Stanford Heuristic Programming Project
Memo HPP-79-17

July 1979

Computer Science Department
Report No. STAN-CS-79-749

Applications-oriented AI Research: Education

by

William J. Clancey, James S. Bennett, and Paul R. Cohen

a section of the

Handbook of Artificial Intelligence

edited by

Avron Barr and Edward A. Feigenbaum

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Applications-oriented AI Research: Education

by

William J. Clancey, James S. Bennett, and Paul R. Cohen

a section of the

Handbook of Artificial Intelligence

edited by

Avron Barr and Edward A. Feigenbaum

This research was supported by both the Defense Advanced Research Projects Agency (ARPA Order No. 3423, Contract No. MDA 903-77-C-0322) and the National Institutes of Health (Contract No. NIH RR-00785-06). The views and conclusions of this document should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, the National Institutes of Health, or the United States Government.

AI Applications in Education

Table of Contents

Foreword

Those of us involved in the creation of the Handbook of Artificial Intelligence, both writers and editors, have attempted to make the concepts, methods, tools, and main results of artificial intelligence research accessible to a broad scientific and engineering audience. Currently, AI work is familiar mainly to its practicing specialists and other interested computer scientists. Yet the field is of growing interdisciplinary interest and practical importance. With this book we are trying to build bridges that are easily crossed by engineers, scientists in other fields, and our own computer science colleagues.

In the Handbook we intend to cover the breadth and depth of AI, presenting general overviews of the scientific issues, as well as detailed discussions of particular techniques and important AI systems. Throughout we have tried to keep in mind the reader who is not a specialist in AI.

As the cost of computation continues to fall, new areas of computer applications become potentially viable. For many of these areas, there do not exist mathematical "cores" to structure calculational use of the computer. Such areas will inevitably be served by symbolic models and symbolic inference techniques. Yet those who understand symbolic computation have been speaking largely to themselves for twenty years. We feel that it is urgent for AI to "go public" in the manner intended by the Handbook.

Several other writers have recognized a need for more widespread knowledge of AI and have attempted to help fill the vacuum. Lay reviews, in particular Margaret Boden's **Artificial Intelligence and Natural Man**, have tried to explain what is important and interesting about AI, and how research in AI progresses through our programs. In addition, there are a few textbooks that attempt to present a more detailed view of selected areas of AI, for the serious student of computer science. But no textbook can hope to describe all of the sub-areas, to present brief explanations of the important ideas and techniques, and to review the forty or fifty most important AI systems.

The Handbook contains several different types of articles. Key AI ideas and techniques are described in core articles (e.g., basic concepts in heuristic search, semantic nets). Important individual AI programs (e.g., SHRDLU) are described in separate articles that indicate, among other things, the designer's goal, the techniques employed, and the reasons why the program is important. Overview articles discuss the problems and approaches in each major area. The overview articles should be particularly useful to those who seek a summary of the underlying issues that motivate AI research.

Eventually the Handbook will contain approximately two hundred articles. We hope that the appearance of this material will stimulate interaction and cooperation with other AI research sites. We look forward to being advised of errors of omission and commission. For a field as fast moving as AI, it is important that its practitioners alert us to important developments, so that future editions will reflect this new material. We intend that the **Handbook of Artificial Intelligence** be a living and changing reference work.

The articles in this edition of the Handbook were written primarily by graduate students in AI at Stanford University, with assistance from graduate students and AI professionals at other institutions. We wish particularly to acknowledge the help from those at Rutgers University, SRI International, Xerox Palo Alto Research Center, MIT, and the RAND Corporation.

The authors of this report, which contains the section of the Handbook on educational applications research, are William Clancey, James Bennett, and Paul Cohen. Others who contributed to or commented on earlier versions of this section include Lee Blaine, John Seely Brown, Richard Burton, Adele Goldberg, Ira Goldstein, Albert Stevens, and Keith Wescourt.

Avron Barr
Edward Feigenbaum

Stanford University
July, 1979

Handbook of Artificial Intelligence

Topic Outline

Volumes I and II

The following sections of the Handbook are still in preparation and will appear in the third volume:

Theorem Proving
Vision
Robotics
Information Processing Psychology
Learning and Inductive Inference
Planning and Related Problem-solving Techniques

A. Historical Overview

Educational applications of computer technology have been under development since the early 1960s. These applications have included scheduling courses, managing teaching aids, and grading tests. The predominant application, however, has involved using the computer as a device that interacts directly with the student, rather than as an assistant to the human teacher. For this kind of application, there have been three general approaches.

The "ad lib" or "environmental approach" is typified by Papert's LOGO laboratory (Papert, 1970), that allowed students more or less free-style use of the machine. Students are involved in programming; it is conjectured that learning problem-solving methods takes place as a side effect of using tools that are designed to suggest good problem-solving strategies to the student. The second approach uses games and simulations as instructional tools; once again the student is involved in an activity--for example, doing simulated genetics experiments--for which learning is an expected side effect. The third computer application in education is computer-assisted instruction (CAI). Unlike the first two approaches, CAI makes an explicit attempt to instigate and control learning (Howe, 1973). This third use of computer technology in education is the focus of the following discussion.

The goal of CAI research is to construct instructional programs that incorporate well-prepared course material in lessons that are optimized for each student. Early programs were either electronic "page-turners" which printed prepared text or drill-and-practice monitors, which printed problems and responded to the student's solutions using prestored answers and remedial comments. In the intelligent CAI (ICAI) programs of the 1970s, course material is represented independently of teaching procedures so that problems and remedial comments can be generated differently for each student. Research today focuses on the design of programs that can offer instruction in a manner that is sensitive to the student's strengths, weaknesses, and preferred style of learning. The role of AI in computer-based instructional applications is seen as making possible a new kind of learning environment.

This overview surveys how AI techniques have been used in research attempting to create intelligent computer-based tutors. In the next article, some design issues are discussed and typical components of ICAI systems are described. Subsequent articles describe some important applications of artificial intelligence techniques in instructional programs.

Frame-oriented CAI Systems

The first instructional programs took many forms, but all adhered to essentially the same pedagogical philosophy. The student was usually given some instructional text (sometimes without using the computer) and asked a question that required a brief answer. After the student responded, he was told whether his answer was right or wrong. The student's response was sometimes used to determine his "path" through the curriculum the sequence of problems he is given (see Atkinson & Wilson, 1969). When the student made an error, the program branched to remedial material.

The *courseware author* attempts to anticipate every wrong response, prespecifying branches to appropriate remedial material based on his ideas about what might be the underlying misconceptions that would cause each wrong response. Branching on the basis of response was the first step toward *individualization of instruction* (Crowder, 1962). This style of CAI has been dubbed ad-hoc, frame-oriented (AFO) CAI by Carbonell (1970b), to stress its dependence on author-specified units of information. (The term "frame" as it is used in this context predates the more recent usage in AI--see Article Representation.B7--and refers to a block or page or unit of information or text.) Design of ad-hoc frames was originally based on Skinnerian stimulus/response principles. The branching strategies of some AFO programs have become quite involved, incorporating the best learning theory that mathematical psychology has produced (Atkinson, 1972; Fletcher, 1975; Kimball, 1973). Many of these systems have been used succesfully and are available commercially.

Intelligent CAI

In spite of the widespread application of AFO CAI to many problem areas, many researchers believe that most AFO courses are not the best use of computer technology:

> In most CAI systems of the AFO type, the computer does little more than what a programmed textbook can do, and one may wonder why the machine is used at all....When teaching sequences are extremely simple, perhaps trivial, one should consider doing away with the computer, and using other devices or techniques more related to the task. (Carbonell, 1970b, pp. 32, 193)

In this pioneering paper, Carbonell goes on to define a second type of CAI that is known today as "knowledge-based" or "intelligent" CAI (ICAI). Knowledge-based systems and the previous CAI systems both have representations of the subject matter they teach, but ICAI systems also carry on a natural language dialogue with the student and use the student's mistakes to diagnose his misunderstandings.

Early uses of AI techniques in CAI were called "generative CAI" (Wexler, 1970), since they stressed the ability to generate problems using a large database representing the subject they taught. (See Koffman & Blount, 1975, for a review of some early generative CAI programs and an example of the possibilities and limitations of this style of courseware.) However, the kind of courseware that Carbonell was describing in his paper was to be more than just a problem generator--it was to be a computer *tutor* that had the inductive powers of its human counterparts. ICAI programs offer what Brown (1977) calls a *reactive learning environment*, in which the student is actively engaged with the instructional system and his interests and misunderstandings drive the tutorial dialogue. This goal was expressed by other researchers trying to write CAI programs that extended the medium beyond the limits of frame selection:

> Often it is not sufficient to tell a student he is wrong and indicate the correct solution method. An intelligent CAI system should be able to make hypotheses based on a student's error history as to where the real source of his difficulty lies. (Koffman & Blount, 1975)

The Use of AI Techniques in ICAI

The realization of the computer-based tutor has involved increasingly complicated computer programs and has prompted CAI researchers to use artificial intelligence techniques. Artificial Intelligence work in natural language understanding, representation of knowledge, and methods of inference, as well as specific AI applications like algebraic simplification, calculus, and theorem proving, have been applied by various researchers toward making CAI programs that are more intelligent and more effective. Early research on ICAI systems focused on *representation* of the subject matter. Benchmark efforts include SCHOLAR, the geography tutor of Carbonell and Collins (see article C1), EXCHECK, the logic and set theory tutors by Suppes et al. (article F7), and SOPHIE, the electronics troubleshooting tutor of Brown and Burton (article C3). The high level of domain expertise in these programs permits them to be responsive in a wide range of problem-solving interactions.

These ICAI programs are quite different from even the most complicated frame-oriented, branching program.

> Traditional approaches to this problem using decision theory and stochastic models have reached a dead end due to their oversimplified representation of learning. . . . It appears within reach of AI methodology to develop CAI systems that act more like human teachers. (Laubsch, 1975)

However, an AI system that is expert in a particular domain is not necessarily an expert *teacher* of the material--"ICAI systems cannot be AI systems warmed over" (Brown, 1977). A teacher needs to understand what the student is doing, not just what he is supposed to do. AI programs often use very powerful problem-solving methods that do not resemble those used by humans; in many cases, CAI researchers borrowed AI techniques for representing subject domain expertise but had to modify them, often making the inference routines *less powerful*, in order to force them to follow human reasoning patterns, so as to better explain their methods to the student, as well as to understand his methods (Smith, 1976; Goldberg, 1973).

In the mid-1970s, a second phase in the development of ICAI tutors has been characterized by the inclusion of expertise in the tutor regarding (a) the student's learning behavior and (b) tutoring strategies (Brown & Goldstein, 1977). AI techniques are used to construct models of the learner that represent his knowledge in terms of "issues" (see article C4) or "skills" (Barr & Atkinson, 1975) that should be learned. This model then controls tutoring strategies for presenting the material. Finally, some ICAI programs are now using AI techniques to explicitly represent these *tutoring strategies*, gaining the advantages of flexibility and modularity of representation and control (Burton & Brown, 1979; Goldstein, 1977; Clancey, 1979a).

References

The best general review of research in ICAI is Brown & Goldstein (1977). Several papers on recent work are collected in a special issue of the **International Journal of Man-Machine Studies**, Volume 11, 1979.

B. Issues in ICAI Systems Design

The main components of ICAI systems are (a) its problem-solving expertise, the knowledge that the system tries to impart to the student, (b) the student model, indicating what the student does and does not know, and (c) tutoring strategies, which specify how the system presents material to the student). (See Self, 1974, for an excellent discussion of the differences and interrelations of the types of knowledge needed in an intelligent CAI program.) Not all of these components are fully developed in every system. Because of the size and complexity of intelligent CAI programs, most researchers tend to concentrate their efforts on the development of a single part of what would constitute a fully usable system. Each component is described briefly below.

The Expertise Module--Representing Domain Knowledge

The "expert" component of an ICAI system is charged with the task of generating problems and evaluating the correctness of student solutions. The CAI system's knowledge of the subject matter was originally envisioned as a huge static database that incorporated all the facts to be taught. This idea was implicit in the early drill-and-practice programs and was made explicit in *generative CAI* (see Article A). Representation of subject matter expertise in this way, using *semantic nets* (Article Representation.B2), has been useful for generating and answering questions involving causal or relational reasoning (Carbonell & Collins, 1973; Laubsch, 1975; and see Articles C1 and C2 on the SCHOLAR and WHY systems).

Recent systems have used *procedural representation* of domain knowledge, for example, how to take measurements and make deductions (see Article Representation.B9). This knowledge is represented as *procedural experts* that correspond to subskills that a student must learn in order to acquire the complete skill being taught (Brown, Burton, & Bell, 1975). *Production rules* (Article Representation.B3) have been used to construct modular representations of skills and problem-solving methods (Goldstein, 1977; Clancey, 1979a). In addition, Brown & Burton (1975) have pointed out that *multiple representations* are sometimes useful for answering student questions and for evaluating partial solutions to a problem (e.g., a semantic net of facts about an electronic circuit and procedures simulating the functional behavior of the circuit). Stevens & Collins (1978) considered an evolving series of "simulation" models that can be used to reason metaphorically about the behavior of causal systems.

It should be noted that not all ICAI systems can actually solve the problems they pose to a student. For example, BIP, the BASIC Instructional Program (Barr, Beard, & Atkinson, 1975), can't write or analyze computer programs: BIP uses sample input/output pairs (supplied by the course authors) to test students' programs. Similarly, the procedural experts in SOPHIE-I could not debug an electronic circuit. In contrast, the production rule representation of domain knowledge used in WUMPUS and GUIDON enables these programs to solve problems independently, as well as to criticize student solutions (Goldstein, 1977, and Clancey, 1979a). Being able to solve the problems, preferrably in all possible ways, correctly and incorrectly, is necessary if the ICAI program is to make fine-grained suggestions about the completion of partial solutions.

An important idea in this connection is that of an *articulate expert* (Goldstein, 1977).

Whereas typical expert AI programs have data structures and processing algorithms that do not necessarily mimic the reasoning steps used by humans and are, therefore, considered "opaque" to the user, an articulate expert for an ICAI system must be designed to enable the explanation of each problem-solving decision that it makes in terms that correspond (at some level of abstraction) to those of a human problem solver. For example, the electronic circuit simulator underlying SOPHIE-I (see Article C3), which is used to check the consistency of a student's hypotheses and to answer some of his questions, is an opaque expert on the functioning of the circuit. It is a complete, accurate and efficient model of the circuit, but its mechanisms are never revealed to the student since they are certainly not the mechanisms that he is expected to acquire. In WEST, on the other hand, while a (compete and efficient) opaque expert is used to determine the range of possible moves that the student could have made with a given roll of the dice, an *articulate* expert, which only models pieces of the game-playing expertise, is used to determine possible causes for less-than-optimal student moves.

ICAI systems are distinguished from earlier CAI approaches by the separation of teaching strategies from the subject expertise to be taught. However, the separation of subject-area knowledge from instructional planning requires a structure for organizing the expertise that captures the difficulty of various problems and the interrelationships of course material. Modeling a student's understanding of a subject is closely related conceptually to figuring out a representation for the subject itself or for the language used to discuss it.

Trees and lattices showing prerequisite interactions have been used to organize the introduction of new knowledge or topics (Koffman & Blount, 1975). In BIP this lattice took the form of a *curriculum net* that related the skills to be taught to example programming tasks that exercised each skill (Barr, Beard, & Atkinson, 1976). Goldstein (1979) called the lattice a *syllabus* in the WUMPUS program and emphasized the developmental path that a learner takes in acquiring new skills. For arithmetic skills used in WEST, Burton & Brown (1976) use levels of *issues*. Issues proceed from the use of arithmetic operators to strategies for winning the game, to meta-level considerations for improving performance. Burton and Brown believe that when the skills are "structurally independent," the order of their presentation is not particularly crucial. This representation is useful for modeling the student's knowledge and coaching him on different levels of abstraction. Stevens, Collins, & Goldin (1978) have argued further that a good human tutor does not merely traverse a predetermined network of knowledge in selecting material to present. Rather, it is the process of ferreting out student misconceptions that drives the dialogue.

The Student Model

The modeling module is used to represent the student's understanding of the material to be taught. Much recent ICAI research has focused on this component. The purpose of modeling the student is to make hypotheses about his misconceptions and suboptimal performance strategies so that the tutoring module can point them out, indicate why they are wrong, and suggest corrections. It is advantageous for the system to be able to recognize alternate ways of solving problems, including the incorrect methods that the student might use resulting from systematic misconceptions about the problem or from inefficient strategies.

Some early frame-oriented CAI systems used mathematical *stochastic learning models*, but this approach failed because it only modeled the probability that a student would give a specific response to a stimulus. In general, knowing the probability of a response is not the same as knowing what a student understands--the former has little diagnostic power (Laubsch, 1975).

Typical uses of AI techniques for modeling student knowledge include (a) simple *pattern recognition* applied to the student's response history and (b) flags in the subject matter semantic net or in the rule base representing areas that the student has mastered. In these ICAI systems, a student model is formed by comparing the student's behavior to that of the computer-based "expert" in the same environment. The modeling component marks each skill according to whether evidence indicates that the student knows the material or not. Carr & Goldstein (1977) have termed this component an *overlay model*--the student's understanding is represented completely in terms of the expertise component of the program (see Article C5).

In contrast, another approach is to model the student's knowledge not as a subset of the expert's, but rather as a perturbation or deviation from the expert's knowledge--a "bug". (See, for example, the SOPHIE and BUGGY systems--Articles C3 and C6.) There is a major difference between the overlay and "buggy" approaches to modelling: In the latter approach it is not assumed that, except for "knowing" less, the student reasons as the expert does; the student's reasoning can be substantially different from expert reasoning.

Other information that might be accumulated in the student model includes the student's preferred modes for interacting with the program, a rough characterization of his level of ability, a consideration of what he seems to forget over time, and an indication of what his goals and plans seem to be for learning the subject matter.

Major sources of evidence used to maintain the student model can be characterized as: (a) implicit, from student problem-solving behavior; (b) explicit, from direct questions asked of the student; (c) historical, from assumptions based on the student's experience; and (d) structural, from assumptions based on some measure of the difficulty of the subject material (Goldstein, 1977). Historical evidence is usually determined by asking the student to rate his level of expertise on a scale from "beginner" to "expert." Early programs like SCHOLAR used only explicit evidence. Recent programs have concentrated on inferring "implicit" evidence from the student's problem-solving behavior. This approach is complicated because it is limited by the program's ability to recognize and describe the strategies being used by the student. Specifically, when the expert program indicates that an inference chain is required for a correct result and the student's observable behavior is wrong, how is the modeling program to know which of the intermediate steps are unknown or wrongly applied by the student? This is the *apportionment of credit/blame problem*; it has been an important focus of WEST research.

Because of inherent limitations in the modeling process, it is useful for a "critic" in the modeling component to measure how closely the student model actually predicts the student's behavior. Extreme inconsistency or an unexpected demonstration of expertise in solving problems might indicate that the representation being used by the program does not capture the student's approach. Finally, Goldstein (1977) has suggested that the modeling process should attempt both to measure whether or not the student is actually learning and to discern what teaching methods are most effective. Much work remains to be done in this area.

The Tutoring Module

The tutoring module of ICAI systems must integrate knowledge about natural language dialogues, teaching methods, and the subject area to be taught. This is the module that communicates with the student: selecting problems for him to solve, monitoring and critici. ng his performance, providing assistance upon request, and selecting remedial material. The design of this module involves issues like "When is it appropriate to offer a hint?" or "How far should the student be allowed to go down the wrong track?"

> These are just some of the problems which stem from the basic fact that teaching is a skill which requires knowledge additional to the knowledge comprising mastery of the subject domain. (Brown, 1977)

This additional knowledge, beyond the representation of the subject domain and of the student's knowledge, is about how to teach.

Most ICAI research has explored teaching methods based on *diagnostic modeling* in which the program debugs the student's understanding by posing tasks and evaluating his response (Collins, 1976; Brown & Burton, 1975; Koffman & Blount, 1975). The student is expected to learn from the program's feedback which skills he uses wrongly, which skills he does not use (but could use to good advantage), etc. Recently, there has been more concern with the possibility of saying just the right thing to the student so that he will realize his own inadequacy and switch to a better method (Carr & Goldstein, 1977; Burton & Brown, 1979; Norman, Gentner, and Stevens, 1976). This new direction is based on attempts to make a bug "constructive" by establishing for the student that there is something inadequate in his approach, and giving enough information so that the student can use what he already knows to focus on the bug and characterize it so that he avoids this failing in the future.

However, it is by no means clear how "just the right thing" is to be said to the student. We do know that it depends on having a very good model of his understanding process (the methods and strategies he used to construct a solution). Current research is focussing on means for representing and isolating the bugs themselves (Stevens, Collins, & Goldin, 1978; Brown & Burton, 1978).

Another approach is to provide an environment that encourages the student to think in terms of debugging his own knowledge. In one BIP experiment (Wescourt and Hemphill, 1978), explicit debugging strategies (for computer programming) were conveyed in a written document and then a controlled experiment was undertaken to see whether this trainging fostered a more rational approach for detecting faulty use of (programming) skills.

Brown, Collins, and Harris (1978) suggest that one might foster the ability to construct hypotheses and test them (the basis of understanding in their model) by setting up problems in which the student's first guess is likely to be wrong, thus "requiring him to focus on how he detects that his guess is wrong and how he then intelligently goes about revising it."

The Socratic method used in WHY (Stevens & Collins, 1977) involves questioning the student in a way that will encourage him to reason about what he knows and thereby modify his conceptions. The tutor's strategies are constructed by analyzing protocols of real-world student/teacher interactions.

Another teaching strategy that has been successfully implemented on several systems is called *coaching* (Goldstein, 1977). Coaching programs are not concerned with covering a predetermined lesson plan within a fixed time (in contrast with SCHOLAR). Rather, the goal of coaching is to develop the acquisition of skill and general problem-solving abilities, and it works by engaging the student in a computer game (see Article A). In a coaching situation, the immediate aim of the student is to have fun, and skill acquisition is an indirect consequence. Tutoring comes about when the computer coach, which is "observing" the student's play of the game, interrupts him and offers new information or suggests new strategies. A successful computer coach must be able to discern what skills or knowledge the student might acquire, based on his playing style, and to judge effective ways to intercede in the game and offer advice. WEST and WUMPUS (Articles C4 and C5) are both coaching programs.

Socratic tutoring and coaching represent different styles for communicating with the student. All mixed-initiative tutoring involves following some dialogue strategy, which involves decisions about when and how often to question the student and methods for presentation of new material and review. For example, a coaching program, by design, is non-intrusive and only rarely lectures. On the other hand, a Socratic tutor questions repetitively, requiring the student to pursue certain lines of reasoning. Recently ICAI research has turned to making explicit these alternative *dialogue management* principles. Collins (1976) has pioneered the careful investigation and articulation of teaching strategies. Recent work has explored the representation of these strategies as *production rules* (see Clancey, 1979a and Article C2 on Collins and Stevens' WHY system).

For example, the tutoring module in the GUIDON program, which discusses MYCIN-like "case diagnosis" tasks with a student (see Clancey, 1979a, and Article C1 on MYCIN), has an explicit representation of discourse knowledge. Tutoring rules select alternative dialogue formats on the basis of economy, domain logic, and tutoring or student modeling goals. Arranged into procedures, these rules cope with various recurrent situations in the tutorial dialogue, for example: introducing a new topic, examining a student's understanding after he asks a question that indicates unexpected expertise, relating an inference to one just discussed, giving advice to the student after he makes a hypothesis about a subproblem, and wrapping up the discussion of a topic.

Conclusion

In general, ICAI programs have only begun to deal with the problems of representing and acquiring teaching expertise and of determining how this knowledge should be integrated with general principles of discourse. The programs described in the articles to follow have all investigated some aspect of this problem, and none offer an "answer" to the question of how to build a computer-tutor. Nevertheless, these programs have demonstrated potential tutorial skill, sometimes often showing striking insight into students' misconceptions. Research continues toward making viable AI contributions to computer-based education.

References

Goldstein (1977) gives a clear discussion of the distinctions between the modules discussed here, concentrating on the broader, theoretical issues. Burton & Brown (1976) also discuss the components of ICAI systems and their interactions and present a good example. Self (1974) is a classic discussion of the kinds of knowledge needed in a computer-based tutor.

C. ICAI Systems

C1. SCHOLAR

An important aspect of tutoring is the ability to generate appropriate questions for the student. These questions can be used by the tutor to indicate the relevant material to be learned, to determine the extent of a student's knowledge of the problem domain, and to identify any misconceptions that he might have. Given that the knowledge base of a tutoring program can't contain all of the "facts" that are true about the domain, the tutor should be able to reason about what it knows and make *plausible inferences* about facts in the domain. In addition to responding to the student's questions, the tutor should be able to take the initiative during a tutoring dialogue by generating good tutorial questions.

SCHOLAR is one such *mixed-initiative* computer-based tutorial system; both the system and the student can initiate conversation by asking questions. SCHOLAR was the pioneering effort in the development of computer tutors capable of coping with unanticipated student questions and of generating subject matter in varying levels of detail, according to the context of the dialogue. Both the student's input and the program's output are in English sentences.

The original system, created by Jaime Carbonell, Allan Collins, and their colleagues at Bolt, Beranek and Newman, Inc., tutored students about simple facts in South American geography (Carbonell, 1970b). SCHOLAR uses a number of *tutoring strategies* for composing relevant questions, determining whether or not the student's answers are correct, and answering questions from the student. Both the knowledge representation scheme (see below) and the tutorial capabilities are applicable to other domains besides geography. For example, NLS-SCHOLAR was developed to tutor computer-naive people in the use of a complex text-editing program (Grignetti, Hausman, & Gould, 1975).

In addition to investigating the nature of tutorial dialogues and human plausible reasoning, the SCHOLAR research project explored a number of AI issues, including:

1. How can real-world knowledge be stored effectively for the fast, easy retrieval of relevant facts needed in tutoring?

2. What general reasoning strategies are needed to make appropriate inferences from the typically incomplete database of the tutor program?

3. To what extent can these strategies be made independent of the domain being discussed (i.e., be dependent only on the form of the representation)?

The Knowledge Base--Semantic Nets

In SCHOLAR, knowledge about the domain being tutored is represented in a *semantic net* (see Article Representation.B2). Each node or "unit" in the net, corresonding to some geographical object or concept, is composed of the name associated with that node and a set of properties. These properties are lists of attribute-value pairs. For example, Figure 1 shows a representation of the unit for Peru:

```
PERU:
    ((EXAMPLE-NOUN PERU))

    (I 0)                             "importance" of unit is high

    (SUPERC (I 0)  COUNTRY)
    (SUPERP (I 6) SOUTH/AMERICA)
                                      link to superordinate units

    (LOCATION (I 0)                   values of LOCATION attribute follow:

         (IN (I 0) (SOUTH/AMERICA (I 0) WESTERN))
         (ON (I 0) (COAST (I 0) (OF (I 0) PACIFIC))
         (LATITUDE (I 4) (RANGE (I 0) -18 0))
         (LONGITUDE (I 5) (RANGE (I 0) -82 -68))
         (BORDERING/COUNTRIES (I 1)
                (NORTHERN (I 1) (LIST COLUMBIA ECUADOR))
                (EASTERN (I 1) BRAZIL)
                          .
                          .
                          .
```

Figure 1. The unit for PERU.

Attributes can be English words (other units) that are defined elsewhere in the net or one of several special attributes such as "SUPRA" for superattribute, "SUPERC" for superconcept or superordinate, "SUPERP" for superpart, or case structure attributes used in parsing (see below). An example of SUPRA might be the fact that "fertile" refers to "soil" and "soil" refers to "topography"; of SUPERP, that Peru is part of South America; of SUPERC, that Peru is a country. Values can also be *importance tags*, like the expressions "(I 0)" after LOCATION in Figure 1 and "(I 1)" after EASTERN: the lower the number, the more important the property. SCHOLAR uses these tags to measure the relevance of a node with respect to the topic under discussion (see below).

Tutorial Dialogue

An important part of the SCHOLAR research has been an emphasis on the nature of tutorial dialogue, in particular the Socratic style of tutoring, where the tutor attempts to diagnose the student's misconceptions and then present material that will force the student to see his own errors (Collins, Warnock, & Passafiume, 1974). Many human tutorial protocols have been analysed to see what strategies human students follow, in order to try to incorporate these strategies in the system (see also Article C2 on the groups continuing research in this direction). The following example is an excerpt from a session with the South American geography tutor. The student's questions and answers are preceded by "**", and annotations are shown in *italics*:

What is the language of Argentina?
** PROBABLY SPANISH

Very good.
Approx what is the area of Argentina?

Student changes topic...

** TELL ME SOMETHING ABOUT PERU

Peru is a country.
It is located in South America
The capital of Peru is Lima.
Now answer the question you didn't answer before.

...and finally answers the question.

** THE AREA OF ARGENTINA IS 1,100,000 SQUARE MILES

Approximately true, but a more exact value is 1,200,000.
The eastern bordering country of Peru is ---.

Student makes an error and is corrected...

** CATHOLICISM

Wrong. Catholicism is not a country but a religion.
The correct answer is: Brazil
Where is Peru?

...and once again changes topic.

** Is Asuncion hot?
Since Asuncion is part of Paraguay, yes.

** WHAT IS THE MOST IMPORTANT DIFFERENCE BETWEEN PERU AND PARAGUAY?
The primary difference is: The population of Peru is larger by a
factor of 7.8.

The natural language processing used by SCHOLAR is based on a *case grammar* system (Article Natural Language.B3b), and the semantic interpretation of a student's response is heavily guided by the geography knowledge in the semantic net. Natural language processing is important in SCHOLAR because it allows the student to take the initiative and ask questions that were not anticipated by the program. However, comprehension of English input is facilitated by a taxonomy of expected "question types." The case grammar is also used by the system to construct questions and presentations of new material from the semantic network. English output is composed of short, simple sentences, with no embedded clauses and a limited repertoire of verbs--generally some form of the verb "to be."

A simple *agenda* is used to keep track of topics that are being discussed. Timing considerations and relevance (importance tags) affect the generation and pruning of topics on this agenda. Continuity between questions is weak, however, since SCHOLAR does not plan a series of questions to make a point. SCHOLAR is capable of diagnosing a student's confusion only by following up one question with a related question.

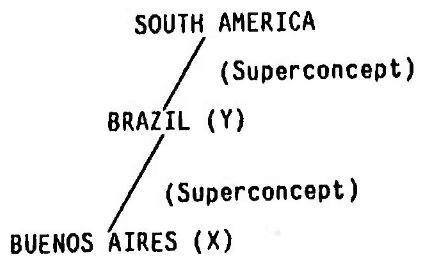
Making Inferences

SCHOLAR's inference strategies, for answering student questions and evaluating student answers to its questions, are designed to cope with the incompleteness of the information stored in the semantic net database. Some of the important strategies used to reason with incomplete knowledge are given below. These abilities have been explored further in current research dealing with default reasoning (Reither, 1978) and plausible reasoning (Collins, 1978).

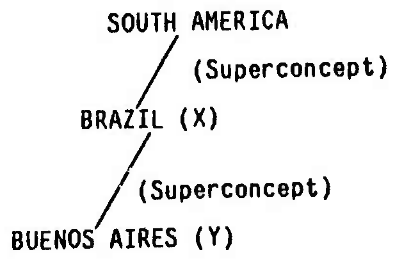
Intersection search. Answering questions of the form "Can X be a Y?" (e.g., "Is Buenos Aires a city in Brazil?") is done by an intersection search: The superconcept (SUPERC) arcs of both nodes for X and Y are traced until an intersection is found (i.e., a common superconcept node is found). If there is no intersection, the answer is "NO." If there is an intersection node Q, SCHOLAR answers as follows:

If Q=Y, then "YES";
If Q=X, then "NO, Y IS AN X."

For example, the question "Is Buenos Aires in Brazil?" is answered YES because Brazil is a SUPERC of Buenos Aires in the net (Q=Y):



But, the question "Is Brazil in Buenos Aires?" gets the response "NO, BRAZIL is a country."



Common superordinate. Otherwise, if Q is not X or Y, the program focuses on the two elements that have Q as a common superordinate. If they are contradictory (contain suitable

CONTRA properties) or have distinguishing, mutually exclusive properties (e.g., different LOCATIONs), the answer is "NO"; otherwise the system answers "I DON'T KNOW." Answering "Is X a part of Y?" questions is similar, except SUPERP (superpart) arcs are used for the intersection process.

Open and closed sets. In order to look for all objects in the system that satisfy some condition (e.g., "How many cities in Columbia are on the Atlantic?"), a distinction must be made about whether the resulting set of these objects is *closed* (explicitly contains all such objects) or *open* (contains some and need not contain all such objects). In SCHOLAR's net, sets are tagged by the course author as either open or closed, thus giving an indication of the *incompleteness* of the system's knowledge.

While it is easy for SCHOLAR to answer questions (such as, "Is France a country in South America?") about closed sets, similar questions about open sets require the use of special techniques--SCHOLAR must be able to reason about the *extent* of its knowledge. For example, if we ask SCHOLAR whether or not rubber is a product of, say, Guyana and if there is no explicit information about Guyanese rubber production in the semantic net, SCHOLAR starts to reason with what it knows: it compares the importance tags of the agricultural products of Guyana with the importance tags for rubber in countries that it knows produce rubber. It happens that Peru is a known rubber producer. SCHOLAR looks at the I-tag of rubber for Peru and compares this tag to the largest (least important) I-tags associated with Guyana. If it finds that these are comparable, SCHOLAR infers that it knows as much about Guyana (up to some large I-tag value) as it knows about Peru. Therefore SCHOLAR infers that it would know about rubber production in Guyana if it were important. Since rubber is not listed for Guyana, SCHOLAR makes the plausible (though uncertain) conclusion that rubber is not produced there. At this point SCHOLAR answers that it doesn't know for certain and gives the information about rubber and Peru.

SCHOLAR's use of knowledge about the extent of its knowledge in this kind of *plausible reasoning* is unique in AI research and represents an application of *meta-level knowledge* (see Representation.Overview).

Summary

The inferencing strategies used by SCHOLAR are independent of the *content* of the semantic net, and are applicable in different domains. The inferences produced are fairly natural; that is, they cope with the incomplete knowledge by employing reasoning processes similar to those that people use. The SCHOLAR project as a whole provides an ongoing environment for research on discourse, teaching strategies, and human plausible reasoning (see Article C2 on recent research, including the WHY system).

References

Carbonell (1970a) is a classic paper, defining the field of ICAI and introducing the SCHOLAR system. Collins (1976) is an illuminating study of human tutorial dialogues. Collins et al. (1975) discusses inference mechanisms, and Collins (1978) reports extended research on human plausible reasoning. Grignetti, Hausman, & Gould (1975) describes NLS-SCHOLAR.

C2. WHY

Recent research by Allan Collins, Al Stevens, and their ICAI research group at Bolt, Beranek and Newman, Inc., has focused on developing computer-based tutors that can discuss complex systems. Their previous research on SCHOLAR (Article C1), a system that tutors facts about South American geography, led them to investigate the nature of tutorial dialogues about subject matter that was not just factual--where the causal and temporal interrelations between the concepts in the domain were of interest and where student's errors could involve not only forgotten facts, but also misconceptions about why processes work the way they do. Stevens & Collins (1977) are building a new system, called WHY, that tutors students in the causes of rainfall, a complex geophysical process that is a function of many interrelated factors; no single factor can be isolated that is both necessary and sufficient to account for rainfall.

In their research on tutorial dialogue of this type, the BBN group has focused on three questions that are central themes throughout ICAI research (Stevens, Collins, & Goldin, 1978):

1. How can a good tutor's use of questions, statements, and examples be characterized? What is the "goal structure" of a Socratic tutor? (See below.)

2. What types of misconceptions do students have? How do tutors diagnose these misconceptions from the errors students make?

3. What are the abstractions and viewpoints that tutors use to explain physical processes?

By analyzing tutorial dialogues between human experts and students, Collins and Stevens identify elements of a theory of tutoring. These are incorporated into a tutorial program, which is then used to find the weak points of the theory for further investigation. The current version of the WHY system is the first of a series of iterations of this sort. The work so far has concentrated on the first topic above, the nature of Socratic tutoring.

Socratic Tutoring Heuristics

Collins (1976) argues that learning to reason about and understand complex processes is best accomplished by dealing with specific problems and cases and trying to generalize from them. Socratic dialogue is especially appropriate for tutoring complex subjects where factors interact and where their interaction accounts for the phenomenon under consideration. In an effort to explicitly model the nature of the Socratic dialogue the current version of the WHY system incorporates 24 *heuristics* which control the student/system interaction. An example heuristic is:

> *If* the student gives as an explanation of causal dependence one or more factors that are not necessary,
>
> *then* pick a counterexample with the wrong value of the factor and ask the student why his causal dependence doesn't hold in that case.

This rule forces the student to consider the necessity of a particular factor. For example, if the student gives rainfall as a reason for growing rice, then the computer-generated counterexample "Why do they grow rice in Egypt, where there isn't much rainfall?" challenges the student's explanation of rice growing. These heuristic rules are designed to facilitate tutorial dialogues in which students must consider combinations of factors that are necessary for rainfall, eliminate irrelevant factors, and attempt to generalize from specific cases to general conditions. (See Collins, 1976, for a complete discussion of the tutoring rules.)

WHY's knowledge base about rainfall is represented in *script*-like data structures, which encode temporal relations in the rainfall process; for example, "First water evaporates from an ocean or sea, then the water-laden air mass is carried over land by winds, which in turn causes the air mass to cool, which causes precipitation." (See Schank & Abelson, 1977, on scripts, as well as Article Representation.B7.) This knowledge representation is adequate for capturing many of the characteristics of a tutorial dialogue, but there are other kinds of knowledge about rainfall that aren't represented here, discussed below.

The following excerpt is from a dialogue with the current version of the WHY system. The system prompts the student to suggest causes of rainfall, to look for prior or intermediate causes, and finally to suggest a general rule. When such a rule is proposed, the system finds a counterexample and forces revision of the rule to account for it. The system proceeds in this way until a rule of sufficient generality is found. The protocol is annotated with the heuristics employed by the system at each juncture in the conversation.

Heuristic 1: Ask for a prediction about a particular case.

Do you think the Amazon jungle has heavy rainfall or light rainfall?
**** HEAVY RAINFALL**

Heuristic 2: Ask for prior causes.

Why does the Amazon jungle have heavy rainfall?
**** BECAUSE ITS NEAR THE ATLANTIC**
Yes, the Amazon jungle is near the equatorial Atlantic.

Heuristic 3: Ask for intermediate causes.

How does that affect rainfall in the Amazon jungle?
**** THE WATER IN THE CURRENT IS WARM**

.
. *(Section Omitted)*
.

Heuristic 4: Ask for subsequent causes.

What happens after the moisture-filled air reaches the land?
**** I DON'T KNOW**

Heuristic 5: Suggest a factor and ask if it is a cause.

Do you think the Andes mountain range has any effect on the rainfall in the Amazon jungle?
** YES

Heuristic 3: Ask for intermediate causes.

How?
** THE PREVAILING WIND HITS THE ANDES AND THE AIR COOLS

Heuristic 3: Ask for intermediate causes.

How does the fact that prevailing wind hits the Andes, lead to air cooling?

.
. *(Section Omitted)*
.

Heuristic 6: Form a general rule for an insufficient cause

Do you think that any place with mountains has heavy rainfall?
** YES

Heuristic 7: Pick a counterexample for an insufficient cause, and Heuristic 2: Ask for prior causes.

Southern California has mountains. Why doesn't Southern California have heavy rainfall?

Current Research

Tutorial goals. One of the shortcomings of the existing system is that it doesn't have long-term "goals" for the tutorial dialogue. Implicit in the tutorial rules is some idea about local management of the interaction, but a global strategy about the tutoring session is absent. Human tutors, however, admit to having goals like "Concentrate on one particular part of the causal structure of rainfall at a time," or "Clear up one misconception before discussing another." Stevens & Collins (1977) set about codifying these goals and strategies for incorporation into the WHY system. They analyzed tutoring protocols in which human tutors commented on what they thought the students did and didn't know, and on why they responded to the students as they did. From this analysis, two top-level goals became apparent:

1. Refine the student's causal structure, starting with the most important factors in a particular process and gradually incorporating more subtle factors.

2. Refine the student's procedures for applying his causal model to novel situations.

Student misconceptions. The top-level goals involve subgoals of identifying and correcting the student's misconceptions. Stevens & Collins (1977) classified these subgoals into five categories corresponding to types of bugs and how to correct them:

> **Factual Bugs.** Dealt with by correcting the student. Teaching facts is not the goal of Socratic tutoring; interrelationships of facts are more important.
>
> **Outside-domain bugs.** Misconceptions about causal structure, which the tutor chooses not to explain in detail. For example, the "correct" relationship between the temperature of air and its moisture-holding capacity is often stated by the tutor as a fact, without futher explanation.
>
> **Overgeneralization.** When a student makes a general rule from an insufficient set of factors (e.g., any place with mountains has heavy rainfall), the tutor will find counterexamples to probe for more factors.
>
> **Overdifferentiation.** When a student counts factors as necessary when they are not, the tutor will generate counterexamples to show that they are not.
>
> **Reasoning bugs.** Tutors will attempt to teach students skills such as forming and testing hypotheses and collecting enough information before drawing a conclusion.

If a student displays more than one bug, human tutors will employ a set of heuristics to decide which one to correct first:

1. Correct errors before omissions.
2. Correct causally prior factors before later ones.
3. Make short corrections before longer ones.
4. Correct low-level bugs (in the causal network) before correcting higher level ones.

Functional relationships. The bugs just discussed are all domain independent, that is, they would occur in tutorial dialogues about other complex processes besides rainfall. But some bugs are the results of specific misconceptions about the functional interrelationships of the concepts of the specific domain. For example, one common misconception about rainfall is that "cooling causes air to rise" (Stevens, Collins, & Goldin, 1978). This is not a simple factual misconception, nor is it domain independent. It is best characterized as an error in the student's functional model of rainfall.

In fact, the script representation used in the WHY system for capturing the temporal and causal relations of land, air, and water masses in rainfall proved inadequate to get at all of the types of student misconceptions. Recent work has investigated a more flexible representation of *functional relationships*, which allows the description of the processes that collectively determine rainfall from multiple viewpoints--e.g., *temporal-causal-subprocess* view captured in the scripts, *functional* viewpoint which emphasizes the roles that different objects play in the various processes (Stevens, Collins, & Goldin, 1978). Misconceptions about rainfall are represented as errors in the student's model of these relationships. A functional relationship has four components: (a) a set of actors, each with a role in the process; (b) a set of factors that affect the process--the factors are all attributes of the actors (e.g., water is an actor in the Evaporation relationship and its temperature is a factor); (c) the result of the process--this is always a change in an attribute of one of the actors; and (d) the relationship that holds between the actors and the result, or how an attribute gets changed. These funtional relationships may be the result of models from other domains that are applied metaphorically to the domain under discussion (Stevens & Collins, 1978).

Summary

The WHY system started as an extension of SCHOLAR by the implementation of rules that characterize Socratic tutoring heuristics. Subsequently, an effort was made to describe the global strategies used by human tutors to guide the dialogue. Since these were directed towards dispelling students' misconceptions, five classes of misconceptions were established, as well as means for correcting them. Many misconceptions are not domain independent and the key to more versatile tutoring lies in continuing research on knowledge representation.

References

The most recent reference on the research reported here is Stevens, Collins, & Goldin (1978). The tutorial rules are discussed fully in an excellent article by Collins (1976). The later work on the goal structure of a tutor is reported in Stevens & Collins, 1977. Finally, recent work on conceptual models and multiple viewpoints of complex systems is discussed in Stevens & Collins (1978).

C3. SOPHIE

SOPHIE (a SOPHIsticated Instructional Environment) is an ICAI system developed by John Seely Brown, Richard Burton, and their colleagues at Bolt, Beranek and Newman, Inc., to explore the objective of a wider range of student initiatives during the tutorial interaction (Brown, Burton, & Bell, 1975). The SOPHIE system provides the student with a learning environment in which he learns problem-solving skills by trying out his ideas, rather than by instruction. The system has a model of the problem-solving knowledge in its domain as well as numerous heuristic strategies for answering the student's questions, criticizing his hypotheses, and suggesting alternative theories for his current hypotheses. SOPHIE enables the student to have a one-to-one relationship with an "expert" who helps him create his own ideas, experiment with these ideas and, when necessary, debug them.

Figure 1 illustrates the component modules of the original SOPHIE-I system (Brown, Rubinstein, & Burton, 1976) and the additional capabilities added for the SOPHIE-II system, discussed later in this article.

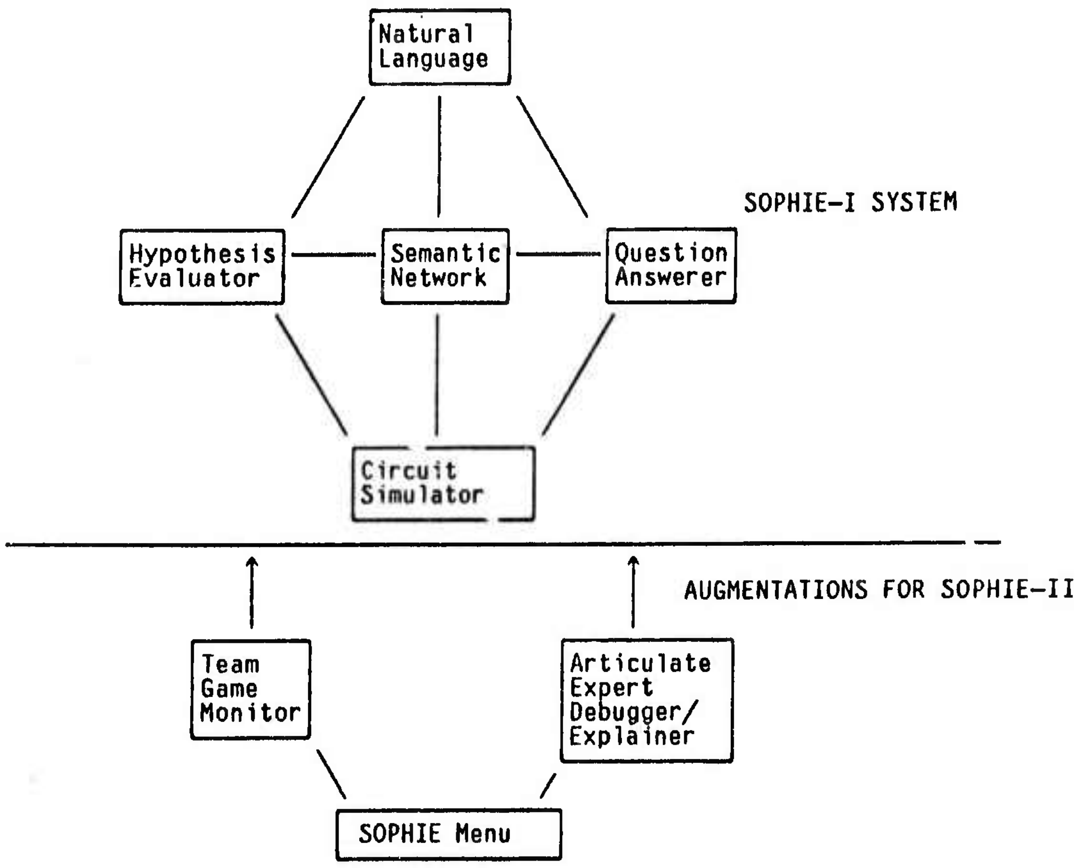


Figure 1. SOPHIE-I and SOPHIE-II.

SOPHIE-I

SOPHIE teaches problem-solving skills in the context of a simulated electronics laboratory. The problem facing the student is to find the "faults" in a malfunctioning piece of equipment whose characteristics have been given to him by taking "measurements." The student obtains a sequence of measurements (such as voltages, currents, resistances, etc.) to determine what is wrong with the equipment and asks questions about the measurements and about what fault hypotheses he should consider, given the information he has discovered so far. SOPHIE evaluates the hypotheses by considering all of the information that the student should have been able to derive from his current set of measurements. The system also judges the merits of a student's suggestion for a new measurement in light of the prior sequence of measurements. SOPHIE can decide if a measurement is valuable, depending on whether it eliminates or isolates a fault, and will inform the student when a proposed measurement provides no new information.

The following portion of a sample protocol illustrates some of the linguistic and inferential capabilities of the SOPHIE system (from Brown & Burton, 1975). SOPHIE randomly selects a fault (in the performance of a circuit component), inserts it into a model or simulation of the circuit, and tells the student how the controls are set. The student is given a schematic diagram of the circuit, and the dialogue refers to components in the schematic diagram (e.g., resistor 8 (R8), node connection 1 (N1), capacitor 5 (C5)). The student begins his troubleshooting by performing measurements in the faulted circuit.

```
Welcome to Sophie, a reactive learning environment. The piece of
equipment you are studying is a regulated power supply. Would you like
to start with a faulted instrument that needs fixing?
** YES

The instrument has a fault and its present control settings are:

        CC      1.0     CURRENT CONTROL
        CR      HIGH    CURRENT RANGE SWITCH
        LOAD    20      LOAD RESISTANCE IN OHMS
        VC      1.0     VOLTAGE CONTROL
        VR      HIGH    VOLTAGE RANGE SWITCH

** WHAT IS THE OUTPUT?

The output voltage is 11.7 volts.
```

Notice that "output" was disambiguated as "output voltage."

```
** WHAT IS IT IN A WORKING INSTRUMENT?

In a working circuit the output voltage is 19.9 volts.
```

In case the student does not know what the required value of measurement should have been, he can ask for any measurement in a properly functioning instrument. The student then requests an internal measurement.

```
** WHAT IS THE VOLTAGE BETWEEN R8 AND Q1?
```

The voltage at N1 is 47.6 volts.

.
.
.

** REPLACE Q6

The student believes that he has discovered the problem (in transistor Q6) and tries to fix it. Before a component is replaced, however, the system queries him as to how he thinks it is faulted.

I am going to ask you some questions about how Q6 is faulted.
Are any junctions shorted?

** I DON'T KNOW

Are any terminals open?

** YES

Which terminal(s) are open?

** THE BASE

Q6 is not faulted in that manner. Q6 not replaced.

After learning that Q6 does not have an open base, the student asks if this was at least a reasonable hypothesis.

** IS IT POSSIBLE THAT Q6 HAS AN OPEN BASE?

Before the system's evaluation procedures can be applied to the student's idea or hypothesis, the hypothesis must be completely specified. To help the student out, the system suggests a plausible complete specification of it in an abbreviated form familiar to the student.

Is the complete fault mode EB/EC/BC = Open/Ok/Open what you intended?

** SURE

That fault is consistent with all of the measurements you have made.

The hypothesis is reasonable, but other measurements will indicate that another component is faulty.

Natural Language Processing--Semantic Grammar

In an instructional environment, it is important that the student be provided with a convenient way in which to communicate his ideas to the system. The student will become quickly frustrated if he has to try several ways of expressing an idea before the system can understand it. SOPHIE's natural language understander copes with various linguistic problems such as *anaphoric references* and context-dependent deletions and *ellipsis*, which occur frequently in natural dialogues.

SOPHIE's natural language capabilities are based on the concept of a *semantic grammar* in which the usual syntactic categories such as noun, verb, and adjective are replaced by semantically meaningful categories (Burton, 1976b, and Burton and Brown, 1979b). These categories represent concepts known to the system--such as "measurements," "circuit elements," "transistors" and "hypotheses." For each concept there is a grammar rule that gives the alternate ways of expressing that concept in terms of its constituent concepts. Each rule is encoded as a LISP procedure that specifies the order of application of the various alternatives in each rule.

A grammar centered around semantic categories allows the parser to deal with a certain amount of "fuzziness" or uncertainty in its understanding of the words in a given statement; that is, if the parser is searching for a particular instantiation of a semantic category, and the current word in the sentence fails to satisfy this instantiation, it skips over that word and continues searching. Thus, if the student uses certain words or concepts that the system doesn't know, the parser can ignore these words and try to make sense of what remains. In order to limit the negative consequences that may result from a misunderstood question, SOPHIE responds to the student's question with a full sentence that tells him what question is being answered. (See Article Natural Language.F7 about the semantic grammar used in the LIFER system).

Inferencing Strategies

In order to interact with the student, SOPHIE performs several different logical and tutorial tasks. First, there is the task of answering *hypothetical questions*. For example, the student might ask, "If the base-emitter junction of the voltage limiting transistor opens, then what happens to the output voltage?"

A second task SOPHIE must perform is that of *hypothesis evaluation*, where the student asks, "Given the measurements I have made so far, could the base of transistor Q3 be open?" The problem here is not to determine if the assertion "the base of Q3 is open" is true, but whether this assertion is logically consistent with the data that have already been collected by the student. If it is not consistent, the program explains why it is not. When it is consistent, SOPHIE identifies which information supports the assertion and which information is independent of it.

A third task that SOPHIE must perform is *hypothesis generation*. In its simplest form this involves constructing all possible hypotheses that are consistent with the known information. This procedure enables SOPHIE to answer questions like, "What could be wrong with the circuit (given the measurements that I have taken)?" The task is solved using the *generate-and-test* paradigm with the hypothesis evaluation task described above performing the "test" function.

Finally, SOPHIE can determine whether a given measurement is *redundant*, that is, if the results of the measurement could have been predicted from a complete theory of the circuit, given the previous measurements.

SOPHIE accomplishes all of these reasoning tasks using an *inference mechanism* that relies principally on a general-purpose *simulator* of the circuit under discussion. For example, to answer a question about a changed voltage resulting from a hypothetical modification to a circuit, SOPHIE first interprets the question with its parser and then, using this interpretation, simulates the desired modification. The result is a Voltage Table that represents the voltages at each terminal in the modified circuit. The original question is then answered in terms of these voltages.

The tasks of hypothesis evaluation and hypothesis generation are handled in a similar manner, using the simulator. When evaluating hypotheses, SOPHIE attempts to determine the logical consistency of a given hypothesis. To accomplish this task, a simulation of the hypothesis is performed on the circuit model and measurements are taken of the result. If the values of any of these measurements are not equivalent to the measurements taken by the student, then a counterexample has been established and it is used to critique the student's hypothesis.

When generating hypotheses, SOPHIE attempts to determine the set of possible faults or hypotheses that are consistent with the observed behavior of the faulted instrument. This task is performed by a set of specialist procedures that propose a possible set of hypotheses to explain a measurement and then simulate them to make sure that they explain the output voltage and all of the measurements that the student has taken. Hypothesis generation can be used to suggest possible paths to explore when the student has run out of ideas for what could be wrong with the circuit or when he wishes to understand the full implications of his last measurement. It is also used by SOPHIE to determine when a measurement is redundant.

SOPHIE-II: The Augmented SOPHIE Lab

Extensions to SOPHIE include: (a) a *troubleshooting game* involving two teams of students and (b) the development of an *articulate expert debugger/explainer*. The simple reactive learning environment has also been augmented by the development of frame-oriented CAI lesson material, used to prepare the student for the laboratory interaction (Brown, Rubinstein, & Burton, 1976). The articulate expert not only locates student-inserted faults in a given instrument but can articulate exactly the deductions that led to its discovery, as well as the more global strategies that guide the trouble-shooting scenario.

Experience with SOPHIE indicates that its major weakness is an inability to follow up on student errors. Since SOPHIE is to be reactive to the student, it will not take the initiative to explore a student's understanding or suggest approaches that he does not consider. However, the competitive environment of the troubleshooting game, in which partners share a problem and work it out together, was found to be an effective means of exercising the student's knowledge of the operation of the instrument being debugged. Finally, an experiment involving a minicourse--and exposure to the frame-based texts, the expert, and the original SOPHIE Lab--indicated that long-term use of the system is more effective than a single, concentrated exposure to the material (Brown, Rubinstein, & Burton, 1976).

Summary

The goal of the SOPHIE project was to create a learning environment in which the student would be challenged to explore ideas on his own and to create conjectures or hypotheses about a problem-solving situation. The student receives detailed feedback as to the logical validity of his proposed solutions. In cases where the student's ideas have logical flaws, SOPHIE can create relevant counterexamples and critiques. The SOPHIE system combines domain-specific knowledge and powerful domain-independent inferencing mechanisms to answer questions that even human tutors might find it extremely difficult to answer.

References

Brown, Burton, & Bell (1975) give a complete description of the early work on SOPHIE, and Brown, Rubinstein, & Burton (1976) report on the later work. Also see Brown & Burton (1975).

C4. WEST

Development of the first *computer coach* was undertaken by Richard Burton and John Seely Brown at Bolt, Beranek and Newman, Inc., for the children's board game called How the West Was Won. The term "coach" describes a computer-based learning environment where the student is involved in an activity, like playing a computer game, and the instructional program operates by "looking over his shoulder" during the game and occasionally offering criticisms or suggestions for improvement (Goldstein, 1977). This research focused on identifying: (a) *diagnostic strategies* required to *infer* a student's misunderstandings from his observed behavior and (b) various explicit *tutoring strategies* for directing the tutor to say the right thing at the right time (Burton & Brown, 1976, and Burton & Brown, 1979). The intention of this work was to use these strategies to control the interaction so that the instructional program took every possible opportunity to offer help to the student without interrupting so often as to become a nuisance and destroy the student's fun at the game. By guiding a student's learning through discovery, computer-based coaching systems hold the promise of enhancing the educational value of the increasingly popular computer-gaming environments.

Philosophy of the Instructional Coach

The pedagogical ideas underlying much of computer coaching research in WEST can be characterized as *guided discovery learning*. It assumes that the student *constructs* his understanding of a situation or a task based on his prior knowledge. According to this theory, the notion of misconception or *bug* plays a central role in the construction process. Ideally, a bug in the student's knowledge will cause an erroneous result in his behavior, which the student will notice. If the student has enough information to determine what caused the error and can then correct it, the bug is referred to as *constructive*. The role of a tutor in an informal environment is to give the student extra information in situations that would otherwise be confusing to him, so that he can determine what caused his error and can transform nonconstructive bugs into constructive ones (see Fischer, Brown, & Burton, 1978 for further discussion).

However, an important constraint on the coach is that it should not interrupt the student too often. If the coach immediately points out the student's errors, there is a danger that the student will never develop the necessary skills for examining his own behavior and looking for the causes of his mistakes himself. The tutor must be perceptive enough to make relevant comments, but not be too intrusive, destroying the fun of the game. The research on the WEST system examined a wide variety of tutorial strategies that must be included to create a successful coaching system.

How the West Was Won

How the West Was Won was originally a computer board game designed by Bonnie Anderson of the Elementary Mathematics Project at the PLATO computer-based education system at the University of Illinois (Dugdale & Kibbey, 1977). The purpose of this original (nontutorial) program was to give elementary-school students drill and practice in arithmetic. The game resembles the popular Chutes and Ladders board game and, briefly, goes something like this: At each turn a player receives three numbers (from spinners) with which he constructs an arithmetic expression using the operations of addition, subtraction, multiplication, and division. The numeric value of the completed expression is the number of spaces the player can move, the object of the game being to get to the end first.

However, the strategy of combining the three numbers to make the biggest valued expression is not always the best strategy, because there are several special features on the game board. Towns occur every ten spaces and if a player lands on one, he skips ahead to the next town. There are also shortcuts, and if he lands on the beginning of one a player advances to the other end of the shortcut. Finally, if the player lands on the space that his opponent is occupying, the opponent is bumped back two towns. The spinner values in WEST are small, so these special moves are encouraged (i.e., landing on towns or shortcuts or on your opponent).

Diagnostic Modeling

There are two major related problems that must be solved by the computer coach. They are (a) when to interrupt the student's problem-solving activity, and (b) what to say once it has been interrupted. In general, solutions to these problems require both techniques for determining what the student knows (procedures for constructing a *diagnostic model*) and explicit tutoring principles about interrupting and advising. These, in turn, require theories about how a student forms abstractions, how he learns, and when he is apt to be most receptive to advice. Unfortunately, few, if any, existing psychological theories are precise enough to suggest anything more than caution.

Since the student is primarily engaged in a gaming or problem-solving activity, diagnosis of his strengths and weaknesses must be unobtrusive to his main activity. This objective means that the diagnostic component cannot use pre-stored tests or pose a lot of diagnostic questions to the student. Instead, the computer coach must restrict itself mainly to inferring a student's shortcomings from what he does in the context of playing the game or solving the problem. This objective can create a difficult problem--just because a student does not use a certain skill while playing a game does not mean that he does not *know* that skill. Although this point seems quite obvious, it poses a serious diagnostic problem: The absence of a potential skill carries diagnostic value if and only if an expert in an equivalent situation would have used that skill. Hence, apart from his outright errors, the main window a computer-based coach has on a student's misconceptions is through a *differential modeling* technique that compares what the student is doing with what the expert would be doing in his place. This difference provides hypotheses about what the student does not know or has not yet mastered. (See the related discussion of *overlay models* in Article C5.)

Constructing the differential model requires that two tasks be performed by the coach, using the computer *Expert* (the subprogram that is expert at playing the game WEST). The first task of the coach is to *evaluate* the student's current move with respect to the set of possible alternative moves that an Expert might have made in the exact same circumstances. The second task is to determine what *underlying skills* were used to select and compose the student's move and each of the "better" moves of the Expert. To accomplish the evaluative task, the Expert need only use the results of its knowledge and reasoning strategies, available as better moves. However, for the second task, the coach has to consider the "pieces" of knowledge involved in move selection and in the generation of better moves, since the absence of one of these pieces of knowledge might explain why the student failed to make a better move.

Tutoring by Issue and Example -- A General Paradigm

One of the top-level goals driving the coach is the objective that its comments be both relevant to the situation and memorable to the student. The *Issues and Examples* tutoring strategy provides a framework for meeting these two constraints. Issues are concepts used in the diagnostic process to identify, at any particular moment, what is relevant. Examples provide concrete instances of these abstract concepts. Providing both the description of a generic issue (a concept used to select a strategy) and a concrete example of its use increases the chance that the student will integrate this piece of tutorial commentary into his knowledge. In the Issues and Examples paradigm, the issues embody the important concepts underlying a student's behavior. They define the space of concepts that the Coach can address--the facets of the student's behavior that are monitored by the Coach.

In WEST, there are three levels of issues on which a Coach can focus: At the lowest level are the basic mathematical skills that the student is practicing (the use of parentheses, the use of the various arithmetic operations, and the form or pattern of the student's move as an arithmentic expression). The second level of issues concerns the skills needed to play WEST (like the special moves: bump, town, and shortcut) and the development of a *strategy* for choosing moves. At the third level are the general skills of game playing (like watching your opponent to learn from his moves), which are not addressed by the WEST program.

Each of the issues is represented in two parts, a *recognizer* and an *evaluator*. The issue recognizer is data-directed; it watches the student's behavior for evidence that he does or does not use a particular concept or skill. The recognizers are used to construct a *model* of the student's knowledge. The issue evaluators are goal-directed; they interpret this model to determine the student's weaknesses. The issue recognizers of WEST are fairly straightforward but are, nevertheless, more complex than simple pattern matchers. For example, the recognizer for the PARENTHESIS issue must determine not only whether or not parentheses are present in the student's expression, but also whether they were necessary for his move, or for an optimal move.

Figure 1 is a diagram of the modeling/tutorial process underlying the issues and Examples paradigm. Figure 1a presents the process of constructing a model of the student's behavior. It is important to observe that without the Expert it is impossible to determine whether the student is weak in some skill or whether the skill has not been used because the need for it has arisen infrequently in the student's experience.

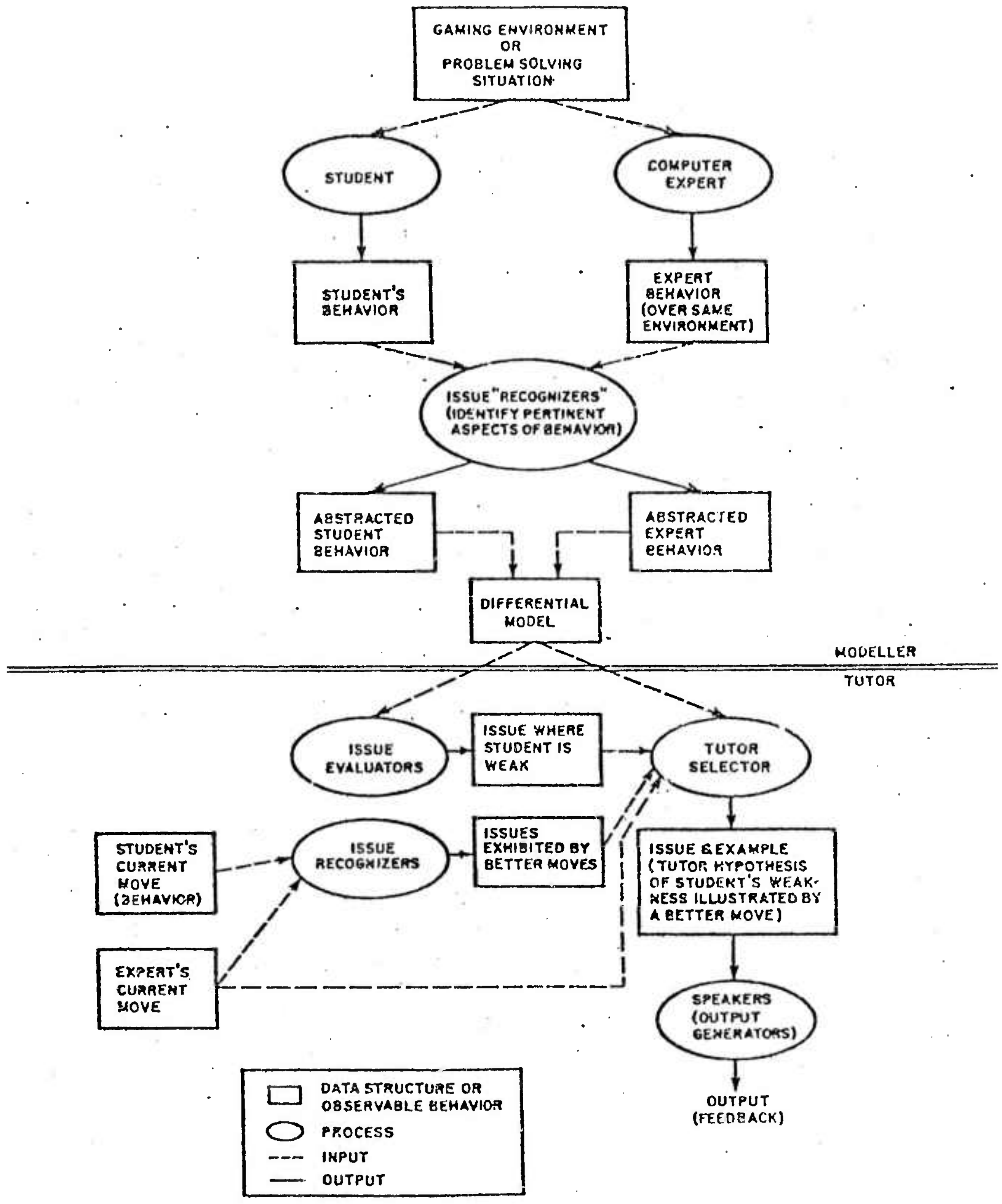

GAMING ENVIRONMENT OR PROBLEM SOLVING SITUATION
STUDENT
COMPUTER EXPERT
STUDENT'S BEHAVIOR
EXPERT BEHAVIOR (OVER SAME ENVIRONMENT)
ISSUE "RECOGNIZERS" (IDENTIFY PERTINENT ASPECTS OF BEHAVIOR)
ABSTRACTED STUDENT BEHAVIOR
ABSTRACTED EXPERT BEHAVIOR
DIFFERENTIAL MODEL
MODELLER
TUTOR
ISSUE EVALUATORS
ISSUE WHERE STUDENT IS WEAK
TUTOR SELECTOR
STUDENT'S CURRENT MOVE (BEHAVIOR)
ISSUE RECOGNIZERS
ISSUES EXHIBITED BY BETTER MOVES
ISSUE & EXAMPLE (TUTOR HYPOTHESIS OF STUDENT'S WEAKNESS ILLUSTRATED BY A BETTER MOVE)
EXPERT'S CURRENT MOVE
SPEAKERS (OUTPUT GENERATORS)
OUTPUT (FEEDBACK)
DATA STRUCTURE OR OBSERVABLE BEHAVIOR
PROCESS
INPUT
OUTPUT

Figure 1. Diagram of the Modeling/Coaching Process

The Coaching Process

Figure 1b presents the top level of the coaching process. When the student makes a less than optimal move (as determined by comparing his move with that of the Expert), the Coach uses the evaluation component of each Issue to create a list of Issues on which it has assessed that the student is weak. From the Expert's list of better moves, the Coach invokes the Issue recognizers, to determine which Issues are illustrated by these better moves. From these two lists of Issues, the Coach selects an Issue and the move that illustrates it (i.e., creates an example of it) and decides, on the basis of *tutoring principles*, whether or not to interrupt. If the two lists have no Issues in common, the reason for the student's problem lies outside the collection of Issues, and the Coach says nothing.

If the Coach decides to interrupt, the selected Issue and Example are then passed to the *explanation* generators, which produce the feedback to the student. Currently, the explanations are stored in a procedures, called *Speakers*, attached to each Issue. Each Speaker is responsible for presenting a few lines of text explaining its Issue. (See also the related discussion of computer coaching in Article C5 on WUMPUS).

Tutoring Principles

General tutoring principles dictate that, at times, even when relevant Issues and Examples have been identified, it may be inappropriate to interrupt. For example, what if there are two competing Issues, both applicable to a certain situation? Which one should be picked? The Issues in WEST are sufficiently independent that there is little need to consider their prerequisite structure, for example, whether the use of parentheses should be tutored before division (but see the description of the *syllabus* in WUMPUS, Article C5). Instead, additional tutoring principles must be invoked to decide which one of the set of applicable Issues should be used.

In WEST, experiments have been conducted using two alternate principles to guide this decision. The first is the Focus Strategy, which ensures that, everything else being equal, the Issue most recently discussed is chosen--the Coach will tend to concentrate on a particular Issue until evidence is present to indicate that it is mastered. The alternative principle is the Breadth Strategy, where Issues that have not recently been discussed tend to be selected. This strategy minimizes a student's boredom and insures breadth of concept coverage.

The rest of WEST's strategies for deciding whether to raise an Issue and what to say can be placed in the four categories listed below, with example rules of each:

1. **Coaching Philosophy.** Tutoring principles can enhance a student's likelihood of remembering what is said. For example, "When illustrating an Issue, use an Example (an alternative move) only when the result or outcome of that move is dramatically superior to the move made by the student."

2. **Maintaining Interest in the Game.** The Coach should not destroy the student's inherent interest in the game by interrupting too often. For example, "Never tutor on two consecutive moves," or "If the student makes an exceptional move, identify why it is good and congratulate him."

3. Increasing Chances of Learning. Four levels of hints are provided by the WEST tutor, at the student's request: (a) isolate a weakness and directly address that weakness, (b) delineate the space of possible moves at this point in the game, (c) select the optimal move and tell why it is optimal, and (d) describe how to make the optimal move.

4. Environmental Considerations. The Coach should consider the game-playing environment. For example, "If the student makes a possibly careless error, one for which there is evidence that he knows better, be forgiving."

Noise in the Model

When the student does not make an optimal move, the program knows only that at least one of the issues required for that move was not employed by the student. Which of these issues blocked the student from making the move is not known. In practice, blame is apportioned more or less equally among all of the issues required for a missed better move. One effect of this apportionment is the introduction of *noise* into the model, that is, blame will almost certainly be apportioned to issues that are, in fact, understood. Also, since the system does not account for the entire process that a person uses to derive a move, the set of issues is, by definition, incomplete. This is the second source of noise in the differential model. A third source of noise in the model is the difficulty of modeling certain human factors such as boredom or fatigue that cause inconsistent behaviors. For example, students are seldom completely consistent. They often forget to use techniques that they know, or get tired and accept a move that is easy to generate but which does not reflect their knowledge.

Another source of noise is inherent in the process of learning. As the student plays the game, he acquires new skills. The student model, which has been accumulating during the course of his play, will not be up to date, that is, it will still show the newly learned issues as "weaknesses." Ideally, the "old pieces" of the model should decay with time. Unfortunately, the costs involved in this computation are prohibitive. To avoid this particular failing of the model, the WEST Coach removes from consideration any issues that the student has used recently (in the last three moves), assuming that they are now part of his knowledge.

To combat the noise that arises in the model, the Evaluator for each issue tends to assume that the student has mastery of the issue. Some coaching opportunities may be missed, but eventually, if the student has a problem addressed by an issue, a pattern will emerge.

Experiences with West

WEST has been used in elementary school classrooms. In a controlled experiment, the coached version of WEST was compared to an uncoached version. The coached students showed a considerably greater variety of patterns, indicating that they had acquired many of the more subtle patterns and had not fallen permanently into "ruts" that prevented them from seeing when such moves were important. Moreover, and perhaps most important of all, the students in the coached group enjoyed playing the game considerably more than the uncoached group (Goldstein, 1979).

References

The most recent and most complete discussion of the WEST coach is Burton & Brown (1979).

C5. WUMPUS

This article describes a *computer coach* for WUMPUS, a computer game in which the player must track down and slay the vicious Wumpus while avoiding pitfalls that result in certain, if fictional, death (Yob, 1975). The coach described here is WUSOR-II, one of three "generations" of computer coaches for WUMPUS developed by Ira Goldstein and Brian Carr at MIT (Carr & Goldstein, 1977). (For discussions of WUSOR-I and -III, see Stansfield, Carr, & Goldstein, 1976, and Goldstein, 1979, respectively.) To be a skilled Wumpus-hunter one must know about logic, probability, decision theory, and geometry. A deficit in one's knowledge may result in being eaten by the Wumpus or falling through the center of the earth. In keeping with the philosophy of computer coaching, students are highly motivated to learn these fundamental skills.

The design of the WUSOR-II system involves the interactions of the specialist programs shown in Figure 1. There are four modules: the Expert, the Psychologist, the Student Model, and the Tutor. The Expert informs the Psychologist of two facts: (a) if the player's move is nonoptimal and (b) which skills are needed for him to discover better alternatives. The Psychologist employs this comparison to formulate hypotheses about which domain-specific skills are known to the student. These hypotheses are recorded in the Student Model, which represents the student's knowledge as a subset of the Expert's skills--an *overlay model* (see Overview B and Carr & Goldstein, 1977). The Tutor uses the student model to guide its interactions with the player. Basically, it chooses to discuss skills not yet exhibited by the player in situations where their use would result in better moves. Goldstein (1977) provides a more detailed discussion of the structure and function of these coaching modules. (Also see the discussion of the WEST computer coach in Article C4.)

The central box of Figure 1 contains a representation for the problem-solving skills of the domain being tutored. It is, in essence, a formal representation of the syllabus. The Expert is derived from the skills represented therein, as is the structure of the student model. The Psychologist derives expectations from this knowledge regarding which skills the student can be expected to acquire next, based on a model of the relative difficulty of items in the syllabus. The Tutor derives relationships *between* skills such as analogies and refinements, which can be employed to improve its explanations of new skills (see Goldstein, 1979).

Theoretical Goals: Toward a Theory of Coaching

The approach to the design of computer coaches in WUSOR-II is to construct *rule-based representation* (see Article Representation.B3) for (a) the skills needed by the Expert to play the game, (b) the modeling criteria used by the Psychologist, and (c) the alternative tutoring strategies used by the Tutor. Each is expanded below:

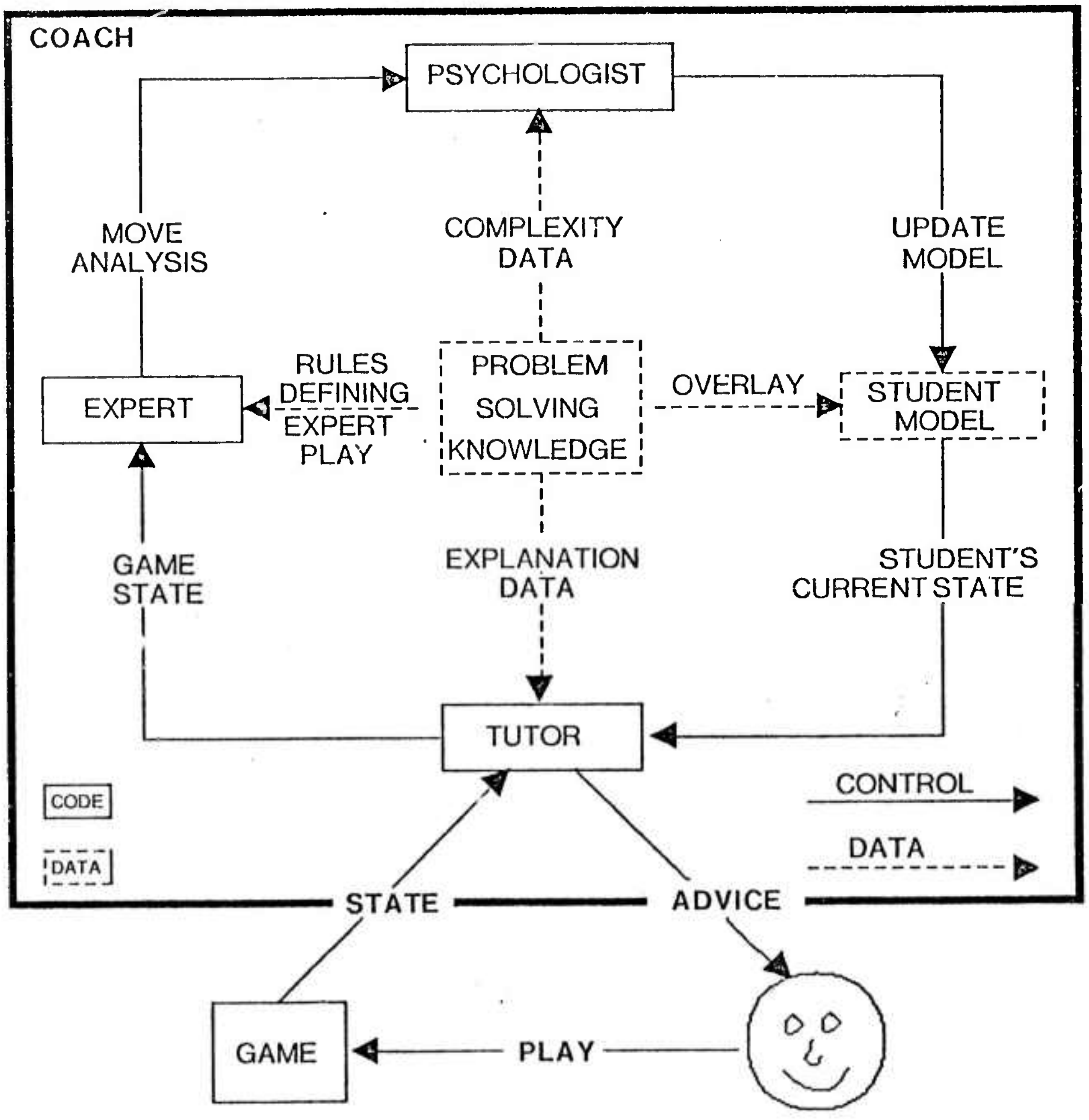


Fig. 1. Simplified block diagram of a computer coach.

The Expert uses rules that embody the knowledge or skills required to play the game to analyze the player's behavior. The virtue of a rule-based representation of expertise is that its modularity both allows tutoring to focus concisely on the discussion of specific skills and permits modeling to take the form of hypotheses regarding which rules are known by the player.

The Psychologist uses *rules of evidence* to make reasonable hypotheses about which of the Expert's skills the player possesses. Typical rules of evidence are:

> Increase the estimate that a player possesses a skill if the player explicitly claims acquaintance with the skill, and decrease the reliability if the player expresses unfamiliarity.
>
> Increase the estimate that a player possesses a skill if the skill is manifest in the player's behavior, and decrease the estimate if the skill is not manifest in a situation where the Expert believes it to be appropriate; hence, implicit as well as overt evidence plays a role.
>
> Decrease the estimate that a player possesses a skill if there is a long interval since the last confirmation was obtained (thereby modeling the tendency for a skill to decay with little use).

The Tutor uses *explanation rules* to select the appropriate topic to discuss with the player and to choose the form of the explanation. These rules include:

> *Rules of simplification* that take a complex statement and reduce it to a simpler assertion. Simplification rules are essential if the player is not to be overwhelmed by the Tutor's explanations.
>
> *Rules of rhetoric* that codify alternative explanation strategies. The two extremes are explanation in terms of a general rule and explanation in terms of a concrete instance.

The WUMPUS Expert

In WUMPUS, the player is initially placed somewhere in a randomly connected warren of caves and told the neighbors of his current location. His goal is to locate the horrid Wumpus and slay it with an arrow. Each move to a neighboring cave yields information regarding that cave's neighbors. The difficulty in choosing a move arises from the existence of dangers in the warren--bats, pits, and the Wumpus itself. If the player moves into the Wumpus's lair, he is eaten. If he walks into a pit, he falls to his death. Bats pick the player up and randomly drop him elsewhere in the warren.

The player can minimize risk and locate the Wumpus by making the proper logistic and probabilistic inferences from warnings that he is given. These warnings are provided whenever the player is in the vicinity of a danger. The Wumpus can be smelled within one or two caves. The squeak of bats can be heard one cave away and the breeze of a pit felt one cave away. The game is won by shooting an arrow into the Wumpus's lair. If the player exhausts his set of five arrows without hitting the creature, the game is lost.

The Wumpus Expert uses a rule-based representation, consisting of approximately 20 rules, to infer the risk of visiting new caves. Five of these rules are shown below:

L1. *Positive Evidence Rule.* A warning in a cave implies that a danger exists in a neighbor.

L2. *Negative Evidence Rule.* The absence of a warning implies that no danger exists in any neighbors.

L3. *Elimination Rule.* If a cave has a warning and all but one of its neighbors are known to be safe, then the danger is in the remaining neighbor.

P1. *Equal Likelihood Rule.* In the absence of other knowledge, all of the neighbors of a cave with a warning are equally likely to contain a danger.

P2. *Double Evidence Rule.* Multiple warnings increase the likelihood that a given cave contains a danger.

A Sample Protocol with the WUSOR-II Computer Coach

A transcript of an interaction with the WUSOR-II coach is illustrated below. The player's responses are preceded by a **.

Hello, Timmy. There are 15 caves, 2 bats, 2 pits, and 1 Wumpus. You are now at cave 15 with neighbors 4, 14 and 0. Brrr! There is a draft. You are near a pit. What a stench! The Wumpus is near. What now?

** 4

You are now at cave 4 with neighbors 15, 14 and 2. Brrr! Squeak! A bat is near. What now?

The goal of the Coach is to tutor a beginner in the relevant logical, probabilistic, and strategic knowledge needed to play the game. For example, the Expert informs the Tutor that cave 14 should be treated as more dangerous than 0 or 2 since there is multiple evidence (from the drafts in 15 and 4) that 14 contains a pit. If the player now moved to cave 14, a coaching situation might occur as follows:

** 14

Timmy, it isn't necessary to take such large risks with pits. One of cave 2 and 14 contains a pit. Likewise one of cave 0 and 14 contains a pit. This is multiple evidence of a pit in cave 14 which makes it quite likely that cave 14 contains a pit. It is less likely that cave 0 contains a pit. Hence, we might want to explore cave 0 instead. Do you want to take back your move?

Although it is not apparent from these simple remarks, every module of the Coach contributed to the dialogue. These contributions are summarized below:

The Expert analyzes all possible moves, using its set of skills. The outcome of its analysis is a ranking of possible moves with an attached list that associates the skills that would be needed to make each move. For example, using the five skills listed earlier, the Expert recognizes that cave 14 is the most dangerous move and cave 0 is the safest move.

Essentially, the Expert provides the following proof for use by the Psychologist and Tutor modules. (The proof is given here in English for readability: the Expert's actual analyses are in the programming language LISP.)

Lemma 1: The Wumpus cannot be in 0, 2, or 14 since there is no smell in 4. (Application of the Negative Evidence Rule, L2, for 2-cave warning of Wumpus.)

Lemma 2: Caves 0 and 2 were better than 14 because there was single evidence that caves 0 and 2 contained a pit, but double evidence for cave 14. (Application of the Double Evidence Rule, P2.)

Lemma 3: Cave 2 is more dangerous than cave 0, since 2 contains a bat, and the bat could drop you in a fatal cave. (We know this fact because the squeak in 4 implied a bat in 14 or 2; but the absence of a squeak in 15 implies no bat in 14. Hence, by Elimination Rule, L3, there is a bat in 2.)

The Psychologist, after seeing Timmy move to cave 14, decreases the Student Model weight indicating familiarity with the Double Evidence Rule, P2, since the Expert's proof indicates that this heuristic was not applied. Table 1 is the Psychologist's hypotheses regarding which skills of the Expert the student possesses.

Table 1.

A Typical Student Model Maintained by the Coach

RULES	APPROPRIATE	USED	PER CENT	KNOWN
L1	5	5	100	Yes
L2	4	3	75	Yes
L3	4	2	50	?
L4	5	5	100	Yes
L5	4	1	25	No

Modeling raises many issues. One subtlety is that the move to 14 above may be evidence of a more elementary limitation--a failure to understand the logical implications of the draft warning--i.e., that a pit is in a neighboring cave. The current state of the Student Model is used by the Psychologist to determine, in the event of a nonoptimal move, which skill is in fact missing. The Student Model indicates the level of play that can be expected from this player--the player might be a beginner with incomplete knowledge of the basic rules of the game, a novice with understanding of the logical skills, an amateur with knowledge of the logical and the more elementary probability skills, etc. The Psychologist would attribute the student's error in the current situation to unfamiliarity with a skill at his current level of play; in this case, Timmy is a player who has mastered the logical skills and is learning the basic probability heuristics. Hence, the coach's explanation focused on explaining the double evidence heuristic.

The Tutor is responsible for abridging the Coach's response to the player's move to cave 14. (The complete explanation generated by the Expert were the three lemmas shown above.) Such pruning is imperative if the Coach is to generate comprehensible advice. Hence, the Tutor prunes the complete analysis on the basis of simplification rules that delete those parts of the argument that are already known to the player on the basis of the Student Model and those portions that are too complex. Here, the coach deleted Lemma 1, the discussion of the Wumpus danger, because it is based on the negative evidence skill that the Student Model attributes to the player. Lemma 2, the elimination argument for bats, is potentially appropriate to discuss; but a simplification strategy directs the Coach to focus on a single skill. Additional information will be given by the Coach if requested by the player.

Conclusions

The novelty of this research is that in a *single system* there is significant domain expertise, a broad range of possible interaction strategies available to the tutor, and a modeling capability for the student's current knowledge state. Informal experience with over 20 players of various ages has shown WUSOR-II to be a helpful learning aid, as judged by interviews with the players. The short-term payoff from this research is an improved understanding of the learning and teaching processes. The long-term payoff is the development of a practical educational technology, given the expected decrease in hardware costs.

References

Carr & Goldstein (1977) describe WUSOR, the overlay model, and related theory. Also see Goldstein (1977), Goldstein (1979), and Stansfield, Carr, & Goldstein (1976).

C6. BUGGY

BUGGY is a program that can accurately determine a student's misconceptions (bugs) about basic arithmetic skills. The system, developed by John Seely Brown, Richard Burton and Kathy Larkin at Bolt, Beranek and Newman, Inc., provides a mechanism for explaining why a student is making an arithmetic mistake, as opposed to simply identifying the mistake. Having a detailed model of a student's knowledge that indicates his misconceptions is important for successful tutoring.

A common assumption among teachers is that students do not follow procedures very well and that *erratic* behavior is the primary cause of a student's inability to perform each step correctly. Brown & Burton (1978) argue that students are remarkably competent procedure followers, but they often follow the wrong procedures. By presenting examples of systematic incorrect behavior, BUGGY allows teachers to practice diagnosing the underlying causes of a student's errors. Using BUGGY, teachers gain experience at forming hypotheses about the relationship between the symptoms of a bug that a student manifests and the underlying misconception. This experience helps teachers become more aware of methods or strategies available for diagnosing their student's problems properly.

Manifesting Bugs

Experience with BUGGY indicates that forming a model of what is wrong with a student's method of performing a task is often more difficult than performing the task itself. Consider, for example, the following addition problems and their (erroneous) solutions. They were provided by a student with a "bug" in his addition procedure:

```
   41      328     989     66       216
  + 9     +917    + 52   +887      + 13
 ----     ----    ----   ----      ----
   50     1345    1141   1053       229
```

Once you have discovered the bug, try testing your hypothesis by simulating the buggy student--predict his results on the following two test problems:

```
   446         201
  +815        +399
  ----        ----
```

The bug is simple. In procedural terms, after determining the carry, the student forgets to reset the "carry register" to zero; he accumulates the amount carried, across the columns. For example, in the student's second problem (328 + 917 = 1345), he proceeds as follows: 8 + 7 = 15 , so he writes 5 and carries 1; 2 + 1 = 3 plus the 1 carried is 4; finally, 3 + 9 = 12 , but the 1 carried from the first column is still there--it has not been reset--so adding it to the final column gives 13. If this is the correct bug, then the answers to the test problems will be 1361 and 700. (This bug is really not so unusual; a child often uses his fingers to remember the carry and might forget to bend them back after each column.)

The model built by BUGGY incorporates both correct and incorrect *subprocedures* that simulate the student's behavior on particular problems and capture what parts of a student's skill are correct and what parts are incorrect. BUGGY represents a skill, such as addition, as a collection of subskills, for example, one of which is knowing how to "carry" a digit into the next column. The subprocedures in BUGGY that correspond to human subskills are linked into a *procedural net* (Sacerdoti, 1974), which is BUGGY's representation of the entire human skill. If all the subprocedures in BUGGY's procedural net for addition work correctly, then BUGGY will do addition problems correctly. On the other hand, replacing correct subprocedures with ones that are faulty will result in systematic errors of the kind shown above. Brown and Burton call a procedural network with one or more faulty subprocedures a *diagnostic model* because it is a way of representing systematic errors. The model has been used in two ways. First, it can diagnose a student's errors and pinpoint the bug(s) in the student's skill. Second, it can help to train a teacher to diagnose student errors by "playing the part" of a student with one or more buggy subskills.

When BUGGY is to diagnose a student's errors, its task is to modify the correct procedural network of, say, subtraction until it accounts for all of the student's answers, both right and wrong. This modification is done by systematically replacing correct subprocedures with incorrect variants until a consistent diagnostic model is found. There are currently 70 primitive faulty subprocedures for subtraction. These are explored exhaustively while attempting to determine a consistent diagnostic model. If a single variant or bug is insufficient to explain a student's behavior, then combinations of two bugs are tried. (Because of the overwhelming number of combinations of three or more bugs, these are not used to form diagnostic models.) In this manner, 330 "bugs" have been identified, each with a bug description. Interactions among bugs and the ramifications of a buggy subprocedure's being called by several high-order procedures constitute major challenges for designing efficient simulations of multiple bugs. Note also that this technique requires a large amount of compute time and is amenable only to domains where bugs can be explicated in a more or less complete way.

The procedural network for subtraction (and its bugs) was tested in an experiment using the results of a test administered to 1,325 students in the fourth, fifth, and sixth grades. Using the BUGGY model, it was found that about 40% of the students demonstrated consistently buggy behavior. That is, BUGGY was able to construct a diagnostic model for student's errors demonstrating that they were using nonrandom, though incorrect procedures.

Using Buggy to Train Teachers

BUGGY can also be used to train teachers to diagnose student errors. This facility is illustrated by a protocol excerpted from Brown (1977) in which BUGGY presents a team of teachers with incorrectly solved math problems and supervises their diagnosis of the bug.

Protocol of a Team Using BUGGY

```
Welcome to Bugland. I have chosen a bug.
Here is an example of the bug.

17
+5
--
13
```

The system presents an example in which the student didn't get the right answer.

```
Now you can give me problems to determine the bug.

** 18    43
** +6   +79
   ---   ---
   15    23
```

The teachers generate problems, and BUGGY solves them incorrectly, following a buggy procedure. The team is trying to diagnose the bug in BUGGY'S addition procedure by presenting BUGGY with problems to solve. However, they are using problems similar to the original one, which is a poor strategy because it may leave other symptoms of the bug undiscovered.

```
** GOT THE BUG!
```

The team thinks that they know what the problem is. Omitted here for space, but shown below, BUGGY asks them to formulate the bug in English. This step has the effect of making the team verbalize the bug clearly. It is a difficult task, and the results are often ambiguous. BUGGY does not "read" the description of the bug, because it does not have the natural language abilities to do so. Instead, it generates a series of problems for the team to solve. The team must solve them incorrectly, using the bug that they think they have identified, to convince BUGGY that they have in fact discovered the bug.

```
Here are some problems to test your theories about the bug.
What is:

  21
 +39
 ---
** 51

That's not the bug I have. Using my bug:
 21
+39
---
 15
Try giving me some more examples...
```

The team fails BUGGY's test by being unable to solve the given problem. That is, their bug is not the same one that BUGGY is using. If they had solved 5 problems "correctly" (that is, incorrectly because of a bug), then BUGGY would credit them with a correct diagnosis. Now, however they must set more problems for BUGGY to solve, in an attempt to deduce the bug.

```
**      51    99     68
**   +1707   +99     +9
     -----   ---     --
        21    36     23
```

```
** GOT THE BUG!

Please describe the bug you think I have:

** THE STUDENT IS ADDING ALL THE DIGITS TOGETHER.

Here are some problems to test your theory about the bug.
What are:

    33   1091     8    28    90
   +99    +60   +34   +70    +6
   ---   ----   ---   ---    --
** 24     17     15    17    15

Very good. My description of the bug is:
The student always sums up all the digits, with no regard to columns.
```

Summary

The central idea of this research is that *procedural networks* can be used to build diagnostic models. This representation scheme facilitates an appropriate decomposition of a skill into subskills, to make explicit the control structures underlying a collection of skills, and to execute directly the resulting *diagnostic model.* Rather than being a subset or simplification of expert rules, as in *overlay modeling* (Carr & Goldstein, 1977; see Articles B, C5 and C4), the procedural network is a model of a student's behavior that is structured in terms of semantically meaningful deviations from the correct procedure. Each subskill, whether correct or incorrect, is explicitly represented as a subprocedure in the network.

References

Brown & Burton (1978) is the most recent and complete report on BUGGY. Also see Brown, Burton, Hausmann, Goldstein, Huggins & Miller (1977) and Brown, Burton, and Larkin (1977).

C7. EXCHECK

EXCHECK is an Intelligent Computer-aided Instruction system designed and implemented by Patrick Suppes and his colleagues at the Institute of Mathematical Studies in the Social Sciences (IMSSS) at Stanford University. It is a general-purpose instructional system used principally to present complete, university-level courses in logic, set theory, and proof theory. In the courses taught using the EXCHECK system, lesson material is presented to the student at his computer terminal, followed by exercises consisting of theorems that he is to prove using the program's theorem prover. The courses are taught on IMSSS's CAI system, which uses computer-generated speech and split-screen displays. Several hundred Stanford students take these courses each year.

From an AI point of view, the most interesting aspects of the EXCHECK system are the procedures and the underlying theories of mathematical reasoning that permit this interaction to take place in a natural style closely approximating standard mathematical practice. These include natural language facilities, natural-deduction-based proof procedures, theorem provers, decision procedures for some simple mathematical theories, procedures for analyzing and summarizing proofs, and procedures for conducting dialogues about some elementary mathematical structures.

Examples of the kind of natural language accepted and generated are given in the proofs and dialogues presented below. The basic logic is a variant of Suppes's (1957) formulation of natural deduction augmented by high-level inference procedures that are the analogs of proof procedures used in standard mathematical practice.

Understanding Informal Mathematical Reasoning

The mathematical reasoning involved in the set theory and proof theory courses is complex and subtle. The fundamental AI problem of EXCHECK is making the program capable of understanding informal mathematical reasoning: The program must be able to follow mathematical proofs presented in a "natural" manner. That is, just as the intent of natural language processing is to handle languages that are actually spoken, the intent of natural proof processing is to handle proofs as they are actually done by practicing mathematicians. In general, such proofs are presented by giving a sketch of the main line of argument along with any other mathematically significant information that might be needed to completely reconstruct the proof. This style should be contrasted with the derivations familiar from elementary logic, where each detail is presented and the focus of attention is on syntactic manipulations rather than on the underlying semantics.

A major aspect of the problem of machine understanding of natural proofs is finding languages that permit users to express their proofs in the fashion described above. Such languages, in turn, must find their basis in an analysis or model of informal mathematical reasoning. Finding these natural proof languages should be compared to the problem of finding high-level "natural" or "English-like" programming languages. For more detailed discussions of these issues, see Blaine & Smith (1977), Smith (1976), and Smith et al. (1975). A simple example of understanding informal mathematical reasoning and fuller discussion of the techniques involved follows.

Student Proof

We present two proofs of the elementary theorem,

Thm: If $A \subset B$ then $\neg(B \subseteq A)$

where "$\subset$" is used for proper subset and "$\subseteq$" is used for subset.

First, the proof is given in the informal style of standard mathematical practice.

We want to show that if $A \subset B$, then $\neg(B \subseteq A)$.

Assume $A \subset B$. We show $\neg(B \subseteq A)$ by indirect proof. Assume that $B \subseteq A$. Since $A \subset B$ then, by definition, $A \subseteq B$ and $A \neq B$. Since $A \subseteq B$ and $B \subseteq A$ then $A = B$. But this is a contradiction and, hence, the assumption that $B \subseteq A$ is false. Therefore, $\neg(B \subseteq A)$.

The following typescript shows how one student did the proof in the EXCHECK system. Input from the student is in **boldface** type and comments are in *italics*; everything else is generated by the program. The program keeps track of the goal that the student is currently trying to establish; the initial goal is the theorem to be proven. EXCHECK indicates acceptance of an inference by returning the top-level prompt **; if a suggested inference is not acceptable, EXCHECK returns an error message.

Derive: If $A \subset B$ then $\neg(B \subseteq A)$

** **hyp** (1) ** $A \subset B$

The hypothesis of the theorem is assumed. The goal is automatically reset to the consequent of the theorem.

** **raa**
assume (2) ** $^G B \subseteq A$

The student begins an indirect proof (the command 'raa' is a mnemonic for **reductio ad absurdum**). *The program assumes the negation of current goal. The goal is now any contradiction.*

** **1definition** Number or Name? ** **proper subset**
1 Df. proper subset
(3) $A \subseteq B$ and $A \neq B$

The definition of proper subset is applied to line 1.

** **2,3establish *B = A**
2,3 Establish
(4) $B = A$

The student asks the theorem prover to check the simple set theoretic inference

```
** 3,4contradiction
Using ** ok
3,4 Contradiction
      (5)  B ¬⊆ A
```

The student indicates that lines 3 and 4 lead to a contradiction. EXCHECK returns the negation of assumption (2).

```
** qed
Correct
```

EXCHECK accepts the derivation.

The following informal *review* printout was generated by the program from the proof given in the above typescript.

```
Derive: If A ⊂ B then ¬(B ⊆ A)

Assume  (1)  A ⊂ B
By raa show: ¬(B ⊆ A)
Assume  (2)  B ⊆ A
From 1, by definition of proper subset,
        (3)  A ⊆ B and A ≠ B
From 2,3 it follows that,
        (4)  A = B
3,4 lead to a contradiction; hence, assumption 2 is false:
        (5)  ¬(B ⊆ A)
```

Natural Inference Procedures

There are no significant structural differences between the detailed informal proof and the student's proof as presented to EXCHECK. The same steps occur in the same relations to each other. Such global or structural fidelity to natural proofs is a major research goal of the EXCHECK project and depends upon the development of *natural inference procedures*. Some of these, such as the HYPOTHESIS and INDIRECT PROOF procedures used in the above proof, are familiar from standard logical systems. The procedure used in the application of the definition of proper subset to line (1) is called IMPLIES. It is used to derive results that, intuitively speaking, follow by applying a previous result or definition. It is considerably more complex than the inference procedures usually found in standard logical systems. An even more complex natural inference procedure used in the above proof is the ESTABLISH procedure. In general, ESTABLISH is used to derive results that are consequences of prior results in the theory under consideration, in this case in the theory of sets. Eliminating the need to cite specific results in the theory, which would disrupt the main line or argument, is important and is discussed further in the section on ESTABLISH, below.

The inference procedures in EXCHECK are intended not only to match natural inferences in strength but also to match them in degree and kind. Howev , there are differences. EXCHECK inference procedures must always be invoked explicitly--in standard practice, particular inference procedures or rules are usually not cited explicitly. For example, compare how the student expresses the inferences that result in lines (3) and (4) with their counterparts in the informal proof. The explicit invocation of inference procedures basically requires that two pieces of information be given: first, the inference procedure to be used; and, second, the previous results to be used--in particular, explicit line numbers must be used.

Explicitness is not disruptive of mathematical reasoning--neither is the reduction of complex inferences to smaller inferences nor the use of explicit line numbers disruptive, in the sense of distracting the student from the main line of the mathematical argument. They are both simple elaborations of the main structure. However, having to think about what inference rule to use can interrupt the main line of argument. The success of a system for interactively doing mathematics depends crucially upon having a few powerful and natural inference procedures with clear criteria of use, which are sufficient to handle all the inferences.

IMPLIES

IMPLIES is used to derive results by applying a previous result or definition as a rule of inference in a given context. This form of inference is probably the most frequent naturally occurring inference. While the basic pattern is simple, the refinements that must be added to the basic form to get a procedure that handles most of the naturally occurring cases result in a computationally complex procedure. The following is a simple example of the basic pattern:

(I) A is a subset of B

I definition (Name or number) *subset

(i) $(\forall x)(x \in A \rightarrow x \in B)$

In this example, the student directed the program to apply the definition of subset to line (i) and IMPLIES generated the result: $(\forall x)(x \in A \rightarrow x \in B)$. While the student thinks he is applying the definition of subset to line (I), the procedure actually invoked is the IMPLIES procedure. It is important to note that in a use of the IMPLIES procedure, the student indicates what axiom, definition, theorem, or line to apply to which lines, and the IMPLIES procedure generates the formula that is the result of the inference.

The IMPLIES procedure seems to correspond closely to naive notions of inference, in that logically unsophisticated but mathematically sophisticated users can use it very well after seeing the basic explanation and a few simple examples. However, the IMPLIES rule does have a fault: It is a purely logical inference procedure and that can occasionally cause problems for users, because mathematicians tend to think in terms of set theoretic rather than logical consequence. (See the discussion of the ESTABLISH rule for more on this distinction.)

ESTABLISH

The following example of a simple use of ESTABLISH is taken from the typescript above.

```
                (2)  B ⊆ A
                (3)  A ⊆ B and A ≠ [illegible]

    *2,3establish *B = A
    2,3 Establish
                (4)  B = A
```

The ESTABLISH rule allows users to simply assert that some formula is an elementary set-theoretic truth or is an elementary set-theoretic consequence of prior results. In the above example, ESTABLISH is used to infer from $A \subseteq B$ and $B \subseteq A$ that $A = B$. $A = B$ is a set-theoretic consequence but not a logical consequence of $A \subseteq B$ and $B \subseteq A$. If ESTABLISH handled only logical consequence, the student would have had to explicitly cite the relevant set theoretic theorems or definitions needed to reduce the inference to a purely logical inference. This is not only disruptive of the line of argument but also difficult to do. Even the most experienced logicians and mathematicians have difficulty ferreting out all the axioms, definitions, and theorems needed to reduce even simple inferences to purely logical inferences.

All of the examples so far are extremely simple if considered in terms of the full capabilities of the ESTABLISH procedure. ESTABLISH uses a theorem prover that can prove about 85% of the first 200 theorems in the set theory course.

Proof Analysis and Summarization

EXCHECK contains procedures that generate informal summaries and sketches of proofs. Such analyses and summaries are useful not only as a semantic basis for the program, to better understand proofs and to better present proofs, but also to give guidance to the student (see the proof summary below for an example of the kind of guidance that can be generated). The summarization procedures analyze the proof by breaking it into parts (or "subproofs") and isolating the mathematically important steps. They also permit a goal-oriented interpretation of the proof where the program keeps track of what is to be established at that point (i.e., the current goal); which lines, terms, etc., are relevant; and how the current line or part fits into the whole structure. MYCIN's consultation explanation system (see article C1) uses a similar approach. Goldstein (1977) also uses summarization techniques in the rhetorical modules of the WUMPUS coach (article C5).

The summaries presented below were generated by EXCHECK from a student proof of the Hausdorff maximal principle. The original line numbers have been retained (in parentheses) in order to give a sense of how much of the proof has been omitted in the summary. In the first summary only the top-level part of the proof is presented; the proofs of its subparts are omitted. Also, all mathematically or logically insignificant information is omitted. In these proofs and summaries "D contains E " is synonymous with "$E \subseteq D$". Also, C is a chain iff both C is a set of sets, and given any two elements of C, at least one is a subset of the other.

```
Derive: If A is a family of sets then
  every chain contained in A is contained in some maximal chain in A

Proof:
        Assume (1) A is a family of sets
           Assume (2) C is a chain and C ⊆ A
           Abbreviate: {B: B is a chain and C ⊆ B and B ⊆ A}
                     by: C!chains
           By Zorn's lemma,
                 (23) C!chains has a maximal element
           Let B be such that
                 (24) B is a maximal-element of C!chains
           Hence,
                 (25) B is a chain and C ⊆ B and B ⊆ A
           It follows that,
                 (31) B is a maximal chain in A
           Therefore,
                 (32) C is contained in some maximal chain in A
```

Figure 1. Informal summary of a proof of the Hausdorff maximal principle.

The summary above is not the only one that could be generated; it essentially presents only the main part of the proof. Subparts of the main part could have been included or even handled independently if so desired.

The proof analysis and summarization procedures will also generate the following kind of summary, which is an attempt to sketch the basic idea of the proof.

```
Derive: If A is a family of sets then
  every chain contained in A is contained in some maximal chain in A

Proof:
  Use Zorn's lemma to show that

          {B: B is a chain and C ⊆ B and B ⊆ A}

contains a maximal element B. Then show that B is a maximal chain in
A which contains C.
```

Figure 2. An example summarization.

The summarization in Figure 2 was obtained from that in Figure 1 by tracing backwards the history of the maximal chain in A that contains C. That is, the general form of the theorem to be proven is $(\exists x)FM(x)$, which is proven by showing $FM(t)$ for some term t. Usually, in proofs of this form, the most important piece of information is the term t. Tracing backwards in this particular proof yields that there are two terms involved. The first is the set of all chains in A containing C, and the second is any maximal element of the set of all chains in A containing C.

Elementary Exercises and Dialogs

Another form of reasoning done by students is the solution of problems. A great many problems in elementary mathematics take the form of asking the student to give finite objects satisfying certain conditions. For example, given the finite sets A and B the student might be asked to give a function F that is a bijection (i.e., 1-1 and onto) from A to B. For a large class of such problems there are programs that will generate a tree of formulas and other information from the original statement of the problem. We call such trees *verification trees* for the problem. Essentially, the verification tree for a problem constitutes a reduction of the original (usually not directly verifiable) condition to a collection of directly verifiable conditions (the formulas at the leaves). These trees have the property that the failure of the formula at a node in the tree explains the failure of formulas at any of its ancestors. Similarly, the failure of a formula at a node is explained by the failure of formulas at any of its descendants.

For example, in the above problem of supplying a bijection F from A onto B, suppose that the student forgets to specify a value for some element of A, say, 3. The first response to the student might be: "The domain of F isn't A." The student might then ask: " Why?" The program would then answer (going towards the leaves), "Because there is an element of A that has not been assigned a value in B." The student might then ask, "Which one?" Since the routines that evaluate the formulas at the leaves provide counterexamples if those formulas fail, the program could then respond, "3." Or going back to the first response by the program ("The domain of F isn't A"), the student might say, "So?" The program could then move a step towards the root (the original statement of the conditions) and say, "Then F is not a map from A into B." The student might then again say, "So?", to which the program could respond, "F is not a bijection from A onto B."

The highly structured information in the verification tree provides the semantic base for a dialogue with the student in which the program can explain to the student what is wrong with the answer. It should be noted that more complex forms of explanation are available. In particular, the program could have said at the beginning that, "Because 3 is not given a value by F, the domain of F is not A and hence F is not a bijection from A onto B."

Summary

A primary activity in mathematics is finding and presenting proofs. In the EXCHECK system an attempt is made to handle natural proofs--proofs as they are actually done by practicing mathematicians--instead of requiring that these proofs be expressed as derivations in an elementary system of first order logic. This objective requires the analysis of inferences actually made and the design and implementation of languages and procedures that permit such inferences to be easily stated and mechanically verified. Some progress has been made in handling natural proofs in elementary mathematics, but there is a considerable amount of work yet to be done.

References

See Blaine & Smith (1977), Smith et al. (1975), Smith & Blaine (1976), Suppes (1957), and Suppes (1960).

References

Atkinson, R. C. Ingredients for a theory of instruction. **American Psychologist**, 1972, 27, 921-931.

Atkinson, R. C., & Wilson, H. A. (Eds.) **Computer-assisted instruction**. New York: Academic Press, 1969.

Barr, A., & Atkinson, R. C. **Adaptive instructional strategies**. Paper presented at the IPN **Symposium 7: Formalized Theories of Thinking and Learning and their Implications for Science Instruction**, Kiel, September 1975.

Barr, A., Beard, M., & Atkinson, R. C. A rationale and description of a CAI program to teach the BASIC programming language. **Instructional Science**, 1975, 4, 1-31.

Barr, A., Beard, M., & Atkinson, R. C. The computer as a tutorial laboratory: The Stanford BIP project. **International Journal of Man-Machine Studies**, 1976, 8, 567-596.

Blaine, L. H., & Smith, R. L. Intelligent CAI: The role of curriculum in suggesting computational models of reasoning. **Proceedings: 1977 Annual Conference**, ACM, Seattle, 1977.

Brown, J. S., Rubinstein, R., & Burton, R. **Reactive Learning Environment for Computer Assisted Electronics Instruction** (BBN Report No. 3314). Cambridge: Bolt, Beranek, & Newman, 1976.

Brown, J. S. Uses of artificial intelligence and advanced computer technology in education. In R. J. Seidel & M. Rubin (Eds.), **Computers and Communications: Implications for Education**. New York: Academic Press, 1977.

Brown, J. S., & Burton, R. Multiple Representations of Knowledge for Tutorial Reasoning. In D. G. Bobrow & A. Collins (Eds.), **Representation and Understanding: Studies in Cognitive Science**. New York: Academic Press, 1975. Pp. 311-349.

Brown, J. S., & Burton, R. R. Diagnostic models for procedural bugs in basic mathematical skills. **Cognitive Science**, 1978, 2(2), 155-192.

Brown, J. S., Burton, R. R., & Bell, A. G. Sophie: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI). **International Journal of Man-Machine Studies**, 1975, 7.

Brown, J. S., Burton, R. R., Hausmann, C., Goldstein, I., Huggins, B., & Miller, M. **Aspects of a theory for automated student modelling** (BBN Report No. 3549). Cambridge, Mass.: Bolt, Beranek and Newman, 1977.

Brown, J. S., Burton, R. R., & Larkin, K .M. Representing and Using Procedural Bugs for Educational Purposes. **Proceedings of the Annual Conference of the Association for Computing Machinery**, Seattle, Oct. 1977, 247-255.

Brown, J. S., Burton, R. R., Miller, M., deKleer, J., Purcell, S., Hausmann, C., & Bobrow, R. **Steps toward a theoretical foundation for complex, knowledge-based CAI** (BBN Report No. 3135). Cambridge, Mass.: Bolt, Beranek and Newman, 1975.

Brown, J. S., Collins, A., & Harris, G. Artificial Intelligence and Learning Strategies. In H. O'Neil (Ed.), **Learning Strategies**. New York: Academic Press, 1978.

Brown, J. S., & Goldstein, I. P. **Computers in a Learning Society**, Testimony for the House Science & Technology Subcommittee on Domestic and International Planning, Analysis, & Cooperation, October 1977.

Burton, R. R. **Semantic grammar: An engineering technique for constructing natural language understanding systems** BBN Report 3453. Cambridge, Mass.: Bolt, beranek and Newman, December, 1976. (b)

Burton, R. R., & Brown, J. S. A Tutoring and Student Modelling Paradigm for Gaming Environments. **Proc. for the Symposium on Computer Science and Education**, Irvine, CA, February 1976. (Also, **SIGCSE Bulletin**, 1976, **8**, 236-246.).

Burton, R. R., & Brown, J. S. An investigation of computer coaching for informal learning activities. **International Journal of Man-Machine Studies**, 1979, 11, 5-24. (a)

Burton, R. R., & Brown, J. S. Toward a natural-language capability for computer-assisted instruction. In H. O'Neil (Ed.), **Procedures for Instructional Systems Development**. New York: Academic Press, 1979. Pp. 273-313. (b)

Carbonell, J. R. AI in CAI: An artificial intelligence approach to computer-aided instruction. **IEEE Transactions on Man-machine Systems**, 1970, MMS-11(4), 190-202. (a)

Carbonell, J. R. **Mixed-Initiative Man-computer Instructional Dialogues** (BBN Rep. No. 1971). Cambridge, Mass.: Bolt, Beranek, & Newman, 1970. (b)

Carbonell, J. R., & Collins, A. Natural Semantics in Artificial Intelligence. **IJCAI 3**, 1973, 344-351.

Carr, B., & Goldstein, I. **Overlays: A theory of modeling for computer aided instruction**, AI Memo 406, Massachusetts Institute of Technology, Cambridge, Mass., 1977.

Clancey, W. Tutoring rules for guiding a case method dialogue. **International Journal of Man-Machine Studies**, 1979, 11, 25-49. (a)

Clancey, W. Dialogue management for rule-based tutorials. **IJCAI 6**, 1979. (b)

Collins, A. Processes in Acquiring Knowledge. In R. C. Anderson, R. J. Spiro, & W. E. Montague (Eds.) **Schooling and the Acquisition of Knowledge**. Hillsdale, N.J.: Erlbaum Assoc., 1976. Pp. 339-363

Collins, A. Fragments of a Theory of Human Plausible Reasoning. **TINLAP-2**, 1978, 194-201.

Collins, A., Warnock, E. H., Aiello, N., & Miller, M. L. Reasoning from Incomplete Knowledge. In D. G. Bobrow & A. Collins, Representation and Understanding. New York: Academic Press, 1975. Pp. 383-415.

Collins, A., Warnock, E. H., & Passafiume, J. J. Analysis and synthesis of tutorial dialogues (BBN Report 2789). Cambridge, Mass.: Bolt, Beranek, & Newman, 1974.

Crowder, N. A. Intrinsic and extrinsic programming. In J. E. Coulson (Ed.), Proceedings of the conference on application of digital computers to automated instruction, New York: Wiley, 1962, 58-55.

Dugdale, S., & Kibbey, D. Elementary mathematics with PLATO. Urbana, IL: University of Illinois (Computer-based Education Research Laboratory), July 1977.

Fischer, G., Brown, J. S., & Burton, R. R. Aspects of a theory of simplification, debugging, and coaching. Proceedings of the 2nd Annual Conf. of Canadian Society for Computational Studies of Intelligence, July 1978.

Fletcher, J. D. Modeling the learner in computer-assisted instruction. Journal of Computer-Based Instruction, 1975, 1, 118-126.

Goldberg, A. Computer-assisted instruction: The application of theorem-proving to adaptive response analysis (Tech. Rep. 203). Stanford, CA: Stanford University, Institute for Mathematical Studies in the Social Sciences, 1973.

Goldstein, I. The Computer as Coach: An athletic paradigm for intellectual education, AI Memo 389, Massachusetts Institute of Technology, Cambridge, Mass., 1977.

Goldstein, I. The genetic epistemology of rule systems. International Journal of Man-Machine Studies, 1979, 11, 51-77.

Goldstein, I., & Papert, S. Artificial intelligence, language, and the study of knowledge. Cognitive Science, 1977, 1(1), 84-123.

Grignetti, M. C., Hausmann, C., & Gould, L. An *intelligent* on-line assistant and tutor--NLS-SCHOLAR. Proceedings of the National Computer Conference, San Diego, Calif., 1975, pp. 775-781

Groen, G. J. The theoretical ideas of Piaget and educational practice. In P. Suppes (Ed.), Impact of research on education: Some case studies. Washington, D.C.: National Academy of Education, 1978. Pp. 267-317.

Hart, R. O., & Koffman, E. B. A Student Oriented Natural Language Environment for Learning LISP. IJCAI 4, 1976, 391-396.

Howe, J. A. M. Individualizing computer-assisted instruction. In A. Elithorn & D. Jones (Eds.) Artificial and human thinking. Amsterdam: Elsevier, 1973. Pp. 94-101.

Kimball, R. B. **Self-optimizing computer-assisted tutoring: Theory and practice** (Tech. Rep. 206). Stanford, Calif.: Stanford University, Institute for Mathematical Studies in the Social Sciences, 1973.

Koffman, E. B., & Blount, S. E. Artificial Intelligence and automatic programming in CAI. **Artificial Intelligence**, 1975, 6, 215-234.

Laubsch, J. H. Some Thoughts about Representing Knowledge in Instructional Systems. IJCAI 4, 1975, 122-125.

Miller, M. L., & Goldstein, I. Problem solving grammars as formal tools for intelligent CAI. **Proc. of the Fall Conference of the Association for Computing Machinery**, Seattle, October, 1977.

Miller, M. L. A structured planning and debugging environment for elementary programming. **International Journal of Man-Machine Studies**, 1979. In press.

Norman, D. A., Gentner, D. R., & Stevens, A. L. Comments on learning schemata and memory representation. In D. Klahr (Ed.), **Cognition and Instruction**. Hillsdale: Erlbaum Associates, 1976.

Papert, S. Teaching children programming. **IFIP Conference on Computer Education**. Amsterdam: North Holland, 1970.

Reither, R. On Reasoning by Default. **TINLAP-2** 1978, 210-218.

Ruth, G. **Analysis of algorithm implementations** (MAC TR-130). Cambridge, Mass.: Massachusetts Institute of Technology, 1974.

Sacerdoti, E. D. Planning in a hierarchy of abstraction spaces. **Artificial Intelligence**, 1974, 5, 115-135.

Schank, R. C., & Abelson, R. P. **Scripts, Plans, Goals, and Understanding**. Hillsdale, N.J.: Lawrence Erlbaum, 1977.

Self, J. A. Student models in computer-aided instruction. **International Journal of Man-Machine Studies**, 1974, 6, 261-276.

Smith, R. L. **Artificial Intelligence in CAI**. Unpublished working paper, IMSSS, Stanford University, 1976.

Smith, R. L., & Blaine, L. H. A generalized system for university mathematics instruction. **SIGCUE Bulletin**, 1976, 8(1), 280-288.

Smith, R. L., Graves W. H., Blaine, L. H., & Marinov, V. G. Computer-assisted axiomatic mathematics: Informal rigor. In O. Lacarme & R. Lewis (Eds.), **Computers in education**, IFIP (Part 2). Amsterdam: North-Holland, 1975. Pp. 803-809.

Stansfield, J. L., Carr, B. P., & Goldstein, I. P. **WUMPUS Advisor I: A First implementation of a program that tutors logical and probabilistic reasoning skills**, MIT AI Memo 381, October 1976.

Stevens, A. L., & Collins, A. **The Goal Structure of a Socratic Tutor** (BBN Rep. No. 3518). Cambridge, Mass.: Bolt, Beranek, & Newman, 1977.

Stevens, A. L., & Collins, A. Multiple Conceptual Models of a Complex System (BBN Rep. No. 3923). Cambridge, Mass.: Bolt Beranek & Newman, 1978. To appear in R. Snow, P. Federico, and W. Mantague (Eds.), Aptitude, learning and instruction: Cognitive **Process Analysis**, forthcoming.

Stevens, A. L., Collins, A., & Goldin, S. Diagnosing Student's Misconceptions in Causal Models (BBN Rep. No. 3786). Cambridge, Mass.: Bolt, Beranek, & Newman, 1978.

Suppes, P. Introduction to logic. New York: Van Nostrand Reinhold, 1957.

Suppes, P. **Axiomatic set theory**. New York: Van Nostrand, 1960. (Slightly rev. ed. publ. by Dover, New York, 1972)

Suppes, P., & Morningstar, M. Computer-assisted instruction at Stanford, 1966-68: **Data, models, and evaluation of the arithmetic programs**. New York: Academic Press, 1972.

Wescourt, K. T., & Hemphill, L. Representing and teaching knowledge for troubleshooting/debugging. IMSSS Tech. Report No. 292, Stanford University, 1978.

Wexler, J. D. Information networks in generative computer-assisted instruction. IEEE Trans. Man-Machine Systems, 1970, 11, 181-190.

Yob, G. Hunt the Wumpus. **Creative Computing**, Sept.-Oct. 1975, pp. 51-54.

Index