4-7-1986

Computer assisted instruction for students studying basic logic at the 9th grade level

Edith Shortt

Rochester Institute of Technology
School of Computer Science and Technology

Computer Assisted Instruction for Students Studying Basic Logic at the 9th Grade Level

by
Edith S. Shortt

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fullfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by: John A. Biles
Professor John A. Biles

Lawrence A. Coon
Dr. Lawrence A. Coon

James Robert Carbin
Professor James R. Carbin

April, 1986

Title of Thesis Computer Assisted Instruction for Students Studying Basic Logic at the 9th Grade Level

I Edith S. Shortt hereby grant permission to the Wallace Memorial Library, of R.I.T., to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date April 7, 1986

Table of Contents. Pages.

Abstract.

A system of Computer Assisted Instruction for Students Studying Basic Logic at the 9th Grade Level was written in Prolog running on a Digital Equipment Vax 11/780 under the Unix1 operating system.

The program is a straight-forward approach to teaching propositional logic. A three layer menu structure is used to inform students of lessons, subtopics, and help available. Example questions are generated after each subtopic, and student scores are displayed for immediate reinforcement or correction. Scores are also saved for storage on the student's progress chart file.

A help function enables the student to review explanations of terminolgy, review truth tables, print a student handbook, and either display or print the student's progress chart. In addition, the teacher may open a student account, close a student account, print a classlist, print progress charts, or print the teacher's handbook.

The input/output is not up to par due to the limitations imposed by the Prolog C-Interpreter. Pretests and Posttests are only available interactively with the aid of the teacher.

Chapter 1. Introduction.

1.1. A History of CAI.

The American Educational System is experiencing another technological revolution. The first "classroom technological" revolution was launched soon after the U.S.S.R. launched Sputnik, with the advent of programmed instruction. However, use of programmed instruction declined in the late sixties due to poor software design [Bell, p. 36].

The current technological revolution was prompted by Japanese "high-tech" achievements and by the introduction of microcomputers into classrooms. In an effort to meet market demand, publishers released much software before it was properly tried and tested [Bell, p. 36], and the lack of quality instructional computing materials was identified as the major impediment to the use of computer aided instruction [Spreecher, p. 47].

The introduction of CAI into school systems has been hampered by many factors since its inception: good software was difficult to design; teachers and administrators lacked expertise in the use of computers for education; hardware and software compatibility was limited; and hardware and software used in classrooms for CAI was expensive [Pressman, p. 183].

Until recently, many school boards were unwilling to back

CAI in their schools. New York state, in its Regents Action Plan, has recommended that a kindergarten through twelfth grade curriculum for the use of computers be developed by school districts.

Microcomputers have proven to be adequate for most classroom computing needs. Today, microcomputers are not only less expensive (compared to hardware costs during the 1950's and 1960's) but have greater capabilities. Microcomputers today are faster, have more memory, allow graphics with high resolution monochromatic or color monitors, support letter quality printers, graphics printers and plotters, are capable of supporting a variety of languages and operating systems, communicate with other computers, and handle files [Pressman, p. 187].

Recent research has shown the effectiveness of using computers over conventional classroom instruction. CAI has been found particularly effective when used with high-achieving or low-achieving students, when used with science, mathematics and foreign language curriculum areas, when used as a supplement to regular classroom instruction, and when used to change student behavior and attitudes for the better [Kontos, p. 4]. "Evidence has lent support to the contention that superior and more rapid learning is associated with more personalized, self-paced, and self-directed computerized

instruction, especially when training adults [Kamouri, p. 291]."

CAI was once considered a threat to teachers, however, today educators are beginning to realize the many capabilities of CAI. Students using CAI programs are actively involved, proceed at their own pace, and have instant feedback for their efforts. Teachers are looking to computers to support their curriculums. Students in today's schools are learning how to do word processing, how to use spreadsheets, how to do CAD (Computer Aided Design), how to program in several languages (LOGO, BASIC, and Pascal), and how to use other CAI software.

1.2. Types of CAI.

A major factor in the development of CAI has been the availability of computing equipment for use in education. Leased time-sharing terminals once popular in schools have given way to school owned microcomputer labs, and educators are demanding that CAI software be developed to run on the latest available computing equipment. The following discussion of CAI approaches reflects educational use trends rather than a strict chronological order of CAI software development.

O'Shea and Self have identified eleven approaches to using the computer in education: 1. linear programs; 2. branching programs; 3. generative computer-assisted learning; 4. mathematical models of learning; 5. TICCIT; 6. PLATO; 7. simulation; 8. games; 9. problem-solving; 10. emancipatory modes; and 11. dialogue systems [O'Shea, pp. 68-121]. A brief description of each of these eleven approaches of using computers in education follows, because it is important that anyone interested in developing CAI systems should know the types of CAI systems already in existence.

Linear programs

Skinner's principle of operant conditioning led to the development of linear programs in which a student followed through a course frame by frame, giving answers. The frames were set up in such a way that the student was led to a correct answer. If the student gave an incorrect answer, there was no

individual diagnosis or feedback. In other words, every student studied the same material in the same order and with the same feedback [Yazdani, p. 107].

Branching Programs

Tutorial programs allowed the student to learn by viewing sequential frames of material. Individualized pacing was permitted by branching to extra frames when additional work was needed by a student [Pressman, p. 187].

A technical fault with branching programs was the great number of branching statements necessary to individualize the program that caused the programs to become exceedingly complex.

Generative Computer-Assisted Learning

Drill and practice programs allow the student to reinforce previous learning by working on automatically generated practice exercises. This type of program also allows for some individualization by varying the level of difficulty of the exercises generated. Drill and practice programs are good for supplementing classroom instruction [Manion, p. 27].

The most popular drill and practice programs are arithmetic drills, but any subject matter where basic skills need to be taught is a candidate for this type of CAI. Drill and practice programs are still popular in schools.

Mathematical Models of Learning

Programs, with an underlying mathematical model of teaching strategies, allowed the computer to choose a teaching strategy according to the desired student learning response. The selection of the teaching strategies for the mathematical model were results of theoretical predictions of outcomes of various teaching strategies [Yazdani, p. 108]. Perhaps the reason this type of CAI was not widespread is due to the difficulty encountered in developing a mathematical model that imitates student learning. Students do not all learn in the same way, so trying to develop a mathematical model to predict teaching strategies for desired student learning outcomes was neither easy nor necessarily valid.

TICCIT

The Time-shared Interactive Computer Controlled Information Television project was funded by the National Science Foundation in 1971. The purpose was to use TICCIT as the main method of teaching calculus and English composition to see if CAI could be cost effective [Yazdani, p. 109].

Course material was developed by specialists in design, evaluation, and packaging. Presentation included outlines, examples of principles outlined, and exercises for practice. Mixed reactions to the success or failure of TICCIT may be the reason for its not being widely adopted.

PLATO

Programmed Logic for Automatic Teaching Operation started originally in 1960 and was later funded by the National Science Foundation. PLATO is essentially an expansion of TICCIT with greater numbers of terminals available to users, and end users developing their own courseware.

PLATO in the mid 1970's allowed CAI to be developed with the aid of an authoring system, Coursewriter [Pogue, p. 76]. This authoring system allowed great quantities of courseware to be developed for the PLATO system by users who did not have programming skills. Unfortunately, the courseware developed for PLATO was not as easily modifiable or as transportable as CAI written in programming languages such as BASIC or Pascal [Kontos, p. 5].

PLATO was impressive and did show that many students' motivation (wanting to learn on their own), rate of learning, and amount of learning were increased through the use of CAI [Spreecher, p. 47].

Simulations

In simulations, "the computer acts as an environment, permitting the student to manipulate parameters and observe the outcome" [Pressman, p. 188]. The use of simulations in education allows students to learn by watching as the parameters for the simulation are manipulated. Simulations are especially useful in areas where real life training is too costly or not possible. Marketing simulations and flight

simulations are two examples that would otherwise be too costly to allow students to gain experience. For example, marketing simulations require the manipulation of money within the stock market, and flight simulations require the manipulation of controls of an airplane during flight.

Simulations are only as good as the programmer makes them since correct modeling is needed to make the simulation be realistic and accurate. Some simulations are extremely complicated and may not be educationally beneficial for the intended audience unless care is taken in the selection process. Simulations are exciting and encourage student thinking as long as there is feedback on all important parameters to reveal the results of the student's actions [Fisher, p. 53].

Games

Games have gained student interest. Games can adapt to the ability of the player, allow students to fantasize, and can rate the student. Games are being developed now that will allow the computer to coach the player. The resulting consequencies of a coach in game playing are to optimize the learning situation so students are shown the best strategies and to give aid to students who are having difficulty and might otherwise give up.

Skills necessary to win a game include the development of problem-solving skills, use of previously acquired skills, and

accuracy and efficiency in the use of those acquired skills [Manion, p. 27]. Games teach students to be competitive and to analyze different situations. Games also can be used to change social behaviors and lengthen attention spans.

Games that involve following moving objects and use of a joy stick or a mouse for aiming are useful for students who have visual perception problems and need to practice coordinating eye-and-hand movements [Fisher, p. 52].

Problem-Solving

A problem-solving system allows a student to write a computer program. It is believed that by writing the program the student is participating in a problem solving activity. The emphasis is on "learning by doing" rather than "learning by being told" [Ward, p. 271].

LOGO, a computer language developed at MIT under Seymour Papert, is a problem-solving system. In the late 1960's Papert set out to create a computer environment in which mathematics could be learned easily, and the LOGO environment of the 1980's is an extension of that work. The LOGO learning environment allows the user to control a robot turtle to draw and explore geometric concepts or use a screen editor. The LOGO commands are words that are taken from everyday life and the programs which are created by the user are in the user's very own "natural" programming language [Watt, p. 50]. Complex programs that simulate "intelligent" behavior may be written in

LOGO.

LOGO has become very popular in a short time and many school districts are offering LOGO workshops to encourage their teachers to use LOGO with their classes.

Emancipatory Modes

The emancipatory program allows the computer to perform tasks for the student. The computer can retrieve information from a database or perform calculations and allow the student to concentrate on the subject matter [Yazdani, p. 112].

Spreadsheets are a type of the emancipatory programs that are being used in many classrooms to predict purchase costs without becoming bogged down with the arithmetic involved. Spreadsheets force the students to be concerned with the formulas needed and the correct organization of material.

Another emancipatory program is a word-processor for computer-assisted writing. Word-processors make the job of organizing, reorganizing, reviewing, and changing written material easier for the beginner as well as for the experienced writer [Manion, p. 28]. The use of a spelling checker enables students to find many of their spelling errors.

Dialogue Systems

Dialogue programs allow unstructured "conversation" between the computer and student [Pressman, p. 187]. To

implement a pure dialogue system is not possible at this time, but AI researchers are working on developing computer systems that can understand natural language [Yazdani, p. 113].

Tutor programs, a variation of dialogue systems, allow some dialogue between computer and user. Basic concepts and rules pertaining to some subject are presented to the student, and the student is led through exercises that call upon his or her understanding of the concepts and rules presented. The computer evaluates the student's comprehension, and provides additional practice of the specific skills and concepts being taught [Manion, p. 27].

CAI has come a long way in its development, however, further advancement in computer technology and educational software development is needed. ICAI, Intelligent Computer Assisted Instruction, seems to be the next step in the development of educational software, but hardware to run this new software will need to exceed the capabilities now available in most school districts.

1.3. Artificial Intelligence and Expert Systems.

Since ICAI is possibly the next step in educational software development, the following background information is included. The underlying concepts of artificial intelligence and expert systems need to be understood to appreciate the work being done to develop "intelligent" educational software.

The term "artificial intelligence" was invented in 1956 by John McCarthy to describe a now famous summer workshop at Dartmouth College [Waldrop, p. 1279]. "Artificial Intelligence (AI) is the study of how to make computers do things at which, at the moment, people are better" [Rich, p. 1].

AI, along with advanced technology, is the basis for a fifth generation of computers. Both the Japanese government and the U.S. government are spending great sums of money for AI research and "high-tech" development [Waldrop, p. 802]. New programming languages have been developed for AI. Imperative languages such as FORTRAN, COBOL, BASIC, Pascal, and Ada must have precise algorithmic directions for the actions which the computer is to perform. Descriptive languages have been developed for AI use which allow relations to be drawn among objects. LISP is an early example, and PROLOG, developed in 1972 by Colmerauer, is a descriptive language which has as its base formal logic. The Japanese plans for a fifth generation of computers are based on logic programming (Prolog) as the

core language [Ennals, p. 19].

One of the early AI programs was called the General Problem Solver (GPS). To attack a complex problem, the GPS breaks the problem down into smaller more manageable parts, each of which are then further broken down until a solution for each subpart is found [Lenat, p. 204]. The GPS failed to solve even the simplest problems because the solution demanded a "combinatoric explosion" of simpler problems to be solved. By the mid-1970's the idea of the general problem solver seemed impossible to achieve [Waldrop, p. 1280].

As a result of the limited power of general-purpose problem solvers to solve complex problems, many researchers began work on expert systems to solve problems in narrowly defined application domains [Hayes-Roth, p. 7]. Now, AI software, computer vision systems, natural language programs, and "expert" systems have found a spot in the marketplace.

Out of 20 years of AI research has come the fact that intelligence requires knowledge. One common way of representing knowledge in the database of an expert system is as a production system. A production system is a set of rules that describe actions to be performed, the conditions under which those actions apply, and a set of facts related to the particular tasks to be performed. Rules that specify the strategy used to decide what comes next must be included [Rich, p. 31], and rules of thumb or heuristics (techniques that

improve the efficiency of the search process) may be needed to alter control.

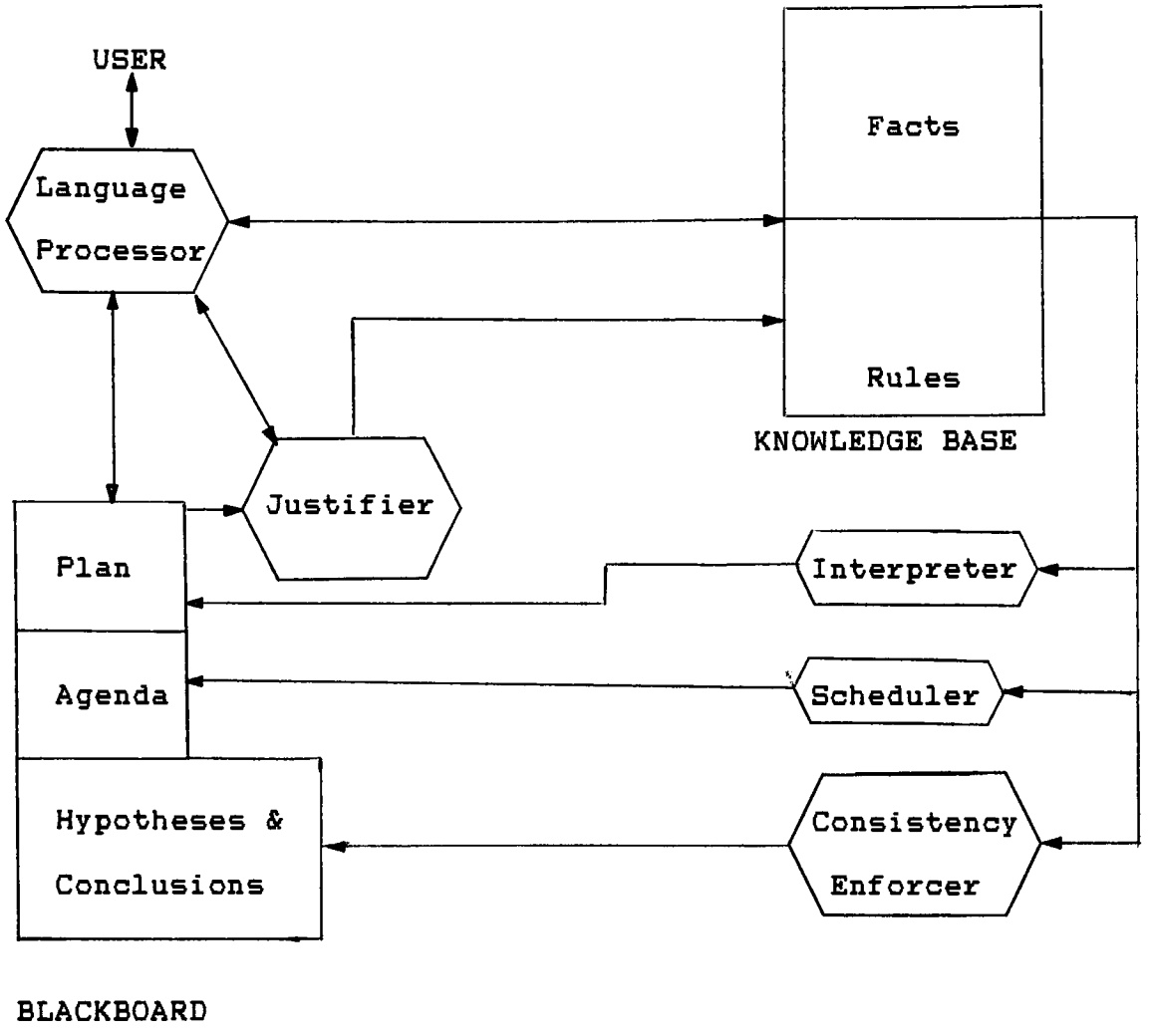

Figure 1. Anatomy of an ideal expert system [Hayes-Roth, p. 17].

An ideal expert system as indicated by figure 1. should include a natural language processor for interactions between

the user and the expert system; a "blackboard" for recording intermediate plans for attacking the problem, the agenda of potential actions awaiting execution, and the hypotheses and conclusions reached; a knowledge base that contains facts, heuristic planning strategies, and problem-solving rules; an interpreter to apply the rules according to the stored agenda; a scheduler to control the order of rule processing in the agenda; a consistency enforcer to ensure that conclusions reached are plausible and to eliminate those that are inconsistent; and a justifier that explains the system's actions to the user [Hayes-Roth, pp. 16-19].

Expert systems should be developed gradually with the experts and programmers communicating in "brainstorming" sessions. From these knowledge acquisition sessions come the facts, rules, and heuristics used by the programmers to develop the expert system. An expert system evolves slowly, beginning as a system that solves simple tasks and gradually expanding until it can solve harder tasks. Expert systems need to modify their "knowledge" as they perform and use this new "knowledge" to derive the next conclusion [Waldrop, p. 804].

Instructional expert systems must be able to diagnose student behaviour and present a plan for helping the student when weaknesses are diagnosed. The system must be able to record the student's knowledge and provide a tutorial interaction with the student to remedy any weaknesses [Hayes-Roth, p. 15]. An instructional expert system and a

medical expert system have much in common, i.e. both diagnose irregularities in student or patient performance, both prescribe remedies to correct those irregularities, and both give a plan of attack for administering the prescribed remedies [Hayes-Roth, pp. 13-14].

The development of ICAI software is still in its infancy. A look at several of the ICAI prototypes will point out that ICAI system development is costly and complex, and so far, they are not totally rewarding for either the designers or users of the systems.

1.4. Educational Prototypes Using Artificial Intelligence.

The following ICAI prototypes have been developed by teams of experts, and they show the complexity involved in producing instructional expert systems even for a narrowly defined domain. Producing an expert system to diagnose irregular student behaviour and prescribe remedies to modify that student behaviour is not easily accomplished.

DEBUGGY, developed at the Xerox Palo Alto Research Center, was a tutor program that specialized in subtraction. The program diagnosed student errors and made suggestions to the student to help him achieve success. If an error was made the program needed to isolate the problem and make suggestions for correcting the error. This was no easy matter as a student's answers may seem to be random [Waldrop, p. 803].

GUIDON used MYCIN to teach about infectious diseases. MYCIN, a very successful medical expert system, diagnoses and prescribes remedies for the treatment of infectious blood diseases [Hayes-Roth, p.9]. GUIDON appraised the student's progress in diagnosing a case by comparing steps taken by the student with steps taken by MYCIN to diagnose the same case. MYCIN was able to give explanations that related to requests for clinical information and reasons for pursuing a particular hypothesis. MYCIN could not explain the reasoning behind the sequence taken to reach its diagnosis, since the order of rule

processing was fixed by the order of the rules appearance in the expert system and not according to a strategy developed while diagnosing the case [Yazdani, p. 117].

SOPHIE III was developed by the same team which developed DEBUGGY. SOPHIE III trained repairmen to troubleshoot electronic circuitry [Gladwin, p. 22]. It uses a simulation model of electronics as well as an updated semantic net of each circuit to determine intermediate voltages and currents when a fault is introduced into the circuit. It responds in restricted English to provide the missing fact or rule that the student needs to reach the correct answer [Hayes-Roth, p. 41].

SPADE was a research program that was not fully developed. Its primary aim was to provide tutoring for students as they attempt to design a structured program. SPADE contained a model of the design process, and as students interacted with SPADE, their problem-solving strategies were compared with the design process model. SPADE's purpose was to help a student debug his or her program design [Yazdani, pp. 118-122]. Since designing programs was not a well understood activity, SPADE was limited in its ability to tutor, but it provided an opportunity to learn more about the design of programs.

WEST, devloped by Xerox, was a computer coach that gave students advice on how to play the game, How the West was Won. The program watched the student's performance in playing the game and built a statistical model of their behaviour. WEST,

an expert player of How the West was Won, occasionally interrupted the player to give suggestions about how to play the game better [Waldrop, p. 803].

WUSOR was a coach for the game of WUMPUS. It compared the model of solutions possible with how the student was interacting to give "expert" advise. Student learning models were developed that showed how new knowledge was gained from old knowledge, i.e. the order in which the student learned the skills being taught was important. One serious limitation of that type of student learning model was that the model could not differentiate between blind luck and actual knowledge [Yazdani, p. 123].

It is important not to underestimate the complexity of an ICAI system that incorporates not only expertize in a subject area but also diagnoses, prescribes, and administers remedies for any student's irregular performance. Equally important are the lessons learned from noting system limitations whether they lead to a greater understanding of the subject area or to a greater understanding of student modeling. To build a really good ICAI system requires the work of a team of experts who not only understand the subject area and system programming techniques, but who understand the management techniques needed to modify student behaviour.

Chapter 2. CAI Evaluation.

2.1. Choosing CAI Systems.

In developing a marketable system, the designer should meet certain criteria set forth by consumers who will be using the system. The CAI Basic Logic System (see Chapter 3) developed for this Masters Thesis should also include a simple language interface and some tutor capabilities present in ICAI systems. The rating of CAI systems in use, the information given in chapter 1 about developing ICAI instructional systems, and the information about ICAI prototypes should enable the system designer to build upon or to try a completely different approach than has been done historically.

The NEA Educational Computer Service guide for rating educational software was developed in 1983 [Gold, pp. 43-49]. According to the guide, technically sound software should be reliable, allow permanent records to be kept, allow use of an external printer, allow for individualization by branching according to student input, allow students to control the pace of screen presentation, allow users to access different areas via menus, collect data on students, and prepare reports for teachers.

An educational program also should be flexible, and should allow a student to end a session when desired or to start a

session at any point within the program. Students should be allowed to correct input to the program when a mistake in entering an answer is made. Programs should provide cues and prompts to aid the students as well as a help option that offers assistance to the user. The mode of presentation should be flexible to allow ease of use by either an expert or a novice. Programs should be easy to learn to use with lesson session lengths appropriate for student attention spans.

The screen displays of the program should be uncluttered, attractive, and easily read. The interaction between computer and student should emphasize good grammar as well as appropriateness of responses. Student progress should be rewarded or redirected, and the difficulty of material should be altered according to student responses. Objectives of each lesson should be clearly stated at the beginning of each lesson, and appropriate teaching should be presented to meet those objectives. Along with good use of graphics, sound and educational techniques, attention retaining devices should be available for keeping a student's interest in using the program. Student progress reports should be presented to both the student and the teacher, and these progress reports should be generated in a format that is easily understood.

A technical manual should include information on such technical features as program installation, start-up procedures, hardware configuration, the operating system, and programming

code. The manual should include an explanation of how all functions work as well as conditions and restrictions of their use.

A quick guide to choosing educational software should include: check completeness of educational documentation for teacher and student use, check achievement of the programs stated aims or goals, check appropriateness of the program with the intended audience, check screen presentation, check friendliness and flexibility of the program, check technical documentation, and check overall impression of the program [Preece, pp. 14-15].

Since a system designer is interested in producing the best possible CAI software, he should incorporate into his system the features that have proven to be successful while staying away from features that have failed. There is also room for completely fresh approaches especially in the area of ICAI.

2.2. Current Development Trends and Designing Methods for ICAI.

Development of ICAI systems using an expert team approach and rigorous testing of the systems before their availability in the marketplace is extremely important. These systems must be able to diagnose students' weaknesses and prescribe and administer remedies for those weaknesses. Student behaviour needs to be understood and models of that behaviour developed and incorporated into expert teaching systems. Moreover, there will also be a need for other types of CAI such as drill and practice, computer-assisted writing, and problem-solving systems.

In the late 1950s and 1960s, the behaviorist philosophy was used to develop programmed instruction and related computer-assisted instruction (CAI). CAI depended heavily on the specification and breaking down of content into small learnable units which helped the learner to be more successful [O'Neil, p. 164]. Even though there have been mixed reactions to the success or failure of CAI during those years, important lessons were learned, and ICAI systems also deal with narrowly defined domains.

ICAI expert systems will be very important and will be written according to the types of learning outcomes desired such as verbalization of information, discrimination, problem solving, perception, development of motor skills, and the alteration of

attitudes [Gold, p.42]. ICAI systems will communicate with the user in natural language, will create a model of the student's knowledge, skills, and strategies by judging the student's responses, and will act as tutors [Gladwin, p. 22]. The instructional systems will include ways of gaining the student's attention, let the learner know the objectives, stimulate recall of past learning, present the material in a stimulating manner, providing guidance to the learner, bring out student performance, provide feedback about student performance, assess student performance, and help the student retain what has been learned [Gold, p. 42]. In addition, the expert teaching systems will be able to diagnose irregular student behaviour, and modify that student behaviour.

It is important to present a language for conversations between the learner and the program that will give the student more initiative in decision making [Ward, p. 269]. This virtual interface should be designed for the intended user as the needs of the novice and expert vary [Goldes, p. 162]. The program's personality should be carefully considered since the student's reaction to the computer's responses should be positive to ensure a good learning situtation. Students can be alienated by computer responses that are extremely neutral or extremely friendly [Burkhardt, p.79]. An example of extreme neutrality would be a computer conversation consisting of only one response when an answer is correct and only one response when an answer is

incorrect. An example of extreme friendliness would be a computer conversation using words of endearment such as "honey" or "dear".

Use of menus to allow logical organization of available functions are usually a convenience for the user. Use of complex layers of menus, however, may cause the user to lose control of the program. A user who is not well oriented will not have the confidence to continue [Goldes, pp.161-164]. Similarly, going from a novice interaction mode to an expert interaction mode can cause great confusion if the change from one mode of conversation to another is not indicated when it occurs. The expert teaching systems will be dealing with narrowly defined domains so complex menuing, if any at all, may not be necessary.

CAI programs must be appealing for students to use. The programs must be both powerful and "friendly", i.e. easy to learn to use. The student should not be penalized for trivial errors, and there should be interesting sets of problems. Also, the student should not be unduly restricted in what he wants to do [Yazdani, pp. 111-112].

> "What will finally emerge from the artificial intelligence laboratories will be intelligent systems that have the capability to understand concrete domains and make inferences. These labs may also unravel the non-trivial problem of comprehending a natural language used in context. What will finally emerge from the learning and development centers will be models of individual learning and sets of articulated strategies for

learning [Burns, p. 181]."

Development of quality courseware will depend upon both involvement of educators and professional programmers working as teams. Banks of CAI courseware are foreseen before the year 2000 [Kontos, pp. 11-12]. Large scale production of educational courseware will be prepared by using production strategies that include several teams of professional educators and programmers who will design, review, and revise the system. The system's conversations, screen displays, code, and testing also will be scrutinized separately by the teams [Bork, pp. 78-79], the end result being the development of the highest quality material by the best teachers who will incorporate many different learning strategies. This educational courseware will utilize the full capability of the computer while meeting reasonable development costs. All courseware will have undergone extensive evaluation before it is marketed. The units will be easily modified to allow for easy maintenance and transportability and to encourage large-scale curriculum development. Authoring systems will increase CAI productivity and will allow transportability of CAI materials [Pogue, p. 76].

The above development methods and designing trends will be very costly and require more hardware than available to most high schools today. Some of the programs may be modified to fit

existing hardware in school systems, but if enough school districts want the new ICAI expert systems with full capabilities, they will upgrade their hardware to meet the needs of those systems. If a natural language processor can be developed to interface with expert teaching systems and if satisfactory student behaviour models can be built for these systems, we may see a complete revolution in the way our children are taught.

2.3. CAI Logic Systems of the Past.

The following CAI Basic Logic Systems have been developed, and at least one, VALID seems to fit the category of an ICAI expert system.

In 1972, Tennessee State University, in cooperation with Stanford University, tested a computer-assisted instruction curriculum funded by the National Science Foundation. One of the courses offered was Symbolic Logic and included the Basic Logic concepts: negation, conjunction, disjunction, conditional, biconditional, and valid argument. This course also covered the construction of logical derivations and was offered to approximately 60 NSF students [Searle, p. 5].

Another computer assisted instructional system for "first-order logic" was used at Stanford University for the academic year, 1972-1973. This system was written in LISP and was also funded by the National Science Foundation. Its importance was the use of a "computer-tutor", a theorem prover able to analyze the paths that students had taken in the derivation of a proof. The theorem prover also kept track of the paths that should have been taken to complete the proof for checking the student's work and giving help to the student. The system told the student that an error had been made and suggested what path to try next. This computer-based instructional system was developed

for the purpose of discovering what kinds of strategies could be used to tutor students to solve these formal proofs [Goldberg, p. 884].

VALID (Voice-Assisted Logic Instruction Dialogue), an extremely large and complex CAI course for teaching Elementary Symbolic Logic was developed at Stanford University and used extensively. This logic program was funded by the National Science Foundation and had undergone more than 10 years of development by 1979. VALID was a self-contained course rather than a supplement to a lecture. Each student received a manual and possible student assistant aid. Students had to meet mandatory milestones. Each student spent approximately 70 hours working on-line and 10-15 hours off-line to complete the course. The student's grade depended upon how much of the course had been completed, and the pass rate was around 75% [Ager, p. 336].

The course content of VALID included elementary logic, axiomatic theories, and applications of logic. The heart of the course was derivations. A proof checker was not only able to correct the statements entered by the students but was also able to give suggestions. If the statements entered by the student were not leading the student to the desired proof, VALID offered help. As students built up their knowledge of logic, more VALID capabilities opened up for their use. The degree of difficulty of questions also increased with the students proficiency.

VALID has been transported to between 15 and 20 installations in several U.S. universities as well as Australia, Saudi Arabia, and England since 1981. Standard Lisp by Hearn was chosen as the transport language. The transported version is mute as an auxillary computer handled the speech synthesis. Authoring capabilities are provided to allow logic instructors to write their own logic lessons. VALID is available from Stanford University for a $500 tape preparation and licensing fee and has been installed on DEC 10, DEC 20, and IBM/CMS systems. The underlying LISP system that runs the program is included on the tape so the system is entirely self-contained. Anyone wishing to acquire the program or who has further questions should contact Dr. Tryg A. Ager.

There are relatively few, if any, CAI basic logic systems available for high school student use, and since this is an area that is included in the New York State syllabus for the ninth, tenth, and eleventh grade integrated mathematics courses, it could possibly prove to be quite lucrative to develop such a system. The available hardware in most high schools would necessitate a much simpler version of basic logic than any of the systems mentioned in this section.

Chapter 3. System Structure.

3.1. System Design.

The purpose of the CAI Basic Logic System for this thesis is to teach elementary logic to high school students. The topics in the program are those normally taught as a unit in the ninth year integrated mathematics regents course offered in New York state [Buchman, pp. 3-10].

Topics in the system include teaching the student to identify statements from a list of sentences and phrases, recognizing the truth value of a statement, identifying open sentences, locating variables in open statements, and finding the domain and solution sets for variables. The student is taught how to write the negation of a statement as an English sentence and in symbolic form; how to construct a truth table for negations, conjunctions, disjunctions, conditionals, and biconditionals; how to recognize equivalent statements; how to write the converse, inverse, and contrapositive of conditional statements as English sentences and recognize them in symbolic form; how to complete truth tables; how to construct the headings of truth tables; how to identify tautologies and contradictions; and how to identify, the Law of Detachment, Law of Contrapositive Inference and Law of Syllogism, write them in symbolic form, and use them for valid reasoning [Rising, pp. 31-60].

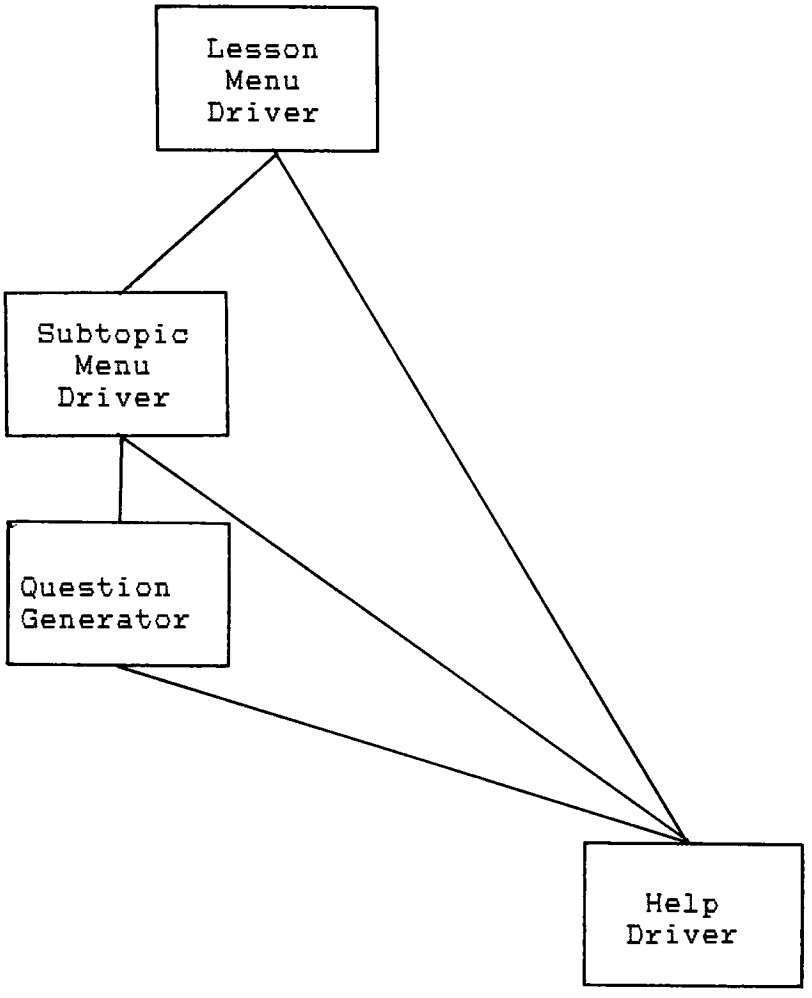

Figure 2. Architecture of the System.

The four modules that make-up the CAI Basic Logic System are shown in Figure 2. Briefly, the Lesson, Subtopic, and Help drivers present menus to the user and allow access to the system's functions. The Question Generator produces the practice examples for the subtopics. Examples of the psuedocode for each of the modules shown in Figure 2 can be found in Appendix A.

To get a feel for the behavior of the system, an example of an interactive lesson with the CAI Basic Logic System follows: The student presumably has gained access to the system by following the directions in the Student Handbook, Appendix C, p. 4.

LESSON MENU

Lesson 1 Introduction to Logic
Lesson 2 Negation
Lesson 3 Conjunction
Lesson 4 Disjunction
Lesson 5 Conditional (Implication)
Lesson 6 Converse, Inverse, and Contrapositive
Lesson 7 Constructing Truth Tables
Lesson 8 Biconditional
Lesson 9 Valid Reasoning

Choose the lesson you wish to study.
Type the number of the lesson (1-9)
or type E to EXIT the program.

|:

Figure 3. Lesson Menu produced by the Lesson Menu Driver.

The student selects lesson 6 and the following subtopic menu is displayed by the Subtopic Menu Driver. Additional subtopic menus are listed in the student handbook, Appendix C, pp. 7-10.

Converse, Inverse, and Contrapositive.

Topic 1 Write the CONVERSE of a CONDITIONAL statement.
Topic 2 Write the INVERSE of a CONDITIONAL statement.
Topic 3 Write the CONTRAPOSITIVE of a CONDITIONAL statement.
Topic 4 Write the CONVERSE, INVERSE, and CONTRAPOSITIVE of a CONDITIONAL statement given in symbolic form.

Type the number of the topic choice or type E to Exit to the LESSON MENU.

|:

Figure 4. A subtopic menu produced by the Subtopic Menu Driver.

The student selects subtopic 4 and the following information concerning the chosen topic is displayed by the Subtopic Menu Driver.

USING SYMBOLS

Let p represent the 1st given statement.
Let q represent the 2nd given statement.

p -> q is the GIVEN CONDITIONAL.
p is the HYPOTHESIS.
q is the CONCLUSION.

The CONVERSE of p -> q is q -> p.
Notice the HYPOTHESIS and CONCLUSION were SWITCHED.

The INVERSE of p -> q is ~p -> ~q.
Notice the HYPOTHESIS and CONCLUSION were BOTH NEGATED.

The CONTRAPOSITIVE of p -> q is ~q -> ~p.
Notice the HYPOTHESIS and CONCLUSION were BOTH SWITCHED AND NEGATED.

Press the RETURN key to continue.

Figure 5. Subtopic Lesson produced by Subtopic Menu Driver.

After the student has read the lesson, the system prompts the student to answer questions about the lesson.

```
Type a whole number greater than or equal to 0
to indicate the number of examples you wish to try.

|:
```

Figure 6. The Subtopic Menu Generator contains the looping structure for the number of examples requested.

After the student has selected a number of examples to try, the questions are generated by the Question Generator. Each question is randomly selected from facts in the system's database that have been set up for each subtopic. The first object in each fact is a numerical value that is matched to the value generated by the random rule [Clocksin, p. 46], and the rest of the objects in the fact include the answer and the information for asking the question. The Question Generator and database fact file used to generate the questions in Figures 7 and 8 can be seen in Appendix E. The Question Generator contains the rules for asking the questions and for checking the answers. Figures 7 and 8, contain both the system and user responses (symbols entered after the prompt |: that are within single quotes).

```
Question 1:


Please use single quotes around the answer.

Type the CONVERSE OF: ~f -> ~g.

|: 'f -> g'.
Sorry, you typed the INVERSE.

The CONVERSE is ~g -> ~f.

Press the RETURN key to continue.
```

Figure 7. Question 1 as generated by the Question Generator with incorrect student response and reason for student error explained by the system.

```
Question 2:


Please use single quotes around the answer.

Type the INVERSE of: ~h -> ~i.

|:  'h -> i'.
You are correct.

Press the RETURN key to continue.
You have correctly answered
1 question  out of 2 tried.

You have earned 50 %.

You have NOT passed the work in this section.
You should redo the lesson.

Press the RETURN key to continue.
```

Figure 8. Question 2 as generated by the Question Generator with correct student response and report of student performance indicated by the system.

The Lesson Menu Driver was designed to accept only valid student account ids, set up by the teacher, to enable the system to keep accurate student records. This module was designed to display the lesson menu, and it branches to either the Subtopic Menu Driver or to the Help Driver according to user input. The student records are written to his/her personal file at the conclusion of a student session, i.e. when the student asks to exit the system.

The Subtopic Menu Driver is accessed by the Lesson Menu Driver. This module was designed to display the subtopic menu, allow the student to choose a subtopic to study, present the subtopic lesson to the student, and allow the student to choose the number of practice exercises to try. This module also keeps count of the number of questions tried, and it computes and updates the student's score after each practice set of exercises is completed.

The Question Generator is accessed by the Subtopic Menu Driver, and randomly generates, or selects, the questions from the database of facts for each subtopic. This module displays each question, accepts and checks the student's answer for each question, and gives a reward or correction. This module also counts the number of questions tried and the number of questions answered correctly, and it assigns weights to question parts which will be used for computing the student's score.

The Help Driver is accessable from any of the other modules where the user is allowed to input a response to the system. This module displays a menu of the help functions available to the user, and it gives directions about their use (see Appendix B). One of the functions performed by this module is altering the number of questions that the student has asked to try in the Subtopic Menu Driver module. It also allows the user to enter a question mode where the system tutors the student by presenting definitions and truth tables for topics covered within the system (see Appendix C). The question mode allows the student to obtain a progress chart or a student handbook. The question mode also allows the teacher to open student accounts; to close student accounts; to obtain a classlist; to obtain an individual student progress chart or a progress chart of an entire class; and to print out a teacher handbook (see Appendix D).

Also, included with the system are modules that have been developed to do specific tasks (see Chapter 4). These are the basic tools needed in a prolog system such as member, append, efface, delete, sublist, prefix, reverse, and substitute to handle lists [Clocksin, pp. 46-142]. Additional functions used include a random number generator and a read_in function.

Chapter 4. Results.

4.1. Prolog routines developed.

The CAI Basic Logic System was written in Prolog, and several Prolog tricks were used to write routines to tackle problems encountered in developing the CAI Basic Logic System. Experience was also gained in transporting a system since the CAI Basic Logic System was started on a Rainbow 100 using a Prolog-86 Interpreter [Prolog-86]. Transporting the system from the Prolog-86 Interpreter to the C-Prolog Interpreter [Pereira] at R.I.T. was done without a great deal of difficulty as the two interpreters are quite similar and only minor variations needed to be made to the existing system. The C-Prolog Interpreter has many additional features not available on the Prolog-86 Interpreter so transporting back to a Prolog-86 Interpreter would require writing the missing rules in Prolog-86.

A problem of generating questions in a new random order each time the system was restarted was handled by seeding a random number generator with a new seed each time the system was restarted. To do this, the last value of the seed produced by the random generator module [Clocksin, p. 149] was saved by writing it to a file before exiting the system and that file was consulted each time the system was restarted.

```
addtot:-                        /* Increment T call        */
     tvalue(S),                 /* Current value of T      */
     T is S + 1,                /* Add 1 to current value */
     retract(tvalue(S)),        /* Remove old value of T  */
     asserta(tvalue(T)),!.      /* Save new value of T     */
```

Figure 9. Routine to increment the value of a variable within a module.

A counter, addtot in figure 9, was set up so the system could keep track of the number of questions that a user wished to try. Since the value of a variable within a module remains the same and cannot be reassigned a value within that module, a way was needed to save the current value of the counter and then increment it after a question had been generated. A value of 0 was asserted as the initial value of the counter. The counter value was then sent through the question generator where it was incremented, and a dummy variable returned the new value to the Subtopic Driver where the new value of T was then asserted into the system with a call to addtot. The value of the counter was then checked against the number of questions the user had wished to try. If the counter was smaller than the number of questions, the check failed and caused failures back to a repeat statement which allowed the next question to be generated.

In lesson one, the answers that the user supplied consisted only of atoms, however, the rest of the lessons required that the user supply complete sentences. The read_in function in Programming in Prolog by Clocksin and Mellish, pp. 87-88, was used for sentence input. Since, the statements in the database files started with capital letters, and the

read_in function converted all letters to lower case, a function that would take a statement from the database file and convert the first letter of the first word to lower case needed to be developed. The input from the user could then be tested against the statement from the database. A routine to capitalize the first letter of the first word input by the read_in function that had been developed by Professor Biles was modified. The new routine in figure 10 replaced the capital letter from the first word in the statement with the corresponding lower case letter.

```
low_case(Word,Lcaseword):-        /* Call to replace cap    */
     name(Word,[First|Rest]),     /* Make a list of codes   */
     rem_cap(First,Newfirst),     /* Send first code value  */
     name(Lcaseword,[Newfirst|Rest]).  /* Make new word     */
rem_cap(X,Newx):-              /* Cap in X lower case in Newx */
     X >= "A",                    /* Is X a capital letter ?     */
     X =< "Z",
     Newx is X - "A" + "a".       /* Computer lower  case code   */
rem_cap(X,X).                    /* X was already lower case    */
```

Figure 10. A routine to uncap a word starting with an upper case letter.

The read_in function refused to recognize tilde, "~", the symbol for negation, even though the "~" had been added to the read_in function as a word itself. A way to allow students to use the symbols that most accurately depict those used for logic in secondary mathematics textbooks, although awkward, was to ask the student to include the answers that are in symbolic form within single quotes. This allowed the answer to be considered an atom. Later, a routine to delete spaces from the atom input by the user and from the atom contained in the

database for better equality checking was added.

In the beginning of the system's development, the database files were setup to represent groups of general facts that could be accessed by all subtopic question generators. For example, the first subtopic question generator accesses the states and their capitals or the states and their cities to present questions to the students as well as generating questions about numerical equations and inequalities. Since a variety of questions concerning different facts were asked in the first lesson, separate rules were developed to generate the questions for each set of facts and separate rules were developed for checking the answers. The original form of the database files, of containing general information facts that could be used by several different subtopics, was changed to allow the coding of the question generator to be less complex. With all information for a particular subtopic being placed in the same fact, the new database files only need one rule to generate questions from them and far fewer rules to check the answer input by the user.

Since the database had each statement stored in list form, a way was needed to print the statements. Type(X) in figure 11, a function that types out, as a sentence, any list which ends in a period was developed, and later function typenp(X) was added to type all of the list except for the period at the end.

```
type(X):-          /* Type a statement given a list       */
     member(K,X), /* Look at each member of the list     */
     wrout(K),    /* Write out the member of the list    */
     member(K,[.,!,?]).    /* Backtrack until ., !, or ? */
wrout(K):-
     member(K,[.,!,?]),    /* If ., !, or ? write out    */
     write(K),!.           /* the character without space*/
wrout(K):-
     write(' '),write(K),!. /* Separate words with space */
```

Figure 11. Type a statement given a list of words and ending punctuation.

The truth table database file was setup to include a set of all the headings that belong on the table, a set of the number of characters in each heading, and the truth values for the table asserted by columns. A separate routine was developed to construct the truth tables and check the column values that were input by the student, and the routine was generalized to allow truth tables of size 3 columns by 2 rows up to 7 columns and 4 rows.

A routine for correcting errors and giving explanations when entering truth table headings was developed. It works by checking the number of characters in the expression and checking for key characters to determine the type of heading that should have been chosen.

The truth table section also needed to have new ways of keeping track of the number of parts in each question generated. This number varied each time a question was generated because the truth tables had different numbers of columns and rows. A dummy variable was sent through the

question generator function and returned a value which was added to the current number of parts tried.

The section in the truth table lessons that identifies logically equivalent statements works by locating the position of the equivalent statements within the table heading of the database. The equivalent statements are found by the getelement(X,Y,S) function where X is the position of the element in the set S and Y is the element returned.

The helpfile created problems of its own. In several places the system was to send a file to the printer. This was accomplished by modifying Professor Biles routine to go to the operating system and do vi editing. In each case the file is created and then the filename and printer name are sent to the operating system to allow the file to be printed.

Several housekeeping functions needed to be included to wipe out any values asserted during a previous student session. Each time a student session is initiated any values of the counter variables left in the system are removed and initialized to 0. Any scores which had been asserted into the system are removed before the new student file is consulted.

In order to allow the system to check the inverse and contrapositive of given statements, a module was developed to negate statements. This module only works for verbs presently in the database.

Unfortunately, some of the modules developed for the system are too lengthy and complex to present here. The work with the truth table constructions was very challenging, but it was also very rewarding.

Prolog is a powerful system language that made writing a CAI system much more straight forward than if nested levels of branching statements in an imperative language had been used.

4.2. Discrepancies and shortcomings of the system.

The CAI Basic Logic System, as it stands now, is a rough first attempt to create a CAI system using Prolog.

Few ICAI methods were incorporated as most of the time spent in developing the system was used to cover the introduction of the logic topics to the student and generate questions about those topics. Many of the criteria of a good Instructional System could not be met due to nature of the C-Prolog Interpreter. For example: To allow for ease of reading, the page of information given to the student should not scroll up. The page should be printed from the top of the screen down. I was not able to achieve this type of output due to the input and output capabilities, or lack of, in the C-Prolog Interpreter. The input for the truth tables is also rather crude. I had wished for the cursor to go to the exact spot in the truth table where the truth value was to be input. This type of screen addressing was not possible with the input functions available in the C-Prolog Interpreter. The decision was made to avoid building a cursor addressing package, and instead to concentrate on the instructional aspects of the project. The system is not robust due to the unpredictable handling of input errors by the C-Prolog Interpreter.

Statements input by a student in response to writing the negation, converse, or inverse of a statement may not be recognized by the system as being correct since the system only

recognizes statements rewritten according to its own rules.

The tutor capability is extremely limited. The system does not diagnose input for spelling (or typing) errors. The system does diagnose answers input in many cases and explains what the correction should be, but due to the time constraints of the designer, the system does not allow the student a second chance to type in a correct answer unless the entry resulted in a syntax error for input to the system.

The read_in function [Clocksin, pp. 87-88] did not recognize the "~" character even when modified so I used the read function of the C-Prolog Interpreter. Answers that included symbols, therefore, needed to be put inside single quotes which is a clumsy way of doing input.

The system is straight forward and it may not be interesting enough to keep the attention of students who are not self motivated to learn logic. The typing skills of students will become a factor where there are long sentences to be entered. The precision and patience of each student will be tested as directions must be followed.

The system was intended to be an intelligent tutor of basic logic, but it does not really meet that goal. The system does cover the basic logic topics taught in ninth grade mathematics in New York state [Dressler, pp. 119-184]. There is no provision for management of lessons according to the

diagnosis of student results. The teacher has no control over which lessons the student may try.

The system is not easily transportable to computer systems available to most New York State students. Most school systems are using microcomputers, and the Prolog interpreters available for them would impose additional limitations on the system.

Although a crude first attempt at developing a CAI Basic Logic System using the Prolog language was made, the designer was able to cover the subject area intended. Student testing of the system is needed at this time to pinpoint additional areas where work is needed.

Chapter 5. Conclusion.

5.1. Future Extensions.

The Basic Logic system should be made transportable to the microcomputers available to most students in New York state schools. The system would need to be transferred to diskettes and the code modified to allow it to interact with an appropriate Prolog interpreter for each microcomputer and to compensate for the decrease in available memory. Penn Yan Central School, where I teach, has purchased a Prolog-86 Interpreter using IBM DOS, and I plan to transport the Basic Logic System to run on the IBM PC.

Improve the input/output quality of the C-Prolog Interpreter, i.e. possible interfacing with another language that would allow screen oriented output and input. The quality of the lessons could be improved dramatically if the pages could be set up on the video display in a nicer manner. Also, having each page print from top to bottom rather than scrolling up from the bottom would greatly improve the readability, and it would also lessen the eyestrain of students who would be using the system. Student interaction could be greatly enhanced if the cursor could be positioned at the point in the truth table where the answer should be placed each time a value was needed.

A motivational aspect should be included to make the system more interesting. Competition based on student scores would possibly encourage the system's use, i.e. who in the class is the best. However, it would not necessarily motivate the slower student, but the development of logic games might solve the motivational problem for slower students who have successful completed a lesson. These logic games would be needed after each subtopic lesson, and they would need to be of varying difficulty to keep students at all levels of performance interested.

One type of logic game that could be included would allow the user at each level to choose actions, objects, people, etc. that would create a story. After the first lesson these choices might represent statements, variables, and domains. In succeeding lessons the actions become more complex as compound statements must be formed, and later the student can make conclusions about the story that can be tested for valid reasoning. The system would create the story given the facts, the student would be questioned about the story, and any student conclusions would be tested. Each story would probably be different as it would depend on student input, and if the student needed help, the system would offer suggestions. Students would be motivated by helping to create a story, by answering questions about their story, and finally, by drawing conclusions through valid reasoning about the characters in their story.

Students scores are computed and placed on each student's personal file so it would be possible to develop an improved diagnostic ability within the Basic Logic System and couple it with a management system to set up student lessons and route students through those assigned lessons. The teacher would be able to access the management system and alter assignments. The management system would allow for the difficulty level of questions generated at each subtopic lesson to vary according to student performance. Additional databases would be needed for each subtopic lesson, and the database chosen would result from the student's performance level.

Additional types of questions should be generated at each subtopic level to increase the student's exposure to the topic. The amount of information in each data file should be increased to avoid repetition of questions. A way of keeping track of all the questions asked of any student to avoid the same question from being asked more than once, especially within the same lesson, needs to be incorporated.

The system should allow students more than one chance to input the correct answer, and it should check for spelling errors with partial credit given for answers which are close to being correct. Additional help should be given to students who have not answered a question correctly to lead them to the correct answer.

Pretests and posttests should be available in both interactive and printed forms. The results of these tests should be included in the student and teacher report. These tests should also be used to allow the management system to set up student assignments.

The system should be tested with actual students. Once the system is transported, it could be used with students who are studying basic logic in their ninth grade mathematics course. Testing the system in this way would point out any inconsistencies in the system so it could be reviewed and improved. Also, the testing should reveal if its use helped students make a significant gain in their understanding of basic logic.

The system needs to be expanded to include more of the tutor capabilities. The learning environment is too structured to be in keeping with modern ICAI systems.

5.2. Final Remarks.

I feel that the CAI Basic Logic System would work well with the students I presently teach at Penn Yan Central School. I do not feel that they are ready for a self-contained CAI system to teach them logic, but this system would be good for supplementing a basic logic unit. The system relies heavily on the student's ability to read and follow directions. The ninth graders that I teach need to be constantly reminded to read and follow directions. I have taught ninth grade students for over twenty years and find that they are extremely insecure in what they are doing and need constant reassurance that they are doing what was asked for in the directions.

There are many areas where the system needs improvement as indicated in the future extensions section but I feel that it is a good first try at a CAI Logic system. The section that allows the students to identify the converse, inverse, and contrapositive of a statement would enable even the slowest of students to practice until they could discriminate which statement was given. The sections dealing with writing headings for truth tables and completing truth tables would be especially valuable to use with slower kids who need drill and practice and constant review to remember how to do something. The last section in valid reasoning would be used by me as enrichment for faster students as this is an optional topic at the ninth grade level in the course I teach. The overall

system does what I intended it to do, i.e. teach basic logic and allow the students to do practice examples to reinforce what they have been taught.

The help mode should prove to be of value not only to the student who needs to review a topic and see his progress but also to the teacher who can retrieve a progress chart of everyone in her class. This section also allows the user to obtain as many copies of the handbooks as needed very easily and economically.

As a final analysis of the CAI Basic Logic System, I feel, it was a good first attempt, but I also feel that the topic was too broad. Too much time was spent in covering the basic educational concepts for all of the lessons and not enough time was left to explore the use of AI for student tutoring.

Bibliography.

Ager, T.A., and McDonald, J., The Stanford Logic Course: Design, Dissemination, and Demonstration, Proceedings of NECC 1979 National Educational Computing Conference, University of Iowa, 1979, pp. 335-342.

Bell, Margaret E., The Role of Instructiional Theories in the Evaluation of Microcomputer Courseware, Educational Technology, Vol. 25, No. 3, March 1985, pp. 36-40.

Bork, Alfred, Producing Computer Based Learning Material at the Educational Technology Center, Journal of Computer-Based Instruction, Vol.7, No. 3, Summer 1984, pp. 78-81.

Brien, Robert, Sequencing Instruction: a Cognitive Science Perspective, Programmed Learning and Educational Technology, Vol. 20, No. 2, May 1983, pp 102-114.

Buchman, Paul and Buchman, Aaron, Three-Year Sequence for High School Mathematics, Course I, The University of the State of New York, State Education Department, Bureau of General Education Curriculum Development, Albany, N.Y., October 1976, pp. 1-10.

Bumby, Douglas and Klutch, Richard, Mathematics: A Topical Approach Course I, Charles E. Merrill Publishing Co., Columbus, Ohio, 1978.

Burkhardt, Hugh, Fraser, Rosemary, and Wells, Colin, Teaching Style and Program Design, Computers and Education, Vol. 6, No. 1, 1982, pp. 77-84.

Burns, Hugh, The Challenge for Computer-Assisted Rhetoric, Computers and the Humanitites, Vol. 18, No. 3/4, July-December 1984, pp. 173-181.

Cercone, Nick and McCalla, Gordon, Artificial Intelligence: Underlying Assumptions and Basic Objectives, Journal of the American Society for Information Science, Vol. 35, No. 5, September 1984.

Clocksin, W.F. and Mellish, C.S., Programming in Prolog, Springer-Verlag, New York, 1981.

Davis, Randall, & Lenat, Douglas. B., Knowledge-Based Systems in Artificial Intelligence, McGraw Hill, Inc., 1982.

Dressler, Isidore and Keenan, Edward P., Integrated Mathematics: Course I, AMSCO School Publications, Inc., New York, 1980.

Duquetee, Cheryll, Formative Evaluation of Courseware: One Instrument, Educational Technology, Vol. 25, No. 2, February 1985, pp. 20- 23.

Edwards, Judith B., "CAI and Training Needs", Professional Development and Educational Technology, Association for Educational Technology, 1980, pp. 117-124.

Ennals, Richard, Beginning Micro-Prolog, Harper & Row, Publishers, New York, Second Revised Edition, 1984.

Ferraris, M., Midoro, V. and Olimpo, G., Petri Nets as a Modeling Tool in the Development of CAL Courseware, Computers and Education, Vol. 8, No. 1, 1984, pp. 41-49.

Fisher, Glenn, Computer Games in the Classroom, Recreational Computing, Issue 49, January-February 1981, pp. 52-53.

Gladwin, Lee A., Computer-Based Training in the Second Computer Age, Performance and Instruction Journal, Vol. 23, no. 7, September 1984, pp. 21-23.

Gold, Patricia Cohen, Educational Software- New Guidelines for Development, Association for Educational Data Systems Journal, Vol. 18, No. 1, Fall 1984, pp. 41-50.

Goldberg, A., Design of a Computer Tutor for Elementary Mathematical Logic, Proc. IFIP Congress 4, Amsterdam, North-Holland.

Goldes, Harold, User Problems in Interactive Environments, British Journal of Technology, Vol. 15, No. 3, October 1984, pp. 161-174.

Good, Ron, Scientific Problem Solving by Expert Systems, Journal of Research in Science Teaching, Vol. 21, No. 3, March 1984, pp. 331-340.

Hayes-Roth, Frederick, Waterman, Donald A., and Lenat, Douglas B., (eds.), Building Expert Systems, Addison Wesley, Reading, Massachusetts, 1983.

Kamouri, Anita L., Computer-Based Training: A Cognitive Framework for Evaluating Systems' Designs, Journal of Educational Technology Systems, Vol. 12, No. 4, 1983-1984, pp. 287-305.

Kinnucan, Paul, Software Tools speed expert System Development, High Technology, Vol. 5, No. 3, March 1983, pp. 16-20.

Knapper, Christopher Kay, Evaluating Instructional Technology, Halsted Press, 1980.

Kontos, George, Instructional Computing: In Search of Better Methods for the Production of CAI Lessons, Journal of Educational Technology Systems, Vol. 13, No. 1, 1984-1985, pp. 3-14.

Langley, Pat and Carbonell, Jaime G., Approaches to Machine Learning, Journal of the American Society for Information Science, Vol. 35, No. 5, September 1984.

Lenat, Douglas. B., Computer Software for Intelligent Systems, Scientific American, Vol. 251, No. 3, September 1984, pp, 204-209, 211-213.

Manion, Mary H., CAI Modes of Delivery and Interaction: New Perspectives for Expanding Applications, Educational Technology, Vol. 25, No. 1, January 1985, pp. 25-28.

Megarry, Jacquetta, Walker, David R.F., Nisbet, Stanley, and Hoyle, Eric, eds., Computers and Education, Kogan Page, London, 1983.

Occhiogrosso, Marilyn, Reviewing Sequential Mathematics Course I, Amsco School Publications, Inc., New York, 1985.

O'Neil, Harold F., ed. Computer-Based Instruction A State-of-the-Art Assessment, Academic Press, 1981.

O'Shea, T. and Self, J. Learning and Teaching with Computers, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1983.

Percival, Fred and Ellington, Henry, A Handbook of Educational Technology, Kogan Page Ltd, London, 1984.

Pereira, Fernando, Ed., C-Prolog User's Manual, Version 1.4, April 24, 1985, SRI International, Menlo Park, California.

Preece, Jenny, and Jones, Ann, Training Teachers to Select Educational Computer Software: results of a formative evaluation of an Open University pack, British Journal of Education Technology, Vol. 16, No. 1, January 1985, pp. 9-20.

Prolog-86 User's Guide and Reference Manual, Software version 1.12, Document version 1.12, MICRO-AI, Rheem Valley, Ca., 1984.

Pogue, Richard E.,Ph.D., What is an Authoring System?,Journal of Educational Technology Systems, Vol. 13, No. 2, 1984-1985, pp. 75-89.

Pressman, Israel and Rosenbloom, Bruce, CAI: Its Cost and its Role, Journal of Educational Technology Systems, Vol. 12, No. 3, 1983-1984, pp. 183-208.

Rich, Elaine, Artificial Intelligence, McGraw Hill, 1983.

Rising, Gerald R., Bailey, William T., Blaeuer, David A., Frascatore, Robert C., and Partridge, Virginia, Unified Mathematics Book 1, Houghton Mifflin Company, Boston, 1981.

Ross, Steven M., Matching the Lesson to the Student: Alternative Adaptive Designs for Individualized Learning systems, Journal of Computer-Based Instruction, Vol. 11, No. 2, Spring 1984, pp. 42-48.

Schmieder, Marjorie F., The Computer and the Secondary School English Curriculum, Rochester Institute of Technolgy Master's Thesis, April 1983.

Searle, Barbara, Computer-Assisted Instruction Program: Tennessee State University. Technical Report Number 198., February, 1973, 40 p.

Self, J.A., Artificial Intelligence Techniques in Computer Assisted Instruction, Australian Computer Journal, Vol. 9, No. 3, September 1977, pp. 118-127.

Spreecher, Jerry W. and Chambers, Jack A., Computer Assisted Instruction: Factors Affecting Courseware Development, Journal of Computer-Based Instruction, Vol. 7, No. 2, November 1980, pp. 47-57.

Strang, Harold and Loper, Ann Booker, A Microcomputer-Based Simulation of Classroom Interaction, Journal of Educational Technology Systems, Vol. 12, no. 3, 1983-1984, pp. 209-219.

Waldrop, M. Mitchell, Artificial Intelligence (I): Into the World, Science, Vol. 223, No. 4638, February 24, 1984, pp. 802-805.

Waldrop, M. Mitchell, Natural Language Understanding, Science, Vol. 224, no. 4647, April 27, 1984, pp. 372-374.

Waldrop, M. Mitchell, The Necessity of Knowledge, Science, Vol. 223, no. 4642, March 23, 1984, pp. 1279-1282.

Ward, R.D., Sewell, D.F., Rostron, A.B., and Phillips, R.J. Interactive Computer Learning for the Classroom: Problems and Principles, Programmed Learning and Educational Technology, Vol. 20., No. 4, November 1983, pp. 269-275.

Watt, Dan, Teaching Turtles Logo as an Environment of Learning, Popular Computing, Vol. 1, No. 9, July 1982, pp. 48-56.

Wilson, R.N. and McCrum, E., Use of Modular Design in the Production of Portable CAL Software: A Case Study, Computers and Education, Vol. 8, No. 2, 1984, pp. 229-237.

Yazdani, M. and Narayanan, A. (eds.), Artificial Intelligence Human Effects, Halsted Press: John Wiley & Sons, New York, 1984.

Appendix A. Basic Logic System Code.

Section 1. Start Up and Lesson Menu Driver.

Section 2. Subtopic Menu Driver.

Section 3. Question Generator.

Section 4. Help Driver:

Section 1. Start Up and Lesson Menu Driver:

```
Start Up the Basic Logic System;
Enter User Id;
if  Id NOT on file
     then  End
     else
          repeat
          Display Lesson Menu (see Appendix C);
          Enter Choice;
          if Choice is Equal to Help
               then (see Help Driver Section 4)
               else (see Subtopic Menu Driver Section 2)
          until Choice is Exit;
Update User File;
End.
```

Section 2: Subtopic Menu Driver:

```
(From Lesson Menu Driver Section 1)
repeat
Display Subtopic Menu;
Enter Choice;
If Choice is Equal to Help
     then (see Help Driver Section 4)
     else begin
          Prompt for Number of Questions;
          Enter Number;
          (see Question Generator Section 3)
          Assert Score;
until Exit;
(Return to Lesson Menu Driver)
```

Section 3: Question Generator:

```
(From Subtopic Menu Driver Section 2)
Display Question;
Enter Answer;
if Answer is Not Correct
     then  if  Answer is Help
               then (see Help Driver Section 4)
               else Give Assistance
     else begin
            Give Praise;
            Update Number of Questions
               Answered Correctly
          end;
Update Number of Questions Tried;
(Return to Subtopic Menu Driver)
```

Section 4: Help Driver:

```
(From Point of Entry Sections 1, 2, & 3)
repeat
Display Helpmenu;
Enter choice;
if choice is Directions
     then Display Directions
     else if choice is Question Mode
          then Output Information Requested (see page 37)
          else if choice is Alter Number of Questions
                    then Enter New Number of Questions
until Exit;
(Return to Point of Entry)
```

Appendix B. Example Listings.

Section 1. Sign On.

Section 2. Sample Session: Topic 1 from Lesson 1.

Section 3. Helpmenu and Sample Listings.

Section 1. Sign On.

PROGRAM LOGIC

A Computer Assisted Instruction program for teaching
lementary Logic to students in Secondary High School.

rogrammer: Edith S. Shortt
Rochester Institute of Technology
1985

ype your ACCOUNT ID.
lease end all input with a dot (.).

Section 2. Sample Session: Topic 1 from Lesson 1.

LESSON MENU

_esson 1 Introduction to Logic
_esson 2 Negation
_esson 3 Conjunction
_esson 4 Disjunction
_esson 5 Conditional (Implication)
_esson 6 Converse, Inverse, and Contrapositive
_esson 7 Constructing Truth Tables
_esson 8 Biconditional
_esson 9 Valid reasoning

:hoose the lesson you wish to study.
'ype the number of the lesson (1 - 9)
)r type E to EXIT the program.

'ype a whole number greater than or equal to 0
·o indicate the number of examples you wish to try.

: 5.

The topic is the TRUTH-VALUE of a sentence.

The TRUTH VALUE of a sentence is indicated by either

T for TRUE,
or F for FALSE.

Examine the following sentence.

Albany is the capital of New York.

The above sentence has a truth value of T because it is TRUE.

Press the RETURN key to continue.

lemember that the TRUTH VALUES are
T for a TRUE STATEMENT, and
F for a FALSE STATEMENT.

xamine the following sentence.

5 + 6 = 10

he truth value of the above sentence is F because it is FALSE.

ress the RETURN key to continue.

Question 1:

Type the TRUTH VALUE for the following sentence:
Montgomery is the capital of Alabama.

|: t.
Very good.

Press the RETURN key to continue.

Question 2:

Type the TRUTH VALUE of the following sentence:
9 + 22 = 31

|: f.
You are wrong. This statement was TRUE.

Press the RETURN key to continue.

uestion 3:

ype the TRUTH VALUE of the following sentence:

2 > 89

: f.
ou are correct.

ress the RETURN key to continue.

Question 4:

Type the TRUTH VALUE of the following sentence:

7 < 4

: f.
You are correct.

Press the RETURN key to continue.

Question 5:

Type the TRUTH VALUE for the following sentence:

39 - 20 = 39

: t.
You were correct.

Press the RETURN key to continue.
You have correctly answered
4 questions out of 5 tried.

You have earned 80 %.

Nice going!. You have done very well.

Press the RETURN key to continue.

Section 3. Helpmenu and Sample Listings.

'here are several ways in which you can receive help.

q question mode

'ou may enter the question mode by typing q.
'he question mode allows you to ask simple questions
ıbout the subject matter. You should consult the student
ıandbook before you use this mode.

'ress the RETURN key to continue.

a alter number of examples requested

You may alter the number of examples chosen by typing a. To change the number of examples you wish to try, type a new whole number for the number of examples you wish to do. Remember the whole number 0 indicates that you will try NO examples.

You will still be expected to answer the question you were on when you typed help.

Press the RETURN key to continue.

x exit help mode

When you type x you will be returned to the exact spot in the program where you were before you typed help.

Press the RETURN key to continue.

You are NOT in a position to alter the number of examples!

Please type d, q, or x.

|:

'ou have chosen to alter the number of examples
:hat you wish to answer. Please type a whole
iumber greater than or equal to 0 to indicate the
iew number of examples that you wish to try.

:

lould you like to ask a question?

: yes.
lhat seems to be your problem?

: explain truth table.

› express all the
possible combinations of the TRUTH VALUES for a given
statement.

Suppose we were given:

p:	5 is prime.	Truth Value: t
q:	3 is even.	Truth Value: f

How can we find the truth value of p -> q?
Construct the TRUTH TABLE

p	q	p -> q
t	t	t
t	f	f
f	t	t
f	f	t

You can now conclude that p -> q is false by looking at
row 2. Notice p is t and q is f as in the example above.

What seems to be your problem?

|:

Appendix C. Student Handbook.

STUDENT HANDBOOK

FOR USE WITH CAI BASIC LOGIC

A COMPUTER AIDED INSTRUCTION SYSTEM

BY

EDITH S. SHORTT

1985-1986

ROCHESTER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

PREFACE:

CAI Basic Logic is intended for use by Secondary mathematics students who are beginning a study of logic. The topics covered include: truth value, statement, variable, domain, open sentence, solution set, negation, conjunction, disjunction, conditional, converse, inverse, contrapositive, constructing truth tables, biconditional, and valid reasoning.

Students will work with symbol and sentence manipulation.

TABLE OF CONTENTS. Pages

INTRODUCTION TO STARTUP SYSTEM:

You should have an ACCOUNT ID which has been given to you by your teacher. If you do not have an ACCOUNT ID you should get one from your teacher.

Write your ACCOUNT ID : _____

TO LOAD CAI BASIC LOGIC, FOLLOW THE STEPS BELOW:
(Typing ERRORS can be CORRECTED by using the BACKSPACE KEY and RETYPING.)

1. After the operating system prompt: %
   a) Type: prolog -A 512 -H 512
   b) press the RETURN key

2. After the prolog prompt: |?-
   a) Type: [loadfiles].
   b) press the RETURN key

3. After loadfiles has been loaded and you see: |?-
   a) Type: fileload.
   b) press the RETURN key
   (Please be patient as this takes about 5 minutes.)

4. After fileload had loaded all the system files and you see: |?-
   a) Type: logic.
   b) press the RETURN key

5. From this point on you will interact with the system. Usually you will want to use lower case rather than upper case letters.
   a) Answer each question asked.

b) Type a dot (.) at the end of each answer before pressing the RETURN key. (If you forget the dot you will receive the prompt: | and nothing will happen until you type a dot and press the RETURN key.)

6. In cases where SYMBOLS are used in the answer you must place 'single quotes' around the answer.

Examples: '~p'.
'p -> q'.
'(~p v q)'.

The reason single quotes are needed is to help the C-Prolog Interpreter understand that you mean the entire expression is the answer. Otherwise, the it takes each symbol as a a separate character.

If you forget the single quotes the system will treat the symbols individually. The behaviour of the system may be unpredictable so please be sure to follow instructions and use single quotes when you are asked to use them around your answer. Otherwise, you will be credited with a wrong answer.

MENU ARRANGEMENT:

(There are 3 layers of MENUS.)

Layer 1 LESSON MENU

Layer 2A SUBTOPIC MENUS

Layer 2B HELPMENU

Layer 1 LESSON MENU

Lesson 1 Introduction to Logic

Lesson 2 Negation

Lesson 3 Conjunction

Lesson 4 Disjunction

Lesson 5 Conditional (Implication)

Lesson 6 Converse, Inverse, Contrapositive

Lesson 7 Constructing Truth Tables

Lesson 8 Biconditional

Lesson 9 Valid reasoning

Choose the lesson you wish to study.

Type the number of the lesson (1 - 9)

or type E to EXIT the system.

Layer 2A SUBTOPIC MENUS

1. Introduction to Basic Logic Subtopics:

Topic 1 truth value

Topic 2 statement

Topic 3 variable

Topic 4 domain

Topic 5 open sentence

Topic 6 solution set

Type the number of your choice (1 - 6)

or type E to Exit back to the lesson menu:

2. Negation

Topic 1 Writing the Negation of a statement.

Topic 2 Using the ~ symbol for Negation.

Topic 3 Constructing Truth Tables for Negation.

Type the number of your choice (1 - 3)

or type E to Exit back to the lesson menu:

3. Conjunction

Topic 1 Write the Conjunction of two statements.

Topic 2 Using the ^ symbol for Conjunction.

Topic 3 Constructing a Truth Table for Conjunction.

Type the number of your choice (1 - 3)

or type E to Exit back to the lesson menu:

4. Disjunction

Topic 1 Write the Disjunction of two statements.

Topic 2 Using the v symbol for Disjunction.

Topic 3 Constructing a Truth Table for Disjunction.

Type the number of your choice (1 - 3)

or type E to Exit back to the lesson menu:

5. Conditional (Implication)

Topic 1 Write the Conditional of two statements.

Topic 2 Using the -> symbol in a Conditional.

Topic 3 Constructing a Truth Table for a Conditional.

Type the number of your choice (1 - 3)

or type E to Exit back to the lesson menu:

6. Converse, Inverse, and Contrapositive

Topic 1 Write the Converse of a Conditional statement.

Topic 2 Write the Inverse of a Conditional statement.

Topic 3 Write the Contrapositive of a Conditional statement.

Topic 4 Write the Converse, Inverse, and Contrapositive of a Conditional given in symbolic form.

Type the number of your choice (1 - 4)
or type E to Exit back to the lesson menu:

7. Constructing Truth Tables

Topic 1 Complete a Truth Table.

Topic 2 Write Truth Table heading and complete the truth table.

Topic 3 Recognize a Tautology or Contradiction.

Topic 4 Truth Tables which include the Biconditional.

Type the number of your choice (1 - 4)
or type E to Exit back to the lesson menu:

8. Biconditional

Topic 1 Write the Biconditional of two statements.

Topic 2 Using the <-> symbol for Biconditional.

Topic 3 Constructing a Truth Table for Biconditional.

Type the number of your choice (1 - 3)
or type E to Exit back to the lesson menu:

9. Valid Reasoning

Topic 1 Law of Detachment

Topic 2 Law of Contrapositive Inference

Topic 3 Law of Syllogism

Topic 4 Identifying Valid Arguments

Type the number of your choice (1 - 4)
or type E to Exit back to the lesson menu:

Layer 2B HELPMENU

Type the letter indicated for the topic where you need help:

d help directions

q enter questions mode

a alter the number of questions chosen

x exit help mode

INTERACTION WITH QUESTION MODE:

SYSTEM DISPLAYS: Would you like to ask a question?
;? -

STUDENT RESPONSE: yes or no

SYSTEM DISPLAYS: What seems to be your problem?
;?-

STUDENT RESPONSE: (Any question, sentence or phrase which contains any one of the following possibilities:)

1. explain (succeeded by any one of the following)

truth value	disjunction	valid reasoning
open sentence	conjunction	truth table
statement	conditional	equivalent statements
domain	implication	
replacement set	biconditional	
variable	converse	
solution set	inverse	
negation	contrapositive	

2. display (succeeded by any one of the following)

negation	conjunction	disjunction
conditional	biconditional	

3. progress chart (preceeded by one of the following)

display to screen	print on paper

4. sthndbk

Appendix D. Teacher Handbook.

TEACHER HANDBOOK

FOR USE WITH CAI BASIC LOGIC

A COMPUTER AIDED INSTRUCTION SYSTEM

BY

EDITH S. SHORTT

1985-1986

ROCHESTER INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY

Preface:

This handbook is to be used in conjunction with CAI Basic Logic and the Student Handbook for CAI Basic Logic. This handbook will only give additional features available to teachers not covered in the Student Handbook.

This system should be used as a supplement to a class lecture. If you have a class of highly motivated students, you may wish to try this as a unit of independent study.

Table of Contents. Pages.

1. STARTUP PROCEDURES.

CAI Basic Logic has been installed on a Digital Equipment Vax at Rochester Institute of Technology. This system was developed as a Masters Thesis project. The system is written in Prolog. The system utilizes the C-Prolog Interpreter which is running under a Unix Operating System.

In order to access this system you must have an account on RIT installation where the system resides.

Assuming all of the above are satisfied and you have permission to use the system.

Type:	prolog -A 512 -H 512	after the RIT insallation prompt %
Type:	[loadfiles].	after the C-Prolog Interpreter prompt ¦?-
Type:	fileload.	after the C-Prolog Interpreter prompt ¦?- (You will need to wait about 5 minutes for all the system files to load.)
Type:	logic.	after the C-Prolog Interpreter has consulted the CAI Basic System and returns the prompt ¦?-

If this is your first time, you do NOT have an ACCOUNT ID so just type: teacher.
Access to the system will then be allowed. If you wish to work through the system, you should open up an ACCOUNT ID for yourself.

2. OPENING A STUDENT ACCOUNT.

In order for students to use the system an ACCOUNT ID must be set up for them. To access this function type: help. for the Lesson Menu choice. A helpmenu will be displayed. Type: q as the menu choice. You are now in question mode.

After the question displayed:

Would you like to ask a question?

Type: teacher.

The computer should respond with:

You are the teacher!

Do You have a request?

Type: open account.

You will be prompted for the students name and ACCOUNT ID which is whatever you assign.

3. CLOSING A STUDENT ACCOUNT.

Same procedure as you used to open the account except type: close student account.

4. OBTAINING A PRINTOUT OF THE CLASSLIST.

A classlist is generated when you opened your student accounts. To have it printed out follow the same procedures as above except type: print classlist.

5. OBTAINING PROGRESS CHARTS.

Same procedures as above except type: print class progress report. You will be prompted for the name of the printer.

You may also obtain a progress chart for an individual student by typing: individual progress chart.

6. SUGGESTIONS FOR GIVING STUDENTS INTERACTIVE PRETEST AND POSTESTS.

To give a student a PRETEST, enter pretest instead of date, if you want both they can be entered together with no spaces between them.

Advance the student to the question section that you are pretesting. Student may need to be monitored so help is not accessed.

To give a student a POSTEST, enter postest as above. You may advance the student to the question section or let the student review by paging through the explanations of the section being tested.

7. EXPLANATION OF SYSTEM EVALUATION OF STUDENTS.

As questions are generated, the number of questions answered correctly is counted. Also, the number of questions, tried is stored. At the conclusion of the set of questions. The student's score is put on the student's personal file.

The student is given the percentage computed by (number of questions answered correctly/number of questions tried) * 100. A percentage of 100 is assumed to be mastery of the section. A percentage of 70 or above is considered to be passing. A percentage below 70 is considered failing.

Appendix E. Sample Question Generator and Database Fact File.

```
/******************** CONVSYM ********************************/

/*************************************************************/
/* THIS SECTION GENERATES THE EXAMPLES FOR CONVERSE, INVERSE,
AND CONTRAPOSITIVE OF A CONDITIONAL STATEMENT USING SYMBOLS*/
generate61(4,C,W):-
        cls,
        write('Question '),write(C),write(':'),nl,nl,
        random(52,S),
        symbol6(S,A,B,C1,C2,I,C3),  /* There are 52 possible
                                       choices in the file. */
        random(3,Y),       /* There are 3 question types.  */
        askques(Y,C1,C2,I,C3), /* Ask appropriate question*/
        W is C + 1,        /* Add 1 to number completed.   */
        pause,!.           /* wait for user to read results*/

/*************************************************************/
/* THIS SECTION ASKS THE APPROPRIATE QUESTION, CONVERSE,
INVERSE, OR CONTRAPOSITIVE.*/
askques(1,C1,C2,I,C3):-
        write('Please use single quotes around the answer.
              '),nl,nl,
        write('Type the CONVERSE of: '),
        write(C1),nl,nl,
        read(S),
        check64(S,1,C1,C2,I,C3),!.
askques(2,C1,C2,I,C3):-
        write('Please use single quotes around the answer.
              '),nl,nl,
        write('Type the INVERSE of: '),
        write(C1),nl,nl,
        read(S),
        check64(S,2,C1,C2,I,C3),!.
askques(3,C1,C2,I,C3):-
        write('Please use single quotes around the answer.
              '),nl,nl,
        write('Type the CONTRAPOSITIVE of: '),
        write(C1),nl,nl,
        read(S),
        check64(S,3,C1,C2,I,C3),!.

/*************************************************************/
/* THIS SECTION CHECKS THE ANSWER.  */
check64(help,1,C1,C2,I,C3):-
        helpmenu(1),
        write('Please use single quotes around the answer.
              '),nl,nl,
        write('Type the CONVERSE of :'),
        write(C1),nl,nl,
```

```
        read(S1),
        check64(S1,1,C1,C2,I,C3),!.
check64(help,2,C1,C2,I,C3):-
        helpmenu(1),
        write('Please use single quotes around the answer.
              '),nl,nl,
        write('Type the INVERSE of :'),
        write(C1),nl,nl,
        read(S1),
        check64(S1,1,C1,C2,I,C3),!.
check64(help,3,C1,C2,I,C3):-
        helpmenu(1),
        write('Please use single quotes around the answer.
              '),nl,nl,
        write('Type the CONTRAPOSITIVE of :'),
        write(C1),nl,nl,
        read(S1),
        check64(S1,1,C1,C2,I,C3),!.
check64(S,1,C1,C2,I,C3):-
        checksymb(S,S1),
        checksymb(C2,C),
        S1 == C,
        addtop,
        write('You are correct.'),nl,!.
check64(S,2,C1,C2,I,C3):-
        checksymb(S,S1),
        checksymb(I,I1),
        S1 == I1,
        addtop,
        write('You are correct.'),nl,!.
check64(S,3,C1,C2,I,C3):-
        checksymb(S,S1),
        checksymb(C3,C),
        S1 == C,
        addtop,
        write('You are correct.'),nl,!.
check64(S,1,C1,C2,I,C3):-
        checksymb(S,S1),
        checksymb(I,I1),
        S1 == I1,
        write('Sorry,'),
        write(' you typed the INVERSE.'),nl,nl,
        write('The CONVERSE is '),
        write(C2),nl,nl,!.
check64(S,1,C1,C2,I,C3):-
        checksymb(S,S1),
        checksymb(C3,C),
        S1 == C,
        write('Sorry,'),
        write(' you typed the CONTRAPOSITIVE.'),nl,nl,
```

```
        write('The CONVERSE is '),
        write(C2),nl,nl,!.
check64(S,1,C1,C2,I,C3):-
        write('Sorry,'),
        write(' you were NOT correct.'),nl,
        write('The CONVERSE is '),
        write(C2),nl,nl,!.
check64(S,2,C1,C2,I,C3):-
        checksymb(S,S1),
        checksymb(C2,C),
        S1 == C,
        write('Sorry,'),
        write(' you typed the CONVERSE.'),nl,nl,
        write('The INVERSE is '),
        write(I),nl,nl,!.
check64(S,2,C1,C2,I,C3):-
        checksymb(S,S1),
        checksymb(C3,C),
        S1 == C,
        write('Sorry,'),
        write(' you typed the CONTRAPOSITIVE.'),nl,nl,
        write('The INVERSE is '),
        write(I),nl,nl,!.
check64(S,2,C1,C2,I,C3):-
        write('Sorry,'),
        write(' you were NOT correct.'),nl,
        write('The INVERSE is '),
        write(I),nl,nl,!.
check64(S,3,C1,C2,I,C3):-
        checksymb(S,S1),
        checksymb(C2,C),
        S1 == C,
        write('Sorry,'),
        write(' you typed the CONVERSE.'),nl,nl,
        write('The CONTRAPOSITIVE is '),
        write(C3),nl,nl,!.
check64(S,3,C1,C2,I,C3):-
        checksymb(S,S1),
        checksymb(I,I1),
        S1 == I1,
        write('Sorry,'),
        write(' you typed the INVERSE.'),nl,nl,
        write('The CONTRAPOSITIVE is '),
        write(C3),nl,nl,!.
check64(S,3,C1,C2,I,C3):-
        write('Sorry,'),
        write(' you were NOT correct.'),nl,
        write('The CONTRAPOSITIVE is '),
        write(C3),nl,nl,!.
```

```
/*************************** SYMBPRS6 ********************/

symbol6(1,'a','b','a -> b','b -> a','~a -> ~b','~b -> ~a').
symbol6(2,'b','~c','b -> ~c','~c -> b','~b -> c','c -> ~b').
symbol6(3,'c','d','c -> d','d -> c','~c -> ~d','~d -> ~c').
symbol6(4,'d','~e','d -> ~e','~e -> d','~d -> e','e -> ~d').
symbol6(5,'e','f','e -> f','f -> e','~e -> ~f','~f -> ~e').
symbol6(6,'f','~g','f -> ~g','~g -> f','~f -> g','g ->~f').
symbol6(7,'g','h','g -> h','h -> g','~g -> ~h','~h -> ~g').
symbol6(8,'h','~i','h -> ~i','~i -> h','~h -> i','i -> ~h').
symbol6(9,'i','j','i -> j','j -> i','~i -> ~j','~j -> ~i').
symbol6(10,'j','~k','j -> ~k','~k -> j','~j -> k','k -> ~j').
symbol6(11,'k','l','k -> l','l -> k','~k -> ~l','~l -> ~k').
symbol6(12,'l','~m','l -> ~m','~m -> l','~l -> m','m -> ~l').
symbol6(13,'m','n','m -> n','n -> m','~m -> ~n','~n -> ~m').
symbol6(14,'n','q','n -> q','q -> n','~n -> ~q','~q -> ~n').
symbol6(15,'o','~p','o -> ~p','~p -> o','~o -> p','p -> ~o').
symbol6(16,'p','q','p -> q','q -> p','~p -> ~q','~q -> ~p').
symbol6(17,'q','r','q -> r','r -> q','~q -> ~r','~r -> ~q').
symbol6(18,'r','~s','r -> ~s','~s -> r','~r -> s','s -> ~r').
symbol6(19,'s','t','s -> t','t -> s','~s -> ~t','~t -> ~s').
symbol6(20,'t','u','t -> u','u -> t','~t -> ~u','~u -> ~t').
symbol6(21,'u','~v','u -> ~v','~v -> u','~u -> v','v -> ~u').
symbol6(22,'v','w','v -> w','w -> v','~v -> ~w','~w -> ~v').
symbol6(23,'w','x','w -> x','x -> w','~w -> ~x','~x -> ~w').
symbol6(24,'x','~y','x -> ~y','~y -> x','~x -> y','y -> ~x').
symbol6(25,'y','z','y -> z','z -> y','~y -> ~z','~z -> ~y').
symbol6(26,'z','~y','x -> ~y','~y -> x','~x -> y','y -> ~x').
symbol6(27,'~a','b','~a -> b','b -> ~a','a -> ~b','~b -> a').
symbol6(28,'~b','~c','~b -> ~c','~c -> ~b','b -> c','c -> b').
symbol6(29,'~c','d','~c -> d','d -> ~c','c -> ~d','~d -> c').
symbol6(30,'~d','~e','~d -> ~e','~e -> ~d','d -> e','e -> d').
symbol6(31,'~e','f','~e -> f','f -> ~e','e -> ~f','~f -> e').
symbol6(32,'~f','~g','~f -> ~g','~g -> ~f','f -> g','g -> f').
symbol6(33,'~g','h','~g -> h','h -> ~g','g -> ~h','~h -> g').
symbol6(34,'~h','~i','~h -> ~i','~i -> ~h','h -> i','i -> h').
symbol6(35,'~i','j','~i -> j','j -> ~i','i -> ~j','~j -> i').
symbol6(36,'~j','k','~j -> k','k -> ~j','j -> ~k','~k -> j').
symbol6(37,'~k','l','k -> l','l -> k','~k -> ~l','~l -> ~k').
symbol6(38,'~l','~m','~l -> ~m','~m -> ~l','l -> m','m -> l').
symbol6(39,'~m','n','~m -> n','n -> ~m','m -> ~n','~n -> m').
symbol6(40,'~n','~o','~n -> ~o','~o -> ~n','n -> o','i -> n').
symbol6(41,'~o','p','~o -> p','p -> ~o','o -> ~p','~p -> o').
symbol6(42,'~p','~q','~p -> ~q','~q -> ~p','p -> q','q -> p').
symbol6(43,'~q','r','~q -> r','r -> ~q','q -> ~r','~r -> q').
symbol6(44,'~r','~s','~r -> ~s','~s -> ~r','r -> s','s -> r').
symbol6(45,'~s','t','~s -> t','t -> ~s','s -> ~t','~t -> s').
symbol6(46,'~t','~u','~t -> ~u','~u -> ~t','t -> u','u -> t').
symbol6(47,'~u','v','~u -> v','v -> ~u','u -> ~v','~v -> u').
symbol6(48,'~v','~w','~v -> ~w','~w -> ~v','v -> w','w -> v').
symbol6(49,'~w','x','~w -> x','x -> ~w','w -> ~x','~x -> w').
symbol6(50,'~x','~y','~x -> ~y','~y -> ~x','x -> y','y -> x').
symbol6(51,'~y','z','~y -> z','z -> ~y','y -> ~z','~z -> y').
symbol6(52,'~z','~y','~z -> ~y','~y -> ~z','z -> y','y -> z').
```