANGLE MEASURE"
                  ""
                  "9  RETURN-TO LESSON")
                 gl-helpwin
                 11)
                (textsize 12 gl-helpwin)
                (textfont 17 gl-helpwin))
        )



;GL-OPTION ARRAY 2 WILL BE FOUND IN THE PARTICULAR LESSON FILE
; IT IS DEPENDENT ON WHICH LESSON IS IN EFFECT
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;HELP ON RULES
(GL-OPTION-ARRAY 4
        (lambda ()
             (cs gl-helpwin)
             (gl-helpwin 'setwtitle "RULE CHOICES")
             (textsize 10 gl-helpwin)
             (draw-text-lines
               '("      "
                 "       CHOOSE BY NUMBER"
                 ""
                 ""
                 "        *** DEFINITIONS ***"
                 ""
                 "12  SEGMENT BISECTOR"
                 "13  CONGRUENT SEGMENTS"
                 "14  CONGRUENT ANGLES"
                 "15  MIDPOINT"
                 ""
                 "9  RETURN-TO LESSON")
                gl-helpwin
                11)
              (textsize 12 gl-helpwin)
              (textfont 17 gl-helpwin))
        )
;
;[31] ------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GL-HSYNTAX-ARRAY
;;;;;;;;;;;;;;;;;;

(setq GL-HSYNTAX-ARRAY (make-array 10))

;FOLLOWING IS SYNTAX HELP FOR CONGRUENT SEGMENTS
(GL-HSYNTAX-ARRAY 0
          (lambda ()
            (provide-reactions
              (list "GRAMMAR HELP"
                  "To write proper grammar "
                  "You must use a language "
                  "the computer understands!"
                  "Choose a number at right"
                  "for help ===>")
                  )))
(GL-HSYNTAX-ARRAY 1
          (lambda ()
               (cs gl-helpwin)
               (gl-helpwin 'setwtitle "CONGR. SEGMENT SYNTAX")
               (draw-segments
                '(((-80 -30 "R") (-15 -30 "S"))
                  ((15 -30 "T") (80 -30 "U")))
                GL-helpwin)
               (moveto -85 0 gl-helpwin)
               (drawstring "TO ENTER:" gl-helpwin)
               (moveto -75 12 gl-helpwin)
```

```
                (textsize 10 gl-helpwin)
                (drawstring
                 "SEGMENT RS IS CONGRUENT"
                 gl-helpwin)
                (moveto -60 22 gl-helpwin)
                (drawstring
                 "TO SEGMENT TU"
                 gl-helpwin)
                (textsize 12 gl-helpwin)
                (textfont 17 gl-helpwin)
                (moveto -85 40 gl-helpwin)
                (drawstring "WRITE:" gl-helpwin)
                (moveto -40 60 gl-helpwin)
                (drawstring "[RS / [TU" gl-helpwin)
                )
          )

;FOLLOWING IS SYNTAX HELP FOR CONGRUENT ANGLES
(GL-HSYNTAX-ARRAY 2
          (lambda ()
                (gl-helpwin 'setwtitle "CONGR. ANGLE SYNTAX")
                (cs gl-helpwin)
                (draw-segments
                '(((-80 -22 "K") (-15 -22 "L"))
                  ((-80 -22 "K") (-20 -52 "J"))
                  ((15 -22 "Q") (80 -22 "R"))
                  ((15 -22 "Q") (75 -52 "P")))
                GL-helpwin)
                (moveto -85 0 gl-helpwin)
                (drawstring "TO ENTER:" gl-helpwin)
                (moveto -75 12 gl-helpwin)
                (textsize 10 gl-helpwin)
                (drawstring
                 "ANGLE JKL IS CONGRUENT"
                 gl-helpwin)
                (moveto -60 22 gl-helpwin)
                (drawstring
                  "TO ANGLE PQR"
                  gl-helpwin)
                (textsize 12 gl-helpwin)
                (textfont 17 gl-helpwin)
                (moveto -85 40 gl-helpwin)
                (drawstring "WRITE:" gl-helpwin)
                (moveto -40 60 gl-helpwin)
                (drawstring ",JKL / ,PQR" gl-helpwin)
                )
           )


;FOLLOWING IS SYNTAX HELP FOR THE MIDPOINT
(GL-HSYNTAX-ARRAY  3
           (lambda ()
                 (gl-helpwin 'setwtitle "MIDPOINT SYNTAX")
                (cs gl-helpwin)
                (draw-segments
                 '(((-55 -30 "X") (0 -30 "Y"))
                   ((0 -30 "Y") (55 -30 "Z")))
                 GL-helpwin)
                (moveto -85 -5 gl-helpwin)
                (drawstring
                  "TO ENTER:"
                  gl-helpwin)


                (moveto -75 10 gl-helpwin)
                (drawstring
                  "Point Y is the midpoint"
                  gl-helpwin)
                (moveto -75 25 gl-helpwin)
                (drawstring
                 "of segment XZ"
                 gl-helpwin)
                (moveto -85 43 gl-helpwin)
                (drawstring "WRITE: " GL-helpwin)
                (moveto -75 60 gl-helpwin)
                (drawstring
                 "Y IS THE MIDPOINT OF [XZ" gl-helpwin)
                )
            )

;FOLLOWING IS SYNTAX HELP FOR = SEGMENT MEASURE
(GL-HSYNTAX-ARRAY  4
            (lambda ()
                 (gl-helpwin 'setwtitle "EQUAL SEGMENT SYNTAX")
                (cs gl-helpwin)
                (draw-segments
                 '(((-80 -45 "R") (-15 -45 "S"))
                   ((15 -45 "T") (80 -45 "U")))
                 GL-helpwin)
                (moveto -85 0 gl-helpwin)
                (drawstring "TO ENTER:" gl-helpwin)
                (moveto -75 12 gl-helpwin)
                (textsize 10 gl-helpwin)
                (drawstring
                 "SEGMENT RS HAS LENGTH"
                 gl-helpwin)
                (moveto -60 22 gl-helpwin)
                (drawstring
                 "EQUAL TO SEGMENT TU"
                 gl-helpwin)
                (textsize 12 gl-helpwin)
                (textfont 17 gl-helpwin)
                (moveto -85 40 gl-helpwin)
                (drawstring "WRITE:" gl-helpwin)
```

```
          (moveto -40 60 gl-helpwin)
          (drawstring "RS = TU" gl-helpwin)
          )
     )

;FOLLOWING IS SYNTAX HELP FOR = ANGLE MEASURE
(GL-HSYNTAX-ARRAY 5
     (lambda ()
          (gl-helpwin 'setwtitle "EQUAL ANGLE SYNTAX")
          (cs gl-helpwin)
          (draw-segments
           '(((-80 -22 "K") (-15 -22 "L"))
            ((-80 -22 "K") (-20 -52 "J"))
            ((15 -22 "Q") (80 -22 "R"))
            ((15 -22 "Q") (75 -52 "P")))
           GL-helpwin)
          (moveto -85 0 gl-helpwin)
          (drawstring "TO ENTER:" gl-helpwin)
          (moveto -75 12 gl-helpwin)
          (textsize 10 gl-helpwin)
          (drawstring
            "THE MEASURE OF ANGLE JKL "
            gl-helpwin)
          (moveto -60 22 gl-helpwin)
          (drawstring
            "AND ANGLE PQR ARE EQUAL"
            gl-helpwin)
          (textsize 12 gl-helpwin)
          (textfont 17 gl-helpwin)
          (moveto -85 40 gl-helpwin)
          (drawstring "WRITE:" gl-helpwin)
          (moveto -40 60 gl-helpwin)
          (drawstring "M,JKL = M,PQR" gl-helpwin)
          )
     )
```

```
;FOLLOWING IS help to return to the system from help on syntax
(GL-HSYNTAX-ARRAY 9
          (lambda ()
               (gl-helpwin 'setwtitle "GEOMETRY")
               (pop-mode-stack gl-present-lesson))
          )
;
;[32] ----------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;FOLLOWING ARE FUNCTIONS FOR HELP ON RULES


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;GL-EXRULES-ARRAY
;;;;;;;;;;;;;;;;

(setq GL-EXrules-array (make-array 20))

;THIS ELEMENT GIVES YOU HELP ON RULES

(GL-EXrules-array 0
        (lambda ()
              (provide-reactions
                '("RULE HELP"
                  "You must first select the"
                  " rule which you want help on!"
                  "CHOOSE A NUMBER ===>")))
        )
;THIS IS HELP ON THE GIVEN RULE
(GL-EXrules-array 1
        (lambda ()
              (gl-helpwin 'setwtitle "GIVEN")
              (cs gl-helpwin)
```

```
                (moveto -140 -30 gl-helpwin)
                (drawstring
                  "YOU CHOSE GIVEN " GL-helpwin)
                (drawstring "LOOK AT " gl-helpwin)
                (drawstring "YOUR PROOF" gl-helpwin)
                )
            )

;THIS ELEMENT DROPS YOU BACK TO THE LESSON FROM HELP ON RULES

(GL-EXrules-array 9 (lambda ()
            (pop-mode-stack gl-present-lesson)
            (setq gl-present-mode
                (pop-mode-stack gl-present-lesson)))
              )

;FOLLOWING IS RULE HELP FOR SEGMENT BISECTOR
(GL-EXRULES-ARRAY  12
            (lambda ()
                (setq GL-INV-RECT '(((-5 . 9) . (28 . 57)) .
                              ((-5 . 9) . (40 . 57))))
                (gl-helpwin 'setwtitle "SEGMENT BISECTOR RULE")
                (cs gl-helpwin)
                (draw-segments
                  '(((-60 -35 "R") (0 -35 "S"))
                    ((0 -35 "S") (60 -35 "T")))
                  GL-helpwin)
                (moveto 0 -0 gl-helpwin)
                (lineto 0 50 gl-helpwin)
                (textfont 0 gl-helpwin)
                (moveto -140 -8 gl-helpwin)
                (drawstring "GIVEN:" gl-helpwin)
                (moveto -140 25 gl-helpwin)
                (drawstring "CONCLUDE:" gl-helpwin)
                (moveto 20 -8 gl-helpwin)
                (drawstring "GIVEN:" gl-helpwin)
                (moveto 20 40 gl-helpwin)
                (drawstring "CONCLUDE:" gl-helpwin)
                (textfont 17 gl-helpwin)
                (moveto -130 7 gl-helpwin)
                (drawstring "S BISECTS [RT" gl-helpwin)
                (moveto -130 38 gl-helpwin)
                (drawstring "S IS THE " gl-helpwin)
                (moveto -130 53 gl-helpwin)
                (drawstring "MIDPOINT OF [RT" gl-helpwin)
                (moveto 30 7 gl-helpwin)
                (drawstring "S IS THE " gl-helpwin)
                (moveto 30 22 gl-helpwin)
                (drawstring "MIDPOINT OF [RT" gl-helpwin)
                (moveto 30 55 gl-helpwin)
                (drawstring "S BISECTS [RT " gl-helpwin)

               )
         )

(GL-EXrules-array 13
         (lambda ()
                (setq GL-INV-RECT '(((-5 . 9) . (40 . 65)) .
                                ((-5 . 9) . (40 . 65))))
                (gl-helpwin 'setwtitle "CONGRUENT SEGMENT RULE")
               (cs gl-helpwin)
               (draw-segments
                '(((-80 -30 "R") (-15 -30 "S"))
                  ((15 -30 "T") (80 -30 "U")))
                GL-helpwin)
               (moveto 0 -0 gl-helpwin)
               (lineto 0 50 gl-helpwin)
               (textfont 0 gl-helpwin)
               (moveto -110 0 gl-helpwin)
               (drawstring "GIVEN:" gl-helpwin)
```

```
                (moveto -110 40 gl-helpwin)
                (drawstring "CONCLUDE:" gl-helpwin)
                (moveto 15 0 gl-helpwin)
                (drawstring "GIVEN:" gl-helpwin)
                (moveto 15 40 gl-helpwin)
                (drawstring "CONCLUDE:" gl-helpwin)
                (textfont 17 gl-helpwin)
                (moveto -75 20 gl-helpwin)
                (drawstring "RS=ST" gl-helpwin)
                (moveto -75 60 gl-helpwin)
                (drawstring "[RS / [TU" gl-helpwin)

                (moveto 30 20 gl-helpwin)
                (drawstring "[RS / [TU" gl-helpwin)
                (moveto 30 60 gl-helpwin)
                (drawstring "RS=ST" gl-helpwin)
                )
        )


;FOLLOWING IS SYNTAX HELP FOR CONGRUENT ANGLES
(GL-EXRULES-ARRAY  14
        (lambda ()
                (setq GL-INV-RECT '(((-5 . 9) . (40 . 65)) .
                                    ((-5 . 9) . (40 . 65))))
                (cs gl-helpwin)
                (gl-helpwin 'setwtitle "CONGRUENT ANGLE RULE")
                (draw-segments
                 '(((-80 -22 "K") (-15 -22 "L"))
                   ((-80 -22 "K") (-20 -52 "J"))
                   ((15 -22 "Q") (80 -22 "R"))
                   ((15 -22 "Q") (75 -52 "P")))
                 GL-helpwin)
                (moveto 0 -0 gl-helpwin)
                (lineto 0 50 gl-helpwin)
```

```
            (textfont 0 gl-helpwin)
            (moveto -140 0 gl-helpwin)
            (drawstring "GIVEN:" gl-helpwin)
            (moveto -140 35 gl-helpwin)
            (drawstring "CONCLUDE:" gl-helpwin)

            (moveto 15 0 gl-helpwin)
            (drawstring "GIVEN:" gl-helpwin)
            (moveto 15 35 gl-helpwin)
            (drawstring "CONCLUDE:" gl-helpwin)

            (textfont 17 gl-helpwin)
            (moveto -130 15 gl-helpwin)
            (drawstring "M,JKL=M,PQR" gl-helpwin)
            (moveto -130 55 gl-helpwin)
            (drawstring ",JKL / ,PQR" gl-helpwin)
            (moveto 25 15 gl-helpwin)
            (drawstring ",JKL / ,PQR" gl-helpwin)
            (moveto 25 55 gl-helpwin)
            (drawstring "M,JKL=M,PQR" gl-helpwin)
            )
        )


;FOLLOWING IS SYNTAX HELP FOR MIDPOINT
(GL-EXRULES-ARRAY 15
        (lambda ()
            (setq GL-INV-RECT '(((-5 . 9) . (45 . 65)) .
                                ((-5 . 9) . (27 . 65))))
            (gl-helpwin 'setwtitle "MIDPOINT RULE")
            (cs gl-helpwin)
            (draw-segments
             '(((-60 -35 "M") (0 -35 "N"))
               ((0 -35 "N") (60 -35 "P")))
```

```
                 GL-helpwin)
                (moveto 0 -0 gl-helpwin)
                (lineto 0 50 gl-helpwin)
                (textfont 0 gl-helpwin)
                (moveto -140 -8 gl-helpwin)
                (drawstring "GIVEN:" gl-helpwin)
                (moveto -140 40 gl-helpwin)
                (drawstring "CONCLUDE:" gl-helpwin)
                (moveto 20 -8 gl-helpwin)
                (drawstring "GIVEN:" gl-helpwin)
                (moveto 20 25 gl-helpwin)
                (drawstring "CONCLUDE:" gl-helpwin)
                (textfont 17 gl-helpwin)
                (moveto -130 7 gl-helpwin)
                (drawstring "N IS THE " gl-helpwin)
                (moveto -130 22 gl-helpwin)
                (drawstring "MIDPOINT OF [MP" gl-helpwin)
                (moveto -130 60 gl-helpwin)
                (drawstring "MN = NP" gl-helpwin)
                (moveto 30 7 gl-helpwin)
                (drawstring "MN = NP" gl-helpwin)
                (moveto 30 38 gl-helpwin)
                (drawstring "N IS THE " gl-helpwin)
                (moveto 30 53 gl-helpwin)
                (drawstring "MIDPOINT OF [MP" gl-helpwin)
                )
        )

(GL-EXRULES-ARRAY 16
        (lambda ()
                (gl-helpwin 'setwtitle "ALGEBRA RULES")
                (cs gl-helpwin)
                (moveto -80 0 gl-helpwin)
                (textsize 18 gl-helpwin)
```

```
          (drawstring "NOT YET IMPLEMENTED" gl-helpwin)
          (textsize 12 gl-helpwin)
          )
     )

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;FOLLOWING SETS UP AN ARRAY OF RULES DESCRIBING * AND NAME
;;;;;
;THE FOLLOWING ARRAY IS A LIST OF FORMAL RULES
(setq GL-formal-rule (make-array 20))

(GL-formal-rule 12
  '("A Bisector of a segment is a set of points"
    "that intersects the segment at its midpoint"))


(GL-formal-rule 13
  '("Congruent segments are segments"
    "that have the same length"))

(GL-formal-rule 14
  '("Congruent angles are angles"
    "that have the same measure"))

(GL-formal-rule 15
  '("The Midpoint of segment [MP "
    "is a point N such that MN = NP" ))

;FOLLOWING IS A DOUBLEY INDEXED SET OF HEURISTIC MESSAGES
; FOR WRONG REASONS GIVEN FOR ASSERTIONS

(setq GL-WRONG-REASON (make-array '(7 7 2)))
```

```
(GL-wrong-reason 1 2 0 0)
(GL-wrong-reason 1 3 0 4)
(GL-wrong-reason 1 4 0 4)
(GL-wrong-reason 1 5 0 4)
(GL-wrong-reason 1 6 0 9)

(GL-wrong-reason 1 2 1 4)
(GL-wrong-reason 1 3 1 4)
(GL-wrong-reason 1 4 1 4)
(GL-wrong-reason 1 5 1 0)
(GL-wrong-reason 1 6 1 9)

(GL-wrong-reason 3 1 0 1)
(GL-wrong-reason 3 2 0 2)
(GL-wrong-reason 3 4 0 2)
(GL-wrong-reason 3 5 0 2)
(GL-wrong-reason 3 6 0 9)

(GL-wrong-reason 3 1 1 1)
(GL-wrong-reason 3 2 1 2)
(GL-wrong-reason 3 4 1 2)
(GL-wrong-reason 3 5 1 2)
(GL-wrong-reason 3 6 1 9)

(GL-wrong-reason 5 1 0 1)
(GL-wrong-reason 5 2 0 2)
(GL-wrong-reason 5 3 0 2)
(GL-wrong-reason 5 4 0 2)
(GL-wrong-reason 5 6 0 9)

(GL-wrong-reason 5 1 1 1)
(GL-wrong-reason 5 2 1 2)
(GL-wrong-reason 5 3 1 3)
(GL-wrong-reason 5 4 1 2)
(GL-wrong-reason 5 6 1 9)
```

```
(setq GL-WRONG-RESPONSE (make-array 20))

(GL-WRONG-RESPONSE 0 '("PRETTY GOOD! "
                "Your new assertion"
                "matched the CONCLUSION"
                "of this rule ========>"
                "" ))

(GL-WRONG-RESPONSE 1 '("Was your last assertion"
                "Given in the problem?"
                ""
                "LOOK IN THE PROOF WINDOW"
                "AT YOUR ASSERTIONS"
                ))

(GL-WRONG-RESPONSE 2 '("Look at your assertion "
                "in the PROOF window"

                "Look at the CONCLUSION"
                "in the rule ===========>"
                "THEY DO NOT MATCH!"
                ""))

(GL-WRONG-RESPONSE 3 '("PRETTY GOOD TRY"
                "Your new assertion"
                "matched the CONCLUSION"
                "of this rule ========>"
                ""))


(GL-WRONG-RESPONSE 4 '("Look at your assertion "
                "in the PROOF window"

                "Look at the CONCLUSION"
```

```
                    "in the rule ===========>"
                    "THEY DO NOT MATCH!"
                    ""))

(GL-WRONG-RESPONSE 9 '("NOT YET IMPLEMENTED"))


(GL-WRONG-RESPONSE 10 '("HOWEVER"
                    "The GIVEN of this rule"
                    "has not been proven"

                    "HINT:"
                    "See the Problem GIVEN!"
                    ""))

(GL-WRONG-RESPONSE 12 '("HINT:"
                    "Look at rules that"
                    "have the same CONCLUSION"
                    "as your new assertion"
                    ""))

(GL-WRONG-RESPONSE 13 '(
                    "HOWEVER You have not proven"

                    "that GIVEN!! ===============>"

                    ""

                    "HINT: match earlier assertions "
                    "to the GIVEN of a rule"
                    ""))

(GL-WRONG-RESPONSE 14 '("HINT:"
                    "Look at the GIVEN"
                    "in the DRAWING window"
                    ""))
```

```
;--------------------------/D-INPUT-OUTPUT/-----------------

;FUNCTIONS WRITTEN IN THIS FILE FOR INPUT-OUTPUT

;   [33] fill-screen
;   [34] refresh
;   [35] rubout-character
;   [36] read-line
;   [37] event-control
;   [38] translate-line

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;[33] ------------------------------------>

;THIS FUNCTION SAVES A NEW STRING IN THE ARRAY OF STRINGS FOR
; REFRESHING THE SCREEN
;it puts the first 8 lines in the screen on startup

(defun fill-screen (screen save-string)
  (let ((a-loc (get screen 'array-loc))
        (size (get screen 'size)))
       ((get gl-curwin 'lines) a-loc save-string)
       (setq a-loc (add1 a-loc))
       (putprop screen  a-loc 'array-loc)
       (moveto (get screen 'x-start)
         (+ (cadr (penpos screen)) 17)
         screen)
       (cond
        ((> a-loc  size)
         (putprop screen 'refresh 'upkeep)
         (putprop screen  (remainder a-loc (add1 size))'array-loc))
        )
       )
  )
```

```
;
;[34] ----------------------------------->
;THIS FUNCTION DOES SCREEN REFRESH AND SCROLLING FOR A SCREEN
; GIVEN AS AN ARG.  IT ALSO SAVES NEW STRING IN ARRAY OF STRINGS
; FOR REFRESHING THE SCREEN.

(defun refresh (screen new-string)
  (cs screen)

  (do ((x-loc (get screen 'x-start))
       (y-loc  (get screen 'y-start) (+ y-loc 17))
       (location 0 (add1 location))    ;curline array location
       (new-loc (get screen 'array-loc))
       (a-loc (remainder (add1 (get screen 'array-loc))
                    (add1(get screen 'size)))
           (remainder (add1 a-loc) size))
       (size (add1(get screen 'size)))
       (last-loc  (get screen 'size))  ;lastline array location
       )
      ((> location last-loc)
       (putprop screen  a-loc 'array-loc)
       (moveto x-loc y-loc screen)
        )
      (cond ((= location 0)
           ((get screen 'lines)
            new-loc
            new-string) ))
      (moveto x-loc y-loc screen)
      (drawstring ((get screen 'lines)
                a-loc)
               screen)
      )
  )
;
```

```
;[35] ---------------------------->

;THE FOLLWOING FUNCTION IS CALLED FROM read-line TO RUBOUT A
CHARACTER
;FROM THE INPUT (BLACKBOARD) WINDOW.

(defun rubout-character (screen cur-line line-pos last-loc)
  (let ((rubout-pos (line-pos last-loc)))

       (putprop screen last-loc 'in-loc )
       (penpat white screen)
       (paintrect (list (- (cadr rubout-pos)
                       (cond ((member (cur-line last-loc)
                                      '("[" "]"))
                              15)
                             (t 10)))
                     (car rubout-pos)
                     (+ (cadr rubout-pos) 5)
                     (+ (car rubout-pos) 30))
                  screen
                  )
       (penpat black screen)
       (moveto (car rubout-pos) (cadr rubout-pos) screen)
       (cur-line last-loc nil)
       (line-pos last-loc nil)
       nil)
  )
;
;[36] ------------------------->

;this is an implementation of read-line which accepts a
;character (it is a string) and updates the current new line for
;a given window
;it also does backspace
```

```
(defun read-line (screen new-char)
 (let*(
    (string-char  new-char)
    (cur-line (get screen 'input-line))  ;THIS IS ARRAY WITH THE
    (line-pos (get screen 'input-xval))   ; CURRENT LINE IN IT
    (cur-loc (get screen 'in-loc))
    (last-loc (sub1 cur-loc))
    )
    (uprstring string-char t)
    (cond
      ;this is the end of line
     ((equal string-char (char 13))
      (translate-line screen)
      )

     ;this is the rubout to delete a character
     ((and (equal string-char (char 8)) (>  cur-loc 0))
      (rubout-character screen cur-line line-pos last-loc))

     ;this enters a character and displays on screen
     ((not (equal string-char (char 8)))
      (cur-line cur-loc new-char)
      (line-pos cur-loc (penpos screen))
      (putprop screen  (add1 cur-loc) 'in-loc)
      (drawstring  string-char screen)
      nil)
      ))
 )
;
```

```
;[37] ------------------------------->
;**************************************************************
***********

;this is the event loop for the system

(defun event-control ()
 (initcursor)

 (gl-curwin 'selectwindow)
 (drawstring (get gl-present-mode 'prompt) gl-curwin)
 (setq x 1)
 (setq gl-menu-call nil)
 (while (not  gl-menu-call)
       (cond ((button)
             (setq gl-m-pos (getmouse))
             (cond ((member gl-present-mode gl-help-avail)
                   (cond ((not(null gl-first-click))
                         ;first click since toplevel
                         (cond ((and (> -320 (cadr gl-m-pos))
                                    (< -250 (car gl-m-pos))
                                    (> -140 (car gl-m-pos)))
                                (setq gl-menu-call gl-m-pos))))
                        ((equal gl-curwin gl-cwin)
                          ;currently blackboard
                          (cond ((and (> -250 (cadr gl-m-pos))
                                     (< 100 (car gl-m-pos))
                                     (> 210 (car gl-m-pos)))
                                 (setq gl-menu-call gl-m-pos))
                                ))))

                  (t (provide-reactions
                      '("called for help"))))
            (cond ((and gl-menu-call
                       (not (member   ;its not already on stack
                              gl-present-mode
                              (get gl-present-lesson
                                 'mode-stack))))
                        (push-mode-stack
                          gl-present-mode
                          gl-present-lesson))
                        ))
                   ((keyp)
                    (setq gl-first-click nil)
                    (read-line gl-curwin (printrep (read-char)))

                    )
                   )
               )
           )
;
;[38] ------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;THE FOLLOWING FUNCTIONS TRANSLATE KEYBOARD INPUT
;THEY ARE CONTOLLED BY THE FIRST FUNCTION translate-line
; WHICH IS CALLED FROM read-line two functions above

(defun translate-line (screen)
  ; the following will put redundant info in gl-reason-mode plist
  (putprop gl-present-mode (arr-to-list screen) 'latest-as-list)
  (let ((new-string (arr-to-string screen)))
      ;;;;;;;;;;;;;;;
      ;the following call saves the line in the pres mode object
      ; this may be redundant EG. GL-REASON-MODE
      (putprop gl-present-mode new-string 'save-line)
      (save-new-string screen new-string
                (get GL-present-mode 'prompt))
      (cleanout-array screen)  ;this will set input-array to nils
      (putprop screen  0 'in-loc )
      (apply (get GL-present-mode 'evaluator )
        (list GL-present-mode))
    ;;;the following call returns the next prompt for whatever
    ; is now the present mode which may have changed in the
    ; evaluator of the mode
    (drawstring (get GL-present-mode 'prompt) screen)
    )
)
```

```
                    ;   //  PARSER  //
;FOLLOWING IS A LIST OF THE FUNCTIONS IN THIS FILE

;    [39] get-token
;    [40] peek-token
;    [41] c-membs
;    [42] check-members
;    [43] next-word
;    [44] peek-word
;    [45] make-token
;    [46] new-token
;    [47] is-numb
;    [48] p-identifier
;    [49] p-object
;    [50] addition-error
;    [51] combop-check
;    [52] p-simple-expression
;    [53] p-expression
;    [54] p-assertion
;    [55] p-error
;    [56] error-response
;    [57] find-is-token
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;THESE CALLS establishe the global list for new token
;THESE ARE THINGS NEEDED FOR THE PARSER

 (setq gl-t-list nil)
 (setq gl-p-list nil)
 (setq gl-points '("A" "B" "C" "D" "E" "F"))
 (setq GL-all-symbols  '("," "." "/" ";" "'" "[" "]" "=" "-" "+"))
 (setq gl-figures '("," "[" "]"  "."))
 ;(setq gl-properties '("supplementary" "complementary"
                          "adjacent"))
 (setq gl-relationals '("=" "/" ";" "'"  "bisects"))
 (setq gl-combinationals '("+" "-"))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;[39] --------------------->


;this function get-token changes the global t-list
;it returns the first element of the t-list after getting
necessary tokens

(defun get-token ()
  (let ((nt-list 1))
      (cond ((null gl-t-list)
              (setq nt-list (new-token))
              (setq gl-t-list (append gl-t-list nt-list)))
       )
      (cond ((null nt-list)
               nil)
             (t
              (prog1 (car gl-t-list)
                     (setq gl-t-list (cdr gl-t-list))))
                                                       )
    )
  )
;
;[40] ---------------------------->
;this function returns the next token but does not remove it
from the token list

(defun peek-token ()
  (cond ((null gl-t-list)
         (setq gl-t-list (append gl-t-list (new-token))))
    )
```

```
 (cond ((null gl-t-list)
        nil)
     (t (car gl-t-list))
  )
 )
;
;[41] ----------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;




;this function is a recursive check on a list of atoms to see if
they
;are all contained in another list of atoms

(defun c-membs (id-list g-el-list)
 (cond ((null id-list)
         nil)
        ((member (car id-list) g-el-list)
               (cons (car id-list)
          (c-membs (cdr id-list) g-el-list)))
         (t (let ((string-list nil))

          (setq string-list (cons
                    (string-append
                     (GL-error-message 0)
                     (printrep (car id-list))
                     (GL-error-message 2))
                    string-list))
          (setq string-list (cons (GL-error-message 1)
                      string-list))
          (error-response (reverse string-list))
          )
        'error)
   )
  )
;
;[42] ---------------------------->
;this function is the superior of
;c-membs and sets up to check the elements
;of the first list against the elements of the second list

(defun check-members (id-string g-el-list)
  (let ((id-list nil))
      (cond ((or (equal "" id-string)
                 (null g-el-list))
               (error-response (list
                                "error parsing check-members" ))
             )
                                                       (t
               (setq id-list (explode  id-string))
               (c-membs id-list g-el-list))
            )
      )
  )
;
;[43] ------------------------------->

;the two following functions are simple input to be used by
new-token

;the following function currently returns a single string
;from the gl-p-list
(defun next-word ()
  (prog1 (car gl-p-list)
         (setq gl-p-list (cdr gl-p-list))
                                                       )
  )
```

```
;
;[44] ------------------------------->
;returns a single string currently on the gl-p-list

(defun peek-word ()   ;POSSIBLE BUGGGGGGGG
  (car gl-p-list))
;
;[45] ---------------------------------->
;this function makes a token given the token type and the new
word
;however the word must have a property that tells the atom
name

;this function is set up to return tokens in the form of dotted
;pairs , the first being the token the second being the
represetation

(defun make-token (token-type word)
  (list (cons token-type
          (cond ((equal word ",")
                 'angle)
                ((equal word ".")
                 'triangle)
                ((equal word "[")
                 'segment)
                ((equal word "]")
                 'line)
                ((equal word "+")
                 'plus)
                ((equal word "-")
                 'minus)
                ((equal word "=")
                 'equal)
                ((equal word "/")
                 'congruent)
                ((equal word ";")
                 'perpendicular)
                ((equal word "")
                 'parallel)
                ((equal word "BISECTS")
                 'bisects)

                ))
          )
   )
;
;[46] ------------------------->
;this is the new-token function that returns the next token to
be
;parsed
;from the list of words from the assertion
;it uses several globals prefixed by gl
;it gets a word from next word which takes words off of the
;assertion

(defun new-token()
  (let* ((new-word (next-word))
         (points nil))
      (cond ((null new-word)
               nil)

              ((is-nump new-word)
               (list (cons 'numb (StringToNum new-word))))
             ((member new-word gl-figures)
               (make-token 'g-e new-word))

;((member new-word gl-properties)
;(list (cons 'property (intern new-word))))

            ((member new-word gl-combinationals)
      (make-token 'c-o new-word))

 ((member new-word gl-relationals)
   (make-token 'r-o new-word))

 ((equal new-word "IS")
      (find-is-token))
  ((and (or (equal new-word '"m")
              (equal new-word '"M"))
              (equal (peek-word) '","))
       ;this should take care of measure.
    (list (cons 'measure (intern new-word))))

    ((setq points (check-members new-word gl-points))
 (list (cons 'id points)))


  (t
    nil)

       )
      )
   )
;
;[47] ------------------------------->
;this function determines if a string is a number
; it is used by new-token
; it returns t if it is a number and nil if not

(defun is-nump (str)
  (let ((front (schar str))
       (back nil))
      (cond ((equal str "")
            -999)
          ((member front'("0" "1" "2" "3" "4" "5" "6" "7" "8" "9"))
          (setq back (is-nump (string-butfirst str)))
          (cond ((numberp back)
                 t)
                ((null back)
                 nil)
                (t)))
          (t nil))
      )
  )
;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;[48] ---------------------------->
; this function returns a predicate list beginning with the
; predicate = figure
; arguments = points of the figure
; the m-sign is the measure sign
;from the superior function p-object


(defun p-identifier (figure m-sign)
  (cond ((equal (car (peek-token)) 'id)
      (let* ((id-list (cdr (get-token)))
             (letter-count (length id-list))
             )
          (cond ((null id-list)
                 nil)
                (t
                  (cond ((and (equal figure 'triangle)
                              (= letter-count 3)
                              (null m-sign))
                         (cons figure id-list))
                        ((and (or
                              (equal figure 'segment)
                              (equal figure 'line))
                            (= letter-count 2))
                        (cons figure id-list))
                       ((and (equal figure 'angle)
                            (or (= letter-count 1)
                               (= letter-count 3)))
                        (cons figure id-list))
                       ((and (equal figure 'point)
                            (= letter-count 1)
                            (null m-sign))
                        id-list)
                       (t (p-error 'identifer figure)
                         nil)
                       )
                   )
                 )
             )
       )
      (t
       nil)
      )
  )

;
;[49] -------------------------------->
;this function parses for an object
;it takes no parameters
;it returns a list which describes the object

(defun p-object ()
  (let ((o-list nil)
        (measure-sign (get-token))
        (figure nil))
       (cond ((equal (car measure-sign) 'measure)
        (setq figure (get-token)))
       (t
        (setq figure measure-sign)
        (setq measure-sign nil))
       )
 (cond ((equal (cdr figure) 'error)
        nil)

       ((numberp (cdr figure))
        (setq o-list (cdr figure)))
       ((equal (car figure) 'g-e)
        (setq o-list (p-identifier (cdr figure)
                                   (cdr measure-sign))))
       ((and (= (length (cdr figure)) 1)
             (equal (car figure) 'id))
        (setq o-list (cadr figure)))
       ((equal (car figure) 'id)
                                 ;this happens
                                 ;when you have
                                 ;the syntax that allows
                                 ;the measure of a
                                 ;line segment
                                 ;eg
             (setq measure-sign '(measure . m))
             (setq o-list (cons 'segment (cdr figure))))
       (t
        (error-response (list (GL-error-message 3)
                              (GL-error-message 1)))
        nil)
       )
 (cond ((null o-list)
        nil)
       ((null measure-sign)
        o-list)
       (t (list (car measure-sign) o-list)))  ) )
```

```
;
;[50] ---------------------------->
;this function is an error handler for simple expression
;it takes care of the case when an assertion trys to add two
;incompatible numbers

(defun addition-error (long-list short-list)
  (princ long-list)
  (terpri)
  (princ short-list)
  (terpri))
;
;[51] ------------------------------>
;this function makes sure that comb-op is combining
compatible
;expressions

(defun combop-check (comb-op se-list new-obj)
  (let* ((first-se (car se-list))
        (op-sign (cond ((equal first-se 'plus)  ;VERY IMPORTANT
                   ;THIS REPLACES A GLOBAL
                   ;SO WATCH OUT HERE

                    first-se)))
                                 ;if the first element
                                  ;of se-list
                                 ;is a comb-op then it is given
                                  ;to op-sign
        (first-meas (cond ((null op-sign)          ;if no
                                    ;comb-op sign
                                    ;then must be
                                    ;'measure
                          first-se)
                        ((equal first-se 'plus)
                         (caadr se-list)))))
```

```
                ;SAME GLOBAL PROBLEM
                ; AS ABOVE.

(cond ((not (equal first-meas 'measure))
       (princ (cadr se-list))
       (princ " is not a measured number")
       (terpri))
      ((not (equal first-meas 'measure))
       (princ (cadr new-obj))
       (princ " is not a measured number")
       (terpri))
      (t
        (let ((second-meas (car new-obj))
              (first-geo (cond ((null op-sign)
                                (caadr se-list))
                               (t (caadadr se-list))))
              (second-geo (caadr new-obj)))
             (cond ((and (equal first-meas 'measure)
                         (equal second-meas 'measure)
                         (equal first-geo second-geo)
                         )
                    (list comb-op new-obj se-list))
                   ((null new-obj)
                    (setq se-list nil))
                   (t
                     (princ "Those numbers added ")
                     (princ "are not the same.")
                     (terpri)
                     (addition-error se-list new-obj)
                     nil)
                   )
                                                        )
        )
      )))
;
```

```
;[52] ------------------------->
(defun p-simple-expression ()
 (do ((se-list (p-object)
          (combop-check comb-op se-list new-obj))
     (comb-op nil)
     (new-obj nil))
    ((or (null se-list)
       (not (equal (car (peek-token)) 'c-o))) se-list)
    (setq comb-op (cdr (get-token)))
    (setq new-obj (p-object))
    )
 )

;
;[53] ----------------------------->


; this is similar to p-simple-expression but deals with
gl-log-ops

(defun p-expression ()
 (do ((e-list (p-simple-expression)
          (list (cdr (get-token))
              (p-simple-expression)
              e-list))
     )
    ((not (equal (car (peek-token)) 'l-o)) e-list)
    )
 )
;
```

```
;[54] ---------------------------->
;this is the top level of the parser and demands a form that
; reads expression rel-op expression
; assumes property is in gl-prop-list from new-token


(defun p-assertion ()
  (cond ((null (peek-token))
         (print '(no assertion was made)))
        (t
         (let ((a-list (p-expression))
               (relation (get-token))
               )
              (cond ((null a-list)
                     nil)
                    ((and
                       (equal (cdr relation) 'isa)
                       (equal (car (peek-token)) 'prop))
                     (list (cdr (get-token)) a-list))
                    ((equal (car relation) 'r-o)
                     (let ((lhs (p-expression)))
                          (cond ((null lhs)
                                 nil)
                                (t
                                  (list (cdr relation)
                                        a-list
                                        lhs))




                                                   )
                          ))
                    (t
                     (p-error 'p-assertion relation))
                    ) ) )) )
```

```
;
;[55] -------------------------------->
(defun p-error (funct item)
     (princ "ERROR")
     (terpri)
     (princ "function is:")
     (princ funct)
     (terpri)
     (princ "place is:  ")
    (princ (cdr item))
    (terpri)
    nil

)
;
;[56] ------------------------------->
;THIS IS A FUNCTION THAT WILL ATTEMPT TO HANDLE ALL
SYNTAX ERROR
; MESSAGES. IT WILL RECEIVE A LIST OF INDICES FOR PRINTING
;FROM THE GL-error-message list.

(defun error-response (string-list)
  (provide-reactions string-list)
 )
;
;[57] --------------------------->
;THIS FUNCTION TAKES CARE OF THE CASES WHEN THERE IS AN
IS IN
; THE ASSERTION AND RETURNS THE APPROPRIATE TOKEN IF IT
HAS ONE
; OR NIL IF IT DOESN'T
```

```
(defun find-is-token ()
  (let ((new-word (next-word)))
      (cond ((equal new-word "THE")
            (setq new-word (next-word))
            (cond ((equal new-word "MIDPOINT")
                  (setq new-word (next-word))
                  (cond ((equal new-word "OF")
                        (list (cons 'r-o 'midpoint)))
                       (t
                        (error-response (list
                                   (GL-error-message 4)
                                   (GL-error-message 1)))
                        nil)))
                 (t nil)))
           (t nil))
      )
  )
;*************************************************
******************

;THE FOLLOWING CALLS SET UP AN ARRAY OF ERROR MESSAGES

(setq GL-error-message (make-array 20))
(GL-error-message 0 "**ERROR**  ")
(GL-error-message 1 "***TRY AGAIN***")
(GL-error-message 2 "  is not a point")
(GL-error-message 3 "**ERROR**  This is not complete!")
(GL-error-message 4 "*ERROR* Do you mean \IS THE MIDPOINT
OF\")

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;************************************************
***************
```

```
;NOT YET IMPLEMENTED
*********************************************
;**************************************************
***************
;**************************************************
***************
;**************************************************
***************
;**************************************************
***************
;**************************************************
***************
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;THIS FUNCTION TAKES A STRING AND RETURNS A SORTED LIST
OF ITS
;REPRESENTED ATOMS

;THIS FUNCTION SHOULD BE CALLED FROM check-members
(defun sort-string-list (single-list)
 (do ((new-single (cadr single-list)
             (car rem-list))
     (rem-list (cddr single-list)
            (cdr rem-list))
     (sorted-list (list (car single-list))))
    ((null new-single) (slist-to-alist sorted-list))

    )
  )
```

```
                    ;//        F-LESSON        //

;FUNCTIONS WRITTEN IN THIS FILE
;    [58] GL-LESSON-ONE
;    [30] GL-OPTION-ARRAY


;FOLLOWING ARE THE CALLS THAT INITIATE THE NEW LESSON
;IT REQUIRES THAT GL-lesson-one HAS ALREADY BEEN COMPILED
(record gl-awin)

(setq GL-present-lesson 'GL-lesson-one)

(setq gl-present-mode GL-present-lesson )

(setq GL-help-avail (cons 'GL-lesson-one GL-help-avail))

;FOLLOWING IS A FRAME STRUCTURE FOR GL-lesson-one
;IT INCLUDES THE FOLLOWING SLOT NAMES:
;   prompt
;   assertion-list
;   solution-size
;   solution-step
;   mode-stack
;   reason-choice
;   latest-as-list
;   evaluator
;   save-line
;   drawing

(putprop 'GL-lesson-one "?" 'prompt)
(putprop 'GL-lesson-one '("1" "12" "13" "14" "15" "16") 'rules-avail)
```

```
(putprop 'GL-lesson-one
      '(((midpoint "B" (segment "A" "C"))
        (equal (measure (segment "A" "B"))
             (measure (segment "B" "C")))
        (congruent (segment "A" "B")
               (segment "B" "C"))) . ("1" "15" "13"))
      'assertion-list)

(putprop 'GL-lesson-one 4 'solution-size)
(putprop 'GL-lesson-one 1 'solution-step)
(putprop 'GL-lesson-one '(GL-lesson-one) 'mode-stack)
(putprop 'GL-lesson-one 2 'reason-choice)
(putprop 'GL-lesson-one 'evaluate-assertion 'evaluator)
(putprop 'GL-lesson-one nil 'latest-as-list)
(putprop 'GL-lesson-one nil 'save-line)

;
;[58] ---------------------->
;THE FOLLOWING LAMBDA FUNCTION IS THE DRAWING FOR THIS LESSON

(putprop 'GL-lesson-one
        (lambda ()
              (cs gl-awin)
              (norecord gl-awin)
              (record gl-awin)
              (draw-segments
               '(((-60 -40 "A") (0 -40 "B"))
                 ((0 -40 "B") (60 -40 "C")))
               GL-awin)
              (moveto -100 -5 gl-awin)
              (drawstring
                "GIVEN:"
                gl-awin)
              (moveto -80 10 gl-awin)
```

```
                (drawstring
                  "B IS THE MIDPOINT OF [AC"
                  gl-awin)
                (moveto -100 30 gl-awin)
                (drawstring "PROVE:   [AB / [BC" gl-awin)
                )
            'drawing
            )


(funcall (get GL-present-lesson 'drawing))

;
;[30] ---------------------------------------->
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;PLEASE NOTE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;FOLLOWINING IS THE OPTION MENU WHICH WILL BE ONE OF THE
;GL-OPTION-ARRAY SELECTIONS NAMELY
;GL-OPTION-ARRAY 2

(GL-OPTION-ARRAY 2
            (lambda ()
                  (cs gl-helpwin)
                  (gl-helpwin 'setwtitle "REASON CHOICES")
                  (gl-helpwin 'selectwindow)
                  (textsize 10 gl-helpwin)
                  (draw-text-lines
                    '(""
                      "1   GIVEN"
                      ""
                      "   ***DEFINITIONS***"
                      "12   SEGMENT BISECTOR"
```

```
        "13  CONGRUENT SEGMENTS"
        "14  CONGRUENT ANGLES"
        "15  MIDPOINT"
        ""
        "16  ALGEBRA RULES")
       gl-helpwin
       11)
      (textsize 12 gl-helpwin)
      (textfont 17 gl-helpwin))
)
```

SELECTED BIBLIOGRAPHY

Anderson, John R. "Acquisition of Proof Skills in Geometry." Machine Learning An Artificial Intelligence Approach. Ed. Ryszard S. Michalski, Jaime G. Carbonell, and Tom M Mitchell. Palo Alto Calif. Tioga Publishing. Co. 1983.

Anderson, John R., Boyle, C. Franklin, and Gregg Yost. "The Geometry Tutor". Proceedings of the Ninth International Joint Conference on Artificial Intelligence. Sponser. International Joint Conferences on Artificial Intelligence, Inc. Vol 1. Los Altos, Calif. Morgan Kaufmann Publishers. 1985.

Barr, Arron, and Feigenbaum, Edward R. The Handbook of Artificial Intelligence. Vol 2. Los Altos, Calif.: William Kaufmann, Inc. 1982.

Bundy, Alan. The Computer Modelling of Mathematitical Reasoning. London: Academic Press Inc. Ltd. 1983.

Burton, Richard R., and John Seely Brown. "An investigation of computer coaching for informal learning activities". Intelligent Tutoring Systems. Ed. D. Sleeman and J.S. Brown. London: Academic Press Inc. Ltd. 1982.

Charniak, Eugene, and Drew McDermott. Introduction to Artificial Intelligence. Reading Mass.: Addison-Wesley, 1985.

Clancey, William J. "Tutoring rules for guiding a case method dialogue." Intelligent Tutoring Systems. Ed. D. Sleeman and J.S. Brown. London: Academic Press Inc. Ltd. 1982

ExperTelligence, Inc The ExperTelligence Guide to Expert System Shells. Document-D22. Santa Barbara, Calif: 1986.

Hayes, Brian. "A Mechanic's Guide to Grammar, Part III: A homemade compiler". Computer Language. December, 1985.

Hirsh, Christian, R. et al. GEOMETRY Scott, Foresman. 1984.

Hoffer, Alan. "Geometry Is More than Proof." Mathematics Teacher 74 (January 1981): 11-16.

Inference Corporation. ART Automated Reasoning Tool. Los Angeles, Calif 1985.

Papert, Seymour. Mindstorms. New York: Basic Books Inc. 1980.

Polya, G. How To Solve It. Princeton, New Jersey: Princeton University Press, 1945.

Ramamoorthy, C. V. et al. "Software Engineering: Problems and Perspectives". IEEE Computer, October, 1984.

Richer, Mark H. and William J. Clancey. "GUIDON-WATCH: A Graqhic Interface for Viewing a Knowledge-Based System". IEEE Computer Graphics and Applications Novemeber 1985.

Ritz, Dean A., Celmins, Emily K., and Jon R. Richter. { The ExperLisp Manual. ExperTelligence, Inc 1985.

Senk, Sharon L. "How Well Do Students Write Geometry Proofs?" Mathematics Teacher 78 (September 1985) 448-456.

Shaughnessy, J. Michael, and William F. Burger. "Spadework Prior to Deduction in Geometry." _Mathematics Teacher_ 78 (September 1985): 419-427.

Sleeman, D., and J.S. Sleeman, ed. _Intelligent Tutoring Systems_. London: Academic Press Inc. Ltd. 1982

SunMicrosystems, Inc. _Sun-3/160M Workstation_. Mountain View, Calif: 1986.

Texas Instruments. _The Second Artificial Intelligence Satellite Symposium_. 25 June 1986.

Tucker, Michael. "Expert systems blaze trails to AI success." _Mini-Micro Systems_. March 1986.