EXPLORATIONS OF PROGRAMMING LEARNING BEHAVIOUR OF NOVICES THROUGH COMPUTER-AIDED LEARNING

by

CHEE-KIT LOOI

BSc, Nanyang University, Singapore, 1980

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

Department Of Computer Science

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

March 1984

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date April 9, 1984

Abstract

The goal of Computer-Aided Instruction (CAI) research is to build instructional programs that incorporate well-prepared course material in lessons that attempt to individualize learning. The role of Artificial Intelligence (AI) is to facilitate a new kind of learning environment that stresses a learner-based paradigm instead of the teacher-based paradigm of traditional CAI. Research in Intelligent Computer-Aided Learning (ICAL) is focused on providing instruction that is sensitive to the student's strengths, weaknesses, and preferred style of learning.

In this thesis, research milestones in ICAL are discussed. An Interactive Computer-aided Testing program that seeks to diagnose novices' misconceptions of the assignment statement in Pascal is described. This program was also used to explore the utility of providing explicit models as an aid to learning programming.

Table of Contents

List of Tables

List of Figures

Acknowledgement

I would like to thank Dr. Richard Rosenberg for his invaluable guidance and perspective in supervising my research. Most of the material in Chapter 2 is based on a paper written for CPSC 522 and I would like to thank Dr. Alan Mackworth for his most helpful criticisms of that paper and for reading this thesis.

Chapter 1. Introduction

The goal of Computer-Aided Instruction (CAI) research is to build instructional programs that incorporate well-prepared course material in lessons that attempt to individualize learning. 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. The 1970's saw the evolution of a new generation of CAI programs that represented course material independently of teaching procedures, so that problems and remedial comments could be generated differently for each student. Research in these Intelligent Computer-Aided Learning (ICAL) programs is focused on providing instruction that is sensitive to the student's strengths, weaknesses, and preferred style of learning. The role of Artificial Intelligence (AI) in ICAL is to make possible a new kind of learning environment that stresses a learner-based paradigm instead of the teacher-based paradigm of traditional CAI.

An ICAL system generally incorporates three models : a domain model, a teaching model and a student model. Much research work has been done on the student model which makes hypotheses about a student's misconceptions and suboptimal strategies so that the teaching model can point them out, indicate why they are wrong, and suggest corrections.

In this thesis, we wish to specify a small domain of interest and write a CAL program that seeks to diagnose

misconceptions in a student's understanding of that domain by attempting some limited use of AI techniques. We choose as the knowledge domain the concept of the assignment statement in the Pascal programming language.

We begin in Chapter 2 with a broad survey of the research milestones in ICAL which will provide a broad framework for the rest of the thesis. In Chapter 3, we discuss the motivations for writing our instructional program which we call Computer-Aided Testing, Evaluation and Advice (CATEA). Central to the diagnosis model, we enumerate the possible misconceptions novices may have with the assignment statement. In Chapter 4, we discuss the design and implementation of CATEA. In Chapter 5, we describe the results of an empirical study of CATEA conducted on students of an introductory Pascal course. In Chapter 6, we summarise what we have learned from the whole experiment of developing CATEA.

Chapter 2. A Survey of ICAL Systems

2.1 Historical Perspective

In the 1960s the digital computer was envisaged to play a key role in education through the development and widespread use of programs that would 'teach' topics drawn from a wide variety of disciplines. Reinforced by earlier work on programmed learning (by Pressey, Skinner, and Crowder amongst others), the view that children '*learn by being told*' was much more prevalent at that time than it is today. Subsequent implementation of computational teaching programs, in the persuasive advertising jargon of education, claimed to individualize teaching, and so facilitate learning. A profusion of acronyms resulted in relation to the use of computers in education - CAI, CAL, CBE, CBL, CMI, CFI, CML (where C=Computer, A=Aided or Assisted, B=Based, M=Managed, I=Instruction, L=Learning, E=Education and F=Furthered).

During the mid-60's, a number of generative CAI systems (Suppes, 1971; Woods & Hartley, 1971; Uhr, 1969) were devised to provide drill and practice in arithmetic and in vocabulary recall, and to select problems at a level of difficulty dependent upon and adjusted to the student's overall performance. These systems use an algorithm to generate teaching material as opposed to those programs where each piece of teaching material is pre-stored. These systems are also called adaptive and their sophistication lay in the task-selection algorithms. In these systems, models of the students are based more on parametric summaries of behaviour than

explicit representations of their knowledge.

By the end of the decade, a group of AI research workers gradually emerged who shared a very different belief about the learning and thinking processes. The underlying assumption in their work is that complex cognitive activities like seeing, learning, thinking and using language are knowledge-based. One of their chief concerns is how best to represent them in the computer. The relevance to education is the belief that a learner, working in a particular domain, has to build the domain specific knowledge into an active mental database, active in the sense that the knowledge representation can be assessed and used as the basis for new learning. This process of constructing such a mental representation implies creative mental behaviour on the part of the learner. In this context of '<u>learning by doing</u>', the role of the teacher is to structure a domain in such a way that it facilitates the model-building activity. Underlying this shift to a new paradigm in CAI is the belief that truly individualized teaching systems could not be built until deep conceptual problems , about such matters as knowledge representation, student modelling and language understanding, were better understood.

The domain of CAL is interpreted broadly (Zinn, 1978) to include learning about computers, with computers, through computers, and computer-managed learning. In this thesis, the terms <u>Intelligent Computer-aided Learning</u> (ICAL), ICAI and <u>Intelligent Tutoring Systems</u> (ITS) will be taken to be synonymous, and to include :

(1) learning through the computer (coaches, laboratory

instructors, consultants)

(a) drill and practice

(b) tutorial

(c) diagnostic testing;

(2) learning with the computer (problem-solving monitors)

(a) simulation and gaming

(b) problem-solving

(c) creative activities.

2.2 Towards Intelligent Computer-aided Learning

An ICAL system should in some sense model an intelligent teacher, but we have only elementary notions why some human teachers are good at teaching while others are not. As an alternative strategy, we will select the most obvious deficiencies of the traditional CAL approach, and look at AI work which provides some insight into the underlying difficulties. The 4 main limitations are :

(1) inability to know or understand the subject being taught, in the sense that the system cannot accept unanticipated answers or answer questions;

(2) inability to conduct dialogues with the student in natural language;

(3) inability to understand the nature of the student's mistakes or misconceptions;

(4) inability to profit from experience with students or to experiment with the teaching strategy (O'Shea, 1979).

In contrast, to tutor well,

> The [ICAL] system must have its <u>own</u> problem-

> solving expertise, its own diagnostic or student modelling capabilities and its own explanatory cabapilities. In order to orchestrate these reasoning capabilities, it must also have explicit control or tutorial strategies specifying when to interrupt a student's problem-solving activity, what to say and how best to say it; all in order to provide the student with instructionally effective advice (Brown & Sleeman, p. 2, 1981).

ICAL programs use AI formalisms to separate out what they intend to teach from their teaching strategy. This approach has several virtues : it becomes possible to keep records of what the student know, the teaching strategy can be generalized and applied to multiple problems in multiple problem-domains, and a model of student knowledge can be inferred from student responses and used as a basis for tutoring (Clancey & Buchanan, 1982).

As a general paradigm for the field, a tutoring system (Figure 1) should incorporate the following 3 models :

(1) a domain model containing the knowledge or expertise we want to teach (what is being taught?)

(2) a teaching model containing the tutoring theory and strategy to improve the student's performance (how shall we teach?)

(3) a student model containing the knowledge that the system expects the student to have acquired (how much does the student know?).

Within the tutoring system, a teaching model accesses the student model to find out what the student knows and the domain model for what remains to be learned. The system then decides what task to present to the student. Once a task is generated, the student draws upon his current knowledge to do the task, and

presents this decision to the tutoring system. From the student response, the system updates its student model and initiates a new decision cycle.

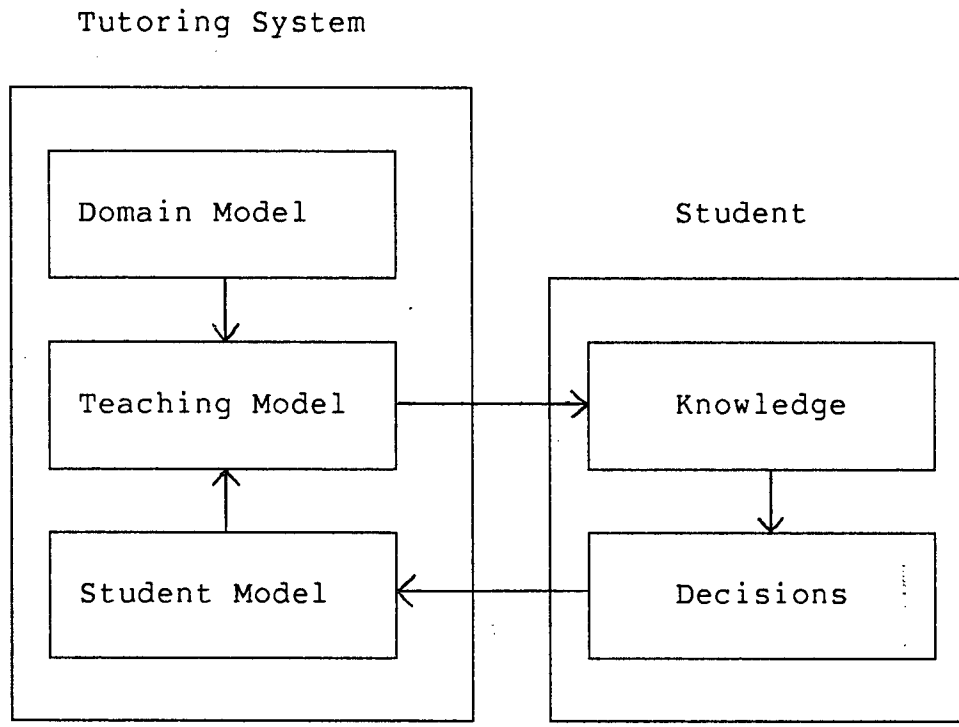

Figure 1. A Paradigm for Tutoring (Osin, 1980)

Not all of the three models are fully developed in every ICAL system. Because of the size and complexity of ICAL systems, most researchers tend to concentrate their efforts on the development of a single part of what would contribute to a fully usable system (Barr & Feigenbaum, 1982). The greatest weakness of current ICAL systems is possibly their elementary teaching models. Clancey & Buchanan (1982, p. 4) note that

> Almost invariably, most researchers have backed off from initially focusing on the [second] question - "How shall we teach?" - to reconsider the [last] question, building a model of the student's knowledge. This follows from the assumption that the student errors are not random, but reflect misconceptions about the procedure to be followed or

> facts in the problem domain, and the best teaching strategy is to directly address the student's misconceptions.

To assist research in building models of misconceptions, we need a sounder understanding of the nature of knowledge and expertise. Comparison studies of experts and novices are revealing how the expert structures a problem; the very concepts he uses for thinking about a problem, distinguish his reasoning from the student's often formal bottom-up approach. These studies suggest that we might convey to the student the kinds of quick associations, patterns and reasoning strategies that experts build up tediously over long exposure to many kinds of problems - the kind of knowledge that tends not to be written down in basic textbooks (Clancey & Buchanan, 1982).

Following this premise that we can be better teachers by better understanding expertise, expert systems research becomes of keen interest in education. These knowledge-based programs contain within them a large amount of facts and inference rules for solving problems in restricted domains of medicine, science and engineering. They have special interest to Cognitive Science research as simulation models that can be used as a "laboratory workbench" for experimenting with knowledge structures and control strategies.

Another natural application for expert systems in education is to use them as the knowledge foundation for an ICAL system (Figure 2). The teaching knowledge is an essential component as an expert system in a particular domain is not necessarily an expert <u>teacher</u> of the material (Barr & Feigenbaum, 1982).

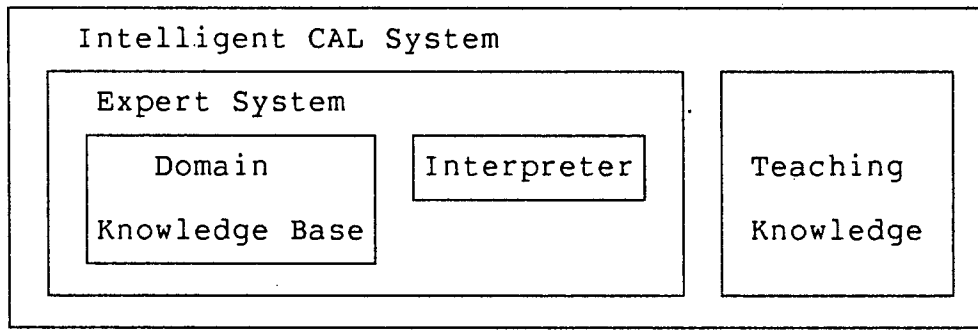

Figure 2 Components of an ICAL system (Clancey, 1981)

2.3 Milestones in ICAL Research

The well-known milestones in ICAL research include[1] :

(1) interacting with the student in a mixed-initiative dialogue (Carbonell, 1970)

(2) tutoring by the Socratic method (Collins, 1976)

(3) evaluating hypothesis for consistency with measurements taken (Brown et al, 1975)

(4) enumerating bugs in causal reasoning (Stevens, 1978)

(5) interpreting student behaviour in terms of expert knowledge (Burton, 1979; Carr & Goldstein, 1977; Clancey, 1979a)

(6) codifying discourse procedures for teaching (Clancey, 1979b)

(7) constructing incorrect plans or procedures (Genesereth 1981; Brown, 1975)

(8) relating incorrect procedures to a generative theory (Brown, 1980)

---

[1]The list of milestones from 1 to 8 was taken from Clancey & Buchanan, 1982. In attempting a comprehensive survey, I have added 9 and 10.

(9) building self-improving teaching systems (O'Shea, 1979)

(10) building learning environments (Feurzeig & Papert, 1969).

Milestones 1,2,3,6,9 and 10 address the teaching model, while 4,5,7 and 8 make contributions to student modelling. We shall now proceed to delve into each milestone and describe its contributions to ICAL research as well as some of its limitations. One broad theme emerges from these milestones : the incorporation of cognitive models of learning or/and teaching. This means that in the near future all CAL courseware will be cognitively-based, capable of diagnosis and remediation at the cognitive level.

2.3.1 Interacting with the Student in a Mixed-Initiative Dialogue

The CAI efforts of the 1960's can be classified into 3 broad categories (Bryan, 1969). In the first, ad-lib CAI, the student is given access to the computer (including one or more languages and perhaps a library of routines), but he is in full control and his input is not controlled by the computer. A typical example is Seymour Papert's LOGO Laboratory (Papert, 1980; Abelson & diSessa, 1981). The second category is games and simulation where the student has some initiative but is constrained by the rules of the game or the logic of the simulation.

In the first two categories, learning takes place as an expected side effect. The third category makes an explicit attempt to instigate and control learning. Drill-and-practice and, in general, classic efforts to use the computer as a tutor

are included here. These efforts involve the construction of frames of questions with anticipated correct and wrong answers and perhaps keywords to be extracted from the student's answer. As their sequencing is deterministic, Carbonell (1969) called them 'ad-hoc frame-oriented' CAI systems and observed that

> [In these systems], the student has limited or no initiative; he cannot use natural language in his responses, and systems usually look fairly rigid to him... From a systems viewpoint, the system controls the student but is in turn tightly ad-hoc programmed by the teacher; the system has no real initiative or decision power of its own; and, of course, it has no real 'knowledge'.

Carbonell (1970) had these limitations in mind when he wrote a dialogue program, called SCHOLAR (the earliest of ICAI systems), to teach facts about the geography of South America. It is a "non-deterministic" CAI program where decisions are probabilistic among dynamically determined alternatives, with weights that are dynamically associated. Instead of storing geographic information in the form of prewritten frames , the program was organized around an associative database (semantic network) of simple geographic facts about industries, populations and capitals. It was designed to manipulate its database to generate factual questions, to check unanticipated answers to these questions, and also to answer the student's questions. Since control of the way in which the conversation could develop is shared between student and system, Carbonell coined the term '<u>mixed-initiative</u>' to describe this kind of interaction.

The semantic network is a collection of named nodes, called concepts, with pairs of these nodes linked by relations. In the database, a fact is represented as a pattern of linked concepts,

the pattern resembling a tree or network. A question, in effect, is an incomplete pattern, and the task of the program's retrieval procedures is to search in the database for a pattern which matches the question pattern, and to supply in its answer any additional relevant information about the topic which is stored in the database.

This method works reasonably well provided that the information required is explicitly represented in the database. However, a great deal of information is encoded implicitly, and this can only be retrieved by adding ad hoc inference rules to the database, and by applying these rules during retrieval. For example, the database might contain the rule 'To find the products of a place find the products of the industry at a place.' An obvious difficulty is that as the amount of knowledge in the network increases, there will be a rapidly expanding number of false paths associated with nodes like industry or place, creating a significant search problem.

2.3.2 Tutoring by the Socratic method

This milestone considers conversations between a tutor and a student as an important method of teaching. Collins (1976) proposes that the best way to teach knowledge and the skills necessary for applying that knowledge to new problems or situations, is through the Socratic method of teaching . The student of a Socratic dialogue is forced to reason for himself, derive general principles from specific cases and apply the general principles that he have learned to new cases. More specifically, he learns 3 kinds of knowledge :

(1) specific information about a variety of cases;

(2) the causal dependencies or principles that underlie these cases;

(3) reasoning skills like forming hypotheses, testing hypotheses, distinguishing between necessary and sufficient conditions, and asking the right questions when there is not enough information to make a prediction.

Collins specifies and formulates the Socratic method as a set of strategies or rules. An ICAL system can incorporate as many of these strategies as possible in tutoring casual knowledge and reasoning. The Socratic method has not been previously considered feasible for education generally because it is a one-to-one teaching strategy. Developing technologies like distributed instructional systems make it possible to teach many more students with such a tutoring strategy.

Collins examines a variety of Socratic dialogues empirically to uncover those features that characterize the tutor's behaviour. He formalizes the tutoring strategy as production rules of the form "If in situation X, do Y." The rules are this written in a procedural formalism that is independent of the particular context. 23 rules are derived, the first two rules of which are listed below with an example in tutoring causal factors affecting rice growing :

> Rule 1 : Ask about a known case.
> If
> (1) it is the start of a dialogue
> then
> (2) pick a well-known case and ask what the value of the dependent variable (the variable that depends on causal factors) is for that case.
> Example : Ask the student "Do they grow rice in China ?"
> Reason for use of rule : It brings out any well-

known facts the student knows about such as rice growing in China.

Rule 2 : Ask for any factors.
If
(1) a student asserts that a case has a particular value for the dependent variable,
then
(2) ask the student why.
Example : if the student says they grow rice in China, ask why.
Reason for use : This determines what causal factors or chains the student knows about.

A second-order theory of teaching strategy is needed to choose what rules are most appropriate to invoke in different situations.

This milestone proposes a psychological theory that will account for behaviour in dialogue situations and a set of rules for constructing effective instructional programs. For the rules to be adequate as a descriptive theory, it must be able to successfully account for new dialogue protocols (Resnick, 1976). In Collins' work, there is no specification of how many or what kinds of dialogue protocols are accounted for. Many more protocols will have to be analyzed before the generality of Collins' rules can be estimated. For the rules to be adequate as a prescriptive theory, it should produce dialogues that look like the real ones and that did promote some kind of learning. Resnick suggested that the condition part of the rules should also include, besides immediate student responses, at least a brief history of the student's responses and the tutor's intentions during the dialogue itself. More empirical studies are needed to judge whether the dialogues do actually teach.

The task of eventually building an automated dialogue expert for tutoring is enormous and may not be realized for a

long time. However, the attempt itself, forcing attention as it does to the details of natural dialogues, will help yield greater understanding of the tutoring process between human actors.

2.3.3 Evaluating Student Hypothesis for Consistency with Measurements taken

An intelligent instructional system which reflects a major attempt to extend Carbonell's notion of mixed-initiative CAL for the purpose of encouraging a wider range of student initiatives is SOPHIE (Brown et al, 1976). Unlike previous systems which mimic the roles of a human teacher, SOPHIE tries to to create a 'reactive' environment in which the student learns by trying out his ideas rather than by instruction.

SOPHIE acts as an electronics laboratory instructor who helps the student transform his classroom knowledge of electronics into an experiential, intuitive knowledge of its meaning and application. It operates by presenting the student with a circuit schemata of an electronic apparatus, into which it has previously introduced a fault of some specified degree of difficulty. The student's task is to trace this fault. He can perform any sequence of measurements, ask specific questions about the implications of these measurements , and even ask for advice about what to consider next, given what he has discovered so far. At any time, SOPHIE may encourage the student to make a guess as to what he thinks might be wrong given the measurements he has made so far. If he does, SOPHIE will evaluate his hypothesis. If the information the student should have been able

to derive from his current set of measurements is logically contradicted by his hypothesis, SOPHIE identifies and explains these contradictions. SOPHIE can also judge the merits of any particular measurement with respect to the prior sequence of measurements he has made. For example, his new measurement may be logically redundant in that no new information can be derived from it.

SOPHIE's expertise is derived from an efficient and powerful inferencing scheme that uses multiple representations of knowledge including

(1) simulation models of its microcosm

(2) procedural specialists (of which the hypothesis evaluation specialist is one) which contain logical skills and heuristic strategies for using those models

(3) semantic nets for encoding time-invariant factual knowledge (Brown & Burton, 1979).

The power and generality of SOPHIE result from the synergism obtained by focusing the diverse capabilities of the procedural specialists on the intelligent manipulation, execution and interpretation of its simulation models.

SOPHIE is a very large program, but the researchers claim that the average response time to a question is 3 seconds. While this is a satisfactory measure of the program's efficiency, we lack evidence about its educational value. One built-in assumption is that the student must have some knowledge of the basic electronic principles underlying the design and operation of DC power supplies. The measurements he makes take on meaning in the context of his mental model of the power supply, and this

enables him to put forward a hypothesis. But if his mental model is wrong, his solution will probably be wrong too. Unfortunately, SOPHIE is not sufficiently intelligent to explain why it is wrong.

2.3.4 Enumerating Bugs in Casual Reasoning

This milestone works towards a theory of tutorial interaction that enumerates the types of conceptual bugs students have in their understanding of physical processes, and tries to diagnose and correct these bugs from different representational viewpoints. Stevens et al (1978) built a system based on this theory, called WHY, which tutors the causes of rainfall. The theory is formalized as a set of Socratic tutoring rules and a script-like knowledge structure (Schank & Abelson, 1977). The script structure represents the different temporal and causal steps in processes that affect rainfall.

Although the first version of the WHY system is able to carry on simple tutorial dialogues about the causes of rainfall, interaction with it reveals that it can detect surface errors like missing steps in reasoning about the causes of rainfall, but fail to diagnose the underlying misconception that the error reflects. The researchers believe that one of the major skills a good teacher possesses is knowledge about the types of conceptual bugs students are likely to have, the manifestations of these bugs and methods for correcting them. So an important component of an ICAL system is a method for representing, diagnosing and correcting bugs.

To enumerate the bugs in students' causal understanding of

rainfall, a questionnaire survey was carried out. It asks 32 questions about the causes of heavy rainfall. From the substantial body of data on errors and misconceptions gathered, the researchers identified 16 different bugs that account for 72% of incorrect answers.

The next step is to represent these bugs in a diagnose-and-correct goal structure. This requires a detailed formalism for representing the knowledge taught. Script structures can be used to represent ordered causal and temporal processes, but this handles only a small number of bug representations and viewpoints from which to discuss them. Collins proposes a functional viewpoint which emphasizes the functional relationship among the attributes of the various objects involved in different processes. The basic unit is a description of some process such as evaporation or cooling. Here is an example of the functional relationship for evaporation :

Actors (with a role in the process)

Source : Large-body-of-water
Destination : Air-mass

Factors (which affect the process)

Temperature (Source)
Temperature (Destination)
Proximity (Source, Destination)

Functional-relationship (which hold amomg the factors and result)

Positive (Temperature (Source))
Positive (Temperature (Destination))
Positive (Proximity (Source, Destination))

Result (of process)

Increase (Humidity (Destination))

Misconceptions are represented as meaningful transformations of the basic knowledge representation. For example, the representation for the cooling-by-expansion bug is :

Actors

Source : Large-body-of-water
Destination : Air-mass

Factors

Pressure (Destination)
Proximity (Source, Destination)

Functional-relationship

Inverse (Pressure (Destination))
Positive (Proximity (Source, Destination))

Result (of process)

Increase (Humidity (Destination))

This bug consists of a substitution of pressure for temperature as the relevant attribute of the destination in the normal representation for evaporation. Bugs can show up in all parts of the representation. Instead of viewing a student model as a simplification of the expert's rules, this milestone treats it as a set of 'semantically meaningful deviations' from an expert's knowledge.

2.3.5 Interpreting Student Behaviour in terms of Expert Knowledge

Past and current research on an theory of student modelling has been focused on the notion that if one has an explicit, well-formulated knowledge base of an expert for a given problem-domain, then one can model a student's knowledge as a

simplification of the rules comprising the expert's procedures. Goldstein (1977) has coined the term 'overlay model' to expand this concept in his Computer Coach research. The main idea is to explain differences between the behaviour of the expert and the student in terms of the lack, on the student's part, of some of the expert's skills. Thus, an overlay model is a set of hypotheses, each of which records the system's confidence that a student possess a given skill.

Overlay modelling has been developed as part of the COACH Project at MIT, whose concern is the development of AI-based CAI programs for tutoring the skills required for successfully playing various computer games (WUSOR 2 for playing WUMPUS and WEST for the PLATO game "How the West has won"). The computer serves as an assistant to a learner who is in the process of acquiring the skills necessary to play the game well.

Overlay modelling is embodied in a Psychologist model, the component of the coach responsible for maintaining such models of the player's current skills (the K model) and learning preferences (the L model). These models are used by the Tutor module to prune complex explanations generated by the expert. Like a human speaker, the coach abbreviates its statements by eliminating those facts that are already known by the listener and those facts that are too complex.

The modelling is performed by a set of P rules that are triggered by different sources of evidence, and whose effect is to modify a set of hypotheses regarding the student's familiarity with the expert's skills. A P critic monitors these rules to detect discontinuities and inconsistencies in the

player's behaviour. Inconsistencies are evidence that the P rules are failing to model the student properly, while discontinuities indicate a change in the player's knowledge state.

As no single source of evidence is a certain indicator of a learner's knowledge, the Psychologist is provided with 4 sources of evidence :

(1) implicit - The student's play indicates his mastery of various skills. The assumption is that the player has learned those skills involved in choosing his particular move and rejecting its inferiors, and has yet to learn those skills needed to recognize superior moves.

(2) structural - The expert's skills are structured into a syllabus which is a network linking the skills in terms of their complexities and dependencies. Structural knowledge suggests conservatively that a student familiar with a certain region of the syllabus (as indicated by the K model) is more likely to acquire a new skill at the frontier of this region rather than deep into unknown territory.

(3) explicit - The tutor can obtain explicit evidence by asking the student two types of questions : test cases and follow-up questions.

(4) background - Every teacher has expectations about the performance of a student on the basis of that student's background. Thus skill levels are classified into various levels, like "novice", "amateur", or "expert", and correspond to a different initialization for the overlay

model.

The contribution of overlay modelling is that it is essentially a linguistic theory of the speaker (Carr, 1977). Each of us, when formulating an explanation, abbreviates the explanation in accord with our model of the speaker. This model is based on our analysis of the listener's behaviour in terms of the knowledge we know and the knowledge we believe he needs to know. Overlay modelling performs a similar function for a computer tutor. Indeed, a person or computer can only judge another in terms of what he, she or it knows itself.

Overlay modelling rests on the assumption that the skills employed by the student are a subset of those of an expert. This is not inevitable for at least 3 reasons (Carr, 1977). First, the student may be solving problems in a fashion completely divergent from the expert - there may be multiple paradigms for the particular problem domain. Second, the student may be using a non-optimal method for his own reasons. A computer game player may be more concerned with finishing quickly than avoiding risk, and choose to a move to a more informative position, despite greater risk. Third, the student may possess a skill of the expert in an incorrect form, perhaps using it inappropriately. Modelling will fail if the complete absence of a skill cannot be distinguished from its inappropriate use. Thus it would be better to have a general theory of learning that address situations in which the student has an incorrect skill or an alternative skill.

2.3.6 Codifying Discourse Procedures for Teaching

So far, ICAL research has focused on the representation of domain expertise, the construction of a student model, and tutoring principles for correcting misconceptions. It has not dealt directly with the problem of carrying on a coherent, purposeful task-oriented dialogue with a student. In WEST and WUMPUS, for example, the tutor's remarks are all interruptions or reactions to the immediately preceding move taken by the student.

Clancey (1979b) discusses a kind of mixed-initiative dialogue that concerns a single, complex task to be solved by a student under a program's guidance. Sequences of student/tutor remarks are grouped into '*discourse situations*', some typical examples of which are : examining the student's understanding after he asks a question that shows unexpected expertise, and giving advice to the student after he makes an hypothesis about a subproblem. Clancey calls the general problem of sharing initiative with the student and having provisions to carry out one's discourse goals as '*dialogue management*'. He discussed it using GUIDON, the first tutor built on top of a complex expert system using MYCIN's 400 production rules and tables for teaching medical diagnosis by the case method.

GUIDON was first conceived as an extension of the explanation system of the MYCIN consultation program. With this foundation, a tutoring program would take MYCIN's solution to a stored consultation problem, analyse it and use it as the basis for a dialogue with a student trying to solve the same problem. The student is given some information about a patient expected

to have an infectious disease, and is expected to request case data, as he sees necessary to draw conclusions about the cause of the infection. The topics of the dialogue are precisely the goals that are determined by applying MYCIN's rules, like the type of infection. As the dialogue proceeds, GUIDON is able to collect information about the student's knowledge of the case and its domain using an overlay model. Tutorial remarks will be given when appropriate to make the students aware of any gaps or inconsistencies in his knowledge with respect to the underlying expert rule base, and to correct these deficiencies. They range from single pertinent remarks to prolonged presentations that go beyond interruption and repetitive question/ answering.

Briefly, the framework of GUIDON incorporates

(1) domain knowledge in the form of MYCIN-like rule base and records from the consultation to be discussed with the student;

(2) discourse knowledge in the form of several hundred domain-independent tutoring rules , organised into discourse procedures for carrying on the dialogue;

(3) a communication record for recording discourse goals and the dynamic state of the tutorial.

Clancey represented the discourse knowledge in the form of a transition diagram in which the nodes are discourse situations and the links represent choice points that lead to alternative dialogues dictated by domain logic, economy of presentation and tutoring goals. These links represent the management decisions in which the tutor takes the initiative to control dialogue situations. GUIDON also provides for student initiative by

allowing him to refer back to an early topic and gather more details, to make a conclusion when he is ready, and to change the dialogue topic.

One limitation of the GUIDON framework of discourse procedures is that it can only discuss rule-based knowledge, that is, topics that can be expressed as production rules. Also, in the course of developing GUIDON some fundamental shortcomings become clear. There are 2 kinds of explanations that MYCIN cannot give and GUIDON cannot teach : it cannot explain why a particular rule is correct and it cannot explain the strategy behind the design of its goal structure. MYCIN is aphasic - able to perform, but unable to talk about what it knows. A human teacher can provide analogies, multiple views, and level of explanations which are unknown to MYCIN. MYCIN did not capture all that an expert knows, for example, how he organizes his knowledge, how he remembers it, and strategies he uses for solving problems. Clancey (1981) reports that many students were unable to remember the rules, even after discussing a single problem with GUIDON many times.

During 1979-80, a study was undertaken to determine how an expert remembered MYCIN's rules and how he remembered to use them. This study (outlined in Clancey, 1981) utilized several common AI methods for knowledge acquisition, built upon them significantly through the development of an epistemological framework for characterizing various kinds of knowledge. A new comprehensible psychological model of medical diagnosis was formulated and implemented in NEOMYCIN which is a consultation program in which MYCIN's rules are reconfigured to make explicit

distinctions that are important for teaching. The knowledge representation separates out the inference rules (simple data/hypothesis associations) from the structural and strategic knowledge, that is, what a heuristic is from when it is to be applied. The strategies and structure model how an expert reasons. Separation of problem knowledge (inference rules) and diagnostic strategy enables a tutorial program to present them separately to a student, as well as look for them in his behaviour. With its theoretical, epistemological underpinning, NEOMYCIN is designed to present the subject material that a new version of GUIDON, GUIDON2 can use to express important teaching points. This provides an example of how work in ICAL has contributed to new AI ideas and methods.

2.3.7 Constructing Incorrect Plans or Procedures

This milestone extends the work of Stevens et al (1978) in diagnosing misconceptions in understanding. Each skill of the expert is explicitly coded, along with a set of potential misconceptions of that skill. The task of inferring a diagnostic model then becomes one of discovering which set of variations or deviations best explains the surface behaviour of the student. This view is also similar to but more structured than the approach taken by Self (1974) in which he models the student as a set of modified procedures taken from a procedural expert problem-solver.

Research in a diagnostic model of basic skills arose from an investigation of the procedural skills necessary to solve high school algebra problems (BUGGY, Brown et al, 1977). To

illustrate the model, we consider an example of teaching arithmetic skills. Brown (1978) noted that it is no great challenge to add or subtract 2 numbers, but diagnosing misconceptions in these skills can be quite subtle.

Consider a case study in which we examine 5 "snapshots" of a student's performance doing addition as might be seen on a home work assignment :

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

Example 1

In computer terms, the student, after determining the carry, forgets to reset the "carry register" to 0 and hence the amount carried is accumulated across the columns. The bug is not so absurd if one considers that a child might use his fingers to remember the carry and forget to bend back his fingers, after each carry is added.

For a system to be capable of diagnosing student's misconceptions such as above, the procedural skill being taught must be represented in a form amenable to modelling incorrect as well as correct subprocedures of that skill. Furthermore, the representation must allow the intermixing of both the correct and incorrect skills, so that the model can capture those parts of a skill that are correct as well as wrong. The breakdown of the skill into shared subskills can also account for the recurrence of similar errors in different skills.

The representational technique for diagnostic models is a procedural network. A procedural network model for a correct

skill consists of a collection of procedures with annotations in which the control structure between procedures are made explicit by appropriate links. Each procedure has 2 main parts : a conceptual part representing the intent of the procedure, and an operational part consisting of methods for carrying out that intent. The methods are programs that define how the results of other procedures are combined to satisfy the intent of a particular procedure. Any procedure can have more than one method, thus providing a way to model different methods for performing the same skill. The possible misconceptions in each skill are represented in the network by incorrect methods associated with subprocedures in its decomposition called "bugs". Each buggy version contains incorrect actions taken in place of the correct ones.

In Example 1, we are provided with several surface manifestations of a bug in the student's addition procedure. To uncover those possible subprocedures which are at fault, we use the procedural network to simulate the behaviour of buggy subprocedures over the set of procedures over the set of problems and note those which generate the same behaviour as exhibited by the student. To catch a student's misconceptions that involve more than one faulty subprocedure, we must be able to simulate various combinations of bugs.

A deep-structure model of the student's errors is a set of buggy subprocedures that, when invoked, replicates those errors. Each buggy version has associated information such as what the underlying causes of the bug may have been, as well as specific remediations, explanations, interactions, and examples of the

bug - all of which may be used by a tutoring system to help correct the student's problem.

The first problem that naturally arises is whether or not it is possible to formulate the modelling problem so that the combinatorics can be contained. Brown and Burton note that, given a sizeable number of buggy procedures, the number of combinations to be considered is extremely large. Sleeman and Smith (1981) propose several heuristics which reduce the size of the search space.

Another issue is the psychological validity of the skill decomposition and buggy variants in the network - how well do the procedures in the network correspond to the skills that we expect people to learn? If the functional breakdown of the skill is not correct, common bugs may be difficult to model while those suggested by the model may be judged to be "unrealistic".

We have also left open the entire issue of a semantic theory of how procedures are understood and learned by a person and why bugs arise in the first place. An interesting theoretical framework that accounts for the entire collection of empirically arrived at bugs will undoubtedly provide insight into how to correct the teaching procedures that produce them in the first place. Moreover, such a theory would be the next step in a semantically-based generative theory of student modelling. In BUGGY, bugs have to be hand-coded into the network. One can envision generatively producing bugs through inappropriate analogy from other operations or incorrect generalization from examples through a "semantic" theory.

2.3.8 Relating Incorrect Procedures to a Generative Theory

This milestone marks current efforts to form a generative theory of bugs in procedural skills. Given a procedural skill, it predicts which systematic errors or bugs will occur in the behaviour of students learning the skill.

A child's errors are said to be systematic if there is a procedure that produces his erroneous answers. BUGGY and more recently DEBUGGY have been used to analyse thousands of student's work (Burton, 1981 ; Vanlehn & Friend, 1980). An extensive data base of bugs was collected and found to be converging in the sense that they cover a substantial number of student errors and only a small number of new bugs are being discovered. The researchers investigate the cause of these bugs and the reasons they occur. Effort is focused on explaining these known bugs in terms of a set of formal principles that transform a procedural skill into all of its buggy variants. A generative theory of bugs was developed that explains how bugs arise as a result of systematic errors in performing a skill. The theory must generate all the known or expected bugs for a particular skill and it must generate no others.

The theory is motivated by the belief that when a student has unsuccessfully applied a procedure to a given problem, he will attempt a repair. Suppose he is missing a fragment of some correct procedural skill, either because he never learned the fragment or he has forgotten it. Instead of quitting because of an impasse, he will often be inventive, invoking his problem-solving skills in an attempt to repair the impasse so that he

can proceed to execute the procedure, albeit in a potentially erroneous way. The researchers believe that many bugs can best be explained as "patches" derived from repairing a procedure that has encountered an impasse. Brown (1980) calls the theory Repair Theory. In the theory, a bug's derivation has 2 parts. The first is a series of operations that generate an incomplete procedure, namely, a procedure that may reach an impasse on certain problems. The second part is a series of operations that represent the repair of the procedure so that it can proceed.

Repair theory defines the set of incomplete procedures by applying a set of deletion principles to a formal representation of the correct procedure for the given skill. The set of repairs is defined by a set of repair heuristics and a set of critics. When an incomplete procedure is applied to a problem and reaches an impasse, a set of repairs is performed by a generate and test problem-solver. The set of repair heuristics suggest repairs and generate bugs, and the critics filter out the "star-bugs" - absurd bugs that expert diagnosticians agree will not be observed.

The theory has been tested on the extensive database of bugs for multidigit subtraction and not only generates the observed bugs, but also make predictions about the frequency and stability of bugs (how often a bug occurs in the students' work and how long a student keeps a bug). While Repair theory has been designed to be relatively domain-independent, it has not been applied to domains other than place-value subtraction. Brown(1980) argues that Repair theory can be applied to other domains and that it can be extended to provide a theory of

procedural learning that accounts for bug acquisition.

2.3.9 Self-Improving Teaching Systems

One of the ways in which CAL programs compare badly with human teachers is that they do not benefit from their teaching experience. A CAL program which teaches poorly in some way will teach poorly in exactly the same way after teaching 10,000 students. This milestone addresses the issue of the lack of learning capability in CAL programs. O'Shea (1979) reports only three CAI programs in the literature that have a self-improving capability - Smallwood's program (1962) that teaches geometry, Kimball's tutor (1973) for integration and O'Shea's system for teaching quadratic equations.

Self-improvement could take the form of improving its internal representation of its teaching material. It would mean learning new facts, concepts or skills from the students being taught. For the most part, such learning is completely beyond the state of the art because of the way domain-specific knowledge is represented in such programs (O'Shea, 1981). The only example of a CAI program that improves the internal representation of its teaching material is that of Kimball's. In his program, if a student's solution to an archive integration problem is better (has fewer steps) than an archive solution, then this student's solution is adopted and becomes the new archive solution for that problem. So the program's ability to carry out integration improves and hence its hints to the students may change, but the extent (response-sensitivity) to which it can adapt to the individualized learning needs of its

remains fixed.

O'Shea proposes a design for a self-improving teaching system aimed at improving the quality of instruction by altering and experimenting with those features of CAI programs which relate to their response-sensitivity. Some specific examples of such features are :

(a) the best order to present a set of concepts to students who have different styles

(b) the point at which a program monitoring a student performing a problem-solving task should intervene.

The design has two components. One component is an adaptive teaching program where the teaching strategy is expressed as a set of production rules. The second component performs the self-improving function of the system by making experimental changes to the set of production rules. This component employs a deduction procedure which operates on a theory of instruction expressed as a set of modally qualified assertions. A theory of instruction is concerned with optimising the learning process and can be regarded as an action-driven production system where the question to be answered by deductive inference on the assertions is "What change in teaching strategy will improve teaching performance wrt educational objective x?". The cycle of operations proposed for the system is as follows : select an educational objective, make an experimental change in teaching strategy, statistically evaluate the resulting performance, and update both the set of production rules and the set of assertions.

A self-improving system was implemented for the teaching of

quadratic equations by the discovery method. The system was used by 51 students, and executed 5 experimental changes on its teaching strategy. This trial demonstrates that it was capable of improving its performance as a result of experimentation.

Self-improvement is particularly applicable to those CAI programs intended to run under very large multi-access systems which affect very large number of students. These environments provide considerable amounts of performance data which should enable self-improving systems to make reliable modifications.

The principal contribution of O'Shea's program has been to demonstrate the feasibilty of constructing an response-sensitive self-improving program. At present the system only makes changes to its tutorial strategy. Future possible research could be done on facilitating the expression and development of richer theories of instruction.

2.3.10 Building Learning Environments

Given the kinds of deep pedagogical problems of CAL and the teacher's traditional dislike of machines in the classroom, we can anticipate that expert systems in the shape of computerised tutors will play a minor role for many years to come (Howe, 1979). An alternative approach to the use of computers in education is to use the computer to simulate a modelling system which a learner can use to carry out model-building experiments. In the last five years, researchers have focused on supportive learning environments which attempt to combine the problem-solving experience and motivation of 'discovery' learning with the effective guidance of tutorial interactions.

Pioneering work has been done by Feurzeig and Papert in mathematics learning. They emphasized the fundamental problem of identifying and naming the concepts that a beginner needs to enable him to express his thoughts in a clear way. With this in mind, they invented a simple but powerful programming language called LOGO which provides a child with a language for describing procedures. The child will learn to invent and carry out exciting projects by having access to computers and with peripheral devices capable of producing on-line real-time action (Papert, 1971). In doing so, he learns to use computers in a masterful way which can change the way he learns everything else. A sound Piagetian principle is brought into play - new learning takes place in the framework of old knowledge, something the child already knows.

There are two views concerning the relevance of the activity of model-building to education. Papert's view is that the programming approach should emphasize problem-solving skills in the expectation that when a child has work in several different domains, he will recognize the similarities between the methods used to make plans and to debug them. In this way, he will build up genuine domain-independent study skills. Others take the view that these skills can only be appreciated when a person has extensive domain-specific knowledge, suggesting that in the early stages of learning emphasis should be placed on content learning.

Evidence of benefit of learning content or problem-solving skills has been scanty. Some evaluation studies (Howe, 1979) has shown that LOGO has been successful in teaching domain-specific

knowledge like mathematics and programming. When it comes to teaching problem-solving skills, there is less evidence of benefit. Statz (1973) records little improvement in children's ability to cope with problem-solving tasks after a year's programming activity.

Work is also done at the Xerox Palo Alto Research Center in California on designing learning activities for 'children' of all ages. The aim is to provide powerful computational facilities and techniques which can be applied to a diverse range of tasks in school and in the office, by children and adults alike (Kay, 1977). A new programming language called Smalltalk was developed and in it, transactions take the form of messages sent to, or received from, 'activities' in the system. Every activity belong to a class, or family of activities, each one of which has the ability to recognize and reply to messages. Each class also has certain capabilities, such as drawing pictures, making musical noises, or adding numbers. For example, to program an airplane simulator in Smalltalk, one would define a class called instrument, and create activities (instances) of that class to draw the particular instruments on the display screen. Each instrument would have its own position on the screen, its own label and its own displayed value. Altering one of the controls would cause messages to be exchanged via the class which locates the instances to obtain any value requested or to make alternations to these values.

A number of features has been added to Smalltalk to build Thinglab (Borning, 1979) which provides an object-oriented environment for building simulations. Thinglab allows the user

to describe complex simulations easily by specifying all the relations (called constraints) among the various objects that must be satisfied and leaving it up to the system to plan exactly how the constraints are to be satisfied. Although the interactions among the parts of simulation system may be numerous, the user can specify each relation without worrying about the others. A Thinglab-style system might prove valuable as part of a geometry curriculum, or as an adjunct to a physics laboratory.

The learning environment is viewed by some investigators such as Papert as a substitute for traditional classroom teaching. Papert saw the classroom as an artificial and inefficient learning environment and perceived that schools as we know them today will have no place in the future (Papert, 1980). Others are less radical and are interested in adapting the learning environment to the existing classroom, instead of abandoning the existing curricula and substituting the learning environment for it. This raises the issue of educational evaluation which we will discuss later.

2.4 Future Prospects

Two important areas which will surely influence ICAL are natural language input and output, and technological advances. Most ICAL systems have some form of natural language input/output (I/O), but the amount of language recognised is rather limited. The usual techniques are used : filling in templates, finding keywords or their synonyms, and a limited degree of spelling correction. Recent work in natural language

research suggests that natural language interfaces of considerable power may be available for CAL systems. Gable & Page (1980) note that

> The improvement in communication between the student and the CAI system which occurs because of improved language capability should not be underestimated. It would seem fair to say that a CAI system without extensive natural language capability is like a tutor who speaks a language foreign to the student.

In the area of human factors engineering, intelligent systems need to exchange information smoothly and efficiently with the student; that is, they should be sensitive to the needs, capacities and limitations of human beings. One possibility is to include some part of the student's psychological state into the student model in order to conduct a more appropriate dialogue. The PARRY system of Colby (1973) simulates a paranoid personality and constructs a model of the person using it in order to be ready to insult them later in some telling way. In an ICAL system, one could analyze unexpected or unrecognized inputs for signs of boredom or anger in order to respond in some appropriate way. The paranoid aspect of PARRY's personality might make it too human to be an effective tutor. It may typically become 'impatient' or 'angry' and terminates the session. But students might enjoy the possibility of choosing the 'personality' of their tutorial system. Some might prefer a mildly paranoid tutor to escape the monotony of endless mechanical patience.

Visual aids have been used in education for many years. The developing technology in video recording has extended the use of recorded lectures and demonstrations in teaching. Video storage technology has been greatly enhanced by the videodisk which has

the advantage of random access and can store over 100,000 television pictures on a single disk. Costs for storage of a single picture are reduced 100 times over printing costs, providing a significant incentive for use in ICAL systems. Computers may be interfaced with the videodisk providing selective access to video frames or sequences. The videodisk can also be used to store sound, large amounts of text, or mixtures of text, sound and moving pictures.

The use of videotex for education also holds much promise. Videotex makes available computer-based information via visual display units, or appropriately adapted television sets, to a dispersed and reasonably numerous audience. In interactive videotex (like Prestel in U. K. and Telidon in Canada), users can interact individually with the computer, instead of just passively receiving information.

Increasing cheap CPU power to cheap imaging systems means that we can supplement or alter the nature of ICAL systems from text-based natural language I/O to graphics or image-based I/O. The advent of inexpensive microcomputers makes available speech, sound, video and colour graphics at a cost significantly less than many of the early expensive video display terminals. CAL should be cost-effective, as one reason traditional CAI has not proved to be as widely accepted, practised and disseminated is that there has been no obvious financial advantages of CAI in comparison to conventional educational methods (Fielden, 1977).

From another perspective, ICAL systems can be seen as explorations into how the dramatic advances in computer technology can be used to produce new kinds of learning

environments. Future paradigms for using computers in education need not be constrained by scarce computational resources.

2.5 The Evaluation Issue

The raison d'etre of CAL is perhaps its ability to provide effective individualized tutoring. To convince people other than its creators that a CAL system is worthwhile, we have to look at the evaluation issue, which poses two problems. On the one hand, there is a current crisis in educational research (Howe, 1978) over the question of what constitutes an acceptable form for an educational evaluation. On the other, we have the problem of evaluating CAL.

One purpose of educational evaluation is to provide decision makers with information about the effectiveness of an educational program, product or procedure. Within this perspective, evaluation is viewed as a process in which data are obtained, analyzed, and synthesized into relevant information for decision making. While these is basic agreement about this fundamental role of evaluation in education, there is considerable variance in the conceptual framework, resulting in numerous paradigms or models used by practitioners (Borich et al, 1981). The task of evaluating a prototype ICAL system becomes one of choosing the paradigm or model most appropriate to the evaluation problem.

The generally high degree of specificity of CAL and its potential for saving student responses offer possibilities for adaption of instruction, and thus for evaluation that are not practical with non-computer approaches. An ICAL system can

perform its own self evalution, called intrinsic evaluation. Venezky (1983) proposed such a framework for evaluating ICAL systems. The objective is to judge whether ICAL systems meet instructional needs and is adaptable to the total curriculum. These types of evaluative information will be needed :

(1) the range of student strategies the system is capable of diagnosing;

(2) the probabilities of diagnosing each correctly;

(3) for each diagnosis, the effectiveness of the instruction that follows, in terms of the achievement outcomes and the time spent learning.

Venezky characterized ICAL programs as sequences of assertions of the form (Si, Iij, Sj), where Si is the learner's current state, Sj is the next desired state, and Iij is the instruction generated for moving from state Si to state Sj. Thus, a program asserts that, based on its own diagnosis, the learner is in state Si. Furthermore, it asserts that instruction Iij is sufficient to move the learner to state Sj. The role of evaluation is to assess the validity of both the state and instruction assertions. To do this, evaluation procedures will need to be designed as an integral part of the lesson itself. ICAL systems should be adaptive and continually evaluate their own behaviour and adjust their instructional paradigms and teaching strategies accordingly. To evaluate such systems (like O'Shea's self-improving program) extrinsically, it will be necessary to determine how the instructional approach changes with student behaviour.

Another perspective in evaluating ICAL is to perceive it

through the expert versus the apprentice view of the automated tutor. We ask this question of any ICAL system : is it aimed at replacing the teacher or assisting him or is it more of a test-bed for experimenting with theories of intelligence, learning or teaching. These 3 approaches use different means to achieve their ends and thus should be evaluated on different criteria. Indeed, a CAL system does not stand or fall on its own merits. Rather, its utility depends on the manner in which it is integrated into the classroom setting and harmonized with instructional objectives.

The apprentice view of an CAL system as a supplement to traditional classroom teaching is more fruitful than the expert view of such a system superseding the teacher. The state-of-the-art in AI is such that the task of modelling human intelligence is still considerable. There are not many examples of expert systems whose performance consistently surpasses that of an expert. Evaluation of traditional CAI has shown that CAI is more effective if it supplements or is supplemented by human instructors. A case in point is that when the TICCIT system was used as a sole source of instruction in mathematics, the completion rate of the course dropped significantly over that of traditional classroom teaching (Kehler, 1982).

ICAL research has abandoned CAI's early objective of providing total courses, and has concentrated on building systems which provide supportive environments for more limited topics (Brown & Sleeman, 1981). Since the learning paradigm of ICAL emphasizes learning by doing, it seems unlikely that the computer-based tutor will be able to handle all situations that

arise. It is probable that the main use of computers in the classroom over the next decade will not be for delivery of complete curricula but as an adjunct to a teacher-directed curriculum.

2.6 Summary

Since its inception about a decade ago with an history not relatively short for a computer-related field, ICAL research has made some significant advances through the milestones described above. Difficult conceptual issues still remain to be solved before the use of ICAL systems can be of educational benefit. Most of the ICAL programs are experimental and we can only speculate whether or not these programs will find their way into the classroom. In the short run, one of the key limitations to the widespread use of ICAL systems is the heavy and exacting demand placed on processing power, memory size and special peripheral devices, compared to the availability of these resources. In the long run, the technological context is likely to change in ways which will facilitate the introduction of complex systems at all levels in the educational system.

In the next few years what we can also expect to see is some further refinement of our information-processing models of complex human activities such as seeing, learning and thinking. ICAL as an application area for AI is likely to lead to the further development of AI techniques. On the other hand, exciting AI ideas, methods and techniques are percolating through into the CAL field.

Chapter 3. Computer-Aided Testing, Evaluation and Advice (CATEA)

3.1 An Experiment in Writing a CAL System

We have seen that one of the main foci of ICAL research is the diagnosis of misconceptions or bugs in understanding. One obvious application of work done in this area is in the field of interactive computer-aided testing (ICAT, Cartwright & Derevensky, 1976) in which the computer is used to administer a test to a student, evaluate his answer and provide immediate feedback after each test item. An ICAT system can thus be used for teaching, learning, diagnosis and evaluation.

Research in the diagnosis of student errors has emphasized the need to know the types of conceptual bugs students are likely to have. Based on this notion, we wish to propose the design and implementation of an ICAT program. The objectives of writing such a CAL program are :

(1) to go through an experiential cycle of developing, testing and evaluating a CAL program;

(2) to evaluate how enumerating bugs and representing them can help in diagnosing a student's misconceptions;

(3) to attempt some limited use of AI techniques in developing the CAL program.

The domain of interest has to be restricted so that the task of writing a CAL program is not overwhelming.

We propose to consider the assignment statement in the Pascal programming language, explore the misconceptions students have in learning the statement, and write a program to diagnose these misconceptions. In the next section, we first look at the

difficulties novices have with the assignment statement.

3.2 Novices' Misconceptions of the Pascal Assignment Statement

3.2.1 Bayman and Mayer's work

In recent years, some work has been done on the subject of how novices learn programming (Boulay & O'Shea, 1981; Soloway et al, 1982; Bayman & Mayer, 1983; Joni et al, 1983). The general objective of these works has been to study misconceptions in understanding in the minds of novice programmers. Of particular relevance to this thesis is the work done by Bayman and Mayer which focuses on 'what is learned' when a student is exposed to his first programming language such as BASIC or Pascal.

Bayman and Mayer postulate that the outcome of learning can be viewed in two distinct ways :

(1) Learning BASIC or Pascal involves the acquisition of new information and new rules, such as how to use quotes in a PRINT or WRITE statement;

(2) Learning BASIC or Pascal involves the acquisition of a mental model, such as the idea of memory locations for holding numbers.

They explore the idea that learning a programming language involves more than just the acquisition of specific facts, rules or skills. The beginning student also develops mental models for the language in the process of learning the essentials of the language. A mental model refers to the student's conception of the 'invisible' information processing that occurs inside the computer. Bayman and Mayer (p. 677) note that :

> Most instructional effort is directed solely at helping the learner acquire the new information and behaviours without giving much guidance to the learner for the acquisition of useful mental models.

A study was carried out in which 30 undergraduate students learned BASIC through a self-paced, mastery manual and simultaneously had hands-on access to an Apple 2 computer. After instruction, the students were tested on their mental models for the execution of each of nine BASIC statements.[1] The results show that beginning programmers, although able to perform adequately on mastery tests in program generation, possessed a wide range of misconceptions concerning the statements they have learned.

In a wider perspective, Honi (1983) provides a categorization for the types of misconceptions that manifest themselves as bugs in novice programmers' code :

(1) A clash between a student's preprogramming knowledge and his budding computer knowledge resulting in a bug;

(2) Faulty/incomplete understanding of programming concepts. This general category can be broken down into :

   a. Overgeneralization of a concept : students have difficulty discerning the specific context in which a concept is appropriate;

   b. Hazy understanding of a concept, that does not manifest itself in simple programs, but comes to light only when more advanced topics are introduced;

---

[1]Of the nine statements, the assignment statement was the fourth most difficult statement based on the proportion of incorrect conceptions, after the INPUT, READ and IF statement.

c. Simply not knowing the rules of programming discourse that guide the composition of programming plans into understandable and executable programs;

(3) Difficulties arising from the coordination of multiple constructs;

(4) Students may decompose the problem differently from that which was intended.

In this thesis, by focusing on the learning of one programming statement, we have excluded from discussion any consideration of 2c, 3 and 4.

3.2.2 The Pascal Assignment statement

The first statement a student learns in his first programming language is most possibly the assignment statement, which is also the most frequently used statement in programming. As one of the most common languages taught in first-year computer science undergraduate curricula is Pascal, we decided to consider how the assignment statement in Pascal is taught, learned, understood or misunderstood.

Appendix A shows how Programming in Pascal (Grogono, 1980), the text for Computer Science (CPSC) 114, "Principles of Computer Programming I", taught at the University of British Columbia, presents the assignment statement. It emphasizes that execution of an assignment statement results in the left-hand operand getting the value of the right-hand side. One might think that the example "nextnumber := nextnumber+1" will force a correct mental model of the assignment statement (not confusing it with the algebraic statement). From their empirical study,

Soloway et al (1982) hypothesize that students might learn or memorize the counter variable update (I := I+1) as an indivisible unit or pattern. In other words, they do not decompose (I := I+1) into a left-hand variable which has its value changed by the right-hand expression, and thus are not viewing (I := I+1) as an example of the assignment statement.

Appendix B shows the 3 lecture slides used by instructors of the same course for teaching the statement. We observe that the slides' presentation is more comprehensive, in that they try to answer many doubts or misunderstandings that may arise in the mind of a student. By tabulating the values of variables before and after execution of an assignment statement, they attempt to illustrate vividly a correct model for the statement.

If we conceptualize the assignment statement as a list of transactions, a correct model for the assignment statement A := B involves the following transactions :

(1) Find the number in memory location B;

(2) Erase the number in memory location A;

(3) Write the number found in (1) into memory location A.

An expert programmer may have developed an accurate conception for a programming statement such as the assignment statement. An novice may differ in his mental model for the same statement or he may lack a coherent mental model. It may be that the beginning student starts off with an incorrect model and as he acquires new programming knowledge or more programming experience, he discovers, refines or corrects his incorrect model on his own. We argue that the teaching process should expedite the early formulation of a correct model in the

student's mind, instead of permitting him to harbour an incorrect or incoherent one and leaving it to other 'educational' media to rectify. How much training one needs to acquire a mental model has not yet been explored in research. A first step in addressing this issue is to study the novice user's understanding of newly learned programming statements (Bayman & Mayer, 1983).

Based on encounters with novices' programs, we attempt to enumerate the possible bugs or misconceptions of the assignment statement. The first bug is the interpretation of A := B to mean assigning the value of variable A to variable B. We postulate that this semantic bug is possible, though bizarre, as we have seen manifestations of it in examinations of students who have actually gone through half a term or one whole term of Pascal programming.

The second bug reflects the confusion of the equal sign in traditional algebra with the assignment operator (Soloway et al, 1981). Soloway et al (1982) observe that a possible manifestation of this bug is writing a Running-Total update using lines of code (Y := X + Z ; Z := Y) instead of just (Z := X + Z). While the presence of such statements as A := A+1 in a student's program may indicate the lack of this bug, we cannot conclude that the student has this bug from the absence of such statements. We postulate that a clear manifestation of this bug is the expectation that execution of the lines (A := 1; A := B; B := 2) will result in both A and B attaining values of 2.

The third bug possibly reflects a hazy understanding of the

assignment statement. Some students when first encountering the task of swapping the values of 2 variables often come up with (P := Q; Q := P). They expect P to attain the value of Q, yet somehow retains its previous value. Conceivably, they have extended the analogy of a variable as a mailbox that can hold one letter at a time to a mailbox that can hold several letters. We conjecture that it is also possible that they have taken a variable to mean a queue. Thus P := Q would remove the first value of Q and append it to the queue of P. This could explain how (P := Q; Q := P) can be expected to swap the values of P and Q.

The fourth bug reflects a confusion of the mailbox with its mail. Thus A := B is taken to mean that character B, not the value of B, is put into variable A.

The first three bugs are used as the foundation for a ICAT program that will ask student to write simple Pascal programs that illustrate use of the assignment statement. The fourth bug is not tested as students who have only just learned the assignment statement may not have known character types in Pascal. The program will attempt to diagnose manifestations of these bugs and possibly detect undiscovered bugs. An empirical study of CATEA was carried out with some 47 volunteer students of CPSC 114 who had just been taught the assignment statement in class. In the next chapter, we shall describe the design and implementation of CATEA. In Chapter 4, we shall describe the results of the empirical study.

Chapter 4. Design and Implementation of CATEA

4.1 Overall Structure

The goal of CATEA as an ICAT program is twofold : evaluation and diagnosis. As an evaluation tool, CATEA tests the ability of students to write simple introductory programs. As a diagnostic tool, CATEA suggests misconceptions in the student's mind which underlie incorrect programs. With the list of our three enumerated bugs, each of which incorporates an incorrect model of the assignment statement, we design three ostensibly simple programming assignments (hereafter called problems) for which the student has to write a program. The problems (Figure 3) are designed so as to entrap the student into manifesting each of our enumerated bugs in their programs, should he possess the underlying misconception.

The structure of CATEA is shown in Figure 4. CATEA displays a problem and the student's response will be a typed-in program. CATEA will attempt to compile the program; if there is unrecoverable syntax error(s), it will prompt the student to try again. Otherwise, CATEA will execute the program. If correct, the student receives a message to that effect and proceeds to the next problem. If incorrect, the diagnostic loop will be invoked.

```
Problem 1 :
Write a Pascal program that does the following :
   (1) Declare X, Y as integer variables
   (2) Assign X the value of 1
   (3) Assign the value 2 to Y
   (4) Assign X to Y
   (5) Write out the values of X and Y.

Problem 2 :
Write a Pascal program that does the following :
   (1) Declare A, B as integer variables
   (2) Assign the value 1 to A
   (3) Assign the value A to B
   (4) Now, make A and B have the value 2
   (5) Write out the values of A and B.

Problem 3 :
Write a Pascal program that does the following :
   (1) Declare P, Q as integer variables
   (2) Read in the value of P and Q
   (3) Now, exchange the values of P and Q
   (4) Write out the values of P and Q.
```

Figure 3. Three Problems tested by CATEA

The diagnostic loop (Figure 5) will attempt to detect the underlying bug which causes the incorrect program. If it cannot, it will display a 'Fail-to-diagnose' message and move on to the next problem. If it can diagnose a bug, remedial advice (prestored for each bug) will be displayed followed by a multiple-choice test. The test is used to find out if the student has gained a better understanding, that is, become aware of the bug. If the student's answer to the test is correct, he will proceed to the next problem or he can retry the same problem. If the student's answer is wrong, he will be offered a choice of more remedial advice or a second multiple-choice test. Should he get this test wrong again, the diagnostic loop will end with a warning that he has yet to correct his misconception

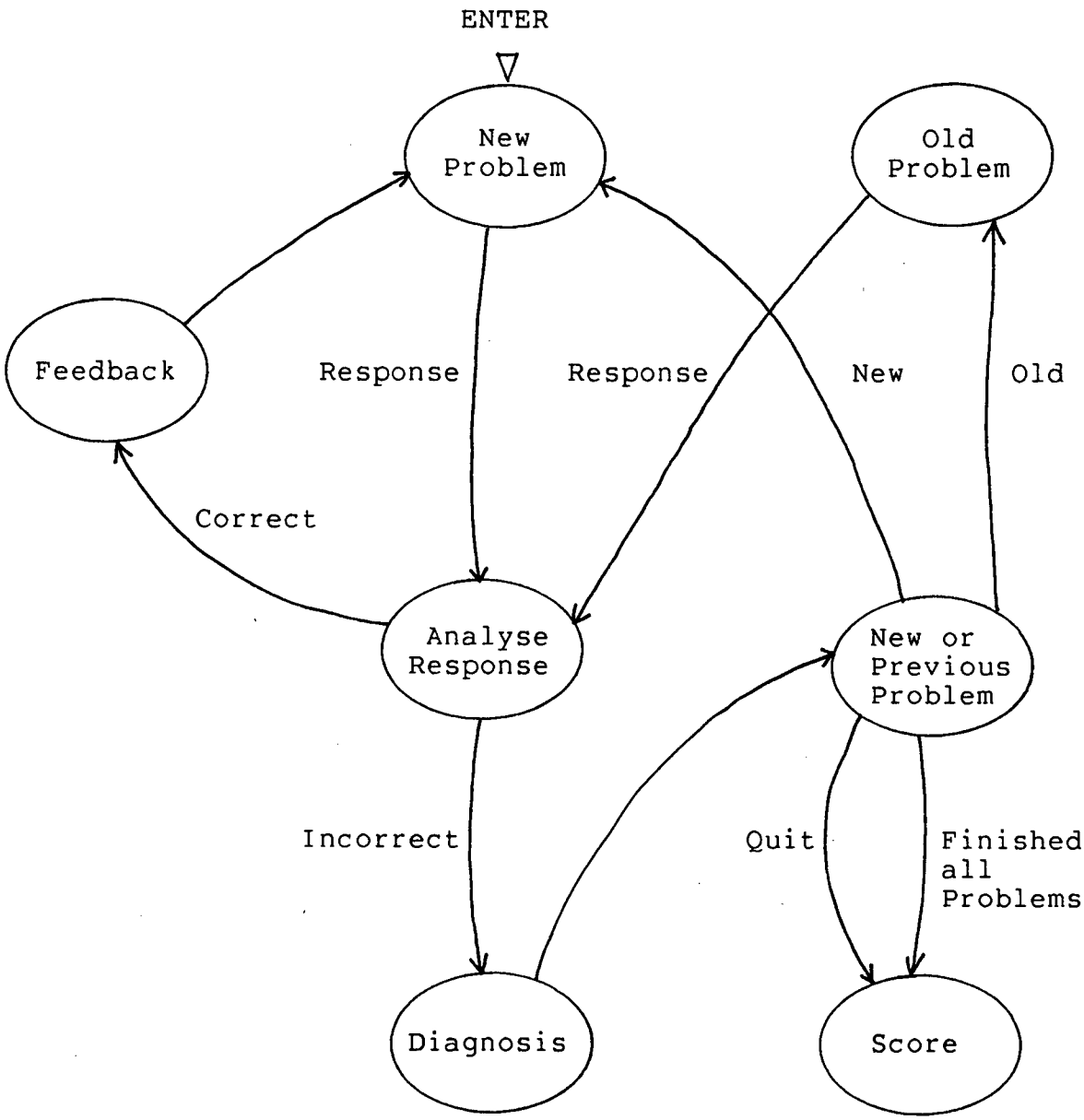

Figure 4. Structure of CATEA

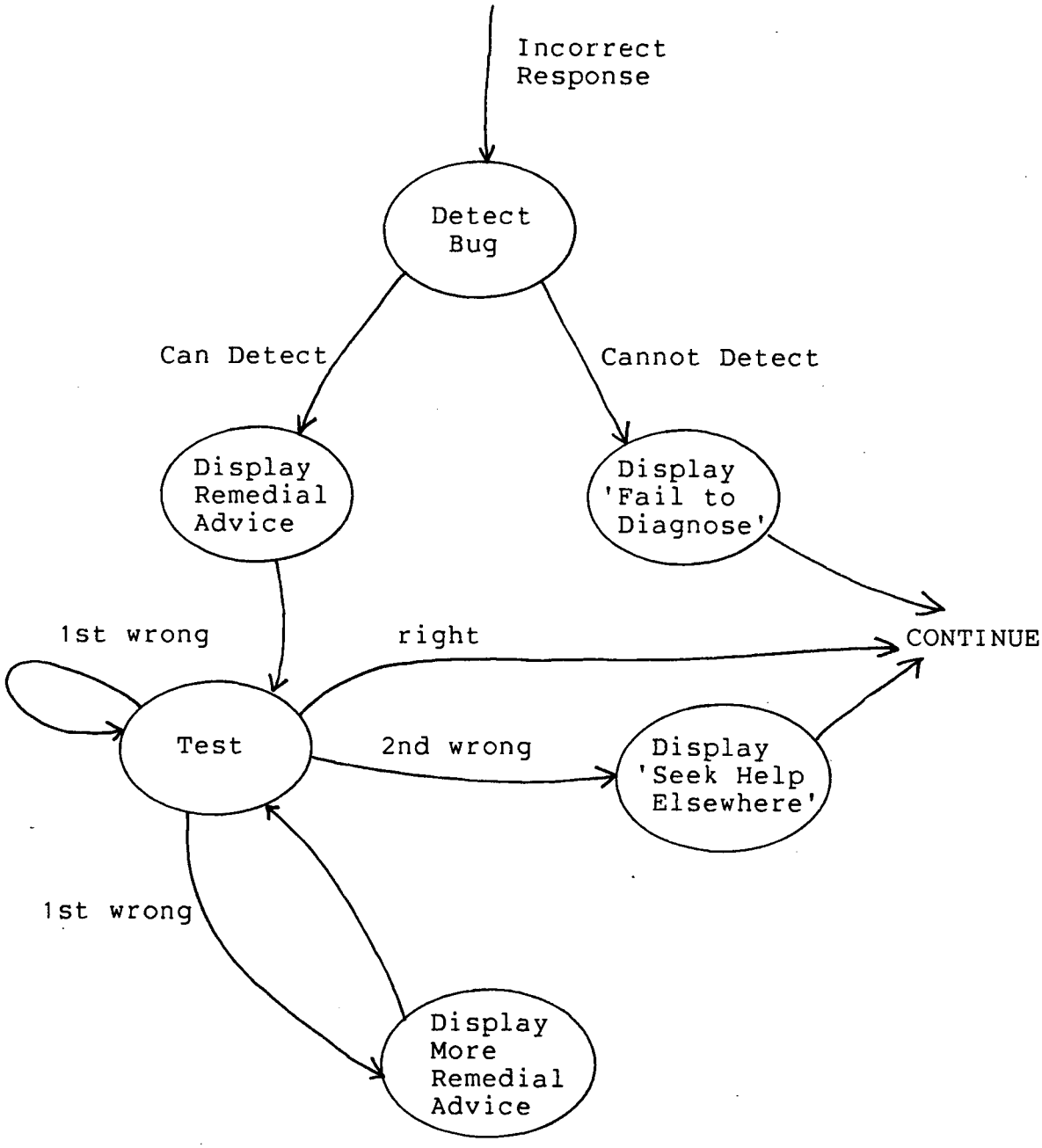

Figure 5. Diagnostic Loop of CATEA
(Blow-up of 'Diagnosis' bubble in Figure 4)

and advises him to try other remedial sources.

When all questions have been attempted or the student decides he has had enough, he will receive a score which indicates how many of the three problems he has coded correctly. If CATEA decides that the student still has misconceptions, the session ends with a summary of these misconceptions. In Appendix C, we illustrate the diagnostic loop in greater detail by tracing through a student's session as he attempts Problem 3 (see Figure 3).

Two important components of CATEA are the Pascal compiler and the diagnostic model. They are described in the next two sections.

4.2 The Pascal Compiler

As our focus is on programs that incorporate only the use of the assignment statement, we consider a subset of standard Pascal for our compiler. Thus, it will accept as valid only a simple main program comprising READ, WRITE or/and assignment statements. In the constant definition part, the only constants considered valid are integer or boolean values. In the variable declaration part, the only types considered valid are INTEGER and BOOLEAN.

The Pascal compiler is designed using the method of recursive descent (Davis & Morrision, 1981). The method centres around the syntax analysis phase which is divided into a number of recognition procedures, each of which has the task of checking whether a particular kind of phrase is present in the input. Each recognition procedure can call upon the services of

other ones to recognise the appearance of subphrases.

The error diagnosis and recovery scheme is that of Turner (1977) in which an error is detected when the compiler encounters a symbol in the input that is not the goal symbol. Syntax analysis continues by skipping the input until the next goal symbol and the input symbol match; this balances the syntax tree and the input. All syntax error messages have been written to make them as lucid and precise as possible.

The target machine code will be LISP structures. For example, the statement (X := 1) will be compiled into the LISP list (ASSIGNSY X 1), which we call a clause. Thus a program will be compiled into a list of such clauses which we call a clause-list and will be the main object for the next phase of program execution.

A spelling corrector is incorporated to detect and correct spelling errors of Pascal reserved words. Spelling checking of a word from the input is done only against the set of reserved words that can be expected to occur in that position of the syntax tree. The spelling correction algorithm is taken from the Ada implementation of a spelling corrector for an user interface of Durham, et al (1983). The spelling errors it looks for are transposition of two adjacent letters, one letter wrong, one letter extra and one letter missing. As the number of reserved words in our Pascal subset is very small, the spelling corrector will match at most one candidate and so correction will be done automatically.

4.3 The Diagnostic Model

For the diagnostic model, we draw upon the work of Brown & Burton (1978). Brown & Burton use a sophisticated procedural network model to represent the correct procedures of a skill. To diagnose bugs, the network is used to simulate the behaviour of buggy variants of the procedures in the network and note those which generate the same behaviour as exhibited by the student (BUGGY, Brown et al, 1977). In CATEA, we adopted a much simplified diagnostic model. A procedural network is not deemed necessary as understanding the assignment statement is unlike understanding a procedural skill such as addition or subtraction. To diagnose bugs, CATEA simulates the behaviour of an incorrect model of the assignment statement to find out if the model explains the surface behaviour of the student's program, that is, the student's mental model which leads him to think that his program is correct when it is not. We feel that this diagnostic strategy is adequate for our purposes of diagnosing bugs in our narrow domain area.

The output of the compilation phase will be a clause-list containing a clause for each executable program statement. CATEA will first execute the program to find out if it is correct. It will execute each clause in the clause-list sequentially, assigning values to variables as specified by each assignment statement or read statement. Program correctness will be determined by matching the values of predetermined variables to prestored values. For example, in Problem 2 a correct program (A := 1; B := A; A := 2; B := 2) will put values of 2 into variables A and B. If the student's program also put values of 2

into A and B, it will be deemed correct, otherwise incorrect.

If a student's program is evaluated to be incorrect, the next task of CATEA will be to find out which of its enumerated bugs can explain the behaviour of the program. CATEA will simulate an incorrect model of the assignment statement and execute the clause-list again. If this execution assigns values to predetermined variables that match that of prestored ones, then CATEA will postulate that the incorrect model is the source of the bug. Thus, if a student indicates the presence of the second enumerated bug, he may code (A := 1; B := A; A := 2) for Problem 2 (see Figure 3, p. 52) which specifies that A and B should attain values of 2. He thinks that B := A is like an algebraic statement and therefore A := 2 will also change the value of B to 2. Let us see how CATEA simulates the second bug in the student's program. Executing A := 1, it assigns 1 to A. Executing B := A, it assigns the value of A, that is, 1 to B. Executing A := 2, it assigns 2 to A, searches for preceding assignment statement(s) that has A in its right-hand side, finds the statement B := A, and assigns 2 to B as well. In the student's mind, his program will end up with values of 2 in both A and B, and therefore works.

If no model can explain an incorrect program's behaviour, then CATEA would have failed to diagnose the underlying bug. The three problems were designed so as to minimize the possibility of more than one bug that can equally explain the behaviour of an incorrect program. Problem 1 is designed to entrap the first enumerated bug, Problem 2 the second bug, and Problem 2 the third bug. This also simplifies our diagnostic model by

obviating the need to consider interaction of two or more bugs.

The guiding metaphor of CATEA's diagnostic model is simulation of bugs. A more sophisticated metaphor is pattern matching at the plan level with a program solution, which is used in MENO-II (Soloway et al, 1981), an intelligent tutoring system being developed for novice Pascal programmers. We posit that the simulation approach is sufficient for our purposes to handle very simple non-looping programs. Moreover, this diagnostic approach allows CATEA to accept multiple ways of doing a task. For example, in Problem 2 (see Figure 3), CATEA will accept all these programs as correct :

```
(A := 1; B := A; A := A+1; B:=A)
(A := 1; B := A; A := 2*A; B:=2*B)
(A := 1; B := A; A := A+1; B:=B+1)
(A := 1; B := A; A := B+1; B:=B+1)
(A := 1; B := A; B := B+1; A:=B)[1]
```

If the fourth statement in each of these programs except the last one is removed,the program will be a manifestation of the second enumerated bug. For Problem 3 (see Figure 3), CATEA will accept these programs as correct :

```
(P := P+Q; Q := P-Q; P := P-Q)
(T1 := P; T2 := Q; Q := T1; P := T2)[1].
```

We also incorporate two demons in the design of CATEA. The first demon is invoked when redundant variables are used as in (Z := 1; ... X := Z) and Z is not used in the left-hand side of

---

[1]All of these programs are actual students' programs in our empirical study. Before the study, we had not thought of such programs as possible solutions to the problems.

any other assignment statement. A message will be displayed, saying that this could be equivalently coded as (X := 1) without introducing Z. The second demon checks for the incorrect use of the third variable in Problem 3. Thus if a student attempts the use of the third variable but does it incorrectly, CATEA will remark that his program is incorrect, but at the same time encourage him that he has correctly perceived the requirement of a third variable.

4.4 Implementation

CATEA is written in LISP/MTS (Wilcox & Hafner, 1974) and runs in 200K bytes (including the LISP interpreter) on an Amdahl V/8.

Chapter 5. An Empirical Study of CATEA

5.1 Description of Study

An empirical study of CATEA was carried out with the primary objective of exploring how well it can detect bugs in students' misunderstandings of the assignment statement. Subjects were volunteers enlisted from the Fall 1983 class of CPSC 114, Principles of Computer Programming I.

The first session of CATEA was administered to a group of 20 students, who had just been taught the assignment statement that very week. Feedback from these students and their session protocols prompted improvements which were later incorporated into CATEA. This modified version of CATEA was administered to a second group of 19 students, who had also been exposed to the assignment statement in the previous week. All of these 39 students had yet to code their first programming assignment, and so CATEA provided them an opportunity to write their first Pascal programs. Feedback from this second session also generated improvements in CATEA.

A week later, the third session of CATEA was administered to 9 students who had done their first programming assignments. Of these 9 students, one had tried CATEA before. This third session did not prompt any further refinement to CATEA.

Subjects were given a questionnaire survey form that canvassed their opinion of their interaction with CATEA. A hardcopy protocol listing of each student's terminal session was generated for subsequent evaluation purposes. In the next section, we describe the results of the empirical study.

5.2 Results of Study

In a session with CATEA, the student will be asked to write programs as solutions to a set of problems. For scoring purposes, based on his or her response for each problem, we classify each student into one of these categories :

(1) Those who program correctly. Their programs follow problem specifications and assign values to predetermined variables which match those of stored ones;

(2) Those who did not program correctly. Either their programs did not follow problem specification or the values of predetermined variables do not match those of prestored ones. We further separate this category into two subcategories :

   (a) Those whose errors CATEA can detect; that is, one of CATEA's enumerated bugs is instantiated in their programs;

   (b) Those whose errors CATEA cannot detect;

(3) Those who give up after some futile attempts and opt for a solution to be given;

(4) Those who did not attempt the problem and instead skipped it.

Table 1 tabulates the results of the first two groups of 39 students. We observe that fewer than half the students got Problem 1 or Problem 3 correct. Not surprisingly, Problem 3 which involves the swapping of variables' values proved the most difficult.

Table 2 shows the breakdown of students who programmed

Table 1. Performance of the first two groups of 39 students

	Problem 1	Problem 2	Problem 3
No. of students who programmed correctly	19 48.7%	26 66.6%	18 46.0%
No. of students who programmed incorrectly	17 43.6%	11 28.2%	16 41.0%
No. of students who opted for solution to be given	2 5.1%	1 2.6%	1 2.6%
No. of students who did not attempt	1 2.6%	1 2.6%	4 10.3%
Total no. of students	39 100%	39 100%	39 100%

Table 2. Breakdown of students in first two groups who program incorrectly

No. of students whose errors CATEA can detect	15 88%	6 55%	14 88%
No. of students whose errors CATEA can't detect	2 12%	5 45%	2 12%
Total no. of students who programmed incorrectly	17 100%	11 100%	16 100%

incorrectly into those whose errors CATEA can detect and those whose errors CATEA cannot detect. 88% (15 students) of students who did Problem 1 incorrectly coded the assignment of X to Y as X := Y, manifestations of our first enumerated bug. Most of the 55% (6 students) of students who did Problem 2 incorrectly, coded (A := 1; B := A; A := 2), manifestations of our second enumerated bug. The majority of the 88% (14 students) of students who did Problem 3 incorrectly coded (P := Q) or (P := Q; Q := P). We believe that this is the surface behaviour of our third enumerated bug, in which the students do not have a correct mental model of the assignment statement.

As each problem has detailed steps on what to write for the next statement, we feel that not all the errors are explainable simply as momentary 'mental slips'. Thus, we believe that the errors indicate misconceptions that students have about the assignment statement and cannot be accounted for by simply saying, "They make silly errors." In Problem 3, some of the students who code (P := Q) or (P := Q; Q := P) were given the hint that a third variable is needed, but they still could not produce the correct program. Some students were observed to still have difficulty even after CATEA has explained and shown them the correct way of swapping variables (R := P; P := Q; Q := R). This is in spite of the fact that swapping variables has already been taught to them in class. It was introduced immediately after the assignment statement was taught. One possible explanation is that these students see only a pattern of the three assignment statements, instead of the model (which imprints more on the mind) behind the statements.

Let us look at those students' programs which contain errors CATEA cannot detect. In the two cases of Problem 1, the programs were (X := 1; Y := 2) which ignore assigning X to Y. We fix CATEA to pattern match (ASSIGNSY Y X) in the clause-list so that it can detect and pinpoint this omission. In two cases of Problem 2, the programs were (A := 1; B := 2) and (A := 1; A := B; A := 2; B := 2). The second program assigns values of 2 to A and B which matches prestored answers and CATEA mistakenly took it to be correct. We patch this by detecting omission of the assignment statement that assigns A to B. For the other 3 cases of Problem 2, the programs were (A := B), (A := 1; B := A; A := 2*A; B := 2*A) and (A := 1; B := A; B := 2). For these, CATEA just offered its fail-to-diagnose apology and proceeded to the next problem. We patch this so that CATEA can now give the remedial hint that either (or both) A or B has not attained the value of 2, as required. For the two cases of Problem 3, the programs were (READ (P,Q)). It seems that these two students were at a loss of what to do. CATEA failed to say anything for these programs. We change CATEA so that it will tell the student that P has not attained the value of Q, and vice versa. The use of a third variable is only suggested if the student has attempted some more statements to swap values of P and Q, without using a third variable.

The results of the third group of 9 students are tabulated in Table 3. 22% did Problem 1 incorrectly, about half that of 43.6% for the first 2 groups. 22% did Problem 2 incorrectly, less than 28.2% for the first 2 groups. 44% did Problem 3 incorrectly slightly more than 41.0% for the first 2 groups.

Table 3. Performance of the third group of 9 students

	Problem 1	Problem 2	Problem 3
No. of students who programmed correctly	6 67%	6 67%	5 55%
No. of students who programmed incorrectly[1]	2 22%	2 22%	4 44%
No. of students who opted for solution to be given	0 0%	0 0%	0 0%
No. of students who did not attempt	1 11%	1 11%	0 0%
Total no. of students	9 100%	9 100%	9 100%

[1]CATEA was able to detect all of the students' errors

While the small sample size of this last group makes the results less statistically reliable, we feel that the results are interesting in that they indicate students even after they have written their first programs still harbour misconceptions. In this last session, CATEA was able to detect all incorrect programs and offer remedial hints or help. The students' protocols of all three sessions revealed no novel bugs that we have not enumerated, possibly because each of the problems is tailored to catch one of the enumerated bugs.

One student was both in the first group and third group. In her first session, she wrote (Q := P; P := Q) for Problem 3. CATEA diagnosed this and showed her the correct way of doing it (R := P; P := Q; Q := R). In her second session two weeks later, she still got it wrong by attempting (Q := P; J := P; P := Q). It seems that after her first session, she remembered a pattern of assignment statements that use a third variable but still could not conceptualize a correct model.

We summarize other observations we have gathered from the student sessions' protocols :

(1) Some students attempted using the multiple assignment statement (A,B := 2 or A+B := 2 or A := B := 2), even though they do not have any previous computing experience (with programming languages like PL/1 that has such a statement). One possible explanation is that the student extends his mental model of the assignment statement to include multiple assignments, seeking his own repair to an impasse on the problem (Repair Theory of Brown, 1980).

(2) Empirical studies of syntactic errors in users' Pascal

programs have been done elsewhere (Ripley & Druseikis, 1978; Pugh & Simpson, 1979). Our results (which apply to novices) reiterate two of their main findings :

(a) The most error-prone construct in Pascal is the (missing) semicolon;

(b) Spelling errors are fairly infrequent;

5.3 Questionnaire Survey

The results of the questionnaire survey are tabulated in Table 4.

Do you think CATEA has helped you in better understanding the assignment statement in Pascal ?		
YES 76%	NO 8%	PROBABLY 16%
Do you think CATEA has helped you in better understanding other features of Pascal ?		
YES 62%	NO 19%	PROBABLY 19%
Do you feel more confident in writing Pascal programs ?		
YES 65%	NO 11%	PROBABLY 24%

Table 4. Student Evaluation

On the whole, student reaction to CATEA has been favourable. It appears that students feel that this experience was beneficial, especially in improving their understanding of the assignment statement. The following questions were also in the questionnaire :

In what ways do you think CATEA can be improved?

What extra capability do you wish CATEA to have?

In response to these questions, 38% of the students suggest better program editing capability. Currently, if a student's program has unrecoverable syntax errors, he has to re-enter the program. In the terminal on which CATEA was run, this inconvenience can be partially alleviated by moving the cursor up to the same line in the previous program, making the modifications on that line, and then hitting <return> to enter the line, instead of retyping the line. As most students do not know how to make use of this keyboard feature, this might account for some of the students' complaints. Nevertheless, we learn something from this; that is, in a CAL program, the student do not like to be distracted by side-issues such as inefficient human-tutor interfaces that have no important bearing on the subject domain. Part of the students' energies may be vented or consumed in coping with a bad human-computer interface, and this makes them less inclined to learn or perform well the main tasks of the CAL program. Indeed, the interface between the tutor and the student is an important component of any CAL system.

Overall, evaluation of CATEA proved satisfactory as it was able to fulfill its limited objective of diagnosing misconceptions in the students' understanding of the assignment statement.

Chapter 6. Conclusion

While we did not set out in this thesis to examine a wide range of issues directly relevant to education, our study suggests some possibilities that are worth considering in the area of teaching novices programming. By looking at the kinds of errors that students make, we have focused on the learning level that they pass through as they learn the assignment statement. We feel that in the learning of any new topic, difficulties in the form of misconceptions do arise, and if the sources of the misconceptions are incorrect mental models, a good aid to learning is to provide correct explicit models. A good tutor should adopt a teaching strategy that anticipates the kinds of misconceptions that can arise and aids the formulation of correct mental models.

By choosing a narrow knowledge domain for our instructional program, we have avoided many difficult conceptual questions and assumptions that have arisen in the design of ICAL systems of much greater sophistication. The design of CATEA is ostensibly simple. In order to extend CATEA to be able to understand and debug programming features beyond the assignment statement, such as looping constructs, more sophisticated methods are needed. Nevertheless, we believe that CATEA is a demonstration of the potential tutorial skill which ICAL systems can manifest.

In sum, we have written an ICAT program which diagnoses a student's incorrect program by simulating enumerated bugs in the program and observing which bug explains its surface behaviour. An empirical study of CATEA shows that it was able to realise

the objective of diagnosing misconceptions in a student's understanding of the assignment statement.

References

Abelson, H. & diSessa, A. 1981. Turtle Geometry : The computer as a medium for exploring mathematics. Cambridge, Mass. : MIT Press.

Barr, A. & Feigenbaum, E. A. (eds) 1982. The Handbook of AI. Volume 2. William Kaufmann, Inc.

Bayman, P. & Mayer, R. E. 1983. A Diagnosis of Beginning Programmers' Misconceptions of BASIC Programming Statements. Sept 1983, CACM, Vol 26 No 9, 677-679.

Borich, G. D. & Jemelka, R. P. 1981. Evaluation. In O'Neil, H.F. Jr. (ed), CBI. A State-of-the-Art Assessment. Academic Press, 1981.

Borning, A. H. 1979. Thinglab - A constraint-oriented simulated laboratory. Ph. D. Thesis, Stanford University.

Boulay, B. Du & O'Shea, T. Teaching Novices Programming. In M. J. Coombs & J. L. Alty (eds), Computing Skills and the User Interface. Academic Press, 1981.

Brown, J. S., Rubinstein , R & Burton, R. 1976. A reactive learning environment for computer-assisted instruction. BBN report No. 3314, Cambridge, Massachussetts.

Brown, J. S. & Burton, R. R. 1975. Multiple representations of knowledge for tutorial reasoning. In D.G. Bobrow & A. Collins (eds), Representation and Understanding, Academic Press, 1975.

Brown, J. S. & Burton, R. R. 1977. A paradigmatic example of an AI instructional system. First International Conference on Applied Systems Research, New York , 1977.

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

Brown, J. S. & Sleeman, D. H. 1981. Intelligent Tutoring Systems. Academic Press, 1981.

Brown, J. S. & Vanlehn, K. 1980. Repair theory : A generative theory of bugs in basic mathematical skills. Cognitive Science 4(4), 1980, 379-426.

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

Burton, R. R. 1981. DEBUGGY : Diagnostic of errors in basic mathematical skills. In D. H. Sleeman & J. S. Brown (eds)

Intelligent Tutoring Systems, Academic Press, 1981.

Carbonell, J. R. 1969. Interactive non-deterministic CAI. International Symposium on Man-Machine Systems, 1969.

Carbonell, J. R. 1970. Mixed-initiative man-computer instructional dialogues. Technical Report 1970, BBN, 1970.

Carr, B. & Goldstein, I. 1977. Overlays : A theory of modelling for CAI. AI Memo 406, MIT, 1977.

Cartwright, G. F. & Derevensky, J. L. 1976. ICAT : A feasibility paper. Paper presented at the Annual Conference of the Canadian Educational Research Association. Laval University, Quebec.

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

Clancey, W. J. 1979b. Dialogue management for rule-based tutorials. In Proceedings of the Sixth IJCAI, 1979.

Clancey, W. J. 1979c. Transfer of rule-based expertise through a tutorial dialogue. Ph. D. Thesis. STAN-CS-769, Stanford University, 1979.

Clancey, W. J. 1981. Methodology for building an intelligent tutoring system. STAN-CS-81-894, Stanford University, 1981.

Clancey, W. J. & Buchanan, B. 1982. Exploration of teaching and problem-solving strategies, 1979-1982. STAN-CS-82-910, Stanford University, 1982.

Colby, K.M. 1973. Simulations of belief systems. In Colby, K.M. & et al (eds), Computer Models of Thought and Language. San Francisco.

Collins, A. 1977. Processes in acquiring knowledge. In R.C. Anderson & et al (eds), Schooling and the acquisition of knowledge, Erlbaum Associates, 1977.

Davie, A. J. T. Recursive Descent Compiling. John Wiley & Sons, 1981.

Durham, I., Lamb, D. A. & Saxe, J. B. 1983. Spelling Correction in User Interfaces. CACM, Oct. 1983, Vol 26, No 10.

Feurzeig, W. & Papert, S & et al, 1969. Programming languages as a conceptual framework for teaching mathematics, Volumes 1 to 4, BBN, Cambridge.

Fielden, J. 1977. The financial evaluation of NDPCAL. British Journal of Educational Technology, 8, 190-200.

Gable, A. & Page, C. V., 1980. The use of AI techniques in CAI :

an overview. The International Journal of Man-Machine Studies , 12, 1980.

Genesereth, M. 1981. The role of plans in ITS. In D. H. Sleeman & J. S. Brown (eds), Intelligent Tutoring Systems, Academic Press.

Grogono, P. 1980. Programming in Pascal. Addison-Wesley, 1980.

Joni, S.-N., Soloway, E. et al, 1983. Just so stories : How the program got that bug. SIGCUE Bulletin, Fall 1983.

Howe, J. A. M. 1978. AI and CAI : Ten years on. In N. Rushby (ed), Selected Readings in CAL, Kogan Page, 1981.

Howe, J. A. M 1979. Learning through model-building. In D. Mitchie (ed), Expert Systems in the Microelectronic Age. Edinburgh Press.

Kay, A. C. 1977. Microelectronics & the personal computer. Scientific American , 237.

Kehler, T. 1982. Future directions for Computer-assisted education. In C. Hernandez-Logan & M. Lewis (eds), Computer Support for Education. R & E Research Associates. 1982.

Kimball, R. B. 1973. Self-optimizing computer-assisted tutoring. Theory & Practice. Technical Report 206. Institute for Mathematical Studies in the Social Sciences, Stanford University.

O'Shea, Tim 1979. Self-improving teaching systems. Ph. D. Thesis, Birkhauser, 1981.

O'Shin, L. 1980. CAI in Israeli disadvantaged elementary schools. Centre for Educational Technology, Ramat Aviv, 1980.

Papert, S. 1971. Teaching children thinking. MIT AI Lab Memo 247, 1971.

Papert, S. 1980. Mindstorms. Children, Computers and Powerful Tools. Basic Books, 1980.

Pugh, J. & Simpson, D. 1979. Pascal errors - empirical evidence. Computer Bulletin, 2, 26-28.

Resnick, L. B. 1977. Holding an instructional conversation. In R.C. Anderson & et al (eds), Schooling and the Acquisition of Knowledge , Erlbaum Associates, 1977.

Ripley, G. D. & Druseikis, F. C. 1978. Statistical analysis of syntax errors. Computer Languages, 3, 227-240.

Schank, R. & Abelson, R. Scripts, Plans, Goals and Understanding. Erlbaum Associates, 1977.

Self, J. A. 1974. Student models in CAI. The International Journal of Man-Machine Studies 6, 1974.

Sleeman, D. H. & Smith, M. J. 1981. Modelling student's problem-solving, AI 16.

Sleeman, D. H. 1975. A problem-solving monitor for deductive reasoning tasks. The International Journal of Man-Machine Studies 7, 1975, 183-212.

Smallwood, R. D. 1962. A decision structure for teaching machines, Cambridge, Massachussetts : MIT Press.

Soloway, E., Lockhead, J. & Clement, J. Does computer programming enhances problem solving ability? Some positive evidence on algebra word problems. In R. Seidel, R. Anderson & B. Hunter (eds), Computer Literacy, Academic Press, 1982.

Soloway, E., Ehrlich, K., et al, 1982. What do novices know about programming? In A. Badre & B. Shneiderman (eds), Directions in Human/Computer Interaction, 1982.

Soloway, E., Woolf, B. , et al. MENO-II : An intelligent tutoring system for novice programmers. IJCAI, 1981.

Statz, J. 1973. Problem-solving in LOGO. Syracuse University LOGO Project, New York.

Suppes, P. 1967. Some theoretical models for Mathematics Learning. Journal of Research & Development in Education, 1.

Stansfield, J. L. 1974. Programming a dialogue teaching situation. Ph. D. Thesis, School of AI, University of Edinburgh : Edinburgh (8).

Turner, D. A. 1977. Error diagnosis & recovery in one-pass computers, Information Processing Letters, Aug 1977, 6, 4, 113-115.

Uhr L. 1969. Teaching machine programs that generate problems as a function of interaction with students. Proceedings of 24th ACM National Conference (8).

Wilcox, B. & Hafer, C. 1974. LISP/MTS User's Guide, Mental Health Research Institute, Ann Arbour, 1974.

Woods, P. & Hartley, J. R. 1971. Some learning models for arithmetic tasks and their use in CAL. British Journal of Educational Psychology, 41, 1, 1971.

Vanlehn, K. & Friend, J. 1980. Results from DEBUGGY : An analysis of systematic subtraction errors. Xerox Palo Alto Science Center Technical Report, 1980.

Venezky, R. L. Evaluating CAI on its own terms. In A.C. Wilkinson (ed), Classroom Computers and Cognitive Science,

Academic Press, 1983.

Zinn, K. L. 1978. An overview of current developments in CAL in the US. In N. Rushby (ed), Selected Readings in CAL, Kogan Page, 1981.

Appendix A : How Programming in Pascal (Grogono, 1980) presents the Assignment Statement

An assignment statement has the form

```
variable := expression
```

The assignment statement is asymmetric : right hand operand answers the question 'what is the value?' and the left hand side answers the question 'to what value is this value to be given?' It is possible to write assignments statements such as

```
firstnumber := 1
circumference := 2 * pi * radius
```

and even

```
nextnumber := nextnumber + 1
```

which has the effect of increasing the value of nextnumber by 1. It is not meaningful, however, to write statements like

```
1 := firstnumber
length * width := area
```

because the left hand sides of these statements cannot be interpreted as a destination for a value.

Appendix B : How Lecture Slides used in CPSC 114 present the Assignment Statement

ASSIGNMENT

The action of changing the value of (i.e. "contained by") a variable.

This action is specified, in Pascal by an assignment statement.

FORM :

```
variable := expression
```

ACTION : The value of 'expression' is computed and becomes the new value of 'variable' (replacing any former value).

Once a value is assigned to a variable, it retains that value until another value is assigned to it in a subsequent step of the program.

Expressions may contain variables as well as constant values. The value represented by a variable is the value most recently assigned to it.

Notes

- Variables don't remember previous values
- Read "a := 3" as "a becomes 3", or "a gets the value 3"; don't read it as "a equals 3".

Examples

1) a := 3;

Before		After	
a		a	3

2) x := x + 1;

Before		After	
x	82	x	83

(Try "x := x + 1" on a mathematician !)

Note x := y and y := x have different effects.

Appendix C : The Diagnostic Loop illustrated by tracing a student's session as he attempts Problem 3

The following is part of an actual student's session protocol annotated with explanatory notes, which can be distinguished by a 'C' as the first character of their lines.
CATEA prompts input from the student with a '*'

```
>
>   Choose one of the following :
>
>   (1) Please display restrictions of the Pascal compiler.
>
>   (2) Try Problem 2 again for more practice.
>
>   (3) Proceed directly to the next problem.
>
>   Enter 1,2 or 3 :
*3
>
>   Programming Assignment 3
>   ------------------------
>   Write a Pascal program that does the following
>     (1) Declare P, Q as integer variables
>     (2) Read in the value of P using READ (P);
>     (3) Read in the value of Q
>     (4) Now, exchange the values of P and Q
>     (5) Write out the values of P and Q
>
>   Now, enter your Pascal program :
>
*PROGRAM TIRED;
*   VAR
*     P,Q :INTEGER;
*   BEGIN
*     READ (P,Q);
*     Q :=P;
*     P :=Q;
*     WRITE (P,Q);
*  END.
C
C CATEA will attempt to compile the program.
C If there are unrecoverable syntax errors,
C it will prompt the student to re-enter the program.
C If a student still can not type in a program free of syntax
C errors after six attempts, CATEA will ask if he wants to give
C up and be provided with the solution.
C
>   Start of Compilation.
>   PROGRAM TIRED;
>      VAR
>        P,Q :INTEGER;
```

```
>       BEGIN
>         READ (P,Q);
>         Q :=P;
>         P :=Q;
>         WRITE (P,Q);
>       END.
>
>    No error detected.
>    End of Compilation.
>
>    Do you want to make any more changes to your program ?
>    To make changes, you have to enter the program again.
>    Enter Y or N :
*N
>
>    To continue, hit any key and then the <RETURN> key.
*x
C
C As the student's program is syntactically correct,
C CATEA proceeds to execute it.
C
>
>    Start of Execution.
>    Executing   READ ( P , Q ) ;
>    Enter input value for  P  >
*5
>    Enter input value for  Q  >
*8
>    Executing   Q := P ;
>    Executing   P := Q ;
>    Executing   WRITE ( P , Q ) ;
>    Program will print :
>                 5     5
>    End of Execution.
>
C
C CATEA realizes that as P and Q have not swapped variables,
C the student's program is incorrect.
C It will then invoke its diagnostic model to find out which
C of its stored bugs can explain the surface behaviour of the
C student's program.
C
>
>    Your program doesn't work as required.
>    Do you know what's wrong with your program ?
>
>    To continue, hit any key and then the <RETURN> key.
*x
>
C
C CATEA points out where the student's program is wrong.
C
>
>    P has NOT attained the value of Q, namely, 8
>
>    To continue, hit any key and then the <RETURN> key.
```

```
*x
>
C
C CATEA was able to diagnose the bug as its third stored
C enumerated bug. It prints out a block of remedial advice
C (prestored for each bug) which is most often
C an example followed by some discussion, as in this case.
C
>
>   Remedial Advice
>   ---------------------------------------------------------------
>   Look at the following assignment statements :
>                P := 1;
>                Q := 2;
>                P := Q;
>                Q := P;
>                                                      P  |  Q
>                                                     ---------
>    P := 1 puts 1 into the variable P                 1  |  -
>    Q := 2 puts 2 into the variable Q                 1  |  2
>                                                      |
>    P := Q puts the value of Q, that is, 2            |
>    into P. P and Q both has the value 2.             2  |  2
>    Note that P := Q destroys the old value           |
>    of P, namely, 1. To retain this old               |
>    value of P, we need a THIRD variable.             |
>                                                      |
>    Q := P puts the value of P, that is, 2            |
>    into Q. BOTH P and Q have the value 2.            2  |  2
>   ---------------------------------------------------------------
>
>   To continue, hit any key and then the <RETURN> key.
*x
>
>   Let me give you a test to see if you have a better
>   understanding.
>
>   Question
>   ========
>   Consider the following program :
>   PROGRAM QUIZ;
>   CONST A=TRUE;
>         B=FALSE;
>   VAR X, Y : BOOLEAN;
>   BEGIN
>     X := A;
>     Y := B;
>     X := Y;
>     Y := X;
>     WRITE (X, Y)
>   END.
>
>   What gets printed ?
>    {1}   FALSE FALSE
>    {2}   FALSE TRUE
>    {3}   TRUE  FALSE
```

```
>     {4}   TRUE  TRUE
>    Type in your answer (1, 2, 3 or 4) :
*2
>    That's WRONG.
>    The answer is 1
>
C
C If the student's answer is correct, CATEA will take it that
C the student has become aware of the bug and proceed to the
C next problem or allow the student to retry the same problem.
C
C CATEA gives the student the choice of trying another test or
C seeking more advice.
C
>    Choose one of the following by entering 1 or 2 :
>       {1} More advice
>       {2} Try another test
>
>    Now, enter 1 or 2 :
*1
>
>    Remedial Advice
>    ---------------------------------------------------------
>    To exchange the values of 2 variables P and Q,
>       we use a third variable R in the following way :
>                  R := P;
>                  P := Q;
>                  Q := R;
>
>     R is used to store the value of P,
>       so that P can then get the value of Q.
>     After that, Q gets the value of R,
>       which is the original value of P.
>
>     Can you observe that P and Q have exchanged values ?
>    ---------------------------------------------------------
>
>    To continue, hit any key and then the <RETURN> key.
*x
>
C
C CATEA gives another test to find out if the student has become
C aware of the bug in his program.
C
>
>    Question
>    ========
>    Consider the following program :
>    PROGRAM QUIZ;
>    CONST A=2;
>          B=4;
>    VAR P, Q, R : INTEGER;
>    BEGIN
>      P := A;
>      Q := B;
>      R := P;
```

```
>      P := Q;
>      Q := R;
>      WRITE (P, Q)
>    END.
>
>    What line of output gets printed ?
>     {1}   4    4
>     {2}   2    2
>     {3}   2    4
>     {4}   4    2
>    Type in your answer (1, 2, 3 or 4) :
*4
>    That's RIGHT.
>    Let's proceed to the next problem.
>
C
C This ends the diagnostic loop.
C If the student's program answer is wrong, that is,
C he has failed to do the two multiple-choice tests correctly,
C CATEA will end the diagnostic loop with a warning that he has
C yet to correct his misconception and advises him to try other
C remedial sources.
C
>    To continue, hit any key and then the <RETURN> key.
*x
>
>
>    Choose one of the following :
>
>    (1) Please display restrictions of the Pascal compiler.
>
>    (2) Try Problem 3 again for more practice.
>
>    (3) Proceed directly to the next problem.
>
>    Enter 1,2 or 3 :
```