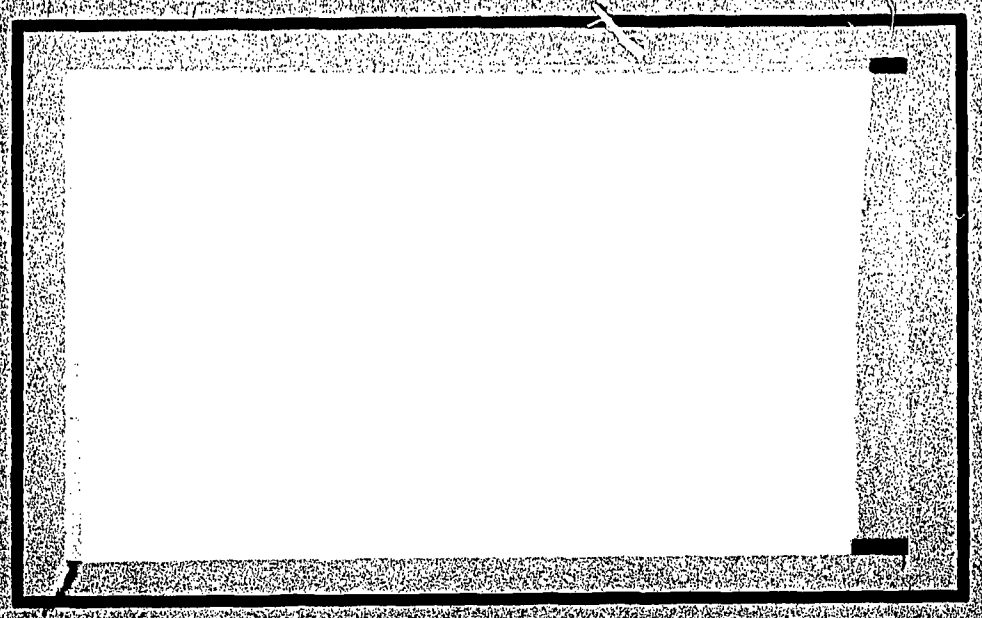
DOCUMENT RESUME

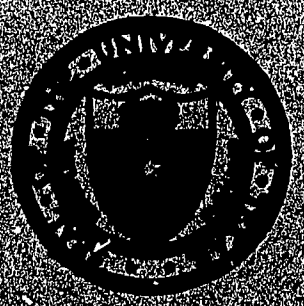
ED 053 534 EM 009 056

AUTHOR	Siklossy, Laurent
TITLE	Topics in CAI: Information Transfers and Review. Paper I; Control and Feedback in the Environment of a Computer Tutor. Paper II; Review in CAI: The Problem and an Implemented Solution.
INSTITUTION	Texas Univ., Austin. Computer-Assisted Instruction Lab.
SPONS AGENCY	Texas Univ., Austin. Computation Center.; Texas Univ., Austin. Electronics Research Center.
PUB DATE	Mar 71
NOTE	45p.; Paper I, to be presented at the International Symposium of the American Society for Cybernetics (4th Annual, Washington, D.C., October, 1971)
EDRS PRICE	EDRS Price MF-$0.65 HC-$3.29
DESCRIPTORS	*Computer Assisted Instruction, Computer Science, Educational Diagnosis, *Feedback, *Instructional Design, *Interaction, Learning Processes, *Review (Reexamination)

ABSTRACT

Two papers are included in this report. "Control and Feedback in the Environment of a Computer Tutor" investigates some control and feedback properties of a tutorial environment comprised of a student, his computer tutor, and the designer of the computer tutor. Three classes of computer tutor are described: rigid, generative, and knowledgeable. These classes are distinguished by an increase of the interactions among tutor and student and a decrease of the control of the tutor over the tutorial environment. Full partnership between tutor and student could be achieved with the help of programs that understand natural language. "Review in CAI: The Problem and an Implemented Solution" discusses review of material, which is made necessary by human memory loss. The computer tutor that incorporates review is built around performance programs that know what the student is to learn, generator programs that generate problems the student must learn to solve, and extensive diagnostic programs that guide the tutor in his interaction with the student. Borrowing compiler-writing techniques, a multi-level system for review has been implemented that avoids extra coding and repetition of material in identical form. (Author/JK)





THE UNIVERSITY OF TEXAS AT AUSTIN

Computer Assisted Instruction Laboratory

AUSTIN

TOPICS IN CAI:

INFORMATION TRANSFERS AND REVIEW

Laurent Siklóssy

March 1971

Computer Sciences Department
and
Electronics Research Center
The University of Texas at Austin
Austin, Texas 78712

P R E F A C E

This report contains two papers. The first paper, entitled "Control and Feedback in the Environment of a Computer Tutor," will be published in the *Proceedings of the Fourth International Symposium of the American Society for Cybernetics*. The inclusion of the paper in this report is justified by the long delay in publication of the *Proceedings*. The paper investigates the interrelationships between a student, his computer tutor, and the designer of the computer tutor.

The second paper, entitled "Review in CAI: The Problem and an Implemented Solution," describes a technique that we have developed to treat the need for review in a tutorial environment. The paper also compares two organizations of a computer tutor: as a single tutor or as a multiplicity of tutors cooperating among themselves. Our approach for review is best understood in the framework of a group approach to teaching.

The interested reader may wish to consult other related publications by the author:

Computer Tutors That Know What They Teach. *Proceedings of the Fall Joint Computer Conference*, 251-255, 1970.

Computer Information Structures Teaching by Computer (with L. D. Shroyer). *Proceedings of the Fifth Annual Princeton Conference on Information Sciences and Systems*, 1971.

Let Us Build Intelligent Computer Tutors. *Computers and Automation*, March, 1971.

The last two papers describe preliminary results in mechanizing the teaching of computer data and storage structures. In the present curriculum, computer structures are usually taught to advanced college undergraduates.

CONTROL AND FEEDBACK IN THE ENVIRONMENT OF A COMPUTER TUTOR

Laurent Siklóssy

To be published in:

Proceedings of the Fourth International Symposium of the American Society for Cybernetics
Spartan Books, 1971

Computer Sciences Department
The University of Texas at Austin
Austin, Texas 78712

TABLE OF CONTENTS

ABSTRACT

An essential ingredient of man's environment is ... man! The training of man to interact harmoniously with his own human environment will be an increasingly important problem. As one means to solution, the role of computers as teaching agents is increasing.

The present study investigates some control and feedback properties of a tutorial environment comprised of a student, his computer tutor and the designer of the computer tutor. Three classes of computer tutors are described: rigid, generative, and knowledgeable. These classes are distinguished by an increase of the interactions among tutor and student and a decrease of the control of the tutor over the tutorial environment. Full partnership between tutor and student could be achieved with the help of programs that understand natural language.

1 INTRODUCTION

Man has changed profoundly his environment. Man has also multiplied to the extent that further population growth poses serious problems. Future generations will need to learn how to interact harmoniously both with the physical world and with the large populations that will constitute an essential part of the human environment. Men and books have been the traditional teachers of man. In the past decade, however, considerable interest has been raised by possible applications of computers to teaching. A book does not carry on a dialogue with a student, and individualized teaching of man by man can be afforded only by the wealthiest. Thus, truly individualized instruction for the masses is conceivable only if some mechanical device, such as a computer, becomes the teacher.

A recent compendium by Lekan (1970) lists 910 computer assisted instruction (CAI) programs. This large number indicates the interest that computer tutors already engender. We shall see how a cybernetic approach to the description of computer tutors helps us to understand their past and present development.

1.1 Cybernetics and the Educational Process.

Couffignal (1964) discussed in general some contributions that cybernetics could make to education. Batteau (1968) has shown that feedback is essential to ascertain that information has been transmitted. The importance of feedback, and its utilization, is at the heart of some recent work in the uses of computers in teaching (Bestougeff et. al., 1970). The arithmetic capabilities of the computer are widely used for the bookkeeping chores (record-keeping, etc.) that accompany

the educational process. We shall not be interested here in these activities of the computer but shall concentrate instead on the tutorial interaction between a student and his computer tutor. We shall show how various stages in the development of computer tutors can be viewed as particularizations of a simple system comprised of two interacting information-processing systems: a student and a tutor.

2 THE TUTORIAL ENVIRONMENT

We shall be concerned primarily with the system comprised of a student and his tutor, in our case a computer tutor. We shall also talk briefly about a larger system in which the designer of a computer tutor interacts with the previous student-tutor system. The designer's task is the development of a computer program with specific goals, namely the teaching of some subject to certain categories of students.

2.1 The Student-Tutor System.

In Figure 1 we have a schematic outline of the paths through which information flows in a system comprised of a student and his tutor. We assume that the designer of the system has stepped out of the picture. The dynamic development of the interaction between student and tutor would constitute input data that the designer could utilize to modify the tutor.

Insert Figure 1 about here.

In Figure 1 we have labelled the information paths in the system. The path T-S carries information from tutor to student. The message carried may be a problem that the tutor is submitting to the student. The path T-T is a feedback path on the tutor. It could carry the same problem to the problem-solving capabilities of the tutor. The student could send his solution of the problem along the S-T path. Along the S-S path the student could send some method that he discovered while solving the problem. This method may be stored or generalized, or it may lead to some questions from the student to the tutor, again along the S-T path.

When we consider actual computer tutors, we find that the kinds of information that can be transmitted fruitfully along the information paths of a tutorial environment are limited to various degrees. For example, in some systems, the S-T path can only carry numbers between one and four. These numbers represent answers to multiple-choice questions transmitted from the tutor along the T-S path. In sections 3, 4 and 5 we shall describe three kinds of computer tutors which are increasingly general in respect to the types of information that can be transmitted, and appropriately interpreted, in the tutorial environment. We shall focus in particular on paths T-S and S-T.

2.2 The Teaching System and its Designer.

At the level of generality of this section it is not possible to be very explicit about the dynamic interactions between a designer and his tutor. The design of a computer tutor results from a long experimentation with a succession of tutors. It is estimated that from fifty to two hundred hours of designer time go into the preparation of a computer tutor for each hour of student interaction with the tutor. If the design involves some search for optimization of the student's learning rate, the designer is faced with the frustrating discovery that what should, logically, count does not always do so. For example, if the designer builds up his course of a logical sequence of units, he may find that scrambling these units, and consequently presenting them to the student in some "cart before the horse" order, does not necessarily influence the learning rate of the student. The evidence is not conclusive, however. (See for instance Roe, 1962 and Roe et. al., 1962). Some theoretical framework is, however, emerging in which computer-aided instruction programs can be designed (Bunderson, 1969).

We shall have more to say about the tools available to a designer as we consider three classes of computer tutors.

3 RIGID COMPUTER TUTORS

The first computer tutors have a structure very similar to a scrambled textbook. The designer has divided the material to be taught into a number of frames. The structure of a frame is shown in the flow-diagram of Figure 2. In this figure, as in the subsequent figures, a box labelled Tn, where n is an integer, indicates an activity of the tutor, while a box labelled Sn indicates an activity of the student.

Insert Figure 2 about here.

At the start of a frame the student will be exposed, typically, to some statements by the tutor (box T1). A question then follows (T2). Control, which had until now rested with the tutor, passes to the student who answers the question. Control immediately returns to the tutor which compares the student's answer to a finite number of stored items or schemata (T3). For example, if the answer is the number 3 (the result of some computation), the system may accept only 3 as an answer, or it may also accept 3.0, or even any expression that the tutor can evaluate to 3, such as 9/3. The tutor takes into account the student's last answer, and possibly some function of past performance, and decides to transfer to some particular frame (T4).

3.1 Discussion.

We shall discuss rigid computer tutors in terms of our schema in Figure 1. The main information path is T-S, carrying statements by the tutor, its questions to the student and its comments on the answers from the student. Uttal (1968) distinguishes between two types of rigid computer tutors, depending on the respective weights of statements and questions.

He calls "degenerate computer teaching machine" a computer tutor that has essentially the appearance of a scrambled text book: long statements (typed out by the system or presented on slides or microfiches) followed by multiple-choice questions, answers to which determine the next frame to be considered. On the other hand, in what he terms a "selective computer teaching machine", statements and questions share more equitably the T-S information path. In either case, the designer has had to preprogram all statements, questions and problems that will be presented to the student. The material is rigidly fixed, and it is this rigidity that made us choose the classificatory name of the tutors that can be described by Figure 2.

The S-T path is atrophied to various degrees. In many cases only multiple-choice answers are allowed. The student may have to choose among answers none of which he feels are accurate. Moreover, the student may not ask any questions that had not been offered by the tutor as possible questions.

3.2 The Designer's Tools.

Although unsatisfactory from many points of view, rigid computer tutors have met with considerable success. Accompanying and contributing to their success is the development of specialized computer languages for CAI: such as COURSEWRITER (the most widespread, IBM produced), PLATO, LYRIC, PLANIT, etc. Computer systems have been developed with various "bells and whistles", including slide or microfiche projectors, scope terminals, film loops, sound tape loops, etc. The programming languages facilitate the writing of text, questions and the diagnostic functions and their resultant branching logic. The languages also incorporate various bookkeeping facilities that allow the designer to fine-tune his system.

The limitations of these special languages are becoming more and more apparent. Attempts have been made to give them some of the facilities of general-purpose computer languages. PLANIT (Feingold, 1967) has a function definition capability, while some other CAI languages can include a compiler/interpreter for a general-purpose algebraic language (see INTERFACE, 1970).

4 GENERATIVE COMPUTER TUTORS

The first attempts at removing the deficiencies of rigid computer tutors were directed neither at the overwhelming control exercised by the computer tutor during the tutorial "dialogue" nor at the very limited informational transfer capabilities of the S-T path; rather they aimed at facilitating the designer's task. In a rigid computer tutor the designer must code explicitly all questions or problems that will be presented to the student, and he must predict all answers to which the tutor can answer knowledgeably. All unpredicted answers are lumped together and result in a typical response of "You are wrong. Try again." by the tutor.

In a generative computer tutor, the questions or problems, instead of being coded explicitly by the designer, are generated by a program, the generator program. The designer now needs only to code the generator program to assure the implicit coding of a very large number of problems.

- -

Insert Figure 3 about here.

- -

Figure 3 is a flow-diagram for the frame of a generative computer tutor as viewed by the author. Again the tutor may present some statements to the student (not represented in the flow-diagram). Then, depending on some input parameter (at first determined by the designer, subsequently determined by the strategy program: see below) the generator program (box T1) generates a sample of some universe. This sample could be a question, a problem, or data for some function that is to be learned. For example, the generator program may have generated a set with a given number of elements - the number depending possibly on past student performance. The question is to determine the number of elements in the set.

Control then branches in parallel both to the tutor (T2) and to the student (S1). (This is the meaning of the parallel horizontal lines in the flow-diagram. See Chapin, 1970). The student attempts to manipulate the sample, i.e. answer the question, solve the problem, or apply the function to the data (box S1). The question also feeds back to the tutor along the T-T path; and a program in the tutor, the performance program, also tries to manipulate the sample (T2). To continue our example, both student and program will try to determine the number of elements in the given set, the tutor by applying some algorithm, the student some method that he has learned.

The student communicates his result to the tutor which compares it with the result that it computed. Both results are then input to a diagnostic program (T3). This program may determine that the student's result was not of the right kind (not a number), was a correct number, or was an incorrect number. The incorrect number may have been, in varying degrees, either too long or too short, or may even have been nonsensical (negative or fractional).

The results of the diagnostic program are kept by the tutor and communicated to its strategy program (T4). The strategy program may then communicate aspects of the diagnosis to the student (T5). In our example, the tutor may choose to indicate only that the answer is incorrect, or be more specific in why it is incorrect. The tutor may terminate the frame (F2), transfer to a new frame (F1), or return to the same frame. In the last instance, the input to the generator routine may be different, in which case a set with a different length will be generated. If the set generator routine randomly selects elements of a set from a pool of elements, then even an unchanged length parameter may result in a different set.

The advantages gained by the designer may be partially upset by the difficulty in comparing the paths taken by various students through a number of frames. In a rigid computer system, the frames for various students are a rearrangement of a fairly small, fixed number of frames. In a generative computer tutor, only the frame schemata are fixed; the actual presentations may vary widely from student to student.

4.1 Discussion.

If the designer can code generator and performance programs for the subject matter that he tries to have the computer tutor teach, then his task becomes significantly easier. Instead of carefully precoding a large number of examples of various degrees of difficulty, and their answers, he only needs to determine some strategy for the generation of input parameters to the generator routine.

In terms of our model (Figure 1), the tutor still has overwhelming control of the tutorial dialogue. The S-T path is more developed than before because now the tutor can process meaningfully a wide variety of student responses since processing is carried out by some evaluative computer program instead of by a simple procedure that matches for identity.

The feedback path T-T is barely used in a rigid computer tutor; here it carries samples to be manipulated from T1 to T2, diagnoses that can be explicated to the student at various levels of detail and diagnoses that help determine the next move by the tutor. The S-S path is not used explicitly in this computer tutor.

4.2 Examples.

As illustrations we shall briefly describe some generative computer tutors. Uhr (1965 or 1969) has described a very

elementary program, written in SNOBOL, to teach addition and word-for-word translation from one language into another. The program generates numbers, gives them to the student, sums them and, upon comparison with the student's response, will print a diagnosis like "your answer is too large." Wexler's system (1968) is a more developed program that teaches the four elementary arithmetic operations. Wexler's program is written in ALGOL, as is Peplinski's (1968), a computer tutor to teach the solution of quadratic equations in one variable. Peplinski's program generates random quadratic equations which are constrained to remain of a certain type and of a certain level of difficulty. Uttal et. al. (1969) describe a system to teach analytic geometry; and their computer tutor is coded in assembly language. Outside of mathematics, but still in a strongly structured domain, Spolsky (1966) has described a program to teach some elementary aspects (such as agreement) of a language to a foreign learner of the language. (Details about an implementation are lacking).

The special-purpose CAI languages mentioned in section 3.2 are not particualrly suitable for the coding of algorithms, and it is no surprise that none of the generative computer tutors mentioned is programmed in one of them. On the other hand, general-purpose programming languages are often harder to learn than CAI languages and are consequently less accessible to some curriculum designers.

5 KNOWLEDGEABLE COMPUTER TUTORS

Generative computer tutors are less tedious to design than rigid computer tutors. They also offer a larger variety of frames and significantly improved diagnostic capabilities. Nevertheless, in both types of tutors, control rests mostly in the tutor. We can extend the capabilities of a generative computer tutor, and thereby diminish the imbalance between tutor and student, if instead of concentrating on the generative capabilities, we focus attention on the performance program that manipulates the samples generated by the generative program.

The performance program of a generative computer tutor can solve problems in some universe. The program may not know the subject matter (and it is not clear what "know" means), but it certainly performs as if it knew: the program "can do." Just as it can answer questions generated by the generative program, the performance program can answer questions generated by the student. The performance program embodies some method to solve some problems and this method can be made explicit by the tutor. The tutor can show the student a trace of its problem-solving behavior. The student can then learn by imitating the methods of the tutor.

- -

Insert Figure 4 about here.

- -

Figure 4 describes a frame of such an extended generative computer tutor. We call such a tutor, in which the know-how of the performance program is fully put to contribution, a knowledgeable computer tutor. The corresponding tutor boxes in Figures 3 and 4 are labelled identically. We have already discussed the path F1, T1, T2, S2, T3, T4, and T5. The tutor also has the option to explain the processes utilized by the performance program (T6).

Initially, control can also be relinquished to the student, who can generate some sample of the universe under consideration (S1). We can justify this process from two points of view: a sample generated by the student could be the equivalent of a question addressed to the tutor, or it could be a quiz given by the student to the tutor.

In the first case, both tutor and student manipulate the sample generated by the student (T2'and S2', S3 is bypassed). The previous path is reentered at T3. At T4, the program may, if the student so desires (not indicated in the flow-diagram), branch to T6 where the introspection program will explain how the tutor computed its answer. Box S2' could also be bypassed and the tutor would simply manipulate the student's sample and, possibly, explain the obtained result.

In an effort to equalize both the tutor's and the student's roles, and to introduce an element of fun, we allow the student to quiz the tutor. The tutor can go into a "dumb" mode (T7), and in this mode it will sometimes make mistakes. The student can catch the tutor in error (S3), and we could conceivably give back to the tutor its knowledge, at which time the tutor could ascertain that the student's diagnosis was correct.

5.1 Example.

We shall postpone general discussion of knowledgeable computer tutors until section 5.2. Presently we shall introduce aspects of a knowledgeable computer tutor that teaches some elementary set theory. This program is presently under development by the author. The subject matter was chosen for its fundamental interest and for the ease with which appropriate performance and diagnostic programs could be programmed.

The tutor tries to teach some elementary concepts about sets, such as set union, intersection, difference, complement,

etc. The student learns by performing, under the guidance of the computer tutor, operations that embody these concepts; he learns by imitation and practice.

Sets are represented as unordered lists. A set can have sets as elements: the set ((HI S7) DOG 32) has three elements, one of which is itself a set of two elements. Atomic elements (not sets themselves) are alphanumeric identifiers. Multiple copies of the same element are not permitted in the representation of a set: for example, ((A B) (B A)) is not an acceptable representation of a set since the element (A B) occurs twice. (Remember that the sets (A B) and (B A) are equal). The empty set is represented as (). The program is coded in LISP.

Assume that the student has just been introduced to the notions of a set and of the elements of a set. The following conversation occurs.

Tutor: Here is a set (DOG (HI S7) 32). Is S7 an element of the set?

Student: Yes.

The set was generated by a random set generator. The atomic elements are randomly drawn from a pool of such elements. The length (3) and depth (1) of the set are given to the set generator. These numbers could in turn be random numbers in a certain range, and could be generated by a standard random number generator. The depth should be at least one, and the tutor selects an atom which is not in the set but in one of the subsets of the set.

Given the incorrect result, the program could decide to go into the introspection mode and exhibit its method to determine whether something is an element of a set.

Tutor: ...Let us examine one by one the elements of the set (DOG (HI S7) 32) to determine whether any of

the elements is equal to S7. Choose an element of the set.

Student: (HT S7).

Tutor: (HT S7) is an element of the set. Is it equal to S7?...

The tutor checked that the student did give a correct element of the set. At this point the student may see his error, interrupt the tutor and request another chance to solve the original problem.

Using a rigid computer tutor, all of the above could have been obtained except for the variety of sets that the generator program can produce. However, the following similar dialogue could not occur in the framework of a rigid tutor.

Tutor: Give me a set with at least one subset as element.

Student: ((CAT RAT) DOG).

Tutor: Correct. Is RAT an element of the set ((CAT RAT) DOG)?

Student: Yes...

If the student has made enough progress, he is allowed to quiz the tutor which can then go into the "dumb" mode.

Tutor: Give me a set and I shall give you an element of the set. Then you can tell me whether I am right or wrong.

Student: ()

Tutor: ( )

Student: Wrong.

Tutor: Oops, how right you are.

There is nothing like giving impossible problems to the tutor (or for that matter to the student)!

The possible diagnostic capabilities are not very evident from these examples. Tables 1 and 2 show the more extensive

diagnostic capabilities of the tutor when teaching set union and intersection. For both tables, inputs are two sets S_1 and S_2 and the student's answer A has been checked to be a set.

- -

Insert Table 1 about here.

- -

- -

Insert Table 2 about here.

- -

	Cases Determined by Diagnostic Program	Possible Partial Comments to Student
1.	$A = S_1 \cup S_2$ (set equality)	Your answer is correct.
2.	$A \neq S_1 \cup S_2$	Your answer is incorrect.
	2-1. $(S_1 \cup S_2) - A \neq ()$	You left out some element(s).
	2-1-1. $((S_1 \cup S_2) - A) \cap S_1 \neq ()$	You left out some element(s) of the first set.
	2-1-2. $((S_1 \cup S_2) - A) \cap S_2 \neq ()$	You left out some element(s) of the second set.
	2-2. $A - (S_1 \cup S_2) \neq ()$	Some element(s) in your answer are neither in S_1 nor in S_2.

Table 1. Diagnostic Program for Set Union and Some Possible Comments.

Cases Determined by Diagnostic Program	Possible Partial Comments to Student
1. $A = S_1 \cap S_2$ (set equality)	Your answer is correct.
2. $A \neq S_1 \cap S_2$	Your answer is incorrect.
2-1. $(S_1 \cap S_2) - A \neq ()$	You left out some element(s) which belong to both S_1 and S_2.
2-2. $A - (S_1 \cap S_2) \neq ()$	Some elements in your answer do not belong to both S_1 and S_2.
2-2-1. $(A - S_1) \cap S_2 \neq ()$	Some element(s) in your answer belong to S_2 but not to S_1.
2-2-2. $(A - S_2) \cap S_1 \neq ()$	Some element(s) in your answer belong to S_1 but not to S_2.
2-2-3. $A - (S_1 \cup S_2) \neq ()$	Some elements in your answer are neither in S_1 nor in S_2.

Table 2. Diagnostic Program for Set Intersection and Some Possible Comments.

5.2 Discussion.

In a knowledgeable computer tutor, the information path S-T can carry not only the student's responses to questions of the tutor but also his questions to the tutor. The feedback path S-S carries the same questions. The S-T path also can carry the student's diagnosis of the performance of the tutor.

Although the roles of the tutor and student are more symmetrical, total symmetry has not been reached yet. The tutor still has overall control of the tutorial environment. The student cannot communicate the reasons for his diagnoses, nor can he explain his procedures to the tutor. These deficiencies in the dialogue between tutor and student are to a large extent due to the inadequacy of programs that understand natural language. Since we do not wish to tackle the problems of natural language in our work, we restrict the English vocabulary that the student may use to a few words such as : yes, no, right, wrong, enough, impossible, etc. Progress is being made in the design of programs that can accept students' responses much freer in nature (Simmons, 1968).

Another lack of symmetry stems from the fact that while the student learns during the tutorial dialogue, the tutor does not. We would like the tutor to learn both about the student and about the subject matter. The scarcity of programs that learn in a manner comparable to human learning (Siklóssy, 1968) does not bode well for the future of learning computer tutors.

It is pedagogically sound to allow students to generate their own problems and examples. Not only is it impossible to preprogram a universally sufficient set of examples to teach some concept, but some experimental evidence (Hunt, 1965; Crothers and Suppes 1967, Ch. 7) indicates that the rate of learning is greater when students generate their own examples.

6 CONCLUSIONS

We have considered a tutorial environment consisting of three interacting information-processing systems: a student, a computer tutor and a designer of the computer tutor. In a rigid computer tutor, control lies almost exclusively with the tutor, and the communication paths initiating from the student are hardly ever used. The designer has many special-purpose programming languages to help him design and test the tutor, but his work is often very tedious.

In a generative computer tutor, programs have taken over some of the chores of the designer. The tutor still has major control over the tutorial environment but can accept and process a much larger class of student's responses. In a knowledgeable computer tutor, the student can ask questions from the tutor and the tutor can explain the methods that it uses to answer questions that it, or the student, generated. The inability of the computer to understand natural language does not make it meaningful for the student to explain his methods to the tutor and prevents, at this stage, the tutorial environment to become a partnership of equals.

7 ACKNOWLEDGEMENTS

J. Peterson, S. Slykhous and A. Duke have contributed to the design and programming of the set theory teacher.

8 REFERENCES

Batteau, D. W., "Feedback and knowledge," Proceedings of the 5th International Congress on Cybernetics, Association Internationale de Cybernétique, Namur, Belgium, 672-675, 1968.

Bestougeff, H., Fargette, J. -P. and Jacoud, R., "Computer-aided control of learning," IEEE Trans. on Education, 12, 1, 4-7, 1970.

Bunderson, C. V., "Ability by treatment interactions in designing instruction for a hierarchical learning task," Paper presented at the annual meeting of the American Educational Research Association, Los Angeles, Calif., February 1969.

Chapin, N., "Flowcharting with the ANSI standard: a tutorial," Computing Surveys, 2, 2, 119-146, 1970.

Couffignal, L., "Que peut apporter la cybernétique a la pédagogie?" Cybernetica, 7, 1, 11-18, 1964.

Crothers, E. and Suppes, P., Experiments in second-language learning, Academic Press, New York, 1967.

Feingold, S. L., "PLANIT - a flexible language designed for computer-human interaction," Proceedings of the Fall Joint Computer Conference, 545-552, 1967.

Hunt, E. B., "Selection and reception conditions in grammar and concept learning," J. Verbal Learn. Verbal Behav., 4, 211-215, 1965.

INTERFACE, Bulletin of the ACM Special Interest Group on Computer Uses in Education, 4, 3, p. 17, June 1970.

Lekan, H. A., Index to computer assisted instruction, Sterling Institute, Boston, Mass., 1970.

Peplinski, C., "A generating system for CAI teaching of simple algebra problems," Computer Sciences Technical Report #24, University of Wisconsin, 1968.

Roe, A., "A comparison of branching methods for programmed learning," Journal of Educational Research, 55, 407-416, 1962.

Roe, V. K., Case, H. W. and Roe, A., "Scrambled vs. ordered sequence in auto-instructional programs," Journal of Educational Psychology, 53, 2, 101-104, 1962.

Siklossy, L., "Natural language learning by computer," doctoral dissertation, Carnegie-Mellon University, Pittsburgh, Pa., 1968.

Simmons, R. F., "Linguistic analysis of constructed student responses in CAI," Report TNN-86, The University of Texas at Austin, Computation Center, 1968.

Spolsky, B., "Some problems of computer-based instruction," Behavioral Science, 11, 6, 487-496, 1966.

Uhr, L., "The automatic generation of teaching machine programs," Center for Teaching and Learning Technical Report, University of Michigan, August 1965.

Uhr, L., "Teaching machine programs that generate problems as a function of interaction with students," Proceedings of the 24th ACM National Conference, 125-134, 1969.

Uttal, W. R., Pasich, T., Rogers, M. and Hieronymus, R., "Generative computer assisted instruction," Communication #243, University of Michigan, Mental Health Research Institute, 1969.

Wexler, J. D., "A self-directing teaching program that generates simple arithmetic problems," Computer Science Technical Report #19, University of Wisconsin, 1968.

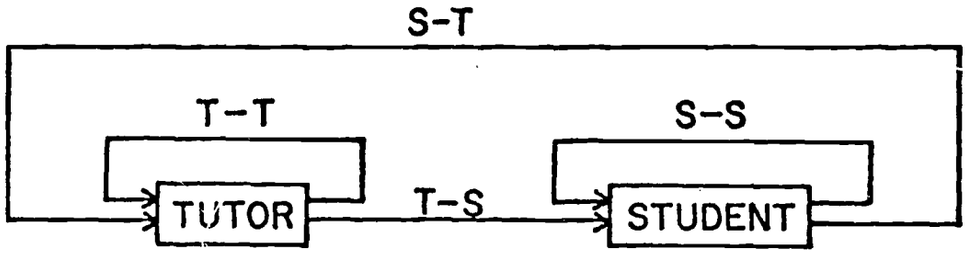


FIGURE 1. TRANSFER OF INFORMATION IN A TUTORIAL ENVIRONMENT

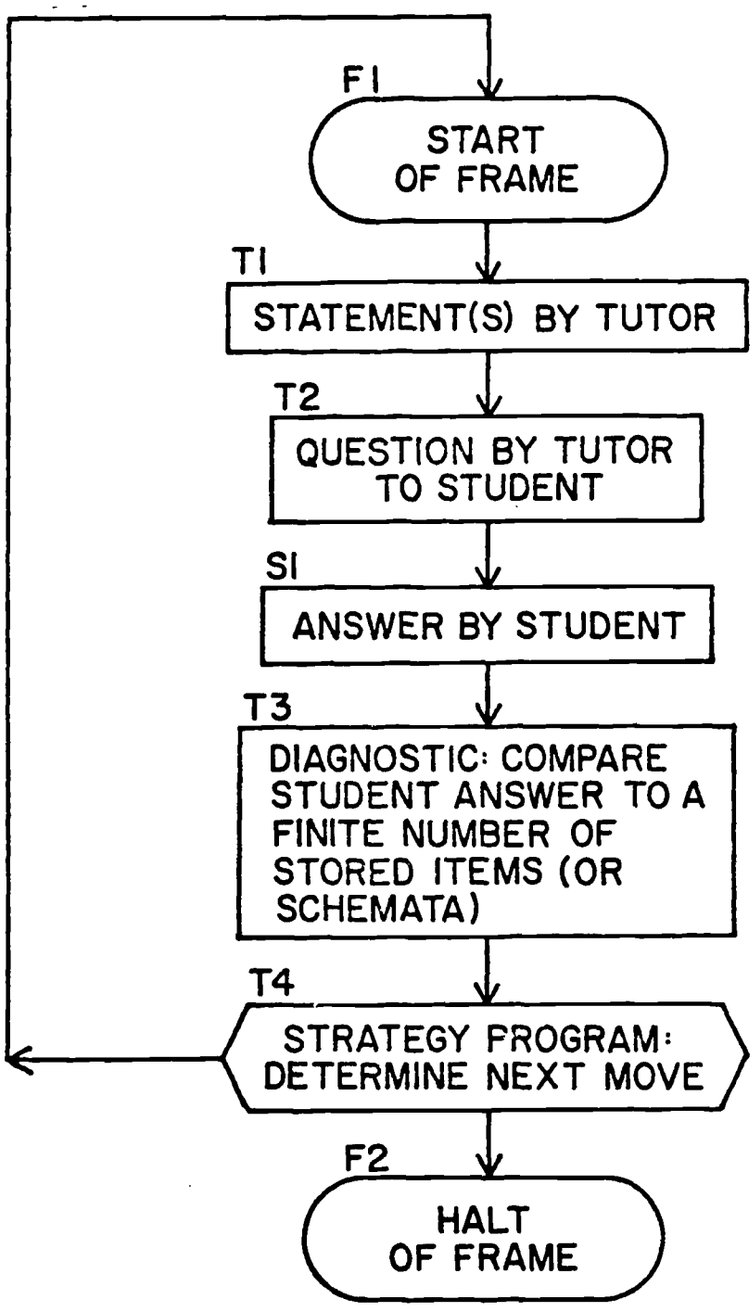


FIGURE 2. FRAME OF A RIGID COMPUTER TUTOR

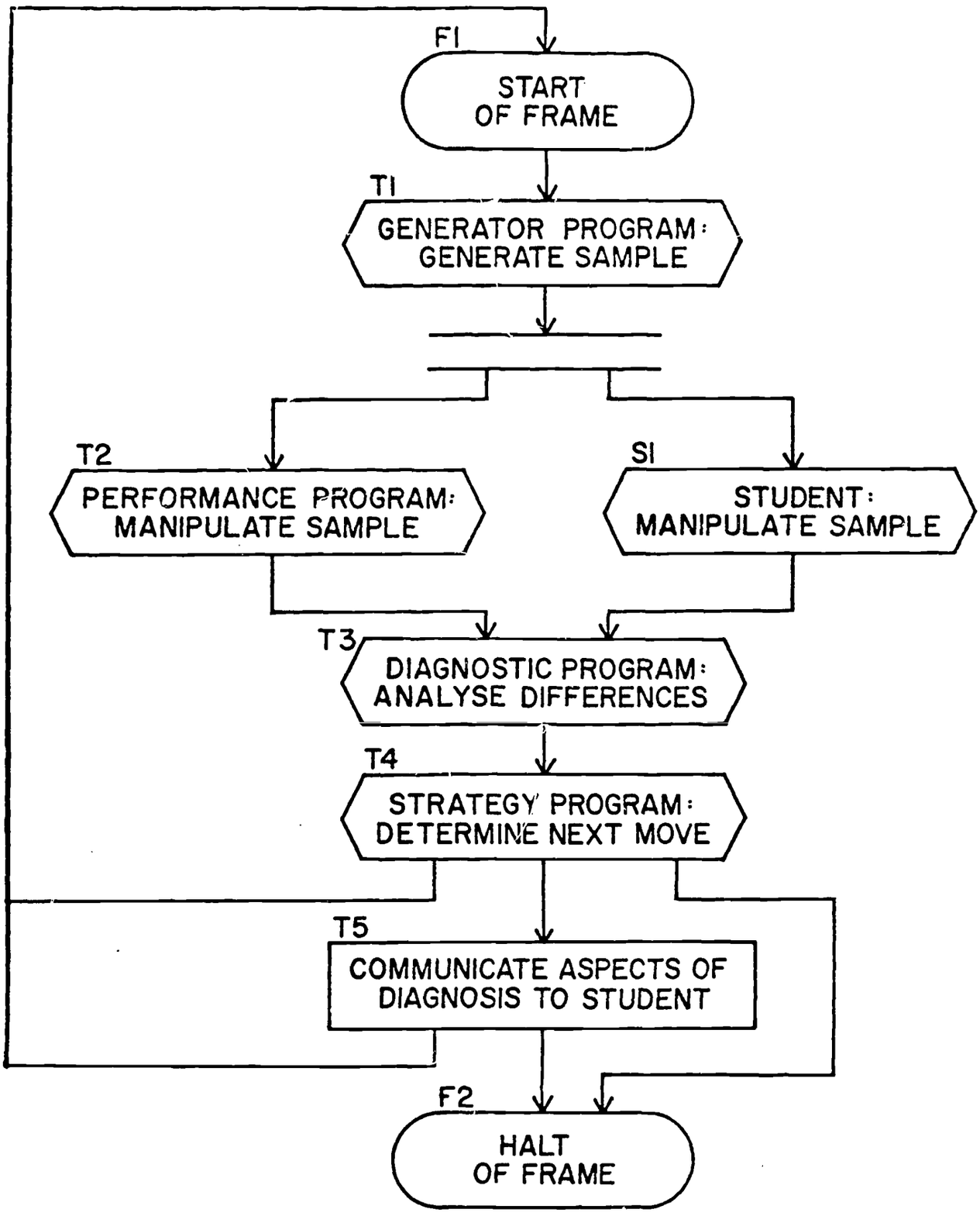


FIGURE 3. FRAME OF A GENERATIVE COMPUTER TUTOR

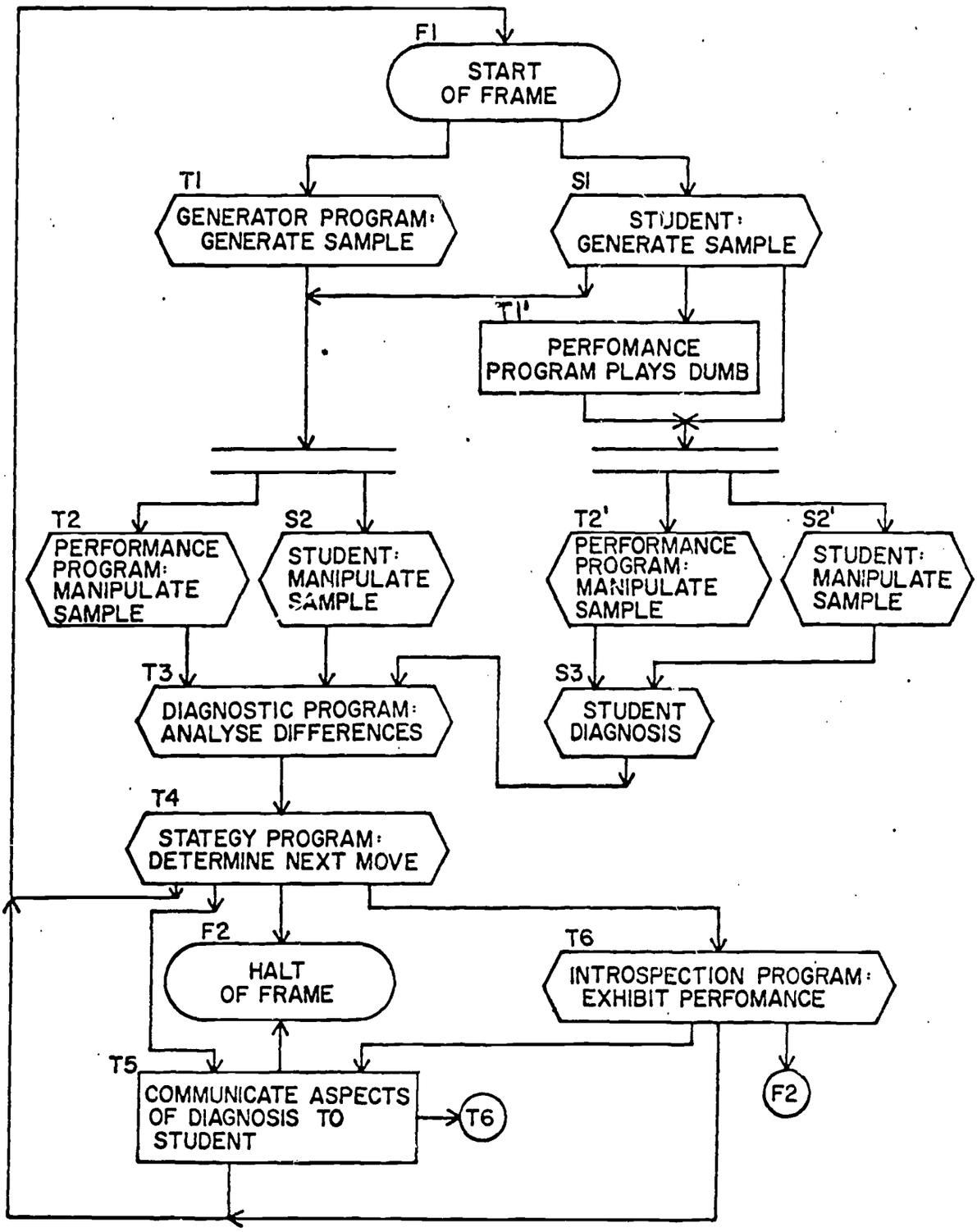


FIGURE 4. FRAME OF A KNOWLEDGEABLE COMPUTER TUTOR

REVIEW IN CAI:

THE PROBLEM AND AN IMPLEMENTED SOLUTION

Laurent Siklóssy

Computer Sciences Department
The University of Texas at Austin
Austin, Texas 78712

ABSTRACT

Review is made necessary by human memory loss. The computer tutor that incorporates our implementation of review is built around: performance programs that know what the student is to learn, generator programs that generate problems that the student must learn to solve, and extensive diagnostic programs that guide the tutor in his interaction with the student.

Conceptually, a tutor that has diagnosed the need for review sends the student to another tutor. Borrowing compiler-writing techniques, we have implemented a multi-level system for review that avoids both the extra coding that has been used for review in other computer teaching systems, and the repetition of material that had already been presented in an identical form.

I. INTRODUCTION

Humans forget. In a tutorial environment, it happens eventually that the student has forgotten some material that he needs for furthering his learning. The student must review the forgotten material if he is to progress. Any tutorial system must incorporate provisions for allowing review to occur, whether it is the student or the tutor who diagnoses the need for review.

In section 2 we describe how, when trying to solve the problem of review, our view of the computer tutor shifted from an emphasis on material to an emphasis on the teacher who interacts with the student on conceptual material that can be particularized by either the teacher or the student. We were led, further, to consider a computer tutor as the team effort of a multiplicity of tutors, mini-tutors, who are each responsible for communicating some material to the student. In section 3 we describe briefly the paths that a mini-tutor can take. The framework of a mini-tutor has been described elsewhere [1,2]. In reference 1 we have shown how such a framework significantly amplifies communication between student and tutor. In reference 2, it was shown that if the mini-tutor does not act quite intelligently, it does act knowledgeably, i.e. it knows what it teaches.

In section 4 we describe the interaction of the mini-tutors and the implementation of review in a program. Examples drawn from a course to teach set theory are given in section 5.

II. TWO VIEWS OF THE TUTORIAL ENVIRONMENT

The goal of a teaching system, whether man or machine, is usually viewed as the transmission of some information to a student. Two different views, and two different implementations, of computer tutors result depending on whether emphasis is placed on the material to be communicated or on the tutor who interacts with the student.

2.1 Emphasis on the Material.

If the material is emphasized, the main concern of the designer of a computer tutor is the organization of the material and its presentation to the student. Typically, most CAI efforts can be viewed as emphasizing material. CAI languages provide multiple ways to break the subject matter to be communicated into fragments of various sizes (frames, chapters, sections, etc.). A CAI system often provides the designer with several media (teletype, audio-equipment, CRT, film, etc.) through which fragments can be presented. CAI languages also facilitate the transfer of attention of the computer teacher from one fragment to another. Implicitly, only one teacher is assumed.

When emphasis in on material, the need for review is diagnosed as the forgetting of some subject matter SM. A student who needs to review SM is presented some review material. This material may be old, having been coded for the original presentation of SM, in which case it may often happen that the student remembers the answers (for example, to multiple-choice questions) without, in fact, understanding SM. The review material may also be new, having been designed specially to provide review at a particular diagnostic point. If, during the total course, the student may need to review the same segment of material

several times, the additional programming chore could be considerable. For each such potential review, separate code must be written.

2.2 Emphasis on the Teacher.

If we shift the emphasis from material to teacher in the tutorial process, the transmission of information to the student can be viewed as the result of the student's interaction with a multiplicity of tutors. We consider each tutor as responsible for communicating some fragment of information. Each tutor is individually responsible for adapting his teaching strategy to the student and, in particular, to the student's previous experience. When a student comes to review some material that is the responsibility of a particular tutor, this tutor is made aware of the fact.

The designer of a computer tutor viewed with this different emphasis must still be concerned about how each individual tutor presents material. But he is also concerned with the interactions of the tutors among themselves. We have solved, programmatically, the interaction of the mini-tutors among themselves by creating a monitor that can call individual mini-tutors into action or fire them! We shall see how such an organization solves, in a very natural way, the problem of review. But we must first describe the tutorial paths that a mini-tutor can take.

III. THE PATHS OF A MINI-TUTOR

A mini-tutor is responsible for transmitting to the student some fragment of information. In figure 1 we have superimposed various paths that a mini-tutor may take during his interaction with a student. Which particular path is actually taken at any particular time may be the result of some decision of the mini-tutor, based on the student's previous performance, for example. The decision may also be left with the student.

Insert Figure 1
about here.

At the heart of a mini-tutor we find performance programs. These are programs that can solve the problems that the mini-tutor is teaching the student how to solve. Along one particular path, the generator program of the tutor generates some problem (box T1 in figure 1). For example, let us assume that our mini-tutor is trying to teach a student about the concept "element of a set".

Tutor: Here is a set (BY HAPPY). Please type an element of this set.

Student: B.

Tutor: No, B is not an element of (BY HAPPY). The elements are: BY HAPPY.*

The set (BY HAPPY) was generated using a guided random set generator. The student has obtained an answer (box S2) which is diagnosed by

* Outputs are taken verbatim from runs of the program. Editorial changes in the form of punctuation and lower-case letters have been made to improve legibility.

the tutor (T3). The tutor calls in an algorithm that checks whether the student's answer, B, is an element of the set (BY HAPPY). The strategy program (T4) branches to T5 and some specific commentary is given to the student. The tutor decides, this time, to go back to F1 and present another randomly generated set to the student.

The tutor could have chosen to exemplify the algorithm it uses to check whether some given item is an element of a set. The algorithm both considers one-by-one the elements of the set, here BY and HAPPY; and tests whether any of these elements is equal (as atom or set) to the given item (T6).

Along another path, the student could generate a set (box S1). The tutor could then calculate some or all of the elements of this set (T2'). This way, the tutor answers a question from the student.

Tutor: Now you give me an example of a set.

Student: (S W E R (T)).

Tutor: That's right. You gave me a set with 5 elements. The elements are: S W E R (T).

If the tutor pretends to make a mistake, by first going into a dumb mode (T1'), then the student can correct the tutor (S3), which is always a satisfaction!

In our implementation, a mini-tutor consists of consecutive elements of an array COURSE. Each element is a LISP expression that is evaluated by the LISP function EVAL. Each element can be a complex program but, in practice, the element is fairly elementary. In this way, the program for a tutor is quite modular and easy to modify. (Modularity is enhanced by designing a tutor in a free-field format; a loader is called to load the array COURSE.) In the next section we shall see how the monitor controls the interaction between the student and the various mini-tutors.

IV. IMPLEMENTATION OF REVIEW

Let us assume that, on some day, a student starts his interaction with the teaching system by meeting the mini-tutor who teaches part of the concept of a subset. The mini-tutor begins, for example, at COURSE(60). The index 60 is pushed on a stack, called STACK, that starts empty. The monitor calls for (EVAL (COURSE 60)), the evaluation of the contents of COURSE(60). After evaluation, control passes back to the monitor who checks the top of STACK.

If the top of STACK is unchanged, it is replaced by the next index (here, 61), and evaluation continues. If the top of STACK has been altered, for example to 25, then monitor calls for the evaluation of COURSE(25). A mini-tutor can change the top of STACK in any of three different ways:

1) by evaluating a transfer function GO1. (GO1 25) would result in changing the top of STACK to 25. This case corresponds to the usual transfer of control in a program.
2) by calling for a mini-tutor to perform review.
3) by terminating review if the tutor is now in charge of providing review.

Let us consider a mini-tutor teaching about subsets:

Tutor: Here is another set (G HI DOG). Type a subset of this set.

Student: G.

Tutor: No, a subset is a set, not an element. You have typed an element of the set (G HI DOG). (G) would be a a subset of (G HI DOG).

After several questions, the mini-tutor decides to send the student to review some material with another mini-tutor teaching about elements of a set. The monitor determines that the reviewing tutor is located at COURSE(25). 25 is pushed down on STACK and COURSE(25) is evaluated.

The above dialogue continues:

Tutor: Here is a set ((FOOT) ()). Please type an element of this set.

Student: (FOOT).

A mini-tutor can establish whether it is in reviewing mode by testing whether the stack has more than one element! When the mini-tutor has terminated its task, the monitor checks whether the tutor was providing review. If so, STACK is popped, the new top of STACK is increased by one and the mini-tutor who asked for review continues teaching with the hope that the student is now better prepared to continue.

The reader will have noticed that the mechanism is similar to the one used to implement recursion in programming languages. The generality of the mechanism implies that a tutor who is helping the student to review can, in turn, call another mini-tutor to perform review on some material, etc. The reader will have noticed also that the mini-tutor teaching about elements uses the same code originally and in review. However, the random set generator will produce new sets, so that the tutor's parts in the dialogue are not identical.

V. CONCLUSION

We have shown how compiler writing techniques could be used to call, "recursively", members of a team of mini-tutors who, together, consitute a computer tutor. A student who needs to review past information is sent to a reviewing mini-tutor who can, if necessary, call other reviewing mini-tutors.

The mini-tutors are written as knowledgeable computer tutors and can perform both initial and review teaching of a concept. As implemented, no significant additional code is needed to give a computer tutor general reviewing capabilities.

VI. ACKNOWLEDGMENT

The computer tutor owes much to the programming help of J. Peterson.

VII. REFERENCES

1. Siklóssy, L., "Control and Feedback in the Environment of a Computer Tutor," Proceedings of the Fourth Annual International Symposium of the American Society for Cybernetics, Washington, D.C., 1970 (in press).

2. Siklóssy, L., "Computer Tutors that Know what They Teach," Proceedings of the Fall Joint Computer Conference, 1970, p. 251-255.

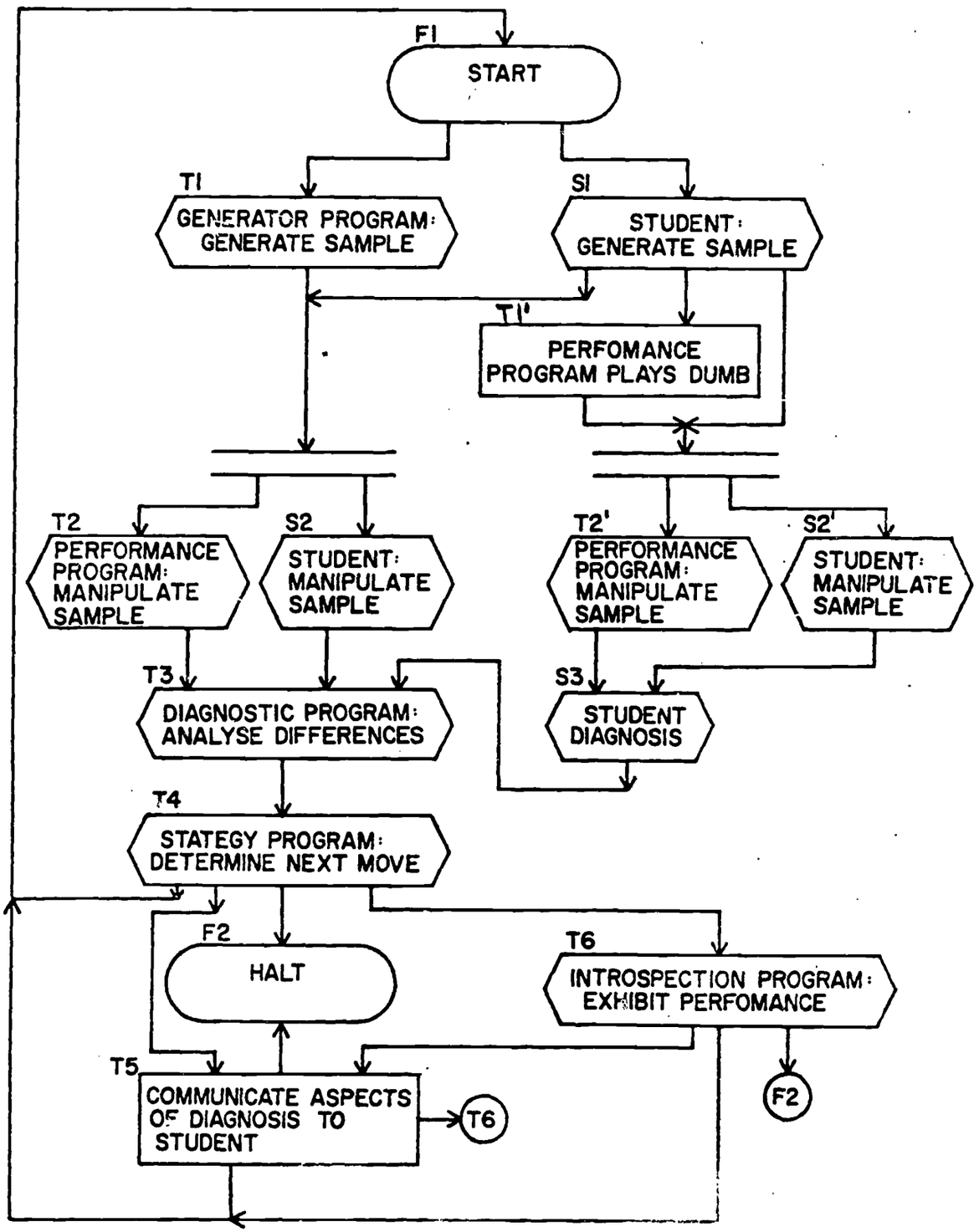


Figure 1. Paths of a Mini-Tutor