University-level

Computer-assisted Instruction at Stanford: 1975

by

Patrick Suppes, Robert Smith, and Marian Beard

TECHNICAL REPORT NO. 265

October 15, 1975

Psychology and Education Series

Institute for Mathematical Studies in the Social Sciences

Stanford University

Stanford, California

ACKNOWLEDGMENTS

The research reported here in logic, set theory, natural-language processing, computer-generated speech, and Chinese was supported by the National Science Foundation under grant NSF EC-43997. The development of the BASIC Instructional Program and the "keyword" mnemonic method was supported by the Office of Naval Research under contract ONR N00014-67-A-0012-0054, and by the Defense Advanced Research Projects Agency under ARPA Order 2284, Program Code 3D20. Research in computer-assisted instruction in Slavonic languages was supported by the National Science Foundation, the Office of Education, and Stanford University. The development of the computer-controlled music system was supported by Stanford University.

Contributions to the various sections of the report were made by Avron Barr, Gerard Benbassat, Lee Blaine, Edward Bolton, Michael Hinckley, Pentti Kanerva, Rosemary Killam, Paul Lorton, Lawrence Markosian, Freeman Rawson, William Sanders, Rainer Schulz, Richard Schupbach, Joseph Van Campen, Thomas Wolpert, and Peter (E-shi) Wu.

Table of Contents

1 Introduction

Work in computer-assisted instruction (CAI) at the Institute for Mathematical Studies in the Social Sciences, Stanford University, began in 1963. During the first decade, most of the work was concerned with the development of computer-based curriculums for elementary schools, especially in mathematics and reading. A history covering these efforts is to be found in Suppes (1972).

Beginning in the late 60's, work at the Institute began to focus more on the possibilities of CAI for university-level courses. The first major effort was under the direction of Professor Joseph Van Campen of the Department of Slavic Languages at Stanford, and was concerned with the development of two years of elementary Russian at the university level. The very positive evaluation results of the first-year Russian course are reported in Suppes and Morningstar (1969). The work in Slavic languages has now shifted from the beginning courses to intermediate courses that have small enrollment. Computer-assisted instruction is offered in such relatively esoteric topics as Old Church Slavonic and the linguistic history of the Russian language (see Section 3 below).

One of the earliest curriculum efforts for the elementary school was the development of a course in logic for gifted elementary school students. In the late 60s, efforts turned to the development of a corresponding course in logic for university students. Over the past several years, the continued development and improvement of this course has been one of the central efforts in CAI at the Institute, and is described in Section 2 of this report. Recently, the efforts in logic have moved to the intermediate level as a primary focus of research and development. A greater effort is now being put into axiomatic set theory, and future efforts at a similar level are anticipated.

Courses in music and computer programming have been developed for use at the university level as well. Individualized CAI in music has been applied to theoretical and instructional investigations in a number of different courses, as reported in Section 6. Previous CAI work in programming has included courses in the BASIC, AID, SIMPER, and LOGO languages; current projects deal with an entirely new approach to teaching BASIC and the integration of a CAI course in LISP into the university curriculum. Both projects are discussed in Section 4.

Summary information on the courses in Slavic languages, logic and set theory, music, and programming is shown in Table 1. It will become apparent at once that the enrollment is quite small in a number of courses that we consider as a primary focus of our research and development efforts. Contrary to some of our thinking a few years ago, it is now our deliberate choice to develop CAI for such courses rather than for the large enrollment elementary courses.

Table 1

University-Level Computer-Assisted Courses

at Stanford, 1972-75

Course	Number of Students per Academic Year 1972-73	1973-74	1974-75	Avg. Number of Student Hours at Computer Terminal
Philosophy 57 Introduction to Logic	56	160	209	70 for A 54 for PASS
Philosophy 161 Set Theory	---	---	12	51 for A 31 for PASS
Slavic Lang. 211 Old Church Slavonic	5	2	1	30
Slavic Lang. Bulgarian	3	0	1	35
Slavic Lang. 212 History of Russian Literary Language	4	4	4	30
BASIC Instructional Program	---	100	200	10 (1)
Computer Science 206 LISP	---	--- (2)	79	93 (3)
Music (ear-training)				
Music 21	---	42	41	(4)
Music 22	18	19	23	
Music 23	5	26	8	
Music 103	5	6	---	
Music 27	---	---	33	

NOTES.

(1) BIP students were limited to ten hours of time for experimental reasons. During unlimited pilot runs, students have taken up to 30 hours to complete the course.

(2) During 1973-74, LISP was taught at Stanford using the IMSSS machine; but students logged in as users, and there was no special CAI for LISP.

(3) LISP students spent an average of 69 hours in the LISP interpreter, and 24 hours in the LISP CAI system.

(4) The students had restricted terminal time.

1.1 Productivity in CAI

The reason for the emphasis on small-enrollment courses is to a very considerable extent a matter of productivity (Suppes, 1975). The initial impetus for the course in axiomatic set theory was the loss of a faculty position in this area due to budget cuts at Stanford. In the face of declining or fixed budgets, it has become apparent that faculty sizes will probably decrease during the remainder of this century. Some state universities require that a specific level of enrollment be maintained for all courses offered, and there is considerable pressure against specialized courses with low enrollments. Still, such courses represent an important function of the university in transmitting intellectual knowledge and skills from one generation to the next.

One of our main purposes in CAI research and development at the Institute is to show how these specialized courses can be maintained at reasonable cost in the future by appropriate use of computer technology. Our aim is to increase the teaching load of faculties in terms of courses offered, and one of our subsidiary aims, consequently, is to improve the ability of CAI to provide effective instruction with little or no intervention required of the instructor. A few teaching assistants, available at scheduled hours through the day and evening, work in conjunction with each course; the instructor himself is free to supervise further development of CAI courses or to consult on an individual basis with students and teaching assistants.

1.2 Problems of evaluation

Unlike the situation in elementary and secondary school, there is an almost nonexistent tradition of evaluation of achievement in college-level courses, especially courses that are beyond the initial introductory level in any subject. A broad survey of what has been done to evaluate alternative media of instruction at the college level is to be found in Jamison, Suppes, and Wells (1974).

Over the decade of work at the college level in computer-assisted instruction at Stanford, two introductory courses have received systematic evaluation. The first was a evaluation of the computer-based teaching of elementary Russian during the academic year1967-1968 and second-year Russian during the academic year 1968-1969. A detailed report of this evaluation is to be

found in Suppes and Morningstar (1969). In the case of the elementary Russian, a fairly strong superiority of the computer-assisted instruction was inferred from the evaluation data. We should remark that this kind of result is unusual; in most cases of evaluating alternative methods of instruction at the college level the method of instruction has proved to make insubstantial difference.

The "standard" conclusion holds for the other evaluation of college-level CAI at Stanford, namely, evaluation of the introduction to logic. A common final examination was given to the CAI group and to the group taught by usual lecture methods in the fall of 1973. The performance of the CAI students on the common final, which was written by the instructor in the lecture course, was slightly but not statistically significantly better than that of the students receiving traditional instruction. Detailed results of this evaluation are as yet unpublished.

The situation is rather different when one considers evaluation of courses like those in axiomatic set theory or Old Church Slavonic, which have at any given occasion a very small enrollment either when taught by traditional methods or by computer-based methods. There is in the university setting no tradition whatsoever of a systematic and objective evaluation based on reportable data for such courses. Because of the small number of students taking such courses and because of their highly specialized character, we anticipate that evaluation of the specialized courses on which we are now concentrating at Stanford will have to depend upon detailed reports of the work actually accomplished and reports of the attitude of the students toward this method of instruction. A detailed empirical summary of the students' work in the course in axiomatic set theory is now under preparation, and reports on some of the specialized courses in Slavic languages are referred to in Section 3.

1.3 Complex Instructional Systems

We have emphasized thus far in this introduction the importance of productivity considerations in our attitude toward computer-assisted instruction at the college level and in our approach to making it a viable and permanent part of college-level instruction. There is, however, an important corollary to this approach that needs to be emphasized and that generates a number of fundamental and intellectually challenging problems of computer science.

It seems likely that elementary courses of several kinds can be given without radically advancing the present techniques of computer-assisted instruction and without calling upon deeper methods of program construction for interaction between student and computer.

However, as we push toward increasingly tutorial topics and topics at an intermediate level of difficulty (exemplified, for instance, by courses that are primarily mathematically based in foundations such as the course in axiomatic set theory), the need for work at the frontiers of computer science becomes evident. The courses at this level will be viable and therefore productive only if the techniques of instruction are rich enough to provide a challenge to the students and a realistic range of options corresponding to what one might expect in such a course taught by traditional methods.

In the case of mathematically based courses, the primary need is clear. Techniques of proof that approach informal methods of mathematical argument are absolutely essential in order not to involve the student in an inordinate amount of tedious detail. Moving from formally explicit proofs to informal ones is a central intellectual problem of artificial intelligence and computer science: the construction of a model of informal mathematical reasoning. Our current efforts here have borrowed from artifical intelligence techniques of theorem proving and axiomatic representation of mathematical reasoning. It has also been necessary to add to these techniques, and our current proof checker for set theory (see Section 2) is believed to be the most sophisticated such program in the world.

Closely associated with the problem of informal proof methods is the problem of the language in which the student gives a proof, and the ways in which the program can explain and elucidate a proof to the student. A proof checker for these advanced courses must be able to accept proofs in an informal style that approaches a fragment of English. We feel that by concentrating on the restricted domain of mathematical proofs, it will be feasible to have a relatively rich and adequate use of natural language in this setting, and that with reasonable time and effort we shall be able to duplicate the kinds of use of natural language characteristic of the writing of mathematical proofs in the best textbooks.

A problem of greater difficulty is the development of techniques for conducting a dialogue between student and program about a given proof on which the student is engaged. We want the program to be able to make helpful and useful comments contingent upon the student's work. We want these comments to be relevant, pertinent and natural in tone. We do not believe that these dialogue problems will be solved in any reasonable completeness in the near future. There are as yet very few examples of natural dialogues between student and computer program. Two systems that show great potential are SCHOLAR (Carbonell & Collins, 1973) and SOPHIE (Brown, Burton, & Bell, 1974). While both of these programs are significant artificial intelligence programs, they have not yet been used on a regular basis by students in a standard educational setting. An important feature of our work at the Institute is that all of our efforts are directed toward real courses and answer known pedagogical needs.

Another interesting use of computer science techniques occurs in the BASIC Instructional Program (BIP), discussed in Section 4. Here the focus of our effort has been to select instructional material and order its presentation for the individual student. This is done by using networks representing skills and tasks, which are compared to profiles for each individual student.

The emphasis on the application of these computer science techniques to university instruction has necessitated a different programming environment than most other CAI projects. For example, we have chosen to use the highly interactive TENEX timesharing system on our PDP10 computer, and have implemented and extended several programming languages originally designed for artifical intelligence research (see Section 9). These languages contain data and control structures that are far more advanced than the structures found in traditional CAI languages such as TUTOR, PILOT, INST, and COURSEWRITER. In many cases, the programming effort has exceeded the curriculum effort, and this is understandable when we are dealing with the situation that the curriculum is relatively fixed and clear, while the necessary programming techniques need to be developed.

For example, in the BIP program, the curriculum is not a script of text and exercises to be presented to the student, but is instead organized in the task network. This means that the computer has a better model of the curriculum and is better able to make decisions about what exercises would be most useful to the student than would be possible with a standard CAI author language.

There is, of course, a certain cost in the development and application of computer science techniques to instructional purposes. We feel that the permanent viability of some of these specialized courses will almost demand that such techniques be employed. One crucial question concerns the amount of effort and expense involved in re-applying a technique already developed. Here, our current experience in using the set theory system EXCHECK (see Section 2) in the development of a course in the foundations of probability suggests that it is going to be quite feasible. Indeed, we envision in a few years that we will be able to implement new courses using the basic EXCHECK program with relatively little new effort.

This report describes our current efforts. This introduction is intended to put in perspective the overall viewpoint that is an important feature of our work. We do not look upon the use of computers for instruction at the university level only as a means of enhancing standard instruction, but rather as a much more fundamental and essential tool for meeting the pressing need for increases in productivity of faculty, and for maintaining important intellectual traditions of offering a variety of specialized courses in a great many different subjects to a small number of students.

2 Logic and Set Theory

Logic has been a traditional subject for CAI at Stanford since 1963 when the first computer-based logic course was demonstrated at IMSSS. For a history of these efforts, see Suppes (1972) and Suppes, Jerman, and Brian (1968). The current efforts of the Logic group focus on providing systems that allow the student to deal more naturally with the logical concepts that are being taught, and to extend the subjects covered into applications of logic.

A further and more ambitious project is the set theory system. Here, and in advanced mathematics generally, the reasoning process, while axiomatic and rigorous, is generally done with little explicit reference to logic itself. The system we have developed, EXCHECK, allows the student to construct proofs of the standard theorems of axiomatic set theory, using rules of inference at an appropriate level of complexity. Thus, proofs are natural and informal, but nevertheless fully rigorous.

Development of the EXCHECK program in the next year will focus on providing more informal ways of expressing mathematical arguments in terms of the overall structure that proofs have. Computer-generated audio (see Section 7) will be introduced into both LOGIC and SET THEORY as a way of differentiating the various levels of dialogue that are inherent in the presentation of CAI in logic and mathematics. For example, we intend to use audio output to distinguish the text of the arguments from comments about those arguments.

2.1 The Logic Course

The Stanford CAI logic course, Philosophy 57, has been run during the 1974-75 academic year using a rather more sophisticated program than was available during the previous years. The earlier work is reported in Goldberg (1971), Goldberg and Suppes (1972), Goldberg and Suppes (in press), and Suppes (1972); extensive analysis of the data are reported in Kane (1972) and Moloney (1972). A new program, written in 1974, embodies various features and developments that were impractical in previous versions.

The heart of the program is a highly sophisticated proof-checker, which verifies that lines entered by a student in a formal derivation do indeed follow, in the sense of logical validity, from the previous lines, by means of the particular justifying rules cited by the student.

2.1.1 Problem Types

The curriculum is basically linear except for the addition of several applications of logic, which are student options in the course. The curriculum contains the following kinds of problems:

1) QUESTION problems, in which the student must respond with one or more items from a list of possible answers, constitute the principal method of introducing textual explanations of various concepts. This part of the course follows the traditional CAI method of presenting concepts and information in "frames" and then asking questions about the material.

2) DERIVATION problems form a good portion of the work done by the student. The student is required to construct a formal proof of a given formula. The resources available to the student, naturally, are a function of his progress in the curriculum (new rules of inference and theorems are justified and made available in the course of the lessons), and of the formal theory he is working in. Thus, for example, the axioms and rules of the propositional calculus are later supplemented with the axioms for a commutative group.

An example of a derivation in the propositional calculus (characters in italics were typed by the student):

```
Derive:   NOT(S → NOT Q) → Q
*HYP
          (1) NOT(S → NOT Q)
*WP
          (2) NOT Q
*WP
          (3) S
*3,2CP
          (4) S → NOT Q
*2,1,4IP
          (5) Q
*1,5CP
          (6) NOT(S → NOT Q) → Q
Correct...
```

Here, *HYP* introduces an hypothesis, *WP* a working premise, *CP* closes a conditional proof, and *IP* closes an indirect proof.

3) INTERPRETATION problems are the principal tool in explicating the concept of validity in the predicate calculus. To show that an argument is valid, the student is required to construct a formal derivation of the conclusion from the premises; to show the argument invalid, he is required to find an interpretation of the formal predicates such that the premises are true and the conclusion false. Currently all interpretations are required to be in elementary number theory (integer arithmetic), with which the student has gained some facility by the time the interpretation problems are presented. The program has the machinery for handling the definition of a number of separate mathematical languages and theories simultaneously without logical confusion.

4) COMBINATION problems permit the student to choose whether to construct a proof or find an interpretation. Thus these problems, more than derivations or interpretations alone, require a studied approach to the question of the validity of a given argument and emphasize the semantic or model-theoretic basis for the decision over the combinatorial methods of constructing a derivation.

The combination-type problems are used to demonstrate other concepts which can be defined in terms of (first-order) validity. Thus the student may be required to show *consistency* or *inconsistency* of premises by finding an interpetation which makes all the premises true or by deriving a contradiction. *Dependence* or *independence* of axioms can be shown by selecting suitable derivations and interpetations.

2.1.2 Finding Axioms

In addition to the linear curriculum -- the "basic logic" section that all students must complete -- there are several "finding axioms" problems which increase in difficulty and form additional grade requirements. Each such exercise consists of a set of statements from which a limited number must be selected as axioms and the remainder proved from these axioms. All inference rules of first-order logic, but no non-logical axioms (other than those selected), are available to the student for these exercises.

```
Exercise 7 on Finding Axioms
B(X,Y,Z) means that the point Y lies between the points
X and Z on a line.  We still call it betweenness when X=Y
or Y=Z.
Take 5 of the 11 statements as axioms and prove the remainder
from these axioms.

1.  B(X,X,X)
2.  B(X,Y,X) → X=Y
3.  B(X,Y,Z) → B(Z,Y,X)
4.  X=Y → B(X,Y,Z)
5.  B(X,Y,W) & B(Y,Z,W) → B(X,Y,Z)
6.  NOT Y=Z & B(X,Y,Z) & B(Y,Z,W) → B(X,Y,W)
7.  B(X,Y,Z) & B(X,W,Z) → B(Y,W,Z) OR B(W,Y,Z)
8.  B(X,Y,Z) & B(Y,X,Z) → X=Y
9.  B(X,Y,Z) OR B(Y,Z,X) OR B(Z,X,Y)
10. NOT Y=Z & B(X,Y,Z) & B(Y,Z,W) → B(X,Z,W)
11. B(X,Y,X) → B(Z,Y,X)
```

The finding axioms (FA) exercises, being quite independent of the linear curriculum, are available to the student at any time (beyond a certain point). In general there is no single "correct" solution to an FA exercise, and consequently there is a premium on semantical considerations, as in the combination-type problems. Indeed students often spend considerable time away from the terminals, finding a solution to these exercises before constructing formal proofs with the computer.

2.1.3 Applications of Logic

Following the basic coursework in first-order logic, required of all students, there are several lesson sequences consisting of applications of logic to various topics. These lesson sequences are chosen at the student's option and are requirements for the A and B grades. Currently there are three application sequences: boolean algebra, qualitative probability, and social decision theory. Other sequences planned or under development are applications to legal reasoning, elementary set theory, and geometry. The sequences on legal reasoning and social decision theory are designed to be of particular interest to students who do not plan further work in mathematics or logic, and the boolean algebra/probability to those working in statistics or related fields.

The qualitative probability lessons develop Suppes' (1973a) non-quantitative foundation for

a probability measure, which can be largely formalized in first-order logic. In the social decision theory sequence the student investigates the logical relations among various ethical constraints on decision methods, and actually proves Arrow's famous "Impossibility Theorem." (See Arrow, 1966.)

2.1.4 The LOGIC Proof Checker

It should be pointed out that the instructional development of these and other formal theories in an interesting and useful way is possible only through a sophisticated proof-checker, which contains fairly powerful rules of inference such as the TAUTOLOGY rule (enabling the student to enter as a line of a derivation, in one step, any tautological consequence of results previously obtained), and the BOOLE rule (permitting any valid formula or consequence of quantifier-free boolean algebra). In addition the rules that introduce and eliminate variables have been made to do simultaneous substitutions of variables, reducing the tedium of constructing proofs, and formulas are considered equivalent under uniform change of bound variables. Also, since we noted that a good percentage of student errors resulted from trivial misapplications of the rules of inference, the rules have been generalized to be more forgiving. (Example: line number arguments to certain rules may be given in any order.) With these improvements and others, many tedious steps are omitted, and the conceptually significant steps in the proof are emphasized.

Naturally, the more powerful rules and simplifications are introduced only after the student has become thoroughly familiar with their justification. For example, the tautology rule is introduced near the end of the lessons on the propositional calculus, where the student has already done many combination-type problems (find a proof or a counterexample); it is preceded by a lesson containing problems about truth tables and truth assignments, tautologies, and consistent and inconsistent sets of propositional formulas.

As another example of the introduction of complex inference rules, within any theory the student is allowed to utilize, in any derivation, any theorems of his own which he has previously proved for that theory; in fact, he is encouraged to do so, especially while working in integer arithmetic, since many of the proofs required for interpretations are quite short and simple, given a few trivial theorems which he is required to recognize by himself.

2.1.5 Semantically Perspicuous Inference Rules

The proof-checker (which verifies the correct application of a given rule of inference to preceding lines) permits a highly perspicuous system of natural deduction (similar to Suppes' *Introduction to Logic* (1957), but with some technical differences). A variety of information about the lines of a derivation (e.g., dependency on working premises, flagged variables, ambiguous names) is available to the student on request. The only major constraints are logical ones (that is, the constraints of the system of natural deduction employed); none are the result of system hardware or software peculiarities. Further, some of the logical constraints themselves are "masked" by program features which make automatic adjustments when necessary and possible. For example, formulas which differ only by choice of bound variables are recognized as equivalent;

certain substitutions which would cause quantifier-variable conflicts cause the offending bound variable to be replaced. Again, these masking effects are described only after the student has become familiar with the underlying constraints. The result is that, once the student has developed some facility with the quantifier inference rules, some restrictions which would block valid inferences are effectively surmounted, and notation is simplified.

In general, simplifications to the natural deduction system and more powerful inference rules are introduced either late in the "basic logic" section of the course or at the beginning of the applications sequences.

Care has been taken to insure that the introduction of any inference rule is fully justified (semantically) by appropriately coordinated text and problems. Although the model- and proof-theoretic approaches are purposely confused in this elementary course, every opportunity is taken to convince the student that what he is doing is semantically valid. Most satisfactorily accomplished with the propositional rules, validity is established informally with quantifier rules as far as possible without introducing formal model-theoretic considerations.

As a tool in developing the sections on semantics, we have administered several midterm examinations in the past year. Students were told that there were no formal derivations on the exams, and that the exams tested only conceptual understanding. Results of these examinations indicate that the especially intense semantical material on propositional logic is effective in countering the tendency to over-concentrate on derivation techniques.

2.2 Axiomatic Set Theory

Axiomatic mathematics on a computer has traditionally involved the construction of derivations in a formal system. Heretofore these formal systems have simply been adopted from mathematical logic. Such systems were designed to make it easy to prove results about the system rather than to construct proofs in the system.

The EXCHECK program has been used at Stanford for the past year to teach Introduction to Set Theory (Philosophy 161) to Stanford undergraduates. The EXCHECK program provides many conveniences to the students such as the use of displays and a forgiving input language. These features are described in detail in a forthcoming IMSSS technical report, and somewhat more briefly in Smith, Graves, Blaine, and Marinov (1975). The major research accomplishments of the EXCHECK system are: (1) we have succeeded in providing proof construction machinery that reduces the detail and tedium in checking a derivation on the computer by making the individual steps sufficiently large and transparent, and (2) in the analysis of the proofs made with EXCHECK (including the 500-odd student proofs) we are beginning to understand how the proof machinery of EXCHECK can help the student in understanding the overall structure of the arguments he produces.

2.2.1 High-Level Rules of Inference

In giving proofs in logic and mathematics courses, it is common to pass over certain features that are considered overly detailed. For example, in logic courses taught in a traditional classroom setting, it is common to introduce a (syntactic) rule of inference for universal instantiation. Such a rule is a (partial) embodiment of the principal:

```
If everything has the property P, then any given thing
    has the property P.
```

The syntactic principle initially offered to the students will generally not, however, justify multiple instantiations of terms, automatically adjusting for changes of bound variables, etc. In other words, the syntactic rule of inference will not really correspond to the semantic idea behind it.

The transition between explicit rules of inference and their actual application is glossed over in an elementary logic course. This is a good pedagogical feature of these courses. We have generalized the standard rules of logical inference to include many of these insights about the way elementary logic is taught; these improvements were included simultaneously into our LOGIC and EXCHECK programs. Computer rules that correspond to the semantic ideas behind them are said to be *semantically transparent*.

2.2.2 Semantic Transparency

A similar principle is, we believe, at work in higher mathematics as well. Thus, in set theory, the rules of inference should relate most closely to the set-theoretical objects at hand and the ways in which they are manipulated. In addition to purely logical principles such as TAUTOLOGY, we have implemented the following:

1) The BOOLE rule, which embodies a decision method for quantifier-free boolean algebra. The use of this rule corresponds to saying that a certain result is a theorem of class algebra. At a certain elementary point in a set theory course, this "bag" of results is simply made available to the student.

2) The VERIFY rule, which uses a resolution theorem prover designed to check inferences. We have tried to tune this prover to correspond to the "obvious" step. This succeeded in some cases but failed on many schematic inferences.

3) The IMPLIES rule, which combines resolution with certain pattern-matching heuristics to be similar to the process of applying a previously-proved theorem or axiom to a certain situation. This top-down heuristic succeeds in many cases where resolution alone fails.

As an example of this machinery, showing how it compares to textbook proofs, consider the

Russell Theorem, which states that the set that contains everything that is not a member of itself is empty. The textbook proof goes as follows:

> Let R be the set of all x such that x is not in x. Suppose that R is not empty. Then by Theorem 1.9 we have
>
> for all x, x in R if and only if x in x
>
> and hence, R in R if and only if R is not in R, which is a contradiction. QED.

A proof done using the EXCHECK proof checker might look like the following:

> Derive: {x: x is not in x} = 0
>
> *ABBREVIATION*
> (1) *R = {x: x is not in x}*
> *WP*
> (2) *R is not equal to 0*
> *(2 IMPLIES Using Th. 1.9) US*
> (3) R in R iff R is not in R
> *3 CONTRADICTION*
> (4) R = 0
> *QED*

The steps of the proof are:

1) The student uses the ABBREVIATION command so that R can be used to refer to the Russell set. This saves typing and makes the proof more natural.

2) The WP command creates a temporary assumption, corresponding to the "suppose" in the text-book proof.

3) The student uses the IMPLIES command to apply Theorem 1.9 to the assumption.

4) The CONTRADICTION command, which also calls the resolution theorem prover, is called to get the contradiction.

5) QED is typed by the student when he thinks that his last line is either the desired result or only trivially different from the desired result.

2.2.3 The Structure of Proofs

The major unresolved issues in the EXCHECK program involve the global structure of proofs. We have succeeded in developing proof machinery that is considerably more powerful than standard natural deduction systems. Proofs that our students gave with ease would have run several hundred lines using a proof checker such as described in Goldberg (1971). This is not to say that these theorems are complex; in fact, most of the theorems in our elementary set theory are quite elementary. The difficulty is in giving the student an overall picture of the proof as it proceeds, in helping him structure and understand that proof more interactively.

In the next year we will implement machinery that will allow the student to describe the overall structure of the proof in a top-down manner so that the program can take care of the details, either supplying the obvious details or asking the student for more information. An instance of this kind of structuring, familiar from standard mathematical practice, would be to say:

```
We show

if a is a limit ordinal & b > 0
    then a . b  is a limit ordinal

by induction on b.
```

At this point, in reading a mathematics text, the reader is expected to fill in the usual details. In the derivation control machinery that we plan to implement the student will be able to guide the proof machinery, and query it for a summary of the current state of a proof, where he can control the depth of detail that the summary has.

Work in the logical theory of proofs has suggested various measures of structural complexity for proofs. We are investigating these and other measures in the light of the empirical data obtained in the past year of operation. Next year's research on the EXCHECK project will concentrate on providing these structural tools, both for the student and for our own analysis of the curriculums.

2.2.4 Production of Curriculum Materials

An important part of our work has been a reduction in the amount of administrative and secretarial staff required to develop and maintain computer-taught courses. For example, to revise the set theory curriculum, a staff member need only use one of the text editors to change the source files. He can then use a program which will check for spelling errors. Another program produces two files, one read by the set theory program and another which is used to make printed copies of the text. In practice it is often possible for a student to send a message about a serious typographical error which will be read and confirmed by a staff member within hours. He will then notify all of the students by typing one message, make the appropriate change in the program, and have printed copies of the revised text available that same day.

The rapidity and ease of producing neat, well-formatted text has made it practical to produce timely and highly accurate curriculum materials as a byproduct of the course development. The same faculty who develop a new course and who have day to day contact with the students can now easily prepare the manuals and other supporting materials. Where formerly a special supporting staff was needed and preparation time was counted in months, now only weeks or days are required and revisions can often be made in hours. The use of report programs to monitor both individual and class progress and to prepare grade sheets also saves time.

2.2.5 Extensions into Higher Mathematics

An important project of the Logic and Set Theory group is to expand the techniques into teaching subjects other than logic and its extensions. We began with set theory for several reasons, including the hope that the tools developed in set theory would be applicable in an incremental way to the foundations of measurement, the foundations of probability, and so on. The next year will also see the development of a curriculum in the foundations of probability, starting with the EXCHECK system and adding to it those proof-building and structural tools that are needed in that curriculum.

3 Slavonic Languages and Culture

3.1 Computer-Based Introduction to Old Church Slavic

A computer-based introduction to Old Church Slavic consisting of 24 lessons was developed by Professor Joseph Van Campen of the Department of Slavic Languages and run during the winter quarter of 1972-73 (see Van Campen, 1973). A revised version of the course, consisting of 34 lessons covering essentially the same material as the first version, was run in the autumn quarters of 1973-74 and 1974-75 (see Van Campen & Schupbach, 1975). The combination of the computer-based lessons, plus written homework assignments, has allowed us to significantly reduce the number of hours during which the instructor meets with the students, while maintaining or even improving the level of student performance.

3.2 Computer-Based Course in Bulgarian

This course, consisting of some 37 computer-based lessons, was written by Robert Karriker, a candidate for the Ph.D. degree in the Stanford Department of Slavic Languages and Literatures. A detailed description of the course will be included in Karriker's Ph.D. dissertation (forthcoming). Since the Bulgarian course uses the driver program and audio capability developed in connection with the computer-based Russian courses created and implemented at IMSSS in 1967-71, a general idea of the nature of the course can be gained from the reports on those courses (see, in particular,

Van Campen (1970); evaluation data on the first-year Russian course is to be found in Suppes and Morningstar (1969)). The course is available "on demand" at the Slavic department and was last utilized by a Stanford undergraduate in the spring of 1974-75. Information on student performance will be included in Karriker's dissertation.

3.3 CAI and the History of the Russian Literary Language

CAI portions of Slavic 212, History of the Russian Literary Language, were first run in 1973. They have since been expanded and updated and are now run yearly as a regular portion (roughly two-thirds) of the instructional material of the course. Their pedagogical effectiveness has been noted by the students as well as the instructor. The basic aim in designing the course--to free much of the instructor's time for other teaching duties without loss of pedagogical effectiveness--has been achieved. In fact, there is reason to believe that the present CAI-lecture-reading mix is more effective than the original format of lecture and reading.

3.4 On-Line Generation of Computer-Based Form Drills

Since 1972, Joseph Van Campen, with the help and guidance of Rainer Schulz and Eleanor Van Campen, has been engaged in the development of software for the on-line generation of drills in Old Church Slavic and Russian. The present software permits the student to drill the forms of a given vocabulary item in three different ways:

> 1) by typing forms called for by their English labels, e.g. 'nominative singular', 'third plural present';
>
> 2) by typing a number indicating the category or categories represented by a given form, e.g. OCS *nogy* 'foot', which is either a genitive singular (2), nominative plural (10), or accusative plural (11);
>
> 3) by typing the form (or forms) of a word that correctly completes a given phrase or sentence, e.g., Russian *on govorit o* ... 'he is talking about...' which can be correctly completed by either a prepositional singular or a prepositional plural of the noun being drilled.

A detailed description of the program and coding system utilized in connection with the driller is still in preparation, but certain aspects of both are discussed in Antolini (1975). (See also Section 3.5.) It is hoped that the on-line generation of drills can be used in connection with Stanford's second-year Russian course in 1975-76.

3.5 On-Line Generation of Pattern Drills for First-Year Russian

During 1974-75, Anthony Antolini, working under the direction of Joseph Van Campen and utilizing the program and coding system developed for the drills discussed in Section 3.4 above, created the software necessary to permit the on-line generation of some 676 pattern drills based on the grammar and vocabulary found in Ben Clark's *Russian for Americans* (1973). A detailed description of Antolini's work, plus considerable information on other aspects of the drill program and coding systems, is given in his Ph.D. dissertation, Antolini (1975). It is hoped that Antolini's material can be adapted for use with other first-year texts and will eventually find application in connection with the first-year Russian course at Stanford. In addition, Antolini is interested in investigating the use of computer-generated audio stimuli in his drills and hopes to do work on the implementation of this feature in 1975-76.

3.6 CAI and Mnemonics for Foreign-language Vocabulary Learning

The mnemonic or "keyword" method was designed by Michael Raugh and Richard Atkinson for teaching foreign language vocabulary. Laboratory-controlled experiments have shown it to be an effective learning tool. Briefly, the method involves the use of an English "keyword" that sounds like the Russian word in some way, and a vivid visual image is used to associate the keyword with the English translation. Thus the keyword is used as the link between the spoken Russian word and its English meaning. The development of the keyword method is described by Atkinson and Raugh (1974) and Raugh and Atkinson (1974). A program using the keyword method was implemented as a vocabulary-building supplement to the regular curriculum for second-year Russian at Stanford in 1974. Results show that the method was popular and effective. (See Raugh, Schupbach, & Atkinson, 1975). Plans call for its continuing use in elementary Russian courses.

3.7 Applications of the MISS Audio System to Russian

IMSSS is developing a computer-generated audio system known as MISS (see Section 7). Efforts are now underway to "tune" the MISS system for use in teaching Slavic languages. Several declension types and stress patterns have been analyzed for computer generation of grammatical forms on the basis of prerecorded segments. Of particular interest are vowel changes caused by stress-shifts and the sharping of certain consonants before grammatical endings.

3.8 Selective Lexical Features

Analysis of the semantics of Russian adverb-adjective combinations is being carried out in order to describe the selective lexical features of these words as well as the rules of their combination. On this basis we hope to produce on-line generation of only "correct" combinations through the use of Van Campen's grammatical drill program.

4 CAI in Programming

The Institute has been involved in CAI projects in computer programming since 1968. Work in teaching computer programming began with the development of a high-school-level CAI course in machine language programming (Lorton & Slimick, 1969). The project, called SIMPER, taught programming via a simulated three-register machine with a variable instruction set. Later, lessons in the syntax of the BASIC language were presented, and the student solved them by linking to a BASIC interpreter, without receiving assistance or analysis of his efforts from the instructional program.

In 1970 the Institute developed a much larger CAI curriculum for a new course to teach the AID programming language at the introductory undergraduate level. This course has been used in colleges and junior colleges as a successful introduction to computer programming (Friend, 1973; Beard, Lorton, Searle, & Atkinson, 1973).

During the summer of 1973, a study was made of children's learning of concepts relevant to programming languages, in the context of instruction and hands-on practice in Simper and Logo (Feurzig, Papert, Bloom, Grant, & Solomon, 1969). This study is reported in Weyer and Cannara (1975).

Currently, the Institute offers two programming courses, both of which provide direct programming experience in the languages they teach. The BASIC Instructional Program originated as a research vehicle to broaden the areas of investigation available through the AID course, and has been funded by the Office of Naval Research. (See Beard, Barr, Fletcher, & Atkinson, 1975.) The LISP CAI System was developed as an adjunct to Stanford's course called Computing in Symbolic Expressions. In general, the BASIC course is more research-oriented, involving experimentation with various models of curriculum organization and student progress, while the LISP course is more attuned to the particular needs of the Department of Computer Science in providing a reliable, immediate source of efficient instruction.

These CAI courses have been used by a large number of students, and will continue to be used, both for instruction and for research in CAI, as described more fully below.

4.1 The BASIC Instructional Program

4.1.1 Background

We have explored the use of information networks to describe the contents of a computer-programming curriculum, and developed a large-scale CAI course in programming (the BASIC Instructional Program, or BIP) as a vehicle for exploration and evaluation.

BIP is a stand-alone, fully self-contained course in BASIC programming at the college or junior college level, and was developed as a vehicle for our research in instructional strategies.

Over 300 undergraduates from DeAnza College, the University of San Francisco, and Stanford have taken the course during the past two years and contributed to its design. BIP's major features are:

- A monitored BASIC interpreter, written in SAIL (VanLehn, 1973) by the Institute staff, which monitors all student programming attempts.

- A HELP system that directs the student to appropriate parts of the student manual for explanations of his specific syntax errors.

- A curriculum consisting of approximately 100 well-written, interesting programming problems at widely varying levels of difficulty.

- A HINT system, which gives both graphic and textual aid in problem solving.

- Individualized task selection based on a Curriculum Information Network, describing the problems in terms of fundamental skills. Problems are selected using a model of the student's acquisition of skills required by his earlier programming problems.

Work on methods of curriculum description appropriate for network representation, suitable models of knowledge acquisition, strategies for task selection based on the network and the student model, and methods for comparative evaluation of different strategies is proceeding within the context of the BIP course. BIP's development is described in reports by Barr, Beard, and Atkinson (1975b), and by Barr, Beard, Lorton, and Atkinson (1974).

Given a curriculum description in the form of a Curriculum Information Network, our goal is to develop more sensitive task-selection algorithms that will further individualize BIP's interaction with students. These algorithms may embody many different strategies and heuristics for finding appropriate material, but are all based on a model of the student's acquisition of previous material, his state of knowledge. This model resides inside the instructional program as another data structure. The detail involved in student models ranges over the full spectrum of representational complexity: from right/wrong counters on past problems to counters on the basic skills used to describe the curriculum, as in BIP, to a full semantic representation of the knowledge acquired by the student.

The work of describing a suitable and sufficient student model and of developing appropriate task-selection algorithms is essentially an empirical process of successive comparisons. Previous methods of evaluating the effects of small design changes, including protocol analysis and large-scale controlled experiments, although effective, are impossibly time-consuming for the exploration of the multitude of empirical questions generated in the design of a system of this complexity. It is necessary to develop a simulation procedure that will conveniently exhibit gross differences in the performance of task-selection strategies based on the characteristics of a statistical student population. This method will be a general tool for the evaluation of design considerations before the actual development of courseware is begun.

The Curriculum Information Network is a key product of the research to date. Its purpose is to provide the instructional program with an explicit knowledge of the structure of an author-written curriculum. It allows meaningful modelling of the student's progress along the lines of his developing skills, not just his history of right and wrong responses, without sacrificing the motivational advantages of human organization of the curriculum material. For example, in the BIP course, the CIN consists of a complete description of each of 100 programming problems in terms of the skills developed in solving the problems. Thus the instructional program can keep track of the student's progress on these skills, and choose the next problem to use an appropriate group of new skills. We feel that it is very important to allow this kind of individualization without reducing the interest of the problems by, for instance, having them deal with only one skill. The algorithm by which BIP selects the student's next task is based on the information stored in the network. The current version of this algorithm is described in detail by Barr, Beard, and Atkinson (1975a).

Throughout BIP's development, student use of the course has been closely monitored, both through observation by staff members and through extensive on-line data collection. Informal evaluation has been continuous, and numerous changes have been made to facilitate interaction between naive student programmers and the BIP system. In addition, a controlled experiment involving 41 Stanford undergraduates was carried out to compare the task-selection algorithm with a fixed path through the curriculum. Preliminary results indicate that the CIN-based approach is indeed an effective tool in the individualization of instruction.

4.2 The LSPCAI System

4.2.1 Background

The LISP Computer Assisted Instructional System was designed as a teaching aid for the Stanford Computer Science Department's course in list processing languages, CS206. The system has been used in that course during the 1974-75 academic year. Student response has been varied, and is currently under study with a view toward improving the system.

The course is intended to teach LISP to upper-level undergraduates and first or second year graduate students. It is assumed that the student has some programming experience, though not in LISP or related languages. The course is, therefore, not intended to teach basic programming skills. Rather, the emphasis is on list processing techniques, with particular attention being given to recursive solution methods.

Our design goals were fourfold:

1) To provide instruction in the basic syntax and semantics of LISP.

2) To provide instruction in recursive solution techniques.

3) To provide a system-independent programming environment which approximates that of a production LISP system.

4) To provide access to other programming facilities, as needed.

The instructional goals have been met, much in the manner of traditional CAI systems everywhere. The system presents instructional text (which may be abbreviated at the request of the student), and asks questions on the material covered. In the abscence of directions to the contrary, the program proceeds sequentially through the curriculum materials. However, the student is free to arbitrarily alter the instructional sequence. This allows the student (or the instructor) to reorder the lessons to suit individual preferences. This mode of presentation has proved to be well suited to the student population using the course.

Largely because of the complexity of the CAI system, the services of a teaching assistant have been required by many students. However, the ability of the system to provide instruction and direct programming experience has allowed it to take over much of the LISP language teaching workload for the CS206 course.

In the later segments of the course, dealing with recursive programming problems, the student is given access to an essentially unmodified UCI-TENEX LISP interpreter (see Wolpert, forthcoming). This interpreter contains an extremely useful form-oriented editor, a powerful debugging package and an online documentation package containing extensive information on LISP. To all intents and purposes, this is a production LISP system (and is, in fact, similar in most important respects to INTERLISP). The student uses this interpreter to define and test his functions. When he is satisfied with the result, he returns control to the instructional system, which tests his function against the results produced by a stored model solution. In the event that errors are discovered, the student is given the option of returning to the LISP environment to make changes. When he has obtained a correct answer (or has given up), he may examine the model solution.

The instructional system is thought to be roughly equivalent in content to the Weissman primer (Weissman, 1967). We are of the opinion that the Curriculum Information Network concept (see Barr et al., 1975a) may be fruitfully applied to the segment of the course dealing with the syntax and semantics of the language. Additional interaction between the student and the instructional system during the debugging of functions is also desirable, particularly in case of semantic errors in the student's functions.

The student has access to the LISP environment at any time. During the second half of the academic quarter, the instructional system is used rarely, if at all. Rather, the students make use of the LISP system (and other facilities listed below) to complete programming projects assigned by the instructor.

Additional facilities available in the system include an unmodified MICRO-PLANNER interpreter, a production LISP compiler, and an experimental system for verifying the equivalence of LISP programs and their compiled representations.

4.2.2 Plans for Future Development

Our work with LISP will involve two projects. In the first, an instructional laboratory will be developed for use with the introductory LISP course at Stanford. Experience gained during the development and classroom use of BIP will be applied to the improvement of computer-assisted instruction in LISP.

The second project deals with an additional augmentation of the laboratory environment to facilitate a top-down or structured approach to the programming process. An editing system will be developed to support the discovery and construction of incompletely specified LISP functions. The student will state his ideas about the algorithm in a form completely independent of the LISP language syntax, and a fully specified LISP program will be achieved by a process of step-wise refinement.

5 Chinese

Development of a program to teach spoken Mandarin (the official Chinese language) was begun in 1974 by Peter E-Shi Wu, Ph.D. candidate in the School of Education. A pilot version will be tested late in 1975. The program will make extensive use of the MISS computer-generated audio system (see Section 7), as all interaction between the computer and students will be carried out over a push-button telephone.

The student (working at home) will prepare for the computer-controlled lesson by reading a handout describing the content of the lesson. During instruction, the computer will "speak" in Mandarin and the student will respond to multiple-choice exercises by pushing the appropriate digits on his telephone pad. The student will spend between 20 and 30 minutes with the program for each lesson, depending on his learning rate.

The content of the course emphasizes spoken Mandarin, with explicit instruction on phonetics and grammar introduced only as needed. Written Mandarin characters will not be taught. The course consists of 60 lessons, each of which introduces at most ten new vocabulary items and uses at most ten sentences. Two new patterns are introduced every two lessons.

Four different learning models have been constructed, and will be evaluated according to how well they fit the results of the multiple-choice tests embedded in the program.

6 Music

6.1 History

Since 1972, we have been developing several applications of a computer controlled system for investigating various theoretical and instructional topics in music. An extensive rationale for this system is discussed in Kuhn (1975).

Music instruction demands a highly individualized approach, and computer-assisted instruction has developed out of efforts to meet this demand in other areas. The development of the CAI-Music project focused on five specific requirements. These were: 1) need for sound, 2) need for real time interaction, 3) need for individualization, 4) need for detailed student records, and 5) need for basic research.

With these five requirements in mind, work on a software system to conduct research and instruction was begun. In the Fall of 1972, the curriculum driver was developed and the first few lessons were implemented. From the outset this work was dedicated toward making the computer program that controls curriculum presentation as knowledgeable as possible about the nature of the musical device being used and about the structure of the potential curriculum. This was done so that as much curriculum as feasible could be generated by the computer program rather than written by the curriculum author. This approach made possible a wider range and better rationalized body of curriculum than might otherwise have been produced.

During the winter of 1973 the program and curriculum were tested with small groups of elementary-school (age 7 to 11) and college-age students. Starting with the 1973 Spring Quarter, the system was used as a supplement to the regular curriculum of first-year music courses at Stanford. Student sessions of 20 to 30 minutes proved to be the optimum length. Sufficient student data were generated during this first quarter to undertake a study correlating attitudinal and biographical student information with their performance on the system (see Herrold, 1973). The system was used as an adjunct to undergraduate music courses for the academic years of 1973-74 and 1974-75. Considerable data have been recorded. Some students are now in their fourth quarter of use of the CAI system, so that considerable longitudinal information is available.

6.2 Description of the Music System

The written portion of the curriculum is presented and student responses are entered on a Teletype Corp. Model 33 KSR terminal. Music is presented via a Thomas solid state organ, Model 145. The organ is linked to the IMSSS PDP-10 computer through an interface which translates 8-bit patterns into notes to be addressed (6 bits) and one of the commands SET-ON, PLAY, SET-OFF, or OFF. At present interaction with the organ is one-way; notes are sent to the organ, but none are received from the organ.

The interface provides access to 64 notes on the organ. These notes are located in two

octaves on the upper keyboard and three octaves on the lower keyboard. The timing of the notes is currently limited by the use of a 110 baud line for the connection with the computer. There is the capability for running the present configuration at up to 300 baud; the faster speed will enhance the range of the possibilities for the system.

The main program developed for this project is divided into two parts. One part has been developed to access the curriculum, generating as much as possible, and to handle the student data and history information. The other part has been developed with as much understanding of the organ and of the relationships among musical notes as the curriculum and research projects demand.

The various options available to the curriculum author are described in Killam and Lorton (1974). In general the curriculum is divided into several homogeneous groupings called "strands." At present these include intervals, triads, melody, rhythm, chords, modulation, and others that are constructed for specific research projects.

Because of the importance of maximum flexibility for both instruction and research, many options are available online to the student or experimental subject. These allow free movement, as appropriate to the particular strand involved, throughout the curriculum. Students may restart at the exact place where they left the strand or the preceding lesson. In addition, students have the option of constructing their own musical examples which can be stored by the program and played on request.

6.3 Uses for Research

The system described above makes possible research in both auditory perception and the acquisition of auditory skills. Those involved in the teaching of music theory agree that students need to learn to identify musical intervals and other basic tonal configurations, such as triads, prior to more advanced study. However, little is known about how such learning takes place, or about the most productive sequence of study of basic skills.

One of the first studies undertaken on the present system was a replication and extension of an earlier study on interval identification. The earlier study was limited to the presentation of simultaneous intervals only; our study was extended to cover presentation of the notes in ascending and descending order. A full report of this research will be available in Killam, Lorton, and Schubert (1975).

In a second study, the perception of triads is being investigated. Students identified traditional triad structures (major, minor, diminished, and augmented) in all possible inversions. Matrices similar to those reported in Killam et al. (1975) will be used to chart areas of student confusion of one structure with others.

Projected studies include accuracy of student identification of melodic patterns with controlled rhythmic variances, correlation of students' vocal range with range of highest accuracy of

auditory recognition, and correlation of identification of tonal chord patterns with tempo of patterns and other factors.

6.4 Uses for Instruction

The current system has proved to be of effective, practical use in the teaching of basic auditory concepts. Development of a successful curriculum has been possible because musical sound is accessed and under computer control. The details and range of currently developed curriculum presentation modes are described in Killam and Lorton (1974). Current curriculum areas are those found in traditional ear-training programs, although drill and practice geared to specific course content has been implemented at instructors' requests. Material in each strand is presented so that written instructions are kept to a minimum and the student's attention is focused on the sound presentation.

The system has been used in conjunction with a variety of courses given in the Music Department at Stanford University. These include:

Music 21 and 22--Elements of Music

Music 23--Functional Harmony

Music 27--Solfege and Ear Training

Music 102B--Eighteenth Century Harmonic Practice

Music 144A--Twelve-Tone and Serial Music

Two completed studies (Herrold, 1973; Killam & Lorton, 1974) document the effectiveness of the system for instruction. More extensive data, similar to those used in these two studies but for the academic years 73-74 and 74-75, are currently being analyzed.

Other areas of the music curriculum are being considered for incorporation in this system. For example, individual tests are given to all music majors on a department-wide basis to determine students' skill levels. A major portion of the test has been implemented under computer control and is being evaluated. This will result in considerable saving of faculty time presently required to give these tests, as well as detailed data for analysis.

7 Computer Generated Speech

IMSSS is developing a speech synthesis system called *Microprogrammed Intoned Speech*

Synthesizer (MISS). MISS is a hardware/software system capable of high quality speech synthesis with user control over the intonation pattern of the speech. When completed, this system will be applied to complex CAI programs such as BIP and EXCHECK.

The MISS vocabulary of words is stored on a drum and disks. The stored vocabulary consists of compressed representations of recorded words. To play a particular word from the vocabulary, the compressed representation is fetched from the drum or disk and a special purpose processor expands the data into real time speech. The compressed representation is called Linear Predictive Coding (LPC) and a digital filter performs the calculations necessary to expand this representation into speech.

The hardware part of MISS is a special purpose, high speed, dual microprocessor designed to generate up to sixteen simultaneous voices to any of forty-eight audio stations. One microprocessor performs the digital filter calculations required to actually generate the speech. It can be programmed to handle direct, cascade, or lattice filter forms with both poles and zeros. With twelve poles per filter, any of these forms requires only one sixteenth of the processor time, so sixteen independent voices can be generated. We are presently using the lattice form of digital filter. The other microprocessor performs prosodic manipulations to the filter parameters and does the necessary housekeeping to keep data flowing from the host computer to the filter.

The software part of MISS is divided into two parts: Analysis programs to generate LPC parameters from recorded speech, and user library procedures to allow curriculum and experimental programs to have easy high-level access to the MISS hardware for generating their speech. A vocabulary of ten thousand words is being recorded and analyzed. The user can tell the system to play any sequence of words from the vocabulary and at the same time specify the intonation pattern that he desires to have superimposed on his "sentence."

An intonation template is a set of rules for adjusting the pitch rate, speed of pronunciation, and volume of the words in a sentence. Intonation templates that give a prosodically neutral sentence particular syntactic or semantic meaning are being investigated. Also under investigation are programs to automatically choose an appropriate intonation template from the syntax of the sentence. Later, attempts will be made to concatenate parts of words together, such as adding prefixes and suffixes to root words. These attempts will be successful only when phonemic concatenation rules are found that result in high quality speech. Professor Jon Allen at M.I.T. is presently investigating such rules. (Personal communication, 1975.)

8 Natural Language Processing

The work on natural language processing at IMSSS focuses on the study of techniques that are designed to handle relatively stereotyped or formalized fragments of English, rather than on techniques that attempt to provide a very general understanding of all of natural language. The reasons for this emphasis are: (1) the more limited enterprise of dealing with semi-formal pieces of

English does, in fact, provide some interesting answers to questions about the nature of the computational apparatus required to process these parts of language; (2) this work is immediately applicable to the CAI systems being developed by the Logic and Set Theory groups at IMSSS.

8.1 An Overview of Constructive Semantics

Using the idea that a simple, rather natural context free grammar can be developed for the fragment of English to be processed, we are attempting to develop an appropriate semantic and pragmatic theory for evaluating the semantics.

Donald Knuth (1967) introduced the notion that the productions of a context free grammar may have associated with them semantic functions that map the semantic values of the items on the right hand side of the rule to a semantic value for the item on the left hand side. Unfortunately, but not too surprisingly, such a simple scheme will not work for English. Initially, we introduced the notion of semantic transformations -- a mapping from semantic forms to semantic forms analogous to the syntactic transformations of Chomsky. These transformations are used to move information from its initial location in the semantic form to a place where it can actually be used. Our initial attempts along these lines were embodied in a system for processing elementary mathematical language, CONSTRUCT, and are documented in Smith, Smith, and Rawson (1974), N. Smith (1974), and Rawson (1973).

We are exploring other extensions to these semantic ideas which permit a much greater interaction between the nodes of the parse tree that the context free syntactical analysis builds. We are currently developing a multi-processing model that will allow individual computations to interact in a way that allows semantic and pragmatic information to be used.

8.2 Applications of the Natural Language Processing Work

We hope to be able to greatly improve the dialogue capabilities of existing CAI programs such as set theory. The work on the CONSTRUCT system has been incorporated into the current logic program and the EXCHECK program for teaching axiomatic set theory. In logic, for example, the languages of several different first order theories are recognized by a single system, depending on the theory that the student is dealing with. In EXCHECK, a context-free grammar with 500-odd rules recognizes a large part of the formal/informal language of sets, with the major current restriction being the explicit mention of bound variables.

Our current work on a multi-processing system has as one of its major goals the development of better ways to handle the scopes of variables and operator-like constructions, especially when these are only implicitly mentioned, such as in the sentence

Find the next largest cardinal after A and then take its powerset

where the word "next" creates an implicit variable.

Additionally, for audio output, we intend to develop output grammars that include prosodic features in the grammar directly and thus drive the MISS speech generation system for prosodically improved speech.

8.3 IMSSS Research on Language Acquisition

In addition to the work on computer-based language processing, for a number of years the Institute has had a program of research on children's language. This work is reported in Gammon (1970), R. Smith (1972), Suppes (1970, 1971, 1973b, 1974), Suppes, Leveille, and Smith (1974), Suppes, Smith, and Leveille (1972), and Wexler (1970). Most of this work concentrated on analysis of the lexicon, grammar, and semantics of transcribed dialogues. Techniques employed were probabilistic models of disambiguation and the development of structure, and set-theoretic models of meaning.

In the area of foreign-language instruction, David Levine developed a system for grading the correctness of syntax for elementary German instruction. This is reported in Levine (1973).

9 IMSSS Computer Facilities

9.1 IMSSS Software System

The current software system is a BBN TENEX-131 system converted to run on the Digital Equipment Corporation KI-10 processor. Very few changes were made to the operating system by the IMSSS staff, but the changes made have been significant. Many of the changes to the system were for interfacing IMSSS hardware peripherals to TENEX. Other changes in the scheduling and memory management parts of the system resulted in markedly superior system performance. The net result of all these changes is that the IMSSS system is the fastest KI-10 (KA-10) hardware system in the world.

Still more changes to the operating system could be made to improve system performance. One central concern motivating the improvements made to date is to meet the requirements of our large and diverse user load. Currently the user community seems to be adequately serviced, but as the load on the system becomes heavier, we may make more changes to improve system throughput.

Another requirement we have set for ourselves concerns reliability, uptime, and long mean time between failures. Currently our uptime (excluding scheduled downtime for installing new hardware or software) is approximately 99%. The mean time between failures, since the installation of the new hardware, is several weeks.

9.2 IMSSS Hardware Configuration

The following outlines the components of the IMSSS KI/TENEX system.

1) DEC KI-10 central processor.

2) 256k words (1,152,000 bytes) of high speed core memory. 350 ns access time, 750 ns cycle time.

3) 2 GI drums for secondary storage. 8,257,000 bytes of storage each; 8 ms latency, transfer 490 ns/byte.

4) 3 AMPEX dual density 3330 disk spindles with IBM compatible controller. Each spindle has a capacity of 178 million bytes.

5) Two IBM 3420 tape drives with 3803 controller. The system has 800/1600 bpi capability.

6) High speed line multiplexer, capable of running over 500 terminals.

7) Remote terminal multiplexers (PDP8's and MICRO 800's).

8) Delta modulation audio system.

9) PCM audio system.

10) LPC audio system.

11) Various graphics and other display terminals, Teletypes, line printers, plotter, etc.

9.3 Application Programming Support

In general, the Institute is not funded to produce utility and applications programs. In a few significant cases we have invested some central facilities effort into producing software that has been justified by increased productivity of the sponsored research. The main efforts of this kind involve the SAIL and ILISP programming languages, and the creation of a display-oriented text editing system, TVEDIT. The SAIL effort was funded directly by a contract to the Jet Propulsion Laboratory, while the other work was supported indirectly by research contracts with other agencies.

9.3.1 SAIL

The SAIL language (see VanLehn, 1973) was developed at the Stanford Artificial Intelligence Project and is a good language for our research in computer-assisted instruction. SAIL is an ALGOL-like language, with complex data- and control-structure extensions for artificial intelligence research. It compiles reasonably efficient code, which is an important consideration for CAI production.

We have given the SAIL system complete access to the facilities provided by TENEX, and have added a number of features (random input-output, interrupt system) designed to facilitate various aspects of the CAI research. These changes were merged into the master files at the AI Project, insuring integrity of the SAIL software.

The features we have added to SAIL are very heavily used at IMSSS. All of the projects described in this report were written in SAIL except for the LISP CAI system and parts of the EXCHECK set theory program, which were written in ILISP (see below).

TENEX SAIL is documented in R. Smith (1975).

9.3.2 ILISP

The ILISP system is a variant of the UCI extensions to LISP 1.6, as originally documented in Quam and Diffie (1967) and in Bobrow, Burton, and Lewis (1972). In order to facilitate the LISP CAI course, and to provide a language for implementing some parts of the EXCHECK system for set theory, we have TENEX-ized the runtime system, and added many features to the interpreter, compiler, and break package. This work will be documented in a report by Wolpert (1975).

9.3.3 TVEDIT

At IMSSS we have developed several page-oriented text editors to be used from our display terminals. We refer to them collectively as TVEDIT. Our first TVEDIT program was written in 1965. The most recent version, written in SAIL, is documented in Kanerva (1975).

TVEDIT provides a window into the user's text file, with a section of the file -- 23 consecutive lines on most of our displays -- visible at once. The user can skip to any part of his file and view it through this window, and he can alter the text in the current window. When an editing command is typed, the window is immediately updated to show the effect of the command on the text. Updating of the file is automatic, requiring no specific transmit commands by the user.

We have found that TVEDIT is easier to learn than line-oriented text editors designed for teletypewriter terminals, and that users prefer it to line editors, probably because the user can see the immediate effect of his commands. The ease of viewing a text file with the editor also reduces the need for temporary listings of a file while editing work is in progress.

The display routines used by the editor service all the display terminals owned by IMSSS. These routines are also available to other programs in the form of a library of functions written by Pentti Kanerva.

References

1. Antolini, A. F. *An investigation of the feasibility of computer-based generation of pattern drills for first-year Russian* (Tech. Rep. 254). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975.

2. Arrow, K. *Social choice and individual values.* New York: Wiley, 1966.

3. Atkinson, R. C., & Raugh, M. R. *An application of the mnemonic keyword method to the acquisition of a Russian vocabulary* (Tech. Rep. 237). Stanford, Calif: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1974.

4. Barr, A., Beard, M., & Atkinson, R. C. Curriculum networks for CAI curriculums. In O. Lecarme & R. Lewis (Eds.), *Computers in education, part 1: IFIP.* Amsterdam: North Holland, 1975. (a)

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

6. Barr, A., Beard, M., Lorton, P., & Atkinson, R. C. A college-level course in BASIC. In G. Goos and J. Hartmanis (Eds.), *Lecture Notes in Computer Science* (Conference RGU, Hamburg). Berlin: Springer-Verlag, 1974.

7. Beard, M., Barr, A., Fletcher, D., & Atkinson, R. C. *The improvement and individualization of computer-assisted instruction: Final report.* Stanford, Calif: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975.

8. Beard, M., Lorton, P., Searle, B., & Atkinson, R. C. *Comparison of student performance and attitude under three lesson selection strategies in computer-assisted instruction* (Tech. Rep. 222). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

9. Bobrow, R. J., Burton, R. R., & Lewis, D. *UCI LISP Manual* (Tech. Rep. 21). Irvine, Calif.: Information and Computer Science Department, University of California at Irvine, 1972.

10. Bolton, E. A., and Markosian, L. Z. Reference manual for introduction to logic (Philosophy 57a). Internal IMSSS document.

11. Brown, J. S., Burton, R. R., & Bell, A. *An intelligent CAI system that reasons and understands* (BBN Report 2790). Cambridge, Mass: Bolt Beranek and Newman, 1974.

12. Carbonell, J. R., & Collins, A. M. Natural semantics in artificial intelligence. *Proceedings of the third international joint conference on artificial intelligence*, Stanford, Calif., August 1973.

13. Clark, B. *Russian for Americans* (2nd ed.). New York: Harper & Row, 1973.

14. Feurzig, W., Papert, S., Bloom, M., Grant, R., & Solomon, C. *Programming languages as a conceptual framework for teaching mathematics* (Report 1889). Boston: Bolt Beranek, & Newman, 1969.

15. Friend, J. *Computer-assisted instruction in programming: A curriculum description* (Tech. Rep. 211). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

16. Gammon, E. M. *A syntactical analysis of some first-grade readers* (Tech. Rep. 155). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1970.

17. Goldberg, A. *A generalized instructional system for elementary mathematical logic* (Tech. Rep. 179). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1971.

18. Goldberg, A., & Suppes, P. A computer-assisted instruction program for exercises on finding axioms. *Educational Studies in Mathematics*, 1972, *4*, 429-449.

19. Goldberg, A., & Suppes, P. Computer-assisted instruction in elementary logic at the university level. *Educational Studies in Mathematics*, in press.

20. Herrold, R. *Computer-assisted instruction: A study of student performance in a CAI ear-training program.* Unpublished DMA Project, Stanford University, 1973.

21. Jamison, D., Suppes, P., & Wells, S. The effectiveness of alternative instructional methods. *Review of Educational Research*, 1974, *44*, 1-67.

22. Kane, M. T. *Variability in the proof behavior of college students in a CAI course in logic as a function of problem characteristics* (Tech. Rep. 192). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1972.

23. Kanerva, P. TVGUID. Internal IMSSS document, 1975.

24. Killam, R., & Lorton, P. Computer-assisted instruction in music: ear-training drill and practice. *Proceedings of the Fifth Conference on Computers in the Undergraduate Curriculum.* Pullman, Wash: Washington State University, 1974.

25. Killam, R., Lorton, P., & Schubert, E. *Interval recognition: A study of student accuracy of identification of harmonic and melodic intervals.* Manuscript submitted for publication, 1975.

26. Knuth, D. E. Semantics for Context Free Languages. *Mathematical Systems Theory*, 1967, *2*, 127-145.

27. Kuhn, W. Computer-assisted instruction in music: Drill and practice in dictation. College Music Symposium, 1975, *XIV*, 89-101.

28. Levine, D. R. *Computer-based analytic grading for German grammar instruction* (Tech. Rep. 199). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

29. Lorton, P., Killam, R., & Kuhn, W. A computerized system for research and instruction in music. In O. Lecarme & R. Lewis (Eds.), *Computers in education, part 1: IFIP*. Amsterdam: North Holland, 1975.

30. Lorton, P., & Slimick, J. Computer based instruction in computer programming: A symbol manipulation-list processing approach. *Proceedings of the Fall Joint Computer Conference*, 1969, 535-544.

31. Moloney, J. M. *An investigation of college student performance on a logic curriculum in a computer-assisted instruction setting* (Tech. Rep. 183). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1972.

32. Quam, L.H., & Diffie, W. *Stanford LISP 1.6 Manual* (Operating Note 28.6). Stanford, Calif.: Stanford Artificial Intelligence Laboratory, Stanford University, 1967.

33. Raugh, M. R., & Atkinson, R. C. *A mnemonic method for the acquisition of a second-language vocabulary* (Tech. Rep. 224). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1974.

34. Raugh, M. R., Schupbach, R. D., & Atkinson, R. C. *Teaching a large Russian language vocabulary by the mnemonic keyword method* (Tech. Rep. 256). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975.

35. Rawson, F. L. *Set-theoretical semantics for elementary mathematical language* (Tech. Rep. 220). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

36. Smith, N. W. *A question-answering system for elementary mathematics* (Tech. Rep. 227). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1974.

37. Smith, R. L., Jr. *The syntax and semantics of ERICA* (Tech. Rep. 185). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1972.

38. Smith, R. L. *TENEX SAIL* (Tech. Rep. 248). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975.

39. Smith, R. L., Graves, H., Blaine, L. H., & Marinov, V. G. Computer-assisted axiomatic mathematics: Informal rigor. In O. Lecarme & R. Lewis (Eds.), *Computers in education, part 1: IFIP*. Amsterdam: North Holland, 1975.

40. Smith, R. L., Smith, N. W., & Rawson, F. L. *Construct: In search of a theory of meaning* (Tech. Rep. 238). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1974.

41. Suppes, P. *Introduction to Logic.* Princeton: Van Nostrand, 1957.

42. Suppes, P. Probabilistic grammars for natural languages. *Synthese*, 1970, *11*, 111-222.

43. Suppes, P. *On the grammar and model-theoretic semantics of children's noun phrases* (Tech. Rep. 181). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1971.

44. Suppes, P. Computer-assisted instruction at Stanford. In *Man and computer.* (Proceedings of international conference, Bordeaux 1970).Basel: Karger, 1972.

45. Suppes, P. New foundations of objective probability: Axioms for propensities. In P. Suppes, L. Henkin, Gr. C. Moisil, & A. Joja (Eds.), *Logic, methodology, and philosophy of science* IV: *Proceedings of the fourth international congress for logic, methodology, and philosophy of science, Bucharest*, 1971. Amsterdam: North Holland, 1973. (a)

46. Suppes, P. Semantics of context-free fragments of natural languages. In K. J. J. Hintikka, J. M. E. Moravcsik, & P. Suppes (Eds.), *Approaches to natural language.* Dordrecht: Reidel, 1973. (b)

47. Suppes, P. The semantics of children's language. *American Psychologist*, 1974, *29*, 103-114.

48. Suppes, P. Impact of computers on curriculum in the schools and universities. In O. Lecarme & R. Lewis(Eds.), *Computers in education, part 1: IFIP.* Amsterdam: North-Holland, 1975.

49. Suppes, P., Jerman, M., & Brian, D. *Computer-assisted instruction: Stanford's 1965-66 arithmetic program.* New York: Academic Press, 1968.

50. Suppes, P., Leveille, M., & Smith, R. L. *Developmental models of a French child's syntax* (Tech. Rep. 243). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1974.

51. Suppes, P., & Morningstar, M. Computer-assisted instruction. *Science*, 1969, *166*, 343-350.

52. Suppes, P., Smith, R., & Leveille, M. *The French syntax and semantics of PHILLIPE, part 1: Noun phrases* (Tech. Rep. 195). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1972.

53. Van Campen, J. A. Project for application of learning theory to problems of second language acquisition with particular reference to Russian. Report to U. S. Office of Education, Contract No. OEC-0-8-001209-1806, 1970.

54. Van Campen, J. A. *A computer-based introduction to the morphology of Old Church Slavonic* (Tech. Rep. 205). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1973.

55. Van Campen, J. A., & Schupbach, R. *Computer-aided instruction in Old Church Slavic and the history of the Russian literary language* (Tech. Rep. 255). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975.

56. Van Lehn, K. *SAIL users manual* (Artificial Intelligence Memo 204). Stanford, Calif.: Stanford Artificial Intelligence Laboratory, Stanford University, 1973.

57. Weissman, C. *LISP 1.5 primer*. Belmont, Calif.: Dickenson, 1967.

58. Wexler, K. N. *An automaton analysis of the learning of a miniature system of Japanese* (Tech. Rep. 156). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1970.

59. Weyer, S. A., & Cannara, A. B. *Children learning computer programming: Experiments with languages, curricula, and programmable devices* (Tech. Rep. 250). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975.

60. Wolpert, T. M. *TENEX UCI-LISP*. Unpublished manuscript, Stanford University, 1975.