STUDENT MODELING IN AN INTELLIGENT TUTORING SYSTEM

THESIS

Jeremy E. Thompson
Captain, USAF

AFIT/GCS/ENG/96D-27

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/96D-27

STUDENT MODELING IN AN INTELLIGENT TUTORING SYSTEM

THESIS

Jeremy E. Thompson
Captain, USAF

AFIT/GCS/ENG/96D-27

Approved for public release; distribution unlimited

AFIT/GCS/ENG/96D-27

STUDENT MODELING IN AN INTELLIGENT TUTORING SYSTEM

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

Jeremy E. Thompson, B.S.

Captain, USAF

December 17, 1996

Table of Contents

List of Figures

List of Tables

Acknowledgements

I thank my research advisor, Dr Steven K. Rogers, for his expert advice and assistance. His patience and encouragement while I was working on my thesis made my research not only educational, but also interesting and bearable. He is the ideal advisor for guiding a wayward thesis student to a realizable goal. There is no better research advisor at AFIT. I also wish to thank my readers, Dr Matthew Kabrisky and Dr Eugene Santos for their comments and suggestions. Their comments helped me recognize shortcomings in my analysis, which enabled me to build a stronger thesis. I also owe great thanks to Dan Zambon and Dave Doak, the system operators for the Hawkeye computer laboratory. Without their help and support, my days would have been consumed with installing and configuring software, chasing down untraceable system errors, and pleading with the printers to actually print my output, leaving me little time for research.

Finally, my wife, Colleen, and my sons, Christopher and Stephen, all deserve my unending gratitude for suffering through my long education, which began with my undergraduate studies in 1988. They have supported me throughout my six years of advanced education.

Jeremy E. Thompson

AFIT/GCS/ENG/96D-27

Abstract

This thesis explores a new approach to modeling the student in an intelligent tutoring system (ITS), by providing a student model which learns new solutions from the student. A prototype of the new approach to ITS is demonstrated in the Euclidean geometry domain. Complete C++, CLIPS, and Tcl/Tk code listings are included in the appendices for reference. Adaptable multiple software agents were targeted for implementation, based on current literature. However, the student model is found to be maintainable *without* multiple software agents, while still allowing for tracking several possible solution paths when monitoring student solutions. This capability contradicts previous research reported in the literature. The student model is extended by providing a learning module, which is capable of recognizing new solutions provided by the student. These new solutions may then be included in the expert knowledge base. In addition to a learning student model, other concepts from the current ITS literature are explored and implemented. Differential modeling and expectation driven analysis are analyzed, as well as the use of production rules and overlay models. Mastery levels are implemented to aid in cognitive diagnosis. Several cognitive and pedagogical concepts, such as symbolic knowledge, procedural skill, and conceptual knowledge, are explored and applied to the research. The student model prototype is both a pedagogic-content model and a subject-matter model. Additionally, a new division of labor between the student model and the instructor module in intelligent tutoring systems is described. Particularly, the student model acts strictly as a pedagogic-content model and subject-matter model, with no inferencing other than that expected of the real student. The instructor module performs all inferencing about the student's actions and knowledge.

[This page intentionally left blank]

STUDENT MODELING IN AN INTELLIGENT TUTORING SYSTEM

I. Introduction

1.1 Motivation

The computer world has been striving to emulate the intelligence of the human brain for many years. These attempts at mimicking the reasoning ability of the brain are frequently described as artificial intelligence (AI). One of the most promising, yet most elusive, goals of AI is to create an intelligent tutoring system (ITS) with the human teacher's ability to infer a pupil's level of understanding. An ITS is desirable because schools could better afford individualized attention for students if that attention were provided by inexpensive computer programs. Currently, all students are required to do *all* problems because teachers cannot provide individualized attention. Since not all students are the same, not all students require the same amount of reinforcement in order to grasp a particular topic. Ideally, with an ITS, the student can progress at a pace exactly corresponding to his understanding. Concepts the student does not understand well will be repeated, while others may be presented only once, thus avoiding the workbook mentality of standard classrooms. As stated by Anderson, Boyle, and Reiser, we want an ITS that "understands the student and responds to the student's special needs." (1)

ITS research is also important to the Air Force and to the military in general. Training personnel is an expensive requirement. As equipment and weapons become more technologically complex, training requirements for maintenance and operations increase. Potentially, automated training and education could reduce the number of hours spent training and retraining troops, while also freeing domain experts from training requirements for actual mission work. A robust ITS could serve the Air Force well: it would not only allow automated training of troops, but could train them at the optimal rate for each individual. Anderson (1) found a four-to-one advantage for tutored students over classroom taught students, when measuring time required to reach proficiency in programming. If

ITSs successfully attain a similar success rate, all organizations requiring extensive training of employees stand to benefit.

1.2 Background

Commonly, an ITS is composed of at least three modules: an expert (or domain) model, a student model, and an instructor (or pedagogical) module (14). The expert model represents information specific to the subject being taught, the student model portrays the current student understanding or misunderstanding of the subject domain (9), and the instructor module contains knowledge required of teachers to select meaningful lessons for their students.

For example, if the educational topic of interest is a subset of Euclidean geometry such as geometric proofs, the expert model might contain information such as the relationships between angles in a triangle, e.g. the sum of the measures of the interior angles in a triangle is 180 degrees. The student model would then contain a representation of the geometric knowledge the student has grasped, possibly including any misconceptions the student has apparently formed. In contrast, the instructor module should be wholly, or at least primarily, independent of the domain. It would contain, for instance, generic information about what criteria determine when a student needs to repeat a portion or all of a topic, and an engine for generating or selecting relevant questions.

One of the most difficult problems when designing an ITS is effectively determining and representing the student's current knowledge of the subject at hand. While important progress has been made in ITS development, effectively modeling the student continues to present the designer with significant challenges (9). These challenges arise primarily from inconsistencies in student reasoning. Students sometimes change their minds, learn new material, unlearn old material, and make careless mistakes (17, 7). Moreover, students do not necessarily have the ability to completely search their knowledge (7). They may know a concept but not recall it for an application.

In order to model the student's state of understanding, the program must attempt to follow the student's reasoning. A common method for modeling the student is through

the use of state models. By providing expert solutions in the form of state models, an overlay method can be used to track the student's progress towards expert solutions (20). The overlay method treats student knowledge as a subset of expert knowledge. When the student attempts a solution to a posed problem, an overlay student model tries to "overlay" the student's solution onto the expert solution. This works fine when there is one expert solution to the problem. In such a case, the student is either working on the expert solution, or not. A problem arises when there are multiple solutions to the given problem (14). Since the student could choose one of many solutions, or possibly try several before settling on one, tracking these many approaches with a state model has been a problem. Recalling the inconsistencies in student reasoning discussed in the previous paragraph, it is evident that multiple expert solutions magnify the problem of the student changing his mind.

One proposed solution is to use multiple agents to simultaneously represent states in several different solution paths (14). Using independent agents allows representing several possible states for the student. These states can then be used to help predict the next state the student will enter. By using expectation driven analysis, the best student model can be selected based on how well it's predicted state agrees with the student's actual behavior. In most other student models, only one path for modeling the student is tracked, thus restricting the expected next states to those that follow the selected line of reasoning. Since it is not uncommon for a student to try multiple approaches to a problem before settling on one, the multi-agent method allows for more flexibility in modeling student behavior, and thus a better match with the student's real behavior.

A shortcoming of the multi-agent model, as presented by Leman, (14), is the inability to learn from student solutions. The student model expects to have a complete set of solutions for the problems presented. Any solutions not found in its knowledge base are considered wrong. Acquiring knowledge from the student would avoid classifying potentially correct student solutions as wrong, and would provide for a more robust, complete domain model. This author is aware of only one ITS research effort that explores learning from the student (16).

Problem Statement:

This research will explore the use of multiple software agents to model student reasoning. Additionally, the multi-agent student model will be extended to include adaptive agents capable of recognizing and learning new solutions provided by the student.

1.3 Current Knowledge

The genesis of interest in AI in Education at the Air Force Institute of Technology (AFIT) was a paper by Matthew Kabrisky (12). Kabrisky discussed the positive impact an artificial tutor could have on education. As a hypothetical example of the potential for AI, he provided a convincing excerpt of a child's interactive learning session with an ITS. However, his emphasis was the treatment of human/computer interfacing problems, leaving the actual tutoring aspects to other authors.

The basis of this research is the paper by Leman, (14). The authors implemented a student model built on an existing ITS known as EDDI (15). EDDI's student model divided knowledge into three levels: static knowledge (S), dynamic knowledge (D), and reasoning knowledge (R). Static knowledge was defined as knowledge about the basic concepts of the domain being studied, while dynamic knowledge concerned more functional aspects of the domain. Reasoning knowledge was defined as knowledge about the student's reasoning.

Again using the geometry domain for an example, static knowledge would be general concepts about geometry, such as geometrical figures and their configuration. Dynamic knowledge would be geometric theorems, which could be used to generate specific static knowledge. Finally, an example of reasoning knowledge is the observation that the student consistently reasons incorrectly concerning equilateral triangles.

The authors devised several different types of agents to assist in modeling the student. Spy agents were used to record local student actions which would be useful in determining the pupil's reasoning method. Task agents were implemented to examine the student's actions while considering the task to be attempted. Three different types of task agents were employed: S-agents for storing the student's static knowledge, D-agents for handling the student's dynamic knowledge, and R-agents for representing the student's reasoning.

D-agents could produce new static knowledge based on provided static and functional knowledge.

Because of the method employed for modeling the student, EDDI had noticeable difficulty in modeling a student's misconceptions. Leman (14) attributed this problem to the model's inability to track multiple lines of reasoning, as well as the lack of knowledge about a global solution before the student even began the problem. Their answer was to seed the knowledge base with multiple solutions to the given problem, then track the student's progress on all the solution paths, attempting to reconcile the student's actions with those required in each solution. By processing the student's reasoning in parallel through the use of multiple software agents, the student's progress along all of the known solutions could be tracked simultaneously, and the student's final solution method would be readily apparent. Simultaneous tracking of multiple solutions was used to justify the introduction of software agents into the student model.

1.4 Assumptions

The support software for the student model, including user interfaces and an ITS, will be fundamentally implemented with a high degree of *human* instructor guidance for the instructor module. Without an existing ITS, the problem of modeling pedagogical knowledge becomes as critical as modeling the student. By allowing a human instructor to make course-work decisions based on information provided by the student model, modeling the instructor can be circumvented. Since the thrust of this research is modeling the student, designing an intelligent instructor module is not relevant.

1.5 Scope

The emphasis of this research is to implement a multi-agent student model similar to that described by Leman, Giroux, and Marcenac (14). The student model will be extended to credit the student for correct solutions not found in the knowledge base. Implementation of support software, to include interfaces and the remainder of the ITS, will be minimized.

C Language Integrated Production System (CLIPS) will be used to implement the ITS. CLIPS is an expert system tool developed by the Software Technology Branch of NASA. Tcl/Tk will be used for the user interface, and C^{++} will be used for the main procedure.

1.6 Approach/Methodology

As discussed earlier, much of this research will include building a basic multi-agent student model. Once this basic model is in place, modifications to the multi-agent system will be accomplished, to allow the ITS to both realize when a student has produced a solution unique to the ITS's knowledge base, and to incorporate that solution into its knowledge base.

A problem domain must also be selected for the ITS. A subset of Euclidean geometry has been identified as suitable. The area of geometric proofs allows for known (expert) solutions, but additional (potentially less efficient) solutions may also be found. Multiple solutions are necessary to test both the multiple agent student modeling and the learning ability of the ITS.

1.7 Organization of Thesis

Chapter II contains a literature review of the history of AI in education, ITSs, and student modeling. Chapter III describes building the ITS, and results from that experience. Finally, Chapter IV identifies contributions and recommendations.

II. Literature Review

2.1 Introduction

In order to provide a comprehensive background on ITSs, the literature is reviewed in order of increasing specificity. To begin with, a brief introduction to AI is provided. A historical perspective on AI applied to education is furnished, including discussion of computer-assisted instruction (CAI) and of ITSs. Finally, methods for modeling student cognition in ITSs are discussed.

2.2 Artificial Intelligence

Many definitions have been attempted for AI, but defining the term continues to present authors a challenge. Dean (6:1) effectively defines AI as "...the design and study of computer programs that behave intelligently." The idea is to construct programs which behave in a manner similar to humans or other intelligent animals. Some idea of why AI has been studied can be gained by examining the following two goals of AI: (22)

- the engineering goal, which is to solve real-world problems, and
- the scientific goal, that of determining which AI related ideas aid in explaining human intelligence.

Some of the most interesting and challenging problems to explore are often interesting *because* of the scientific goal, but the engineering goal actually drives most AI research.

2.3 Artificial Intelligence in Education

A paper by Matthew Kabrisky (12) initiated AFIT's research in the use of AI in education. Kabrisky discussed the positive impact an artificial tutor could have on education. As a hypothetical example of the potential for AI, he provided a convincing excerpt of a child's interactive learning session with an ITS:

> Imagine an elementary school child entering a small comfortably appointed room, perhaps something like a cockpit with a video screen, some desk space and perhaps a mouse or some sort of convenient mechanically driven input to

> the data handling system. The child walks in, says, "Hi, It's me, Linda," and sits down. The machine says "Hi, Linda, How are you?"? Linda says, "Gee, I saw a neat program on bugs last night. I really wanted to watch cartoons but daddy made me, I mean it was neat with all the pretty and squishy looking things and there was a spider that hid in a trap door and grabed [sic] bugs walking by." The machine says, "Yes, I saw it too, would you like to see the part with the trap door spider? I'll play it for you if you want." Linda says, "Oh yea, let me get Joey, he didn't believe me."
>
> After a while, Joey and Linda are through with the nature program, Joey runs outside, and the machine says, "I have a really neat math game. Would you like to play it"? Linda says, "OK, what's it like"? They do math for a while. Note that Linda never has to take tests because the machine always knows where she is in the curriculum and how well she's doing. When Linda's through with math, the computer plays a game tying together geography, bugs, and natural history. The machine knows from its experience with Linda that she now needs a break and sends her out to play.

Though Kabrisky identified the desirability of the machine being able to evaluate the student's progress, his emphasis was on human/computer interfacing problems, identifying three components necessary for a friendly machine:

1. a speech recognizer for converting human speech into text;
2. a natural-language handler for analyzing the content of the text and for generating intelligent responses; and
3. a speech synthesizer for reading back the responses in a near-human voice.

While much progress has been made in items one and three, the natural language handler turns out to be a challenging application for AI. Hidden within this proposed handler is the requirement for generating intelligent responses to whatever the topic of conversation might be. For the hypothetical example, this response mechanism would in fact need to be an ITS, or at least an intelligent CAI system.

2.3.1 Computer-Assisted Instruction. CAIs were an attempt to automate the pedagogical method known as programmed instruction (PI) (20). PI was the structured, goal oriented instruction technique popular in the early 1960's. It required the instructor to determine inputs, the skills known upon entering the PI, and outputs, the behaviors learned when finished with the PI. Learners were led through a PI curriculum in a lock-

step manner, with every incorrect response corrected immediately. Learners were always informed of the accuracy of their solutions before they moved on. When PI is implemented in a computer, CAI results.

CAI doesn't owe its entire origin to PI, though; it also evolved from stimulus-response psychology. When stimulus response psychology is applied to CAI, the computer evaluates a student's answers at every stage of instruction, then determines the next path of instruction. In a standard CAI, the path the computer chooses is already predetermined by the programmer and by the student's success. Intelligent CAI (ICAI), on the other hand, attempts to evaluate *why* a student missed a particular problem, then varies the curriculum based on its evaluation. In standard CAI, if two students missed the same problem (possibly even only the same number of problems in the same section), the two students will invariably receive the same set of problems next. However, with ICAI, if student A missed the problem because he miscarried the hundreds digit, while student B erred in her addition of the nine and seven in the tens place, then the two students could receive different problems to reinforce the particular concepts with which they were struggling.

The step from CAI to ICAI is not an abrupt step, but rather a gradual evolution from a basic computer-interaction method to a refined computer evaluation of student progress. At some point along this progression of refining ICAI, intelligent tutoring systems (ITSs) result.

2.3.2 Intelligent Tutoring Systems. Commonly, an ITS is composed of three modules (14):

1. an expert (or domain) model,
2. a student model, and
3. an instructor (or pedagogical) module.

The expert model represents information specific to the subject being taught, the student model portrays the current student understanding or misunderstanding of the subject domain (9), and the instructor module contains knowledge required of teachers to select meaningful lessons for their students. Hartley and Sleeman (8) identified these same basic

requirements for an ITS in 1973. In fact, an ITS can be regarded as a distinct type of ICAI, with these three components making the distinction (20). Some authors have even come to regard ITSs as the ICAI of the 1980's (23), while ITSs have come much closer to realizing their full potential in the 1990's. The prototype for this research makes use of expert and student models, and the functionality of the instructor module is provided by a human instructor. A graphical interface module is also included in the implementation.

Despite numerous examples of ITSs employing many different approaches to *intelligence* in tutoring, there are two terms most often associated with them (20):

1. cognitive diagnosis, and
2. adaptive remediation.

Cognitive diagnosis entails forming an idea of the student's cognition or knowledge, usually in the form of student modeling. Cognitive diagnosis is addressed again in Chapter III, where it is applied to reasoning about student knowledge. Adaptive remediation is the dynamic application of tutoring based on specific difficulties encountered by the student. Adaptive remediation is typically included in the instructor (or pedagogical) module.

2.4 The Student Model

One of the most difficult problems when designing an ITS is effectively determining and representing the student's current knowledge of the subject at hand (2). Modeling the student remains difficult because students are capable of inconsistent reasoning. Students may be distracted while working on a problem and produce sub-optimal work. They may also learn new material, unlearn known material, or simply not recall a known item at the particular moment it is required (17, 7). An optimal student model would include *all* information about the student, both relevant and seemingly irrelevant (9). The source of a student inconsistency or misunderstanding cannot be *perfectly* derived without a complete history of the student. Obviously, such a complete model is both impossible and impractical. An effective student model is possible, however, by including key elements in the model.

Student modeling is characterized by two principal activities (10):

1. behavior analysis, and
2. model management.

Behavior analysis is a cornerstone of student modeling, and is present throughout the prototype. Cognitive diagnosis (20) and behavior analysis are very similar terms, and are applied in roughly the same circumstances in a student model. Behavior analysis describes the entire process of examining student actions, while cognitive diagnosis includes behavior analysis, with the specific objective of diagnosing the student's state of understanding and misunderstanding.

Although most systems inevitably deal extensively with behavior analysis, few apply anything but ad hoc techniques to the model management (7). A concerted effort at model management has been explored by Giangrandi and Tasso (7) through the use of truth maintenance techniques. Interestingly, this effort includes the use of multiple student models to reflect multiple hypotheses about the student's reasoning. Although their research appears both relevant and useful, the application of their described model management methods must be recommended for further research.

Several different approaches to behavior analysis have been attempted, including the following (9):

1. production rules,
2. logic programming,
3. machine learning,
4. Bayesian networks (4, 21, 13),
5. artificial neural networks (ANNs), and
6. overlay models.

Production rules provide the opportunity to represent both ideal (expert) rules and mal-rules (rules defining incorrect beliefs or logic). For a simpler approach, logic programming offers the ability to implement inferencing and knowledge representation easily, while machine learning eases representing the changing knowledge of the student. Bayesian network

models provide a probabilistic method for handling the uncertainties in student reasoning (4, 21, 13). ANNs allow the student model to adjust tutoring strategies to the student, and even allow initial training of the ANN by an expert, rather than explicitly coding domain knowledge (24). Finally, overlay models incorporate a model of expert knowledge, where known solutions are mapped onto the expert model, and student knowledge is always represented as a subset of the expert knowledge. Overlay models are readily implemented in multi-agent form, thus allowing for tracking multiple solutions to problems. Both overlay models and production rules are used in this research.

Shute (19) identified three categories of knowledge to be learned by the student and to be represented by the student model. Symbolic knowledge is familiarity with symbols and equations. Procedural skill concerns the application of rules (i.e. knowledge) to problem solving. Procedural skill is easily represented as a series of concept and equation applications. Finally, conceptual knowledge corresponds to knowledge about relationships between concepts, schemas, and rules. It represents a higher-level of understanding.

Not all tutoring systems apply, or even require, all three types of learning. For example, intelligent *training* systems frequently require only procedural skill and symbolic knowledge. Training systems are used to train individuals in *skills* requisite for a particular job, and often do not require conceptual knowledge, and only a limited amount of symbolic knowledge. All three concepts will be discussed in the context of the implemented prototype in the next chapter.

Ragnemalm (18) describes two types of student models:

1. a pedagogic-content model, and
2. a subject-matter model.

A pedagogic-content student model contains pedagogical measurements of the student's progress, and therefore requires specific pedagogical goals for the student to attain. A subject-matter model is closely tied to the domain model, and is typically runnable. In a runnable student model, the model may be executed with the student's (perceived) knowledge and misunderstandings determining the model's actions. A subject-matter model al-

lows for differential modeling (3) and expectation driven analysis, which will be described in Chapter III. The ITS prototype is both a pedagogic-content and a subject-matter model.

2.5 Summary

Coaxing computers into becoming effective, personal tutors has developed into an ambitious and elusive goal. The problem began as a question of how best to integrate the computer's abilities into pedagogical uses, but has evolved into a search for the holy grail of computer programming: artificial intelligence. Evolving from simple computer-assisted instruction (CAI), to intelligent CAI (ICAI), and eventually to intelligent tutoring systems (ITSs), implementing computer-assisted education has become increasingly more complex. Fortunately, the added complexity *has* increased the effectiveness of computer-assisted education. As we discover new tools for increasing the power and utility of computers, these tools are applied to producing more effective artificially intelligent tutors. The latest of these tools, software agents, have had some success already (14, 5). The goal of this thesis effort is to further explore the use of software agents in modeling student reasoning. The next chapter begins that effort with a description of the ITS implemented, focusing primarily on the student model.

III. Building the Intelligent Tutoring System (ITS)

3.1 Introduction

Our intent was to build a student model, and a minimal amount of support software to go with that model. In order to accomplish this goal, the manageable domain of high school geometry, specifically parallel bisected lines proofs, was selected as the test domain. Additionally, the instructor module was not coded, but its functionality is provided by a human instructor. The method for accomplishing this task is described, followed by a discussion of the student model used. Next, the addition of *adaptable* software agents to student modeling is discussed, and the problem of identifying student knowledge is examined. Finally, the roles of the student model and the instructor module are redefined, based on insights gained during this research effort.

3.2 The Man Behind the Curtain Experiment

The vision for an artificially intelligent tutor described by Kabrisky (12) includes a natural language interface to allow the student more natural communication with the ITS. While a natural language processor is currently unavailable, it can be simulated using the "man behind the curtain" methodology. By placing a student at a workstation, within earshot (but not eyesight) of a human instructor at another workstation, the student can communicate audibly to the instructor, while the instructor communicates to the student through a remote terminal session. The student is told she is orally communicating with the ITS. The instructor provides educational material to the student by typing text and displaying images on the student's monitor.

The instructor is responsible for receiving the student's answers, relaying them to the domain model, and monitoring the student model's analysis of the student's responses. The instructor then acts on behalf of the instructor module in deciding what material would best be presented next. Thus, the human instructor is acting both as a natural language processor, and as the instructor module of the ITS. In this way, a student model can be developed without building a complete ITS. Hereafter, this implementation of an

ITS will be referred to as the Curtain ITS. The Curtain interface was implemented using Tcl/Tk. For reference, Appendix C includes the complete Tcl/Tk source listings.

3.3 Modeling the Student

The basis of this research is the paper by Leman, Marcenac, and Giroux (14). The authors implemented a student model built on an existing ITS known as EDDI (15). EDDI's student model divided knowledge into three levels: static knowledge, dynamic knowledge, and reasoning knowledge. Static knowledge was defined as knowledge about the basic concepts of the domain being studied, while dynamic knowledge concerned more functional aspects of the domain. Reasoning knowledge was defined as knowledge about the student's reasoning.

Using the vocabulary described by Shute (19), static knowledge falls into the symbolic knowledge category, while dynamic knowledge could be either symbolic knowledge or conceptual knowledge. Reasoning knowledge is obtained by tracking a student's procedural skill, but inferences about the student's symbolic knowledge or conceptual knowledge are possible using cognitive diagnosis (20).

The Curtain prototype makes use of the concepts just discussed, as well as others, yet to be presented. The C^{++} and CLIPS source listings for Curtain are included in Appendices A and B, respectively, for further reference.

Given:	Measure of angle 1 = 120 degrees
Find:	Measure of angle 6 = ?
Solution:	Relate angle 1 to angle 5 Relate angle 5 to angle 6

Table 1 Sample Geometry Problem (see Figure 1)

3.3.1 Overlay models. Leman (14) made use of an overlay model in their research. Likewise, in the Curtain ITS, an overlay model is employed for comparing the student solution to a library of expert solutions. A concrete example from the Curtain ITS may help illuminate the use of overlay models.

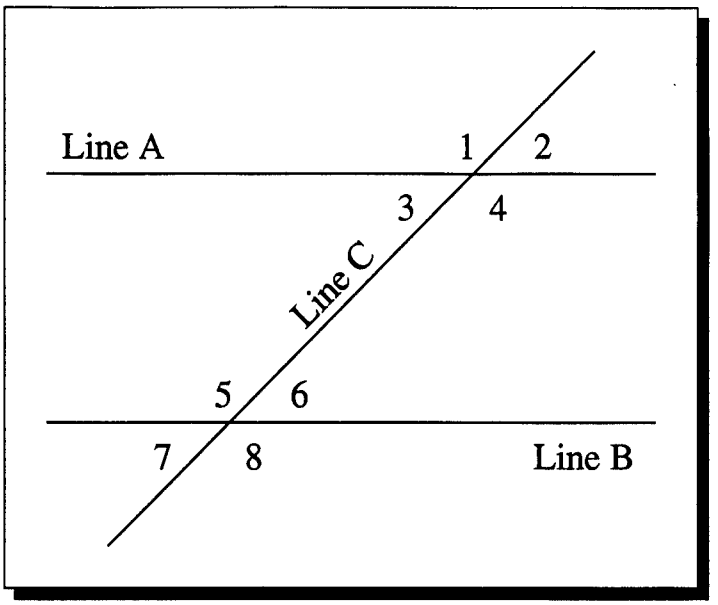

Figure 1 Bisected Parallel Lines

Consider the problem posed in Table 1. The solution provided in Table 1 is only one of many. The actual proof for this solution contains much more detail (see Table 2), but the provided detail is all that is required for matching solution strategies. Figure 2 displays a state model representation of the solution.

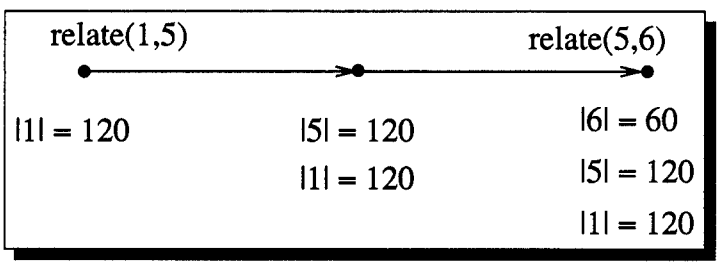

Figure 2 Expert Solution State Diagram

The provided solution is relatively straightforward:

1. define the relationship between angles 1 and 5 in order to determine the measure of angle 5,
2. then define the relationship between angles 5 and 6, thus yielding the desired answer, the measure of angle 6.

Given:	Lines A and B are parallel Lines A and B are bisected by Line C Measure of angle 1 = 120 degrees	
Prove:	Measure of angle 6 = 60 degrees	
PROOF	Statement	Reason
1	Measure of angle 1 = 120	given
2	Angles 1 and 5 are corresponding angles	observation and definition of corresponding angles
3	Angles 1 and 5 are congruent	(2) and definition of corresponding angles
4	Measure of angle 5 = 120	(1), (3), and definition of congruent angles
5	Angles 5 and 6 are adjacent angles	observation and definition of adjacent angles
6	Angles 5 and 6 are supplementary angles	(5) and definition of adjacent angles
7	Measure of angle 6 = 180 - 120 = 60	(4),(6), and definition of supplementary angles
8	Done	

Table 2 Sample Geometry Proof (see Figure 1)

With the overlay method, the student model can check the student's actions against the provided expert solution, applying what has been called a *differential modeling technique* (3). The state model in Figure 2 demonstrates the ease with which solution strategies can be compared. The nodes represent states (known information), and the arcs represent actions required in order to reach the next state.

One of the weaknesses of overlay models is that they do not allow for student knowledge or information outside of the expert's knowledge. Also, by using state models for representing the expert knowledge, overlay models can represent multiple solutions to a proposed problem, but they struggle with tracking the student's line of reasoning when the student makes multiple attempts at the problem. Since the student could possibly try several solution paths before settling on one, representing these many approaches with a state model has been difficult (14).

3.3.2 Multiple agents in student modeling. Because of the overlay method employed for modeling the student, EDDI had noticeable difficulty in modeling a student's misconceptions. Leman (14) attributed this problem to the model's inability to track multiple lines of reasoning, as well as the lack of knowledge about a global solution before the student even began the problem. Their answer was to seed the knowledge base with mul-

tiple solutions to the given problem, then track the student's progress on all the solution paths, attempting to reconcile the student's actions with those required in each solution. By processing the student's reasoning in parallel, the student's progress along all of the known solutions could be tracked simultaneously, and the student's final solution method would be readily apparent.

Using independent agents allows representing several *possible* states for the student. These states represent distinct steps along a solution path, and can be used to help predict the next state the student will enter. Predicting the next state is important because, through the use of differential modeling (3) and expectation driven analysis, the student model's predictions can be compared to the student's actual actions. The student model is validated if the actions match, while corrections may be made to the student model if they do not. In most other models, only one path for modeling the student is tracked, thus restricting the expected next states to those that follow the selected line of reasoning. Since it is not uncommon for a student to try multiple approaches on a problem before settling on one, the multi-agent method allows for more flexibility in modeling student behavior by allowing the selection of the model which produces a predicted action most closely matching that of the student's.

In order to accommodate expectation driven analysis, Curtain's student model was implemented with production rules, thus allowing mal-rules to represent student misconceptions. Ideal rules in the expert model fire to demonstrate correct actions, while ideal rules and mal-rules fire in the student model in an attempt to accurately predict student actions, and thus validate the student model. During the course of this research, it was discovered that implementing software agents to allow for multiple solution paths was unnecessary. By comparing solutions as *sets* of solutions steps, it was possible to distinguish when a student's solution was identical to one of the expert's, a superset of one of the expert's, or completely unrelated to all of the expert's solutions.

Figure 3 shows an alternative solution to the problem in Table 1. This new solution is both correct *and* equal in efficiency to the previous solution. If only one of these solutions was selected for tracking the student's reasoning based on how the student first attempted a solution, while the student first attempted one solution strategy, but settled on another,

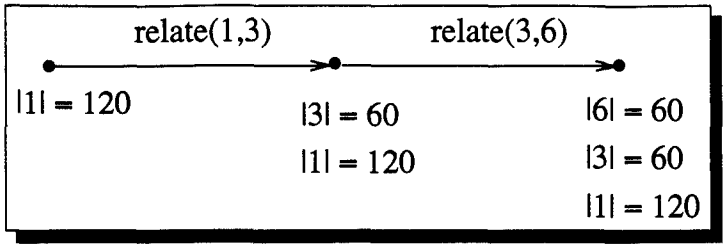

Figure 3 Alternative Expert Solution State Diagram

a standard overlay model would not be able to credit the student with a correct solution. According to Leman (14), a multi-agent student model would be able to track student progress on both expert solutions, and thus correctly deduce which of the strategies the student eventually employed. However, the Curtain prototype successfully tracked the student's strategy *without* the use of software agents.

Tracking multiple solutions was made possible with the use of set functions in CLIPS. If each solution, both expert and student, is treated as a set, it is possible to examine the student solution's relationship to the expert solution(s) with standard subset operations. If the student solution is a subset to the expert solution, then the student hasn't completed a solution. However, if a particular expert solution is a subset of the student solution, then the student has completed an expert solution, but has included more steps than necessary, possibly because the student has wandered through other solution attempts. These extra steps can be readily compared to other expert solutions to determine if the student has attempted, but abandoned, another strategy. If the student solution is a subset of a particular expert solution, and that expert solution is a subset of the student solution, then the student has exactly applied that expert solution. Of course, these comparisons may be made after each student action to maintain a current model of the student's strategy.

3.4 Learning From the Student

A shortcoming of Leman's (14) multi-agent model is the inability to learn from student solutions. The student model expects to have a complete set of solutions for the problems presented. Any solutions not found in its knowledge base are considered wrong. It would be useful if the model could recognize and adopt new solutions when presented

with them. Moreover, the ability to recognize a solution which seems unique, but is really just a combination of existing solutions, would also be of great benefit.

An extension to Leman's current model is to provide a learning agent (L-agent) which checks for unique student solutions. This agent need only be invoked if a match is not found in the expert solution library. If no match is found, then the solution must be evaluated to determine its correctness. If it is indeed a correct solution, the student must be given credit for it. If it is also as efficient as the expert solution(s), the student model must be updated accordingly, and the expert solution library may be updated as well.

Based on the discovery that software agents were not required for processing multiple solutions, the L-agent wasn't required to be an agent at all, and so is described as an agent only in the weakest sense. Here, as in the multiple solution processing, a *module* is all that is required.

In order for the L-*module* to determine if a student solution is correct, the domain model must have a *complete* set of rules governing legal transitions in the solution space, and the student model must be a subject-matter model. The complete set of rules is required to ensure the assertions made by the student are valid. The subject-matter model is necessary so that the model may be executed to determine if the goal was actually achieved. Without a subject-matter model, it would be necessary to search the *student's* solution space to determine if any of the possibly many solution paths leads to the goal.

The solution space in Curtain was completely specifiable because of the procedural solutions required for geometrical proofs. Implementing learning modules such as this is possible in any domain where procedural knowledge dominates. Training environments favor procedural skills (11), as do other types of ITSs.

Some examples from Curtain may help to illustrate the points just discussed. Consider the proposed student solution in Figure 4. This solution is a correct solution. It is also a completely unique solution. The L-module should evaluate the solution as valid, and add it to the knowledge base. In fact, the prototype's L-module did update the student model with the fact that the student satisfactorily solved the problem.

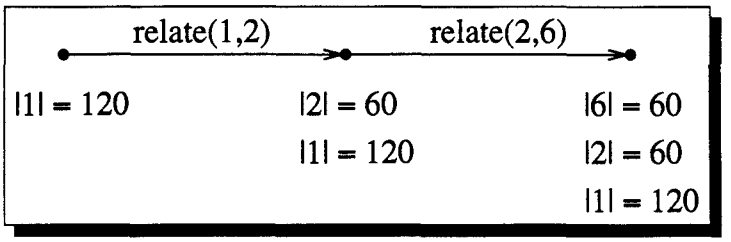

Figure 4 Student Solution State Diagram

In contrast, the solution in Figure 5 is a valid solution, but takes a longer approach. In this case, the solution should not be added to the solution library. The L-module updates the student model with the knowledge that the student solved the problem correctly, but solved it less efficiently, *avoiding the use of corresponding angles.*

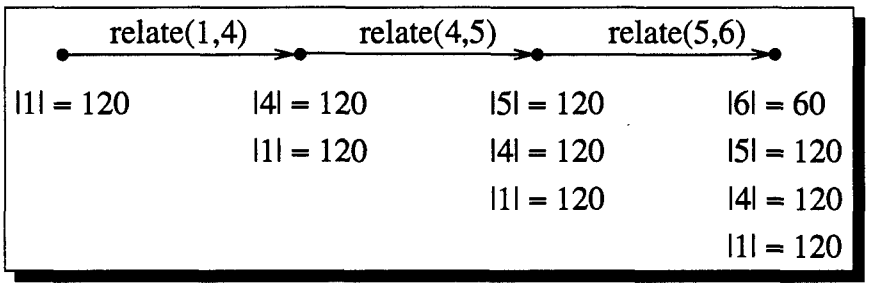

Figure 5 Alternative Student Solution State Diagram

The idea that the student may have avoided the use of corresponding angles intentionally is an important one. If she has not already demonstrated mastery of the concept of corresponding angles, she may be avoiding them out of ignorance. In this case, an appropriate next problem would be one which forces the use of corresponding angles, or asks specifically about them. It is also possible, however, that she avoided them out of boredom, having already solved several problems using corresponding angles, in which case the omission is not critical. Identifying the avoidance of a particular theorem requires more information than that provided in the state diagrams, which is the subject of the next section.

3.5 Identifying Student Knowledge

Though the meta-steps displayed in Figures 2, 3, 4, and 5 are sufficient for comparing solution strategies, they are insufficient for determining the correct application of concepts. By breaking each solution step into sub-steps, student error detection is facilitated (17), through the use of cognitive diagnosis (20) and behavior analysis (10). In interactive tutoring environments, subquestions can be posed to aid in student error detection, but this can be intrusive and distracting for the student (3). In situations where subquestions are impossible, or merely undesirable, the problem of determining student knowledge becomes even more difficult.

In the geometric proof domain, these sub-steps are an inherent requirement for problem completion, and therefore easily obtained. Consider, for example, the problem presented in Table 1. The key concepts the student is required to master in order to *prove* the solution are: (see Table 2)

1. congruent angles,
2. adjacent angles, and
3. supplementary angles.

If the student does not successfully apply these three concepts, then the student *cannot* have a correct solution. However, the student has a multitude of correct solutions to choose from, as demonstrated in Figures 2, 3, 4, and 5. Furthermore, the student may choose from several correct and *efficient* solutions, as shown in Figures 2, 3, and 4. Therefore, making inferences about knowledge the student is avoiding is not as straightforward as one might think; yet, it is not exceptionally difficult either.

The key symptom which lends insight into the student's motivations is the observation that the student takes multiple steps to transition between two nodes that have a direct link. This is the case in Figure 5. As mentioned in Section 3.4, the solution seems to avoid the use of corresponding angles. The formula for discovering such a circumvention makes the nonuse of corresponding angles more apparent, but does not aid in establishing the student's motive. Additional clues to the student's motive can be obtained from the student model.

The most helpful information to be gained from the student model is some form of performance history with respect to the avoided concept, in this case corresponding angles. The most direct information is an actual record of all the student's applications and misapplications of the concept.

While a complete history is possible in small domains, and is available for Curtain, it could certainly become unwieldy as the number of concepts and students increases. Shute (19) suggests mastery levels be defined to assist in determining amount and level of remediation, but they may also be applied to aid decisions regarding student motivations. Curtain was implemented with a binary mastery threshold for exactly this purpose. Enhancing the mastery levels to approach that described by Shute would ease student motivation determinations.

Even with knowledge on how well the student has (apparently) mastered a concept, inferring the student avoids a *mastered* concept because she is bored, or avoids an untried or misunderstood concept because she doesn't know or understand that concept, is still a leap of faith. To make an even more effective estimation, a truth maintenance system, such as that outlined by Giangrandi and Tasso (7), would be beneficial.

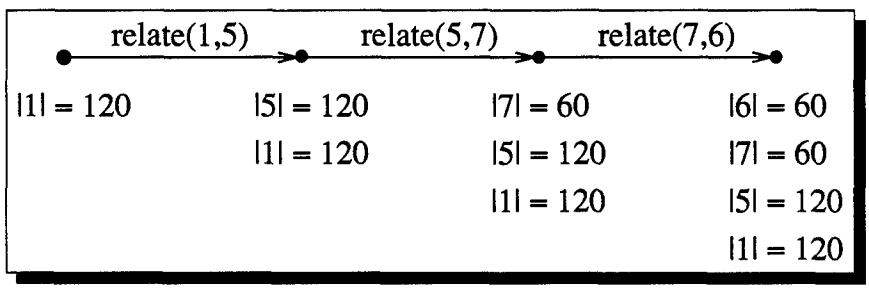

Figure 6 Lengthy Student Solution State Diagram

Although the lengthy but correct solution in Figure 5 has proved useful for cognitive analysis, this is not always the case. Consider, for example, the solution in Figure 6. Like Figure 5, Figure 6 has one more step than the expert solutions in Figures 2 and 3. Unlike Figure 5, Figure 6 makes use of all the essential relations identified earlier in this section: congruent angles, adjacent angles, and supplementary angles. Recognizing this fact is not difficult; flagging the essential components may seem more difficult, but it is only a matter of identifying the intersection of all the concepts applied by all of the expert solutions.

Knowing Figure 6 contains all of the essential relations, it follows that the student has merely included unnecessary steps in her solution. These extra steps, then, can be expected to be a result of distraction, inferior reasoning, etc., but *not* a deficiency in geometry.

3.6 Student Models vs Instructor Modules

While performing the research for this thesis, the definition for a student model seemed to become more blurred and indefinite, rather than more distinct, as one would hope. The cause for the haziness seems to lie in the preponderance of new functions and capabilities being added to the student model. Many authors (19, 18, 3, 4, 7, 16) are including *processing* in the student model which seems more in place in the instructor module. The student model should contain the student's actions and any inferred understandings or misunderstandings.

In contrast, the instructor (or pedagogical) module should contain any functions which reason about the student's actions, or reason about the student's reasoning. The only reasoning the student model should perform is any reasoning the student might be expected to make, and only then if the student model is a subject-matter model.

Thus, the student model would be primarily a *container* for information about the student, including inferences made by the instructor module, with a secondary task of being able to reason in a manner similar to the actual student, if it is a subject-matter model. The instructor module would be responsible for all type of pedagogical matters, including the inferencing a human instructor might do about the student's state of understanding or misunderstanding. With this new division of labor, the instructor module is responsible for all behavior analysis and cognitive diagnosis, while the student model stores the results from the instructor module, acting as a pedagogic-content model /citeragnemalm.

The final chapter discusses author contributions and recommendations.

IV. Contributions and Recommendations

4.1 Contributions

In this thesis, a new method for incorporating learning into the student model of an ITS has been presented. This method was demonstrated effectively with a prototype ITS. The usefulness of subject-matter models for learning was emphasized. Software agents were shown to be unnecessary for processing multiple solutions in the student model, and likewise not required for adding learning to the student model.

While it is recognized that the learning module relies on having a complete set of rules for the domain model, it has also been argued that the main requirement for a domain model with this capability is a domain which relies heavily on procedural knowledge. Many tutoring domains meet this criteria. More importantly, nearly all *training* domains rely primarily on procedural skill (11). Since the Air Force's chief concern is more efficient training, the application of learning to Air Force ITSs is quite feasible.

Even though a learning student model was the initial focus of this thesis, several other useful concepts emerged during the research. First among these was the identification and resolution of some inconsistencies in the division of labor between the student model and the instructor module. Effectively, the student model is a container of information about the student, while the instructor module performs all reasoning or inferencing expected of the human instructor.

Additionally, an alternate use for Shute's (19) mastery levels was demonstrated. As discussed in section 3.5, mastery levels can not only be used for determining remediation as suggested by Shute, but can also be used to gain insight into student motivations.

Also identified in this thesis were appropriate uses for meta-steps and sub-steps in solution analysis. Meta-steps frequently can lead to information about correct and incorrect solution strategies, but sub-step analysis aids making inferences about student thought processes. Sub-steps are particularly effective for cognitive diagnosis. Again, it is recognized that many domains do not have a natural requirement for explicit student identification of sub-steps, unlike geometric proofs, but in these cases a special effort at

deducing these sub-steps can frequently be effective (3). In other cases, explicitly querying the student for solution sub-steps may prove necessary.

Differential modeling (3) and expectation driven analysis were concretely demonstrated as useful for validating student models. These techniques are particularly applicable to overlay methods, which were employed in the Curtain prototype, and lend themselves to intelligent training systems.

Finally, the cognitive terms identified by Shute (19) were successfully applied to Curtain. In section 3.3, the relationships between Leman's (14) static, dynamic, and reasoning knowledge and Shute's symbolic, conceptual, and procedural knowledge were outlined. These relationships are important because new terms are continually being invented for very similar concepts in ITSs, and the relations between these concepts are not always obvious, but remain, nevertheless, important.

4.2 Recommendations

There remains much that can be further explored with respect to this research effort, both in concept and in application. The principal area for further research is the application of the new division of labor between the student and instructor modules. A demonstration of the proposed student and instructor module responsibilities would provide evidence that the division is both logical and practical.

Additionally, the inclusion of a truth maintenance system to the student model would be a significant contribution. As discussed briefly in section 2.4 and in detail by Giangrandi (7), a truth maintenance system aids deconflicting student model inconsistencies generated by student learning, student unlearning, student misconceptions, and others.

The mastery levels in the student model could also be improved. The current binary threshold is serviceable, but a more robust system, similar to that described by Shute (19), could provide more insight into student motivations, as well as aid remediation.

Beyond student modeling, there is the entire realm of the instructor module, which includes topics such as remediation, problem selection, and tutoring strategies. Incorpo-

rating an instructor module with these capabilities can then require enhancements to the student model, such as inclusion of a student's preferred learning strategy.

4.3 Overall Conclusions

This research has proposed and demonstrated a method for expanding the solution library in the expert module through the use of a learning student model. The method demonstrated has application in a wide variety of tutoring domains, including most training scenarios.

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fstream.h>
#include <unistd.h>
//#include <vfork.h>
#include "curtain.h"

#define MAXSTRING 100

extern "C" {
 #include "clips.h"

/************************************/
/*                                  */
/*  Define C functions to CLIPS     */
/*                                  */
/************************************/
// This function may be called from CLIPS, and passes a Tcl command
// and a string of parms to Tcl, thus invoking that Tcl function.
char* TclFunc()
  {
  DATA_OBJECT msg,pipe;
  FILE *write_pipe;
  VOID *MainPtr, *InstPtr;
  if (ArgCountCheck("TclFunc", EXACTLY,1) == -1)
    {
    fprintf(stderr,"Wrong number of arguments! \n");
    return "FALSE";
    }

  if (! ArgTypeCheck("TclFunc",1,STRING,&msg))
    {
    fprintf(stderr,"Wrong type of argument: msg! \n");
    return "FALSE";
```

```
    }
  MainPtr = FindDefmodule("MAIN");
  InstPtr = FindInstance(MainPtr,"pipe-ptr",CLIPS_FALSE);

  DirectGetSlot(InstPtr,"pipe",&pipe);                                    40
  write_pipe = (FILE *) DOToPointer(pipe);
  fprintf(write_pipe,"%s\n",DOToString(msg));
  return DOToString(msg);
  ;;return "TRUE";
  }

 int UserFunctions()
   {
   extern VOID TclFunc();
                                                                          50
   DefineFunction2("TclFunc",'u',PTIF TclFunc, "TclFunc", "11s");
   }
}

/*********************************************************************/
/*                                                                   */
/*  Initialize CLIPS                                                 */
/*                                                                   */
/*********************************************************************/
void C_InitClips(FILE *pipe_ptr, char *student, char *path)           60
{
  char tempBuffer[MAXSTRING];
  char temppath[strlen(path)+25];
  int threshold;
  DATA_OBJECT rtn,vPtr,pipe;
  VOID *MainPtr, *StudentPtr, *ExpertPtr, *InstPtr;

  InitializeCLIPS();
  SetDynamicConstraintChecking (TRUE);
  SetStaticConstraintChecking (TRUE);                                     70
  strcpy(temppath,path);
  strcat(temppath,"/clips/main.clp");
```

```
  Load(temppath);

  MainPtr = FindDefmodule("MAIN");
  ExpertPtr = FindDefmodule("EXPERT");
  StudentPtr = FindDefmodule("STUDENT");

  SetCurrentModule(ExpertPtr);
  strcpy(temppath,path);                                                    80
  strcat(temppath,"/clips/both.clp");
  Load(temppath);
  strcpy(temppath,path);
  strcat(temppath,"/clips/expert.clp");
  Load(temppath);
  SetCurrentModule(StudentPtr);
  strcpy(temppath,path);
  strcat(temppath,"/clips/both.clp");
  Load(temppath);
  strcpy(temppath,path);                                                    90
  strcat(temppath,"/clips/student.clp");
  Load(temppath);
  SetCurrentModule(MainPtr);
  CLIPSFunctionCall("init",student,&rtn);

  // Insert pointer to pipe to Tcl/Tk into CLIPS object
  InstPtr = FindInstance(MainPtr,"pipe-ptr",CLIPS_FALSE);
  SetpType(&pipe,EXTERNAL_ADDRESS);
  SetpValue(&pipe,pipe_ptr);
  DirectPutSlot(InstPtr,"pipe",&pipe);                                     100
  CLIPSFunctionCall("CLIPS-get-threshold",NULL,&rtn);
} // end function C_InitClips

/*********************************************************************/
/* Exec the named cmd as a child process, returning two pipes to   */
/* communicate with the process, and the child's process ID        */
/*********************************************************************/
int start_child(char *cmd, FILE **readpipe, FILE **writepipe)
  {
```

```
int childpid, pipe1[2], pipe2[2];                                    110

if ((pipe(pipe1) < 0) || (pipe(pipe2) < 0))
  {
  perror("pipe");
  exit(-1);
  }

if ((childpid = vfork()) < 0)
  {
  perror("fork");                                                    120
  exit(-1);
  }
else if (childpid > 0)        // Parent
  {
  close(pipe1[0]);
  close(pipe2[1]);

  //  Write to child is pipe1[1], read from child is pipe2[0]
  *readpipe = fdopen(pipe2[0],"r");
  *writepipe = fdopen(pipe1[1],"w");                                 130
  if (setvbuf(*writepipe,NULL,_IOLBF,100) != 0)
    {
    perror("no line buffering");
    exit(-1);
    }
  return childpid;
  }
else  // Child
  {
  close(pipe1[1]);                                                   140
  close(pipe2[0]);

  //  Read from parent is pipe1[0], write to parent is pipe2[1]
  dup2(pipe1[0],0);
  dup2(pipe2[1],1);
  close(pipe1[0]);
```

```
    close(pipe2[1]);

    if (execlp(cmd,cmd,NULL) < 0)
      perror("execlp");                                                    150

    // Never returns
    }
  } // end of procedure start_child

/*********************************************************************/
/*                                                                   */
/*  Man behind the curtain program to simulate an ITS.               */
/*                                                                   */
/*********************************************************************/                                            160
int main(int argc, char **argv)
{
  DATA_OBJECT rtn;
  char *buf, *path;

  // Set up pipe for Tcl process
  FILE *read_from, *write_to;
  char result[MAXSTRING], textBuf[MAXSTRING], cmd[MAXSTRING];
  int childpid;
                                                                           170
  path = getenv("CURTAIN_PATH");  //get path from env var
  if (path == NULL) // if no CURTAIN_PATH env var...
    {
    cout << "You need to set an environment variable CURTAIN_PATH "
         << "with the path to curtain directory.  Continuing with "
         << "current directory." << endl;
    path = getenv("PWD");  //assign path to current directory
    }
  if (path == NULL) exit(0);  // if no current dir, exit program
  childpid = start_child("wish",&read_from,&write_to);                     180
  // Tell wish to read the init script
  fprintf(write_to,"source %s/TCL/curtain.tcl\n",path);
```

```
while(1)
  { // Blocks on read from wish
  if (fgets(result,80,read_from) <= 0) exit(0); // Exit if wish dies

  // Scan the string from wish
  if ((sscanf(result,"%s %[^\"]",cmd,textBuf)) <= 2)
    { // sort out which procedure must be called                          190

    // C_InitClips
    if (strcmp(cmd,"C_InitClips") == 0)
      C_InitClips(write_to,textBuf,path);  // Call C_InitClips

    // CLIPS-quit
    if (strcmp(cmd,"CLIPS-quit") == 0)
      CLIPSFunctionCall("quit",NULL,&rtn);

    // CLIPS-get-student-concept function                                 200
    if (strcmp(cmd,"CLIPS-get-student-concept") == 0)
      { // Call get-student-concept CLIPS function
      char tmp[MAXSTRING];
      if ((sscanf(textBuf,"%s",tmp)) == 1)
        {
        CLIPSFunctionCall(cmd,textBuf,&rtn);
        buf = strdup(DOToString(rtn));
        sscanf(buf,"%s",cmd);
        if (strcmp(cmd,"TRUE")!=0)
          cerr << "CLIPS-get-student-concept returned illegal \          210
            value: " << cmd << endl;
        free(buf);
        }
      else
        cerr << "Bad change-threshold command: "
             << cmd << textBuf << endl;
      }

    // CLIPS-get-threshold function
    if (strcmp(cmd,"CLIPS-get-threshold") == 0)                          220
```

```
  { // Call get-threshold CLIPS function after getting parms
  CLIPSFunctionCall(cmd,NULL,&rtn);
  buf = strdup(DOToString(rtn));
  sscanf(buf,"%s",cmd);
  if (strcmp(cmd,"TRUE")!=0)
    cerr << "CLIPS-get-threshold returned illegal value: "
      << cmd << endl;
  free(buf);
  }

// CLIPS-change-threshold function
if (strcmp(cmd,"CLIPS-change-threshold") == 0)
  { // Call change-threshold CLIPS function after getting parms
  char tmp[MAXSTRING];
  if ((sscanf(textBuf,"%s",tmp)) == 1)
    {
    CLIPSFunctionCall(cmd,textBuf,&rtn);
    buf = strdup(DOToString(rtn));
    sscanf(buf,"%s",cmd);
    if (strcmp(cmd,"TRUE")!=0)
      cerr << "CLIPS-change-threshold returned illegal value: "
        << cmd << endl;
    free(buf);
    }
  else
    cerr << "Bad change-threshold command: "
        << cmd << textBuf << endl;
  }

// CLIPS-get-definition function
if (strcmp(cmd,"CLIPS-get-definition") == 0)
  { // Call the get-definition CLIPS function after getting parms
  char tmp[MAXSTRING];
  if ((sscanf(textBuf,"%s",tmp)) == 1)
    {
    CLIPSFunctionCall(cmd,textBuf,&rtn);
    buf = strdup(DOToString(rtn));
```

```
    sscanf(buf,"%s",cmd);
    if (strcmp(cmd,"TRUE")!=0)
      cerr << "CLIPS-get-definition returned illegal value: "
        << cmd << endl;
    free(buf);
    }
  else
    cerr << "Bad get-definition command: "
        << cmd << textBuf << endl;
  }

// CLIPS-define-problem function
if (strcmp(cmd,"CLIPS-define-problem") == 0)
  { // Call the define-problem CLIPS function after getting parms
  int tmp;
  if ((sscanf(textBuf,"%i %i %i",&tmp,&tmp,&tmp)) == 3)
    {
    CLIPSFunctionCall(cmd,textBuf,&rtn);
    buf = strdup(DOToString(rtn));
    free(buf);
    }
  else
    cerr << "Bad define-problem command: "
        << cmd << textBuf << endl;
  }

// CLIPS-relate function
if (strcmp(cmd,"CLIPS-relate") == 0)
  { // Call the relate CLIPS function after getting parms
  int tmp;
  char tBuf[MAXSTRING];
  if ((sscanf(textBuf,"%i %i %s %s",&tmp,&tmp,tBuf,tBuf)) == 4)
    {
    CLIPSFunctionCall(cmd,textBuf,&rtn);
    buf = strdup(DOToString(rtn));
    sscanf(buf,"%s %[^\"]",cmd,textBuf);
    if (strcmp(cmd,"TRUE")!=0)
```

```
      cerr << "CLIPS-relate returned illegal value: "
        << cmd << endl;
    free(buf);
    }
  else
    cerr << "Bad relate command: " << cmd << textBuf << endl;
  }

// CLIPS-solution function
if (strcmp(cmd,"CLIPS-solution") == 0)
  { // Call the CLIPS-solution function after getting parms
  int tmp;
  if ((sscanf(textBuf,"%i",&tmp)) == 1)
    {
    CLIPSFunctionCall(cmd,textBuf,&rtn);
    buf = strdup(DOToString(rtn));
    sscanf(buf,"%s",cmd);
    if (strcmp(cmd,"TRUE")!=0)
      cerr << "CLIPS-solution returned illegal value: "
        << cmd << endl;
    free(buf);
    }
  else
    cerr << "Bad solution command: " << cmd << textBuf << endl;
  }

// CLIPS-get-mag function
if (strcmp(cmd,"CLIPS-get-mag") == 0)
  { // Call the CLIPS-get-mag function after getting parms
  int tmp;
  if ((sscanf(textBuf,"%i",&tmp)) == 1)
    {
    CLIPSFunctionCall(cmd,textBuf,&rtn);
    buf = strdup(DOToString(rtn));
    sscanf(buf,"%s %[^\"]",cmd,textBuf);
    if (strcmp(cmd,"tcl_IEOmsg")==0)
      fprintf(write_to,"tcl_IEOmsg \"%s\"\n",textBuf);
```

```
      else if (strcmp(cmd,"NULL")!=0)
        cerr << "CLIPS-get-mag returned illegal value: "
          << cmd << endl;
        free(buf);
        }
      else
        cerr << "Bad get-mag command: " << cmd << textBuf << endl;
      }
    }

  else
    cerr << "Bad command: " << cmd << textBuf << endl;
  }

/*****************************************************************/
/*  Clean up before exiting...                                   */
/*                                                               */
/*****************************************************************/
return 0;
}
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                       MAIN CONSTRUCTS                         ;;
;; These are constructs which are defined once, and are exported ;;
;; to both EXPERT and STUDENT.                                   ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defmodule MAIN (export ?ALL))
(defglobal ?*result* = NULL)
(defglobal ?*student* = NULL)
(defglobal ?*given-angle-nbr* = NULL)
(defglobal ?*find-angle-nbr* = NULL)
(defglobal ?*given-angle* = NULL)
(defglobal ?*find-angle* = NULL)
(defglobal ?*given-mag* = NULL)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                          CLASSES                        ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defclass MAIN-SOLUTION "object specifying a solution to a problem"
  (is-a USER)
  (slot given-angle
    (create-accessor read-write)
    (type INSTANCE-NAME)
    (default ?NONE)
    (visibility public)
  )
  (slot find-angle
    (create-accessor read-write)
    (type INSTANCE-NAME)
    (default ?NONE)
    (visibility public)
  )
)

(defclass WORK-SOLUTION "list multiple solutions to a problem"
```

```
  (is-a MAIN-SOLUTION)
  (multislot solution-track
    (create-accessor read-write)
    (type LEXEME))
)                                                                    40

(defclass EXPERT-SOLUTION "list multiple solutions to a problem"
  (is-a WORK-SOLUTION)
  (role concrete)
)

(defclass STUDENT-SOLUTION "list multiple solutions to a problem"
  (is-a WORK-SOLUTION)
  (role concrete)
)                                                                    50

(defclass PIPE-ADDRESS
  (is-a USER)
  (role concrete)
  (slot pipe
    (create-accessor read-write)
    (type EXTERNAL-ADDRESS))
)

(defclass STUDENT-MODEL                                              60
  (is-a USER)
  (role concrete)
  (pattern-match reactive)
  ;;  slot to record number of correct responses required for mastery
  (slot mastery-threshold
    (create-accessor read-write)
    (type INTEGER)
    (pattern-match reactive)
    (default 3)
    (range 1 ?VARIABLE))                                             70
  (multislot student-misconceptions
    (create-accessor read-write)
```

```
  (type SYMBOL)
  (pattern-match non-reactive)
  (allowed-values supplementary congruent adjacent vertical
    alternate-interior corresponding)
  ;; assume new student knows nothing <grin>
  (default supplementary congruent adjacent vertical
    alternate-interior corresponding))
 (multislot supplementary                                          80
  (create-accessor read-write)
  (type SYMBOL)
  (allowed-values correct incorrect))
 (multislot congruent
  (create-accessor read-write)
  (type SYMBOL)
  (allowed-values correct incorrect))
 (multislot adjacent
  (create-accessor read-write)
  (type SYMBOL)                                                    90
  (allowed-values correct incorrect))
 (multislot vertical
  (create-accessor read-write)
  (type SYMBOL)
  (allowed-values correct incorrect))
 (multislot alternate-interior
  (create-accessor read-write)
  (type SYMBOL)
  (allowed-values correct incorrect))
 (multislot corresponding                                         100
  (create-accessor read-write)
  (type SYMBOL)
  (allowed-values correct incorrect))
)

(defclass CONCEPT
 (is-a USER)
 (role concrete)
 (slot defn
```

```
    (create-accessor read-write)                                    110
    (access initialize-only)
    (type STRING)
    (default ?NONE))
)

(definstances CONCEPTS
  (supplementary of CONCEPT
    (defn (str-cat "Two angles are supplementary angles if and only if"
     " the sum of their magnitudes is 180 degrees."
     " e.g. Angle1 and Angle2 are supplementary.")))                120
  (congruent of CONCEPT
    (defn (str-cat "Two angles which have the same measure are"
     " congruent.  e.g. Angle2 and Angle3 are congruent.")))
  (adjacent of CONCEPT
    (defn (str-cat "Two angles are adjacent if and only if they are in"
     " the same plane and share a common side so that their non-common"
     " sides lie in different half-planes determined by the line"
     " containing the common ray.  e.g. Angle1 and Angle2 are adjacent"
     " angles.")))
  (vertical of CONCEPT                                              130
    (defn (str-cat "If the sides of two angles form two pairs of"
     " opposite rays, then the angles are vertical angles.  e.g. Angle1"
     " and Angle3 are vertical angles.")))
  (alternate-interior of CONCEPT
    (defn (str-cat "Two angles are alternate interior angles"
     " if and only if they lie on opposite sides of the traversal, but"
     " between the two lines traversed.  e.g. Angle3 and Angle6 are"
     " alternate interior angles.")))
  (corresponding of CONCEPT
    (defn (str-cat "Two angles are corresponding angles if and"         140
     " only if they lie on the same side of the traversal, and there"
     " exists an angle which is vertical to one of the angles, and"
     " alternate interior to the other.  e.g. Angle2 and Angle6 are"
     " corresponding angles.")))
)
```

```
(definstances PIPES (pipe-ptr of PIPE-ADDRESS))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                          FUNCTIONS                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Convenience function for testing the equality of strings
(deffunction str-eq (?str1 ?str2)
  (= 0 (str-compare ?str1 ?str2))
)

;; Convenience function for returning the last member in a multislot
(deffunction last$ ($?ms)
  (progn$ (?generic $?ms) (bind ?result ?generic))
  (return ?result)
)

;; Convenience function for returning the last word in a string
(deffunction last (?str)
  (return (last$ (explode$ ?str)))
)

;; Convenience function for returning the first word in a string
(deffunction first (?str)
  (return (nth$ 1 (explode$ ?str)))
)

;; Convenience function for appending member to multislot
(deffunction append$ (?ms ?append)
  (return (insert$ ?ms (+ 1 (length$ ?ms)) ?append))
)

;; Convenience function for switching the first member in a multislot
;; to be the last member
(deffunction front-to-back$ (?ms)
  (return (create$ (rest$ ?ms) (first$ ?ms)))
)
```

```
;; Shortcut for front-to-back$
(deffunction ftb$ (?ms)
  (front-to-back$ ?ms)
)

;; Convenience function for switching the first word in a string
;; to be the last word
(deffunction front-to-back (?str)
  (return (implode$ (front-to-back$ (explode$ ?str))))
)

;; Shortcut for front-to-back
(deffunction ftb (?ms)
  (front-to-back ?ms)
)

;; Function to reverse the elements in multi-slot, and quantify
;; correct as -1 and incorrect as +1
(deffunction quantify-model$ (?ms)
  (bind $?result (create$))
  (progn$ (?quality ?ms)
    (if (str-eq ?quality correct)
     then (bind ?quantity -1)
     else (bind ?quantity 1)
    )
    (bind $?result (insert$ ?result 1 ?quantity))
  )
  (return ?result)
)

;; This function initializes the CLIPS environment and sets up
;;   the geometry domain
(deffunction init (?student)
  (unwatch all)
  ;;(dribble-on debug.txt)
  ;;(watch facts)
```

```
  ;;(watch rules)
  ;;(watch deffunctions)
  ;;(watch statistics)
  ;;(watch activations)
  ;;(watch instances)
  ;;(watch focus)

  (reset)
  (close)
  (set-static-constraint-checking TRUE)                                    230
  (set-dynamic-constraint-checking TRUE)
  (bind ?*student* ?student)
  (make-instance ?*student* of STUDENT-MODEL)
  (if (open (str-cat ?*student* ".model") tempFile "r")
   then
    (close tempFile)
    (load-instances (str-cat ?*student* ".model"))
  )
  (open (str-cat ?*student* ".log") log "w")
  (printout log                                                            240
    "Logfile to store student approaches to problem solving." crlf crlf)
  (run)
)

;;  This function displays the message included in the parameters
;;  of the function call to the STUDENT.  The message is
;;  NOT enclosed by quotes.  NOTE:  avoid the use of single and
;;  double quotes.
;;  - textstring: a multivalue variable which is converted to a string
(deffunction stu-msg ($?text)                                              250
  (if (not (str-eq nil (nth$ 1 $?text)))
   then (bind ?message (str-cat "tcl_student_msg " (implode$ $?text)))
  )
  (TclFunc ?message)
  (return)
)
```

```
;;  This function displays the message included in the parameters
;;  of the function call to the INSTRUCTOR only.  The message is
;;  NOT enclosed by quotes.  NOTE:  avoid the use of single and                    260
;;  double quotes.
;;  - textstring: a multivalue variable which is converted to a string
(deffunction IEOmsg ($?text)
  (if (not (str-eq nil (nth$ 1 $?text)))
   then (bind ?message (str-cat "tcl_IEOmsg " (implode$ $?text)))
  )
  (TclFunc ?message)
  (return)
)
                                                                                 270
;;  This function binds results from rules for use in the original
;;  calling function.
(deffunction bindresult ($?textstring)
  (if (= 0 (length$ $?textstring))
   ;; empty parmlist -> NULLify result global
   then (bind ?*result* NULL)
   ;; bind parms to current result global
   else
    (bind ?result (implode$ $?textstring))
    (if (str-eq ?*result* NULL)                                                  280
     then (bind ?*result* ?result)
     else (bind ?*result* (str-cat ?*result* " " ?result))
    )
  )
)

;;  This is a function to get the definition of a geometric term
;;  - term: the term which needs defined
(deffunction get-definition (?term)
  (send (instance-name ?term) get-defn)                                          290
)

;;  Shortcut for get-definition
(deffunction gd (?term)
```

```
  (get-definition ?term)
)

;;  quit saves the current student model before exiting CLIPS
(deffunction quit ()
  (save-instances (str-cat ?*student* .model) local STUDENT-MODEL)
  (exit)
)

;;  This function prints out the specified student-model
(deffunction printstudent (?student)
  (send (instance-name ?student) print)
)

;;  Shortcut for printstudent
(deffunction ps ()
  (printstudent ?*student*)
)

;;  Function to compare two solutions
;;   - if the first solution track is a subset of the second,
;;     then the second solution is a valid one.
;;   - if the first solution matches the second,
;;     then the second is the exact same solution.
(deffunction compare-solutions (?expert-soln ?student-soln)
  (bind ?expert-track (send ?expert-soln get-solution-track))
  (bind ?student-track (send ?student-soln get-solution-track))
  (bind ?result NULL)
  (progn$ (?pair ?expert-track)
    (if (not (member$ ?pair ?student-track))
     ;; student solution completely different from expert
     then
      (bind ?result NO-MATCH)
      (break)
    )
  )
  (if (str-eq ?result NULL)
```

```
   then
    (progn$ (?pair ?student-track)
     (if (not (member$ ?pair ?expert-track))
      ;; student solution _contains_ the expert solution, but also
      ;; contains unnecessary steps
      then
       (bind ?result MATCH)
       (break)
      )                                                              340
     )
    )
   (if (str-eq ?result NULL)
    ;; student solution is _exactly_ the same as the expert solution
    then
     (bind ?result PERFECT-MATCH)
   )
   (return ?result)
)
                                                                     350
;; Shortcut for compare-solutions
(deffunction c-s ()
 (compare-solutions [expert-soln] [student-soln])
)

;; Function to copy all the EXPERT solutions for the current problem
;; into the MAIN module
(deffunction get-expert-solutions ()
  (assert (get-expert-solutions ?*given-angle* ?*find-angle*))
  (run)                                                              360
  (return TRUE)
)

;; Shortcut for get-expert-solutions
(deffunction g-e-s ()
 (get-expert-solutions)
)
```

```
;; Function to copy the STUDENT solution for the current problem
;; into the MAIN module                                              370
(deffunction get-student-solution ()
  (assert (get-student-solution ?*given-angle* ?*find-angle*))
  (run)
  (return TRUE)
)

;; Shortcut for get-student-solution
(deffunction g-s-s ()
 (get-student-solution)
)                                                                    380

;; Function to copy all the EXPERT and STUDENT solutions for the
;; current problem into the MAIN module
(deffunction get-all-solutions ()
  (g-s-s)
  (g-e-s)
)

;; Shortcut for get-all-solutions
(deffunction g-a-s ()                                                390
 (get-all-solutions)
)

;; Function to remove all the solutions from MAIN
(deffunction delete-all-solutions ()
  (do-for-all-instances ((?soln EXPERT-SOLUTION)) TRUE
    (send ?soln delete)
  )
  (do-for-all-instances ((?soln STUDENT-SOLUTION)) TRUE
    (send ?soln delete)                                              400
  )
)

;; Shortcut for delete-all-solutions
(deffunction d-a-s ()
```

```
 (delete-all-solutions)
)

;; function to get all expert magnitudes
(deffunction expert-gma ()                                          410
  (assert (expert-gma))
  (run)
)

;; shortcut for function expert-gma
(deffunction egma ()
  (expert-gma)
)

;; function to get an expert magnitude                              420
(deffunction expert-gm (?angle)
  (assert (expert-gm ?angle))
  (run)
  (bind ?result ?*result*)
  (bindresult)
  (return ?result)
)

;; shortcut for function expert-gm
(deffunction egm (?nbr)                                             430
  (return (expert-gm (sym-cat Angle ?nbr)))
)

;; function to get a student magnitude
(deffunction student-gm (?angle)
  (assert (student-gm ?angle))
  (run)
  (bind ?result ?*result*)
  (bindresult)
  (return ?result)                                                  440
)
```

```
;; shortcut for function student-gm
(deffunction sgm (?nbr)
  (return (student-gm (sym-cat Angle ?nbr)))
)

;; function to get all student magnitudes
(deffunction student-gma ()
  (assert (student-gma))
  (run)
)

;; shortcut for function student-gma
(deffunction sgma ()
  (student-gma)
)

;; function to change the mastery threshold in the current student
(deffunction change-threshold (?threshold)
  (send (instance-name ?*student*) put-mastery-threshold ?threshold)
)

;; shortcut for function change-threshold
(deffunction c-t (?threshold)
  (change-threshold ?threshold)
)

;; Function to get the relation between two angles as defined by
;; the EXPERT module
(deffunction expert-relate (?ang1 ?ang2)
  (assert (expert-relate ?ang1 ?ang2))
  (run)
  (bind $?result (explode$ ?*result*))
  (bindresult)
  (return $?result)
)

;; Shortcut for function expert-relate
```

```
(deffunction e-r (?nbr1 ?nbr2)                                          480
  (bind ?ang1 (sym-cat Angle ?nbr1))
  (bind ?ang2 (sym-cat Angle ?nbr2))
  (expert-relate ?ang1 ?ang2)
)

;; Function to get the expert magnitude relation for a specified angle
;; relation
(deffunction expert-mag-relation (?relation)
  (assert (expert-mag-relate ?relation))
  (run)                                                                 490
  (bind ?result (sym-cat ?*result*))
  (bindresult)
  (return ?result)
)

;; Shortcut for function expert-mag-relation
(deffunction emr (?relation)
  (expert-mag-relation ?relation)
)
                                                                        500
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; The following functions are support functions designed to be
;; called from outside of the CLIPS environment.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; This is a function to get a definition for a geometric term.
;; Designed to be called from outside the CLIPS environment.
(deffunction CLIPS-get-definition (?term)
  (bind ?message (get-definition ?term))
  (stu-msg ?message)                                                    510
  (return TRUE)
)

;; Shortcut for function CLIPS-get-definition
(deffunction C-gd (?term)
  (CLIPS-get-definition ?term)
```

```
  (return TRUE)
)

;; Function to display the magnitude of a STUDENT angle. Designed to
;; be called from outside the CLIPS environment.
;; - nbr: the number of the angle whose magnitude you wish to know
(deffunction CLIPS-stu-get-mag (?angle)
  (bind ?message (str-cat ?angle " = " (student-gm ?angle)))
  (IEOmsg ?message)
  (return TRUE)
)

;; Shortcut for function CLIPS-stu-get-mag
(deffunction C-sgm (?nbr)
  (return (CLIPS-stu-get-mag (sym-cat Angle ?nbr)))
)

;; Function to display the magnitude of an EXPERT angle. Designed to
;; be called from outside the CLIPS environment.
;; - nbr: the number of the angle whose magnitude you wish to know
(deffunction CLIPS-exp-get-mag (?angle)
  (bind ?message (str-cat ?angle " = " (expert-gm ?angle)))
  (IEOmsg ?message)
  (return TRUE)
)

;; Shortcut for function CLIPS-exp-get-mag
(deffunction C-egm (?nbr)
  (return (CLIPS-exp-get-mag (sym-cat Angle ?nbr)))
)

;; This is a function to build parallel, bisected line problems.
;; Designed to be called from outside of the CLIPS environment.
;; - given: the angle number of the given angle magnitude
;; - mag: the magnitude of the given angle
;; - find: the angle whose magnitude the student is to find
(deffunction CLIPS-define-problem (?given ?mag ?find)
```

```
  (bind ?*given-angle-nbr* ?given)
  (bind ?*find-angle-nbr* ?find)
  (bind ?*given-angle* (sym-cat Angle ?given))
  (bind ?*find-angle* (sym-cat Angle ?find))
  (bind ?*given-mag* ?mag)
  (printout log crlf
       "Given:  Magnitude of angle " ?given " = " ?mag crlf)                    560
  (printout log "Find:  Magnitude of angle " ?find " = ? " crlf)
  (assert (define-problem ?given ?mag ?find))
  (run)
  (return TRUE)
)

;;  Shortcut for function CLIPS-define-problem
(deffunction C-dp (?given ?mag ?find)
  (CLIPS-define-problem ?given ?mag ?find)
  (return TRUE)                                                               570
)

;;  This is a function to compare the student provided solution
;;  to the expert derived solution.  Designed to be called from
;;  outside the CLIPS environment.
(deffunction CLIPS-solution (?mag)
  (bind ?expert-mag (expert-gm ?*find-angle*))
  ;; check solution tracks
  (get-all-solutions)
  (do-for-instance ((?stu-soln STUDENT-SOLUTION)) TRUE                         580
    (bind ?student-solution ?stu-soln)
  )
  (progn$ (?exp-soln
           (find-all-instances ((?exp-soln EXPERT-SOLUTION)) TRUE)
          )
    (bind ?result (compare-solutions ?exp-soln ?student-solution))
    (if (or (str-eq ?result "MATCH") (str-eq ?result "PERFECT-MATCH"))
     then
      (bind ?expert-solution ?exp-soln)
      (break)                                                                 590
```

```
  )
)
(bind ?student-track (send ?student-solution get-solution-track))
(if (str-eq ?result "NO-MATCH")
 then
  (bind ?message (str-cat "The student solution strategy does "
                          "not match the expert."))
  ;; Check to see if student solution strategy is a valid approach
  (bind ?valid TRUE)
  (progn$ (?angle-pair ?student-track)                                  600
    (bind ?ang1 (sym-cat (sub-string 2 7 ?angle-pair)))
    (bind ?ang2 (sym-cat (sub-string 9 14 ?angle-pair)))
    (bind ?related (implode$ (expert-relate ?ang1 ?ang2)))
    (if (str-eq "UNRELATED UNRELATED" ?related)
      then (bind ?valid FALSE)
    )
  )
  (if (str-eq ?valid FALSE)
   then
    (bind ?message                                                      610
      (str-cat ?message
        "\\n The student's solution path is NOT valid.")
    )
   else  ;; check for path from given angle to goal
         ;; by executing student model get-magnitude
         ;; for the find angle.  if it has the solution
         ;; magnitude, then there is a correct path from
         ;; given to find
    (bind ?student-mag (student-gm ?*find-angle*))
    (if (str-eq ?expert-mag ?student-mag)                               620
     then ;; new valid solution
      (bind ?message (str-cat ?message
        "\\n However, the student solution path IS valid."))
     else  ;; not a valid solution
      (bind ?message (str-cat ?message
        "\\n And, the student solution path IS NOT valid."))
    )
```

```
  )
 else
  (if (str-eq ?result "MATCH")                                          630
   then
    (bind ?message "The student solution strategy matches the expert,\
         \\n but the student has more steps than necessary:")
   else
    (if (str-eq ?result "PERFECT-MATCH")
     then
      (bind ?message (str-cat "The student solution strategy "
                              "exactly matches the expert: "))
     else
      (bind ?message "ERROR in CLIPS-solution!")                        640
    )
  )
  (bind ?expert-track (send ?expert-solution get-solution-track))
  (bind ?message
    (str-cat ?message
      "\\n\\tStudent relate pairs: " (implode$ ?student-track)
      "\\n\\tExpert relate pairs: " (implode$ ?expert-track)
      "\\n" )
  )
 )                                                                      650
 (IEOmsg ?message)

 (delete-all-solutions)

 ;; check resulting magnitude found
 (bind ?message (str-cat "Expert magnitude = " ?expert-mag
   "\\nStudent magnitude = "?mag "\\n" ))
 (IEOmsg ?message)
 (bindresult)
 (return TRUE)                                                          660
)

;;  Shortcut for function CLIPS-solution
(deffunction C-s (?mag)
```

```
  (CLIPS-solution ?mag)
  (return TRUE)
)

;; Function to define relationship between two angles, as defined by
;; the student.  Designed to be called from outside of the                    670
;; CLIPS environment.
;; - nbr1: the angle number of first angle to relate
;; - nbr2: the angle number of second angle to relate
;; - relation: the geometric relation specified by the student
;; - relation-defn: specifies whether the student believes the
;;      relation indicates congruent or supplementary angles.
(deffunction CLIPS-relate (?nbr1 ?nbr2 ?relation ?relation-defn)
  (assert (relate ?nbr1 ?nbr2 ?relation ?relation-defn))
  (run)
  ;; add to student model                                                    680
  (set-current-module MAIN)
  (bind $?expert-relations (e-r ?nbr1 ?nbr2))
  (bind ?expert-relation (nth$ 1 $?expert-relations))
  (bind ?expert-relation-defn (nth$ 2 $?expert-relations))
  (bind ?message
    (str-cat "Relate Angle" ?nbr1 " to Angle" ?nbr2))
  (if (str-eq ?relation ?expert-relation)
   then
    (bind ?message (str-cat ?message "\\n\\t" ?relation ": correct"))
    (slot-insert$ (instance-name ?*student*) ?expert-relation 1 correct)      690
   else
    (bind ?message (str-cat ?message "\\n\\t" ?relation ": incorrect"))
    (if (str-eq "UNRELATED" ?expert-relation)
     then
      (slot-insert$ (instance-name ?*student*) ?relation 1 incorrect)
     else
      (slot-insert$ (instance-name ?*student*) ?expert-relation 1
        incorrect)
    ) ;; end if
  ) ;; end if                                                               700
  (run)
```

```
  (if (str-eq ?relation-defn ?expert-relation-defn)
   then
    (bind ?message
      (str-cat ?message "\\n\\t" ?relation-defn ": correct"))
    (slot-insert$ (instance-name ?*student*) ?expert-relation-defn
      1 correct)
   else
    (bind ?message                                                    710
      (str-cat ?message "\\n\\t" ?relation-defn ": incorrect"))
    (if (not (str-eq "UNRELATED" ?expert-relation-defn))
     then
      (slot-insert$ (instance-name ?*student*) ?expert-relation-defn 1
        incorrect)
     ;; else do nothing because no data on cong vs suppl
    ) ;; end if
  ) ;; end if
  (run)
  (IEOmsg ?message)                                                   720

  ;; save results to the student log
  (printout log "relating Angle" ?nbr1 " to Angle" ?nbr2 ": " crlf)
  (printout log "  student: " ?relation " " ?relation-defn crlf)
  (printout log "  expert: " ?expert-relation " " ?expert-relation-defn
        crlf)
  (return TRUE)
)  ;; end of functions CLIPS-relate

;;  Shortcut for function CLIPS-relate                               730
(deffunction C-r (?nbr1 ?nbr2 ?relation ?relation-defn)
  (CLIPS-relate ?nbr1 ?nbr2 ?relation ?relation-defn)
  (return TRUE)
)

;;  Function to print all EXPERT solutions to the specified problem
(deffunction CLIPS-expert-solutions (?given ?find)
  (assert (expert-solutions ?given ?find))
```

```
  (run)
)                                                                    740

;;  Shortcut for function CLIPS-expert-solutions
(deffunction C-es (?given ?find)
  (CLIPS-expert-solutions ?given ?find)
)

;;  Function to print all STUDENT solutions to the specified problem
(deffunction CLIPS-student-solutions (?given ?find)
  (assert (student-solutions ?given ?find))
  (run)                                                              750
)

;;  Shortcut for function CLIPS-student-solutions
(deffunction C-ss (?given ?find)
  (CLIPS-student-solutions ?given ?find)
)

;;  Function to change mastery threshold from outside CLIPS
(deffunction CLIPS-change-threshold (?threshold)
  (change-threshold ?threshold)                                      760
  (return TRUE)
)

;;  Function to get the mastery threshold from outside CLIPS
(deffunction CLIPS-get-threshold ()
  (bind ?threshold
    (send (instance-name ?*student*) get-mastery-threshold))
  (TclFunc (str-cat "set curtain_threshold " ?threshold))
  (return TRUE)
)                                                                    770

;;  Function to get concept count in STUDENT-MODEL from outside CLIPS
(deffunction CLIPS-get-student-concept (?concept)
  (bind $?temp (send (instance-name ?*student*)
    (sym-cat get- ?concept)))
```

```
  (bind $?quantity_list (quantify-model$ ?temp))
  (TclFunc
    (str-cat "set curtain_list(" ?concept ")  {" (implode$
    $?quantity_list) "}")
  )                                                                    780
  (return TRUE)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                          RULES                                    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule update-supplementary-misconceptions "checks misconception list"
   (declare (auto-focus TRUE))
   ?inst <- (object (is-a STUDENT-MODEL)                               790
                    (supplementary $?correct&:(> (length$ $?correct) 0))
            )
  =>
   (assert
     (update-generic-misconception supplementary ?inst ?correct))
   (run)
)

(defrule update-congruent-misconceptions "checks misconception list"
   (declare (auto-focus TRUE))                                         800
   ?inst <- (object (is-a STUDENT-MODEL)
                    (congruent $?correct&:(> (length$ $?correct) 0))
            )
  =>
   (assert
     (update-generic-misconception congruent ?inst ?correct))
   (run)
)

(defrule update-alternate-interior-misconceptions "misconception list"  810
   (declare (auto-focus TRUE))
   ?inst <- (object (is-a STUDENT-MODEL)
```

```
            (alternate-interior $?correct&:(> (length$ $?correct) 0))
          )
  =>
   (assert
     (update-generic-misconception alternate-interior ?inst ?correct))
   (run)
)

(defrule update-corresponding-misconceptions "checks misconception list"
   (declare (auto-focus TRUE))
   ?inst <- (object (is-a STUDENT-MODEL)
                    (corresponding $?correct&:(> (length$ $?correct) 0))
            )
  =>
   (assert
     (update-generic-misconception corresponding ?inst ?correct))
   (run)
)

(defrule update-vertical-misconceptions "checks misconception list"
   (declare (auto-focus TRUE))
   ?inst <- (object (is-a STUDENT-MODEL)
                    (vertical $?correct&:(> (length$ $?correct) 0))
            )
  =>
   (assert
     (update-generic-misconception vertical ?inst ?correct))
   (run)
)

(defrule update-adjacent-misconceptions "checks misconception list"
   (declare (auto-focus TRUE))
   ?inst <- (object (is-a STUDENT-MODEL)
                    (adjacent $?correct&:(> (length$ $?correct) 0))
            )
  =>
   (assert
```

```
    (update-generic-misconception adjacent ?inst ?correct))                    850
  (run)
)

(defrule update-generic-misconceptions "generalize update process"
  ?temp <- (update-generic-misconception ?concept ?stu-model $?correct)
 =>
  (retract ?temp)
  (bind ?mastery (send ?stu-model get-mastery-threshold))
  (if (< (length$ ?correct) ?mastery)
   then (bind ?check ?correct)                                               860
   else (bind ?check (subseq$ ?correct 1 ?mastery))
  )
  (bind ?stu-misc (send ?stu-model get-student-misconceptions))
  (bind ?position (member$ ?concept ?stu-misc))
  (if (member$ incorrect ?check)
   ;; student has misapplied concept in last attempts
   then
    (if (not (integerp ?position))
     ;; concept is not already in the misconception list
     then  ;; add concept to misconception list                              870
      (send ?stu-model put-student-misconceptions
        (append$ ?stu-misc ?concept))
      (IEOmsg
        (str-cat ?*student* " has not mastered " ?concept " angles."))
    )
   else  ;; student may have demonstrated mastery over last attempts
    (if (and (integerp ?position) (= ?mastery (length$ ?check)))
     ;; concept is already in the misconception list AND
     ;;   student has successfully (and repeatedly) applied concept
     then  ;; delete concept from misconception list                          880
      (send ?stu-model put-student-misconceptions
         (delete$ ?stu-misc ?position ?position))
      (IEOmsg (str-cat ?*student* " has apparently mastered "
                       ?concept " angles."))
    )
   )
```

```
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                       ADDITIONAL MODULES                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defmodule EXPERT (import MAIN ?ALL))
(defmodule STUDENT (import MAIN ?ALL))
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                 EXPERT and STUDENT CONSTRUCTS                     ;;
;; These are constructs which must be defined for EXPERT and STUDENT;;
;; separately.                                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                          CLASSES                       ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defclass ANGLE
  (is-a USER)
  (role concrete)
  (pattern-match reactive)
  (slot magnitude
    (pattern-match reactive)
    (create-accessor read-write)
    (type INTEGER)
    (range -1 180)
    (default -1)
  )
)

(definstances ANGLE-INSTANCES
  (Angle1 of ANGLE)
  (Angle2 of ANGLE)
  (Angle3 of ANGLE)
  (Angle4 of ANGLE)
  (Angle5 of ANGLE)
```

```
  (Angle6 of ANGLE)                                                    30
  (Angle7 of ANGLE)
  (Angle8 of ANGLE)
)

;; field relate-angle-pairs points to instances of RELATION
(defclass SOLUTION "object specifying a solution to a problem"
  (is-a MAIN-SOLUTION)
  (role concrete)
  (pattern-match reactive)
  (multislot relate-angle-pairs                                        40
    (create-accessor read-write)
    (type INSTANCE-NAME)
  )
)

(defclass RELATION "container class for angle relations"
  (is-a USER)
  (role concrete)
  (pattern-match reactive)
  (slot angle-relation                                                 50
    (pattern-match reactive)
    (create-accessor read-write)
    (type SYMBOL)
    (allowed-values adjacent vertical corresponding alternate-interior)
    (default ?NONE)
  )
  (slot magnitude-relation
    (pattern-match reactive)
    (create-accessor read-write)
    (type SYMBOL)                                                      60
    (allowed-values congruent supplementary)
    (default ?NONE)
  )
  (multislot angles
    (pattern-match reactive)
    (create-accessor read-write)
```

```
    (type INSTANCE-NAME)
    (default ?NONE)
  )
)                                                                    70

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;  Message-handlers for objects
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;  set-mag sets the magnitude for an ANGLE object
(defmessage-handler ANGLE set-mag primary (?mag)
  (if (= ?self:magnitude -1) then
    (bind ?self:magnitude ?mag)
  else (printout werror (instance-name ?self) " already has a "                80
       "magnitude!" crlf)
  )
)

;;  myget-relate-angle-pairs goes out to the RELATION instances
;;  pointed to by the multislot, and retrieves the angle pairs related
(defmessage-handler SOLUTION myget-relate-angle-pairs ()
  (bind $?angle-pairs ?self:relate-angle-pairs)
  (bind $?result (create$))
  (if (progn$ (?angle-pair $?angle-pairs)                                    90
       (bind ?inst (instance-name ?angle-pair))
       (bind ?pair (implode$ (send ?inst get-angles)))
       (bind ?new-pair (sym-cat < (first ?pair) , (last ?pair) >))
       (bind $?result (append$ ?result ?new-pair))
     )
   then (return ?result)
   else (return NONE)
  )
)

                                                                           100
;;  pp pretty prints an instance of a SOLUTION object
(defmessage-handler SOLUTION pp primary ()
  ;;(bind $?angle-pairs ?self:relate-angle-pairs)
```

```
  (printout t (instance-name ?self) " of "
              (get-current-module)::(class ?self) crlf)
  (printout t " (given-angle " ?self:given-angle ")" crlf)
  (printout t " (find-angle " ?self:find-angle ")" crlf)
  (printout t " (Solution track:")
  (progn$ (?pair (send ?self myget-relate-angle-pairs))
    (printout t crlf "  (relate " ?pair ")")
  )
  (printout t ")" crlf)
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                          FUNCTIONS                                 ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; The following functions are user interface functions for use by
;; the instuctor.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Function to set the magnitude of an angle.
;; - angle: the angle to set
;; - mag: the magnitude to assign to the angle
(deffunction set-mag (?angle ?mag)
  (send (instance-name ?angle) set-mag ?mag)
  (run)
)

;; shortcut for set-mag
;; - nbr: the number of the angle to set
(deffunction sm (?nbr ?mag)
  (set-mag (sym-cat Angle ?nbr) ?mag)
)

;; Function to display the magnitude of an angle.
;; - angle: the angle whose magnitude is required
(deffunction get-mag (?angle)
  (bind ?mag (send (instance-name ?angle) get-magnitude))
```

```
  (run)
  (return ?mag)
)

;; shortcut for get-mag
;; - nbr: the number of the angle whose magnitude you wish to know
(deffunction gm (?nbr)
  (return (get-mag (sym-cat Angle ?nbr)))
)

;; Function to get ALL the angles' magnitudes.
(deffunction get-mag-all ()
  (loop-for-count (?nbr 1 8) do
    (printout t "Angle" ?nbr " = " (gm ?nbr) crlf)
  )
  (return)
)

;; shortcut for get-mag-all
(deffunction gma ()
  (get-mag-all)
)

;; This is a function to build parallel, bisected line problems
;; - given: the angle number of the given angle magnitude
;; - mag: the magnitude of the given angle
;; - find: the angle number whose magnitude the student is to find
(deffunction define-problem (?given ?mag ?find)
  (sm ?given ?mag)
)

;; Shortcut for function define-problem
;; - given: the angle number of the given angle magnitude
;; - mag: the magnitude of the given angle
;; - find: the angle whose magnitude the student is to find
(deffunction dp (?given ?mag ?find)
  (define-problem ?given ?mag ?find)
```

```
)

;;  Function to print ALL solutions                                    180
(deffunction print-solutions-all ()
  (do-for-all-instances ((?soln SOLUTION)) TRUE
    (send ?soln pp)
  )
)

;;  Shortcut for function print-solutions-all
(deffunction p-s-a ()
  (print-solutions-all)
)                                                                      190

;;  Function to print all solutions for the specified problem
(deffunction print-solutions (?given ?find)
  (do-for-all-instances ((?soln SOLUTION))
    (and (str-eq ?given (send ?soln get-given-angle))
         (str-eq ?find (send ?soln get-find-angle))
    )
    (send ?soln pp)
  )
)                                                                      200

;;  Shortcut for function print-solutions
(deffunction p-s (?given ?find)
  (print-solutions (str-cat Angle ?given) (str-cat Angle ?find))
)

;;  Function to copy solution to MAIN
(deffunction copy-solution (?old-soln)
  (bind ?soln-instance (instance-name ?old-soln))
  (make-instance of (sym-cat (get-current-module) -SOLUTION)           210
    (given-angle (send ?soln-instance get-given-angle))
    (find-angle (send ?soln-instance get-find-angle))
    (solution-track (send ?soln-instance myget-relate-angle-pairs))
  )
```

```
)

;; Function to make copies of  all solutions for the specified problem
(deffunction copy-all-solutions (?given ?find)
  (do-for-all-instances ((?soln SOLUTION))
    (and (str-eq ?given (send ?soln get-given-angle))                    220
         (str-eq ?find (send ?soln get-find-angle))
    )
    (copy-solution ?soln)
  )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                          RULES                                ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                                                                         230
(defrule congruent-angle "set magnitude on congruent angles"
   (declare (auto-focus TRUE))
   (object (is-a RELATION) (magnitude-relation congruent)
                           (angles ?a1 ?a2))
   (object (is-a ANGLE) (name ?a1) (magnitude ?mag1))
   (object (is-a ANGLE) (name ?a2) (magnitude ?mag2))
   (test (or (= ?mag1 -1) (= ?mag2 -1)))
   (test (!= ?mag1 ?mag2))
  =>
   (if (= ?mag1 -1) then (set-mag ?a1 ?mag2)                             240
   else (set-mag ?a2 ?mag1))
)

(defrule supplementary-angle "set magnitude of supplementary angles"
   (declare (auto-focus TRUE))
   (object (is-a RELATION) (magnitude-relation supplementary)
           (angles ?a1 ?a2))
   (object (is-a ANGLE) (name ?a1) (magnitude ?mag1))
   (object (is-a ANGLE) (name ?a2) (magnitude ?mag2))
   (test (or (= ?mag1 -1) (= ?mag2 -1)))                                250
   (test (!= ?mag1 ?mag2))
```

```
  =>
   (if (= ?mag1 -1) then (set-mag ?a1 (- 180 ?mag2))
   else (set-mag ?a2 (- 180 ?mag1)))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                        EXPERT CONSTRUCTS                               ;;
;;  These are constructs which must be defined for EXPERT module          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                        RELATION INSTANCES                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(definstances ALTERNATE-INTERIOR-RELATIONS
  (Relate3-6 of RELATION (angle-relation alternate-interior)
                         (magnitude-relation congruent)
                         (angles [Angle3] [Angle6]))
  (Relate4-5 of RELATION (angle-relation alternate-interior)
                         (magnitude-relation congruent)
                         (angles [Angle4] [Angle5]))
)

(definstances CORRESPONDING-RELATIONS
  (Relate1-5 of RELATION (angle-relation corresponding)
                         (magnitude-relation congruent)
                         (angles [Angle1] [Angle5]))
  (Relate2-6 of RELATION (angle-relation corresponding)
                         (magnitude-relation congruent)
                         (angles [Angle2] [Angle6]))
  (Relate3-7 of RELATION (angle-relation corresponding)
                         (magnitude-relation congruent)
                         (angles [Angle3] [Angle7]))
  (Relate4-8 of RELATION (angle-relation corresponding)
                         (magnitude-relation congruent)
                         (angles [Angle4] [Angle8]))
)

(definstances VERTICAL-RELATIONS
```

```
  (Relate1-4 of RELATION (angle-relation vertical)
                         (magnitude-relation congruent)
                         (angles [Angle1] [Angle4]))
  (Relate2-3 of RELATION (angle-relation vertical)
                         (magnitude-relation congruent)
                         (angles [Angle2] [Angle3]))
  (Relate5-8 of RELATION (angle-relation vertical)
                         (magnitude-relation congruent)
                         (angles [Angle5] [Angle8]))                    40
  (Relate6-7 of RELATION (angle-relation vertical)
                         (magnitude-relation congruent)
                         (angles [Angle6] [Angle7]))
)

(definstances ADJACENT-RELATIONS
  (Relate1-2 of RELATION (angle-relation adjacent)
                         (magnitude-relation supplementary)
                         (angles [Angle1] [Angle2]))
  (Relate1-3 of RELATION (angle-relation adjacent)                      50
                         (magnitude-relation supplementary)
                         (angles [Angle1] [Angle3]))
  (Relate2-4 of RELATION (angle-relation adjacent)
                         (magnitude-relation supplementary)
                         (angles [Angle2] [Angle4]))
  (Relate3-4 of RELATION (angle-relation adjacent)
                         (magnitude-relation supplementary)
                         (angles [Angle3] [Angle4]))
  (Relate5-6 of RELATION (angle-relation adjacent)
                         (magnitude-relation supplementary)             60
                         (angles [Angle5] [Angle6]))
  (Relate5-7 of RELATION (angle-relation adjacent)
                         (magnitude-relation supplementary)
                         (angles [Angle5] [Angle7]))
  (Relate6-8 of RELATION (angle-relation adjacent)
                         (magnitude-relation supplementary)
                         (angles [Angle6] [Angle8]))
  (Relate7-8 of RELATION (angle-relation adjacent)
```

```
                (magnitude-relation supplementary)
                (angles [Angle7] [Angle8]))                                    70
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                    SOLUTION INSTANCES                                    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(definstances EXPERT-SOLUTIONS
  (of SOLUTION
    (given-angle (instance-name Angle1))
    (find-angle (instance-name Angle2))
    (relate-angle-pairs [Relate1-2])
  )                                                                            80
  (of SOLUTION
    (given-angle (instance-name Angle2))
    (find-angle (instance-name Angle1))
    (relate-angle-pairs [Relate1-2])
  )
  (of SOLUTION
    (given-angle (instance-name Angle1))
    (find-angle (instance-name Angle3))
    (relate-angle-pairs [Relate1-3])
  )                                                                            90
  (of SOLUTION
    (given-angle (instance-name Angle3))
    (find-angle (instance-name Angle1))
    (relate-angle-pairs [Relate1-3])
  )
  (of SOLUTION
    (given-angle (instance-name Angle1))
    (find-angle (instance-name Angle4))
    (relate-angle-pairs [Relate1-4])
  )                                                                            100
  (of SOLUTION
    (given-angle (instance-name Angle4))
    (find-angle (instance-name Angle1))
    (relate-angle-pairs [Relate1-4])
  )
```

```
(of SOLUTION
  (given-angle (instance-name Angle1))
  (find-angle (instance-name Angle5))
  (relate-angle-pairs [Relate1-5])
)
(of SOLUTION
  (given-angle (instance-name Angle5))
  (find-angle (instance-name Angle1))
  (relate-angle-pairs [Relate1-5])
)
(of SOLUTION
  (given-angle (instance-name Angle1))
  (find-angle (instance-name Angle6))
  (relate-angle-pairs [Relate1-5] [Relate5-6])
)
;; here is one multiple solution for testing...
(of SOLUTION
  (given-angle (instance-name Angle1))
  (find-angle (instance-name Angle6))
  (relate-angle-pairs [Relate1-3] [Relate3-6])
)
(of SOLUTION
  (given-angle (instance-name Angle6))
  (find-angle (instance-name Angle1))
  (relate-angle-pairs [Relate2-6] [Relate1-2])
)
(of SOLUTION
  (given-angle (instance-name Angle1))
  (find-angle (instance-name Angle7))
  (relate-angle-pairs [Relate1-5] [Relate5-7])
)
(of SOLUTION
  (given-angle (instance-name Angle7))
  (find-angle (instance-name Angle1))
  (relate-angle-pairs [Relate5-7] [Relate1-5])
)
(of SOLUTION
```

```
  (given-angle (instance-name Angle1))
  (find-angle (instance-name Angle8))
  (relate-angle-pairs [Relate1-4] [Relate4-8])
)
(of SOLUTION
  (given-angle (instance-name Angle8))
  (find-angle (instance-name Angle1))
  (relate-angle-pairs [Relate4-8] [Relate1-4])                    150
)
(of SOLUTION
  (given-angle (instance-name Angle2))
  (find-angle (instance-name Angle3))
  (relate-angle-pairs [Relate2-3])
)
(of SOLUTION
  (given-angle (instance-name Angle3))
  (find-angle (instance-name Angle2))
  (relate-angle-pairs [Relate2-3])                                160
)
(of SOLUTION
  (given-angle (instance-name Angle2))
  (find-angle (instance-name Angle4))
  (relate-angle-pairs [Relate2-4])
)
(of SOLUTION
  (given-angle (instance-name Angle4))
  (find-angle (instance-name Angle2))
  (relate-angle-pairs [Relate2-4])                                170
)
(of SOLUTION
  (given-angle (instance-name Angle2))
  (find-angle (instance-name Angle5))
  (relate-angle-pairs [Relate2-4] [Relate4-5])
)
(of SOLUTION
  (given-angle (instance-name Angle5))
  (find-angle (instance-name Angle2))
```

```
  (relate-angle-pairs [Relate4-5] [Relate2-4])
)
(of SOLUTION
  (given-angle (instance-name Angle2))
  (find-angle (instance-name Angle6))
  (relate-angle-pairs [Relate2-6])
)
(of SOLUTION
  (given-angle (instance-name Angle6))
  (find-angle (instance-name Angle2))
  (relate-angle-pairs [Relate2-6])
)
(of SOLUTION
  (given-angle (instance-name Angle2))
  (find-angle (instance-name Angle7))
  (relate-angle-pairs [Relate2-6] [Relate6-7])
)
(of SOLUTION
  (given-angle (instance-name Angle7))
  (find-angle (instance-name Angle2))
  (relate-angle-pairs [Relate6-7] [Relate2-6])
)
(of SOLUTION
  (given-angle (instance-name Angle2))
  (find-angle (instance-name Angle8))
  (relate-angle-pairs [Relate2-4] [Relate4-8])
)
(of SOLUTION
  (given-angle (instance-name Angle8))
  (find-angle (instance-name Angle2))
  (relate-angle-pairs [Relate4-8] [Relate2-4])
)
(of SOLUTION
  (given-angle (instance-name Angle3))
  (find-angle (instance-name Angle4))
  (relate-angle-pairs [Relate3-4])
)
```

```
(of SOLUTION
  (given-angle (instance-name Angle4))
  (find-angle (instance-name Angle3))
  (relate-angle-pairs [Relate3-4])
)
(of SOLUTION
  (given-angle (instance-name Angle3))
  (find-angle (instance-name Angle5))
  (relate-angle-pairs [Relate3-4] [Relate4-5])
)
(of SOLUTION
  (given-angle (instance-name Angle5))
  (find-angle (instance-name Angle3))
  (relate-angle-pairs [Relate4-5] [Relate3-4])
)
(of SOLUTION
  (given-angle (instance-name Angle3))
  (find-angle (instance-name Angle6))
  (relate-angle-pairs [Relate3-6])
)
(of SOLUTION
  (given-angle (instance-name Angle6))
  (find-angle (instance-name Angle3))
  (relate-angle-pairs [Relate3-6])
)
(of SOLUTION
  (given-angle (instance-name Angle3))
  (find-angle (instance-name Angle7))
  (relate-angle-pairs [Relate3-7])
)
(of SOLUTION
  (given-angle (instance-name Angle7))
  (find-angle (instance-name Angle3))
  (relate-angle-pairs [Relate3-7])
)
(of SOLUTION
  (given-angle (instance-name Angle3))
```

```
  (find-angle (instance-name Angle8))
  (relate-angle-pairs [Relate3-4] [Relate4-8])
)
(of SOLUTION
  (given-angle (instance-name Angle8))
  (find-angle (instance-name Angle3))
  (relate-angle-pairs [Relate4-8] [Relate3-4])                    260
)
(of SOLUTION
  (given-angle (instance-name Angle4))
  (find-angle (instance-name Angle5))
  (relate-angle-pairs [Relate4-5])
)
(of SOLUTION
  (given-angle (instance-name Angle5))
  (find-angle (instance-name Angle4))
  (relate-angle-pairs [Relate4-5])                                270
)
(of SOLUTION
  (given-angle (instance-name Angle4))
  (find-angle (instance-name Angle6))
  (relate-angle-pairs [Relate4-5] [Relate5-6])
)
(of SOLUTION
  (given-angle (instance-name Angle6))
  (find-angle (instance-name Angle4))
  (relate-angle-pairs [Relate5-6] [Relate4-5])                    280
)
(of SOLUTION
  (given-angle (instance-name Angle4))
  (find-angle (instance-name Angle7))
  (relate-angle-pairs [Relate4-5] [Relate5-7])
)
(of SOLUTION
  (given-angle (instance-name Angle7))
  (find-angle (instance-name Angle4))
  (relate-angle-pairs [Relate5-7] [Relate4-5])                    290
```

```
)
(of SOLUTION
  (given-angle (instance-name Angle4))
  (find-angle (instance-name Angle8))
  (relate-angle-pairs [Relate4-8])
)
(of SOLUTION
  (given-angle (instance-name Angle8))
  (find-angle (instance-name Angle4))
  (relate-angle-pairs [Relate4-8])                                    300
)
(of SOLUTION
  (given-angle (instance-name Angle5))
  (find-angle (instance-name Angle6))
  (relate-angle-pairs [Relate5-6])
)
(of SOLUTION
  (given-angle (instance-name Angle6))
  (find-angle (instance-name Angle5))
  (relate-angle-pairs [Relate5-6])                                    310
)
(of SOLUTION
  (given-angle (instance-name Angle5))
  (find-angle (instance-name Angle7))
  (relate-angle-pairs [Relate5-7])
)
(of SOLUTION
  (given-angle (instance-name Angle7))
  (find-angle (instance-name Angle5))
  (relate-angle-pairs [Relate5-7])                                    320
)
(of SOLUTION
  (given-angle (instance-name Angle5))
  (find-angle (instance-name Angle8))
  (relate-angle-pairs [Relate5-8])
)
(of SOLUTION
```

```
  (given-angle (instance-name Angle8))
  (find-angle (instance-name Angle5))
  (relate-angle-pairs [Relate5-8])                                    330
)
(of SOLUTION
  (given-angle (instance-name Angle6))
  (find-angle (instance-name Angle7))
  (relate-angle-pairs [Relate6-7])
)
(of SOLUTION
  (given-angle (instance-name Angle7))
  (find-angle (instance-name Angle6))
  (relate-angle-pairs [Relate6-7])                                    340
)
(of SOLUTION
  (given-angle (instance-name Angle6))
  (find-angle (instance-name Angle8))
  (relate-angle-pairs [Relate6-8])
)
(of SOLUTION
  (given-angle (instance-name Angle8))
  (find-angle (instance-name Angle6))
  (relate-angle-pairs [Relate6-8])                                    350
)
(of SOLUTION
  (given-angle (instance-name Angle7))
  (find-angle (instance-name Angle8))
  (relate-angle-pairs [Relate7-8])
)
(of SOLUTION
  (given-angle (instance-name Angle8))
  (find-angle (instance-name Angle7))
  (relate-angle-pairs [Relate7-8])                                    360
)
)

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

```
;;                              RULES                              ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule expert-solutions "rule to call the EXPERT print-solutions"
   (declare (auto-focus TRUE))
   ?temp <- (expert-solutions ?given ?find)                              370
  =>
   (retract ?temp)
   (p-s ?given ?find)
)

(defrule get-expert-solutions "rule to get all the EXPERT solutions"
   (declare (auto-focus TRUE))
   ?temp <- (get-expert-solutions ?given ?find)
  =>
   (retract ?temp)                                                       380
   (copy-all-solutions ?given ?find)
)

(defrule expert-get-mag "rule to call the EXPERT get-mag"
   (declare (auto-focus TRUE))
   ?temp <- (expert-gm ?angle)
  =>
   (retract ?temp)
   (bindresult (get-mag ?angle))
)                                                                        390

(defrule expert-define-prob "rule reacts and then focuses STUDENT"
   (declare (auto-focus TRUE))
   (define-problem ?given ?mag ?find)
  =>
   ;;  change all angle mags back to -1
   (do-for-all-instances ((?angle ANGLE))
     (!= -1 (send ?angle get-magnitude))
     (send ?angle put-magnitude -1)
   )                                                                     400
   (dp ?given ?mag ?find)
```

```
   (focus STUDENT)
)

(defrule expert-gma "get all magnitudes in EXPERT module"
   (declare (auto-focus TRUE))
   ?temp <- (expert-gma)
  =>
   (retract ?temp)
   (gma)                                                          410
)

(defrule expert-relate "get relations between two angles"
   (declare (auto-focus TRUE))
   ?temp <- (expert-relate ?ang1 ?ang2)
  =>
   (retract ?temp)
   (if (not
     (do-for-instance ((?rel RELATION))
       (and (member$ (instance-name ?ang1) ?rel:angles)              420
            (member$ (instance-name ?ang2) ?rel:angles))
       (bindresult (send ?rel get-angle-relation)
                   (send ?rel get-magnitude-relation))
     ))
    then (bindresult UNRELATED UNRELATED)
   )
)

(defrule expert-mag-relate "get expert magnitude relation"
   (declare (auto-focus TRUE))                                    430
   ?temp <- (expert-mag-relate ?angle-rel)
  =>
   (retract ?temp)
   (do-for-instance ((?rel RELATION))
     (str-eq ?rel:angle-relation ?angle-rel)
     (bindresult (send ?rel get-magnitude-relation))
   )
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                    STUDENT CONSTRUCTS                        ;;
;; These are constructs which must be defined for STUDENT module ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                       FUNCTIONS                          ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Function to define relationship between two angles,
;;  - a1: the first angle to relate
;;  - a2: the second angle to relate
(deffunction relate (?a1 ?a2 ?ang-rel ?mag-rel)
  (if (do-for-instance ((?inst RELATION))
                      (or (and (str-eq ?a1 (nth$ 1 ?inst:angles))
                               (str-eq ?a2 (nth$ 2 ?inst:angles)))
                          (and (str-eq ?a2 (nth$ 1 ?inst:angles))
                               (str-eq ?a1 (nth$ 2 ?inst:angles))))
        (bind ?reln ?inst)
     )
  then  ;; this relation already exists, so just modify it
    ;;(send (instance-name ?reln) put-angle-relation ?ang-rel)
    ;;(send (instance-name ?reln) put-magnitude-relation ?mag-rel)
    (modify-instance (instance-name ?reln)
      (angle-relation ?ang-rel)
      (magnitude-relation ?mag-rel)
    )
  else  ;; this relation DNE, so create it
    (bind ?reln (make-instance of RELATION
                  (angle-relation ?ang-rel)
                  (magnitude-relation ?mag-rel)
                  (angles (instance-name ?a1) (instance-name ?a2))
                )
    )
  ) ;; end if
```

```
  (if (do-for-instance ((?inst SOLUTION))
        (and (str-eq ?*find-angle* ?inst:find-angle)
             (str-eq ?*given-angle* ?inst:given-angle)
        )
        (bind ?soln ?inst)                                          40
      )
   then  ;; modify existing SOLUTION, else no problem defined
     (if (not (member$ ?reln (send ?soln get-relate-angle-pairs)))
       then  ;;  this exact relation DNE in SOLUTION, so add it
         (bind $?old-angles (send ?soln get-relate-angle-pairs))
         (send ?soln put-relate-angle-pairs (append$ ?old-angles ?reln))
     )
  )
  (run)
)                                                                   50

;;  shortcut for relate
;;  - nbr1: the angle number of first angle to relate
;;  - nbr2: the angle number of second angle to relate
(deffunction r (?nbr1 ?nbr2 ?ang-rel ?mag-rel)
  (relate (sym-cat "Angle" ?nbr1)
          (sym-cat "Angle" ?nbr2)
          ?ang-rel ?mag-rel)
)
                                                                    60
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                          RULES                                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defrule student-solutions "rule to call the STUDENT print-solutions"
   (declare (auto-focus TRUE))
   ?temp <- (student-solutions ?given ?find)
  =>
   (retract ?temp)
   (p-s ?given ?find)                                               70
)
```

```
(defrule get-student-solution "rule to get the STUDENT solution"
   (declare (auto-focus TRUE))
   ?temp <- (get-student-solution ?given ?find)
  =>
   (retract ?temp)
   (copy-all-solutions ?given ?find)
)

(defrule student-relate-angles "rule to call STUDENT relate"
   (declare (auto-focus TRUE))
   ?temp <- (relate ?a1 ?a2 ?ang-rel ?mag-rel)
  =>
   (retract ?temp)
   (r ?a1 ?a2 ?ang-rel ?mag-rel)
)

(defrule student-get-mag "rule to call the STUDENT get-mag"
   (declare (auto-focus TRUE))
   ?temp <- (student-gm ?angle)
  =>
   (retract ?temp)
   (bindresult (get-mag ?angle))
)

(defrule student-define-prob "rule reacts to focus by EXPERT"
   ?temp <- (define-problem ?given ?mag ?find)
  =>
   (retract ?temp)
   ;;  change all angle mags back to -1
   (do-for-all-instances ((?angle ANGLE))
     (!= -1 (send ?angle get-magnitude))
     (send ?angle put-magnitude -1)
   )
   (dp ?given ?mag ?find)
   (bind ?given-angle (sym-cat Angle ?given))
   (bind ?find-angle (sym-cat Angle ?find))
   (if (not (do-for-instance
```

```
            ((?soln SOLUTION))                                          110
            (and (str-eq ?given-angle ?soln:given-angle)
                 (str-eq ?find-angle ?soln:find-angle))
            (send ?soln put-relate-angle-pairs (create$)))
      )
    then
      (make-instance of SOLUTION
        (given-angle (instance-name ?given-angle))
        (find-angle (instance-name ?find-angle))
      )
   )                                                                    120
)

(defrule student-gma "get all magnitudes in STUDENt module"
   (declare (auto-focus TRUE))
   ?temp <- (student-gma)
  =>
   (retract ?temp)
   (gma)
)
                                                                        130
```

```
#procedure to force a variable to an integer angle between 0 and 180
proc tcl_forceAng {name element op} {
  upvar $name x ${name}_old x_old
  if { ($x < 0) || ($x > 180) } {
    tcl_IEOmsg "Invalid angle magnitude! Must be between 0 and \
            180 degrees.  Changing $x to previous value of $x_old."
    set x $x_old
    }
  set x_old $x
  } ;# end of procedure tcl_forceAng

# procedure to display a message for the instructor to read
proc tcl_IEOmsg {msg} {
  set index 0
  while {[winfo exists .instructor$index]} { incr index }
  toplevel .instructor$index
  wm title .instructor$index "Instructor"
  frame .instructor$index.f -relief raised -borderwidth 4
  pack .instructor$index.f
  message .instructor$index.m -width 6i -justify left -text $msg \
        -relief sunken -borderwidth 4
  pack .instructor$index.m -in .instructor$index.f -fill both
  button .instructor$index.b -text "Go away" -command "destroy \
       .instructor$index"
  pack .instructor$index.b -in .instructor$index.f
  }

# procedure to display a message for the student to read
proc tcl_student_msg {msg} {
  global curtain_remote curtain_local curtain_remote_display
  global curtain_student
  set mindex 0
  set sindex 0
  while {[winfo exists .message$mindex]} { incr mindex }
  while {[winfo exists .student$sindex]} { incr sindex }
```

```
  toplevel .message$mindex -screen $curtain_remote_display
  toplevel .student$sindex
  wm title .message$mindex $curtain_student
  wm title .student$sindex "Student"
  frame .student$sindex.f -relief raised -borderwidth 4                    40
  pack .student$sindex.f
  frame .message$mindex.f -relief raised -borderwidth 4
  pack .message$mindex.f
  message .message$mindex.m -width 6i -justify left -text $msg \
    -relief sunken
  message .student$sindex.m -width 6i -justify left -text $msg \
    -relief sunken
  pack .message$mindex.m -in .message$mindex.f
  pack .student$sindex.m -in .student$sindex.f
  button .message$mindex.b -text "Go away" -command "destroy \            50
        .message$mindex"
  button .student$sindex.b -text "Go away" -command "destroy \
        .student$sindex"
  pack .message$mindex.b -in .message$mindex.f
  pack .student$sindex.b -in .student$sindex.f
  }

proc tcl_type_message {} {
  toplevel .typemsg
  wm title .typemsg "Enter Message"                                        60
  wm iconname .typemsg "Message"
  frame .typemsg.top
  frame .typemsg.bottom
  label .typemsg.l -text \
    "Please enter the message for the student: "
  entry .typemsg.e -width 100 -relief sunken \
                -textvariable curtain_stu_msg
  .typemsg.e icursor 0  ;# position insertion cursor at beginning
  .typemsg.e selection range 0 end  ;# highlight text for easy deletion
  pack .typemsg.top -side top -padx 3m -pady 1m                            70
  pack .typemsg.bottom -side bottom -padx 3m -pady 1m
  pack .typemsg.l .typemsg.e -side top -in .typemsg.top
```

```
  button .typemsg.ok -text OK -command {
    tcl_student_msg $curtain_stu_msg
    destroy .typemsg
    }
  button .typemsg.c  -text Cancel -command {destroy .typemsg}
  pack .typemsg.ok .typemsg.c -side left -in .typemsg.bottom
  bind .typemsg.e <Return> {focus .typemsg.ok}
  focus .typemsg.e                                                    80
  }

# procedure to build a prompt to select an angle
proc tcl_create_angle_prompt {name title label} {
  global curtain_angle
  toplevel .$name
  wm title .$name $title
  frame .$name.top
  frame .$name.bottom
  pack .$name.top -side top -padx 3m -pady 1m                         90
  pack .$name.bottom -side bottom -padx 3m -pady 1m
  label .$name.l -text $label
  pack .$name.l -in .$name.top
  for {set i 1} {$i < 9} {incr i} {
    radiobutton .$name.b$i -text "Angle $i" -variable curtain_angle \
              -value $i
    pack .$name.b$i -side left -in .$name.top
    } ;# end of for loop
  button .$name.ok -text OK -command " destroy .$name "
  button .$name.can -text Cancel -command "                           100
    set curtain_angle \"\"
    destroy .$name
    "
  pack .$name.ok .$name.can -side left -in .$name.bottom
  focus .$name.ok
  tkwait window .$name
  return $curtain_angle
  } ;# end of procedure tcl_create_angle_prompt
```

```
# procedure to prompt for solution, and call CLIPS-solution                    110
proc tcl_solution {} {
  global curtain_find curtain_mag
  trace variable mag w tcl_forceAng
  if { $curtain_find != 0 } {
    # create prompt for student solution
    set mag $curtain_mag
    toplevel .get_magnitude
    wm title .get_magnitude "Get Magnitude"
    label .get_magnitude.l -text \
      "Please enter the magnitude of Angle$curtain_find:"                      120
    entry .get_magnitude.e -relief sunken -textvariable curtain_mag
    .get_magnitude.e icursor 0  ;# pos insertion cursor at beginning
    .get_magnitude.e selection range 0 end  ;# highlight text
    pack .get_magnitude.l .get_magnitude.e -side left
    bind .get_magnitude.e <Return> { destroy .get_magnitude }
    focus .get_magnitude.e
    tkwait window .get_magnitude
    set mag $curtain_mag
    set curtain_mag $mag ;# mag is traced, so set c_mag to valid mag
    if { $mag != "" } {                                                        130
      puts "CLIPS-solution $mag"
      flush stdout
      }
    } else {
    tcl_IEOmsg "No problem to solve!  Define a problem first."
    }  ;# end of if curtain_find
  }  ;# end of procedure tcl_solution

# procedure to actually make the CLIPS-get-mag call
proc tcl_get_mag {} {                                                          140
  set angle [tcl_create_angle_prompt get_magnitude "Get Magnitude"\
            "Please select the angle whose magnitude you require:"]
  if { $angle != "" } {
    puts "CLIPS-get-mag $angle"
    flush stdout
    }
```

```
  } ;# end of procedure tcl_get_mag

# procedure to prompt for the definition of a relation
proc tcl_get_rel_defn {relate1 relate2} {                                    150
  global curtain_defn
  toplevel .tgrd
  wm title .tgrd "Define Relations"
  frame .tgrd.top
  frame .tgrd.bottom
  pack .tgrd.top -side top -padx 3m -pady 1m
  pack .tgrd.bottom -side bottom -padx 3m -pady 1m
  label .tgrd.l -text "Angle$relate1 and Angle$relate2 are:"
  pack .tgrd.l -in .tgrd.top
  foreach i {congruent supplementary} {                                      160
    radiobutton .tgrd.b$i -text $i -variable curtain_defn -value $i
    pack .tgrd.b$i -side left -in .tgrd.top
    } ;# end of for loop
  button .tgrd.prompt -text "Prompt student" -command "
    tcl_student_msg \"Are Angle$relate1 and Angle$relate2 congruent \
                    or supplementary?\" "
  button .tgrd.ok -text OK -command " destroy .tgrd "
  button .tgrd.can -text Cancel -command " set curtain_defn \"\"
                                          destroy .tgrd "
  pack .tgrd.ok .tgrd.can .tgrd.prompt -side left -in .tgrd.bottom           170
  tkwait window .tgrd
  return $curtain_defn
  } ;# end of procedure tcl_get_rel_defn

# procedure to build a prompt to select a relation
proc tcl_get_relation {relate1 relate2} {
  global curtain_relation
  toplevel .tgr
  wm title .tgr "Select Relation"
  frame .tgr.top                                                             180
  frame .tgr.bottom
  pack .tgr.top -side top -padx 3m -pady 1m
  pack .tgr.bottom -side bottom -padx 3m -pady 1m
```

```
  label .tgr.l -text "Which defines the relation between \
     Angle$relate1 and Angle$relate2?"
  pack .tgr.l -in .tgr.top
  foreach i {vertical adjacent alternate-interior corresponding} {
    radiobutton .tgr.b$i -text $i -variable \
              curtain_relation -value $i
    pack .tgr.b$i -side left -in .tgr.top                                    190
    } ;# end of for loop
  button .tgr.prompt -text "Prompt student" -command "
    tcl_student_msg \"Define the relationship between Angle$relate1\
                  and Angle$relate2. Are they vertical, adjacent,\
                  alternate-interior, or corresponding angles?\"
    "
  button .tgr.ok -text OK -command " destroy .tgr "
  button .tgr.can -text Cancel -command "
                                  set curtain_relation \"\"
                                  set curtain_defn \"\"                      200
                                  destroy .tgr "
  pack .tgr.ok .tgr.can .tgr.prompt -side left -in .tgr.bottom
  tkwait window .tgr
  return $curtain_relation
  } ;# end of procedure tcl_get_relation

# procedure to get the angles to relate and call the CLIPS-relate
proc tcl_relate {} {
  # get the angles to relate
  set relate1 [tcl_create_angle_prompt relate_angles "Relate Angles"\      210
            "Please select the FIRST angle to relate:"]
  if { $relate1 != "" } {
    set relate2 [tcl_create_angle_prompt relate_angles "Relate Angles"\
              "Please select the SECOND angle to relate:"]
    # define the relations between the selected angles
    if { $relate2 != "" } {
      set relation [tcl_get_relation $relate1 $relate2]
      # call CLIPS-relate function if parms present
      if { $relation != "" } {
        set relation_defn [tcl_get_rel_defn $relate1 $relate2]               220
```

```
      if { $relation_defn != "" } {
        puts "CLIPS-relate $relate1 $relate2 $relation $relation_defn"
        flush stdout
        } ;# end of if relation_defn
      } ;# end of if relation
    } ;# end of if relate2
  } ;# end of if relate1
} ;# end of procedure tcl_relate

# procedure to call the CLIPS-get-definition                              230
proc tcl_get_definition {} {
  #get the term to define
  toplevel .get_term
  wm title .get_term "Which term?"
  frame .get_term.fb
  pack .get_term.fb -side bottom
  label .get_term.l -text \
    "Please select the term to define:"
  pack .get_term.l
  foreach t {supplementary congruent vertical adjacent                   240
             alternate-interior corresponding} {
    button .get_term.b$t -text $t -command "
      puts \"CLIPS-get-definition $t\"
      destroy .get_term
      "
    pack .get_term.b$t -side left
    } ;# end of for loop
  button .get_term.bc -text Cancel -command { destroy .get_term }
  pack .get_term.bc -in .get_term.fb
} ;# end of procedure tcl_get_definition                                  250

# procedure to call the CLIPS-procedure define-problem after querying
#  instructor for the parms
proc tcl_define_problem {} {
  global curtain_mag curtain_find
  trace variable mag w tcl_forceAng
  # create prompt for given angle
```

```
set given [tcl_create_angle_prompt given_angle "Given Angle"\
        "Please select the KNOWN angle for the problem:"]
if { $given != "" } {                                                    260
  # create prompt for given magnitude
  set mag $curtain_mag
  toplevel .get_magnitude
  wm title .get_magnitude "Get Magnitude"
  label .get_magnitude.l -text \
    "Please enter the magnitude of Angle$given:"
  entry .get_magnitude.e -relief sunken -textvariable curtain_mag
  .get_magnitude.e icursor 0  ;# pos insertion cursor at beginning
  .get_magnitude.e selection range 0 end  ;# highlight text
  pack .get_magnitude.l .get_magnitude.e -side left                      270
  bind .get_magnitude.e <Return> { destroy .get_magnitude }
  focus .get_magnitude.e
  tkwait window .get_magnitude
  set mag $curtain_mag
  set curtain_mag $mag ;# mag is traced, so set c_mag to valid mag
  if { $mag != "" } {
    # create prompt for find angle
    set find [tcl_create_angle_prompt find_angle "Find Angle"\
            "Please select the angle whose magnitude must be \
             found:"]                                                    280
    if { $find != "" } {
      # actually call the CLIPS-function after sending message
      # also bind global variable for use in tcl_solution
      set curtain_find $find
      set msg "Given: magnitude of Angle $given = \
            $mag, Find:  magnitude of Angle $find."
      tcl_student_msg $msg
      puts "CLIPS-define-problem $given $mag $find"
      flush stdout
      }  ;# end of if find                                               290
    }  ;# end of if mag
  }  ;# end of if given
}  ;# end of procedure tcl_define_problem
```

```
# procedure to display instructions to the student
proc tcl_display_instr {} {
  global curtain_remote curtain_local curtain_remote_display
  global curtain_path
  toplevel .stu_instr -screen $curtain_remote_display
  toplevel .instr_instr                                              300
  foreach tl {stu_instr instr_instr} {
    wm title .$tl "General Instructions"
    wm iconname .$tl "Instructions"
    wm iconbitmap .$tl @${curtain_path}/images/parallel.icon
    wm geom .$tl +0+615
    wm minsize .$tl 680 20

    frame .$tl.bot
    pack .$tl.bot -side bottom -fill both
                                                                     310
    message .$tl.m -width 705 -justify left -text "You will be \
      presented with some geometry problems related to the parallel \
      lines figure above.  Please answer the questions loudly and \
      clearly, so the microphone can pick up your responses.  The \
      tutoring system will prompt you with additional instructions \
      or requests.  Good luck!"
    pack .$tl.m -expand yes -fill both -padx 2m -pady 2m
    }
    button .stu_instr.bot.quit -text "Go Away" -command {destroy \
          .stu_instr}                                                320
    pack .stu_instr.bot.quit
    button .instr_instr.bot.quit -text "Go Away" -command {
      destroy .instr_instr
      }
    pack .instr_instr.bot.quit
  }

# procedure to display the current student model
proc tcl_display_model {} {
  global curtain_list curtain_student curtain_path                   330
```

```
set index 0
while {[winfo exists .student_model_$index]} { incr index }
set tl student_model_$index
toplevel .$tℓ
wm title .$tℓ "Model of $curtain_student"
wm iconname .$tℓ $curtain_student
wm geom .$tℓ +0+0
wm minsize .$tℓ 40 60

frame .$tℓ.bot
pack .$tℓ.bot -side bottom -fill both

set maxx 600
set maxy 300
set minx 10
set miny 10
set basex 130
set basey 50
set deltax 20
set deltay 50
set concepts [array names curtain_list]
image create bitmap .$tℓ.gℓad \
  -file ${curtain_path}/images/glad.bmp \
  -maskfile ${curtain_path}/images/face.msk \
  -background yellow -foreground black
image create bitmap .$tℓ.sad \
  -file ${curtain_path}/images/sad.bmp \
  -maskfile ${curtain_path}/images/face.msk \
  -background red -foreground black
canvas .$tℓ.c -width $maxx -height $maxy -bd 2 -relief sunken
.$tℓ.c create text [expr ($maxx - $minx)/2] $miny \
  -text "Concept hits and misses"
foreach concept $concepts {
  puts "CLIPS-get-student-concept $concept"
  flush stdout
  #set curtain_list($concept) {-1 1 1 -1 -1 -1 1 -1 -1 -1 -1 -1 -1}
  tkwait variable curtain_list($concept)
```

```
    .$tℓ.c create text $minx $basey -text $concept -anchor sw
    set xleft $basex                                                    370
    foreach mod $curtain_list($concept) {
      if {$mod > 0} { set face sad} else {set face glad}
      .$tℓ.c create image $xℓeft $basey -image .$tℓ.$face -anchor sw
      incr xleft $deℓtax
      }
    #set basey [expr $basey + $deℓtay]
    incr basey $deℓtay
    }
  pack .$tℓ.c -expand yes -fill both -padx 2m -pady 2m
  button .$tℓ.bot.quit -text "Go Away" -command "destroy .$tℓ"          380
  pack .$tℓ.bot.quit

  }

# procedure to update the mastery level required for the student
proc tcl_change_threshold {} {
  global curtain_threshold

  # create prompt for student solution
  toplevel .get_threshold                                               390
  wm title .get_threshold "Get Threshold"
  label .get_threshold.l -text \
    "Please enter the number of correct responses required for mastery:"
  entry .get_threshold.e -relief sunken -textvariable curtain_threshold
  .get_threshold.e icursor 0  ;# pos insertion cursor at beginning
  .get_threshold.e selection range 0 end  ;# highlight text
  pack .get_threshold.l .get_threshold.e -side left
  bind .get_threshold.e <Return> { destroy .get_threshold }
  focus .get_threshold.e
  tkwait window .get_threshold                                          400
  set threshold $curtain_threshold
  # threshold is traced, so set curtain_threshold to valid threshold
  set curtain_threshold $threshoℓd
  if { $threshoℓd != "" } {
    puts "CLIPS-change-threshold $threshoℓd"
```

```
    flush stdout
    }
  }

# procedure to display a GIF figure for the student                    410
proc tcl_display_pic {} {
  global curtain_remote curtain_local curtain_remote_display
  global curtain_path
  toplevel .stu_fig -screen $curtain_remote_display
  toplevel .instr_fig
  foreach tl {stu_fig instr_fig} {
    wm title .$tl "Parellel Lines Diagram"
    wm iconname .$tl "Parallel"
    wm iconbitmap .$tl @${curtain_path}/images/parallel.icon
    wm geom .$tl +0+0                                                  420
    wm minsize .$tl 680 520

    frame .$tl.bot
    pack .$tl.bot -side bottom -fill both

    image create photo .$tl.cimage -format gif -file \
      ${curtain_path}/images/parallel.gif
    canvas .$tl.c -width 691 -height 531 -bd 2 -relief sunken
    .$tl.c create image 10 10 -image .$tl.cimage -anchor nw
    pack .$tl.c -expand yes -fill both -padx 2m -pady 2m               430
    }
    button .stu_fig.bot.quit -text "Go Away" -command {destroy .stu_fig}
    pack .stu_fig.bot.quit
    button .instr_fig.bot.quit -text "Go Away" -command {
      destroy .instr_fig
      }
    pack .instr_fig.bot.quit
  }

# procedure to display main selection menu for the instructor          440
proc tcl_menu {} {
  global curtain_local curtain_local_display
```

```
global curtain_remote curtain_remote_display
global curtain_student

exec rsh -n $curtain_remote "setenv DISPLAY :0.0; \
          xhost +$curtain_local"
tcl_display_pic
tcl_display_instr
                                                                    450
# Send C function call to init CLIPS to stdout
# (stdout is a pipe to Curtain)
puts "C_InitClips $curtain_student"
flush stdout
destroy .ok .c .e
.l configure -text "Select an action:"
pack .l -in .bottom
menubutton .mbc -menu .mbc.m -text Commands
menubutton .mbs -menu .mbs.m -text System
menu .mbc.m                                                         460
menu .mbs.m
.mbc.m add command -label "Display current student model" \
  -command tcl_display_model
.mbc.m add command -label "Change student mastery threshold" \
  -command tcl_change_threshold
.mbs.m add command -label "Save and quit" -command {
    puts "CLIPS-quit"
  flush stdout
  exit
  }                                                                 470
.mbs.m add command -label Quit -command exit
pack .mbs .mbc -in .top -side left
button .gd -text "Display a definition to the student" \
  -command {tcl_get_definition}
button .dp -text "Define a geometry problem" \
  -command {tcl_define_problem}
button .ra -text "Relate two angles" -command tcl_relate
button .ss -text "Suggest a solution" -command tcl_solution
button .msg -text "Type a message to the student" -command \
```

```
            tcl_type_message                                          480
  pack .gd .dp .ra .ss .msg -in .bottom -fill x
  wm geom . -0+25
  }

# procedure to display prompt for student's machine
proc tcl_get_remote {} {
global curtain_remote
global env
set curtain_remote $env(HOST)
wm geom . -0+28                                                       490
.l configure -text \
  "Please enter the machine the STUDENT will be working on: "
.e configure -relief sunken -textvariable curtain_remote
.e icursor 0  ;# position insertion cursor at beginning of entry
.e selection range 0 end  ;# highlight text for easy deletion/mod
pack .top -side top -padx 3m -pady 1m -fill x
pack .bottom -side bottom -padx 3m -pady 1m
pack .l .e -side top -in .top
.ok configure -text OK -command {
  set curtain_local $env(HOST)                                        500
  set curtain_local_display $env(DISPLAY)
  if {$curtain_remote == ""} { set curtain_remote $curtain_local }
  set curtain_remote_display ${curtain_remote}:0
  tcl_menu
  }
pack .ok .c -side left -in .bottom
bind .e <Return> {focus .ok}
focus .e
}
                                                                      510
# main portion of Tcl/Tk; ask for remote machine and call tcl_menu
set curtain_mag 0
set curtain_find 0
set curtain_threshold 1
array set curtain_list {
  congruent {0}
```

```
  supplementary {0}
  vertical {0}
  corresponding {0}
  alternate-interior {0}                                          520
  adjacent {0}
  }
set templist [array names env]
if [expr -1 < [lsearch -exact $templist CURTAIN_PATH]] {
  set curtain_path $env(CURTAIN_PATH)/TCL
  } else {
  if [expr -1 < [lsearch -exact $templist PWD]] {
    set curtain_path $env(PWD)/TCL
    } else {exit}
  }                                                               530

wm title . "Curtain ITS"
wm iconname . "Curtain"
wm geom . -10+0
wm minsize . 1 1
frame .top -relief sunken -borderwidth 4
frame .bottom -relief sunken -borderwidth 6
label .l -text \
  "Please enter a unique identifier for the STUDENT: "
entry .e -relief sunken -textvariable curtain_student             540
pack .top -side top -padx 3m -pady 1m
pack .bottom -side bottom -padx 3m -pady 1m
pack .l .e -side top -in .top
button .ok -text OK -command {
  if {$curtain_student != ""} tcl_get_remote
  }
button .c  -text Cancel -command exit
pack .ok .c -side left -in .bottom
bind .e <Return> {focus .ok}
focus .e                                                          550
```

Bibliography

1. Anderson, John R., et al. "Intelligent Tutoring Systems," *Science*, *228*:456–462 (1985).
2. Beller, Sieghard and H. Ulrich Hoppe. "Deductive Error Reconstruction and Classification in a Logic Programming Framework." *Artificial Intelligence in Education, 1993*, edited by Paul Brna, et al. 433 – 440. Association for the Advancement of Computing in Education (AACE), aug 1993.
3. Burton, Richard R. and John Seely Brown. "An investigation of computer coaching for informal learning activities." *Intelligent Tutoring Systems*, Computers and People, edited by D. Sleeman and J. S. Brown. 79–98. Harcourt Brace Jovanich, 1982.
4. Carbonaro, A., et al. "Modelling the student in Pitagora 2.0," *User Modeling and User-Adapted Interaction*, *4*(4):233–251 (1995).
5. de Barros Costa, Evandro, et al. "Mathema: A Learning Environment Based on a Multi-Agent Architecture." *Advances in Artificial Intelligence: Proceedings of the* 12^{th} *Brazilian Symposium on Aritificial Intelligence*, edited by Jacques Wainer, et al. Springer-Verlag Berlin Heidelberg, October 1995.
6. Dean, Thomas, et al. *Artificial Intelligence Theory and Practice.* The Benjamin/Cummings Publishing Company, Inc., 1995.
7. Giangrandi, P. and C. Tasso. "Truth maintenance techniques for modelling student's behaviour," *Journal of Artificial Intelligence in Education*, *6*(2-3):153–202 (1995).
8. Hartley, J.R. and D.H. Sleeman. "Towards More Intelligent Teaching Systems," *International Journal of Man-Machine Studies*, *2*:215 – 236 (1973).
9. Holt, Peter, et al. "The State of Student Modelling." *Student Modelling: The Key to Individualized Knowledge-Based Instruction125*. NATO Special Programme on Advanced Educational Technology, edited by Jim E. Greer and Gordon I. McCalla. 3–35. Springer-Verlag, 1994.
10. Huang, X., et al. "Revising deductive knowledge and stereotypical knowledge in a student model," *User Modeling and User-Adapted Interaction*, *1*:87–115 (1991).
11. Jr., Freeman A. Kilpatrick. *A Generic Intelligent Architecture for Computer-Aided Training of Procedural Knowledge*. PhD dissertation, Air Force Institute of Technology, 1996.
12. Kabrisky, Matthew. "The Promise of an Artificial Intelligence Future." From IEEE meeting in Dayton, OH, 1983.
13. Kambouri, M., et al. "Knowledge Assessment: tapping human expertise by the QUERY routine," *Int J. Humane Computer Studies*, *40*:119–151 (1994).
14. Leman, Stephane, et al. "A Multi-Agent Approach to Model Student Reasoning Process." *Proceedings of AI-ED 95 - 7th World Conference on Artificial Intelligence in Education*, edited by Jim Greer. 258–265. Charlottesville, VA, USA: Association for the Advancement of Computing in Education, August 1995.

15. Marcenac, Pierre. "An authoring system for ITS which is based on a generic level of tutoring strategies." 4^{th} *International Conference on Computers and Learning*. 1992.

16. McCormack, J. S. and J. E. Biegel. "A student model to acquire problem-solving strategies." *Proceedings of the Eighth Florida Artificial Intelligence Research Symposium. FLAIRS-95*, edited by J. H. Stewman. 16–20. St. Petersburg, FL , USA: Florida AI Res. Soc, April 1995.

17. Nour, M., et al. "A proposed student model algorithm for an intelligent tutoring system." *SICE '95. Proceedings of the* 34^{th} *SICE Annual Conference..* 1327–1333. Soc. Instrum. & Control Eng, 1995.

18. Ragnemalm, Eva L. "Student diagnosis in practice; bridging a gap," *User Modeling and User-Adapted Interaction*, *5*(2):93–116 (1995).

19. Shute, Valerie J. "SMART: Student modeling approach for responsive tutoring," *User Modeling and User-Adapted Interaction*, *5*(1):1–44 (1995).

20. Shute, Valerie J. and Josepf Psotka. *Intelligent Tutoring Systems: Past, Present, Future*. Technical Report AL/HR-TP-1994-0005, USAF, Armstrong Laboratory, 1994.

21. Villano, M. "Probabilistic Student Models: Bayesian belief networks and knowledge space theory." *Second International Conference Intelligent Tutoring Systems*. 491–498. 1992.

22. Winston, Patrick Henry. *Artificial Intelligence*. Addison-Wesley Publishing Company, 1992.

23. Yazdani, Masoud. "Intelligent Tutoring Systems: An Overview." *Artificial Intelligence and Education1*, edited by Robert W. Lawler and Masoud Yazdani. 183 – 201. Ablex Publishing Corporation, 1987.

24. Ziegler, U. M. "Use of a neural network to diagnose student errors in an intelligent tutoring system." *World Congress on Neural Netwroks-San Diego4*. 459–464. Hillsdale, NJ, USA: Lawrence Erlbaum Associates, June 1994.

Vita

Captain Jeremy E. Thompson [redacted] graduated from Lavina High School, Lavina, Montana, in May 1984. On June 18, 1986, Jeremy enlisted in the US Air Force. Upon graduation from Basic Military Training School, Jeremy was sent to Lowry AFB, Colorado, where he completed Logistics Management training and was stationed for his first assignment as an Inventory Management Specialist at RAF Greenham Common, United Kingdom. While stationed at RAF Greenham Common, Jeremy was accepted into the Airman's Education and Commissioning Program to complete his Bachelor of Science in Electrical Engineering at the University of Missouri-Rolla (UMR). Graduation from UMR led Jeremy to Officer Training School. Jeremy became Second Lieutenant Thompson on September 25, 1991 and moved to Keesler AFB to complete Basic Communication-Computer Officer Training (BCOT).

Upon graduation from BCOT in March, 1992, Lt Thompson was assigned to USSTRATCOM at Offutt AFB, Nebraska. His duties there included the hosting and testing of an advanced radar model developed by scientists at the Massachusetts Institute of Technology Lincoln Laboratory. In May 1995, Lt Thompson entered the Air Force Institute of Technology (AFIT) at Wright-Patterson AFB, Ohio, to pursue a Master of Science degree with a concentration in Artificial Intelligence. Upon graduation from AFIT in December 1996, Captain Thompson was re-assigned to Air University, Maxwell AFB, Alabama.

[redacted]

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1996	Master's Thesis

4. TITLE AND SUBTITLE
STUDENT MODELING IN AN INTELLIGENT TUTORING SYSTEM

5. FUNDING NUMBERS

6. AUTHOR(S)
Capt Jeremy E. Thompson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology
2950 P Street
WPAFB OH 45433-6583

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AL/HRTI
7907 Lindbergh Drive
Brooks AFB TX 78235

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This thesis explores a new approach to modeling the student in an intelligent tutoring system (ITS), by providing a student model which learns new solutions from the student. A prototype of the new approach is demonstrated in the Euclidean geometry domain. Complete C++, CLIPS, and Tcl/Tk code listings are included in the appendices. Adaptable multiple software agents were targeted for implementation, based on current literature. However, the student model is found to be maintainable *without* multiple software agents, while still allowing for tracking several possible solution paths when monitoring student solutions, which contradicts previously reported research. The student model provides a learning module capable of recognizing new solutions provided by the student. These new solutions may then be included in the expert knowledge base. In addition to a learning student model, other concepts from the current ITS literature are explored and implemented. Mastery levels are implemented to aid in cognitive diagnosis. Symbolic knowledge, procedural skill, and conceptual knowledge, are explored and applied to the research. The student model prototype is both a pedagogic-content model and a subject-matter model. Additionally, a new division of labor between the student model and the instructor module in ITSs is described.

14. SUBJECT TERMS	15. NUMBER OF PAGES
student modeling, intelligent tutoring systems, artificial intelligence	112
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL