PLANIT–A flexible language designed for computer-human interaction*

by SAMUEL L. FEINGOLD
System Development Corporation
Santa Monica, California

INTRODUCTION

Time-sharing has brought about a new age in computer usage: With the advent of time-shared systems, on-line use of the computer will become economically feasible for the average user; and on-line with him will be a wealth of software utility support never before available. No longer faced with a 24-hour turnaround—the average turnaround time in some existing time-sharing systems is a few seconds or less—he can perform program research, development, and checkout with ease undreamed of heretofore.

But time-sharing also has a price: It demands that we do something about an area that has previously been considered too expensive to address properly. We might call that area Computer-Human Interaction (CHI). Before time-sharing, only fragmented CHI could take place unless large blocks of computer time (costing large sums of money) were dedicated to individual users. And during such costly interaction, most of the time the computer was "idling," waiting for the user to make an input. Now that time-sharing provides the capability for many users to converse with the computer simultaneously and for extended periods of time, we are faced with the opportunity—and the need—to refine this interaction.

Many programs have now been written for time-shared use—programs for computer-assisted instruction, computer-based vocational counseling, computer-assisted training, interviewing for medical history, etc. These might all be described as interactive programs: The user may spend an hour or more interacting with the program at the teletype or display console.

The following discussion describes a language (and program) called PLANIT that has proven extremely useful for the preparation and execution of interactive programs.

*The work reported herein was sponsored by SDC and the National Science Foundation under Grant GY-371.

Background of the project

PLANIT (**P**rogramming **LAN**guage for **I**nteraction and **T**eaching) is a system employing a flexible language designed for CHI. PLANIT provides for the designer a powerful and flexible tool for entering communication material into the computer, for modifying the material, for presenting it to the user, and for prescribing the behavior of the computer as a function of the user's current response and past performance. In addition, the language is simple enough to allow a nonprogrammer to use it easily.

The program and language were designed at System Development Corporation by a study team from the Research and Technology Division's Education and Training Staff. The purpose of their project (supported partially by an NSF Grant and partially by SDC's independent research program) was, in general, to explore the use of the computer in the teaching of statistics and, in particular, to develop a set of computer-based instructional materials. Initial efforts produced a first version of the laboratory portion of a course in statistics. A program, called STAT, was coded in a high-level language, JOVIAL.[1] This first program, tested on nine graduate students in psychology at the University of California, Los Angeles, was useful in pointing out a number of problem areas in the material. Moreover, one significant obstacle to our work efforts soon became apparent during the test runs: The capability for changing parts of the lesson during test runs was poor.

There were three team members involved in the development of STAT—a statistician, an educator, and a programmer. Typically, when any one of us wanted a change made, he would first have to find the other two, then explain to them and convince them. Then the material would have to be recoded into JOVIAL, recompiled into another program, and finally checked out. (Meanwhile, often the students were there wait-

ing, being stalled in the conference room.) At any rate, this first go-around showed us that what we needed was a general-purpose program that would assist a nonprogrammer in preparing instructional material for statistics. In addition, the user must be able to enter the material easily and be able to change it as easily. We feel we have succeeded in our attempt to design such a program. (For example, STAT took three men, 14 man months, to produce. We believe that one man—a knowledgeable nonprogrammer—could empty STAT into PLANIT in from two to three months, after he is familiar with the lesson-building and calculation (CALC) modes of PLANIT.)

Related projects

Prior to building PLANIT, the author toured the country in search of a program or ideas that could satisfy the demands of the STAT project. Some of the more interesting programming languages or systems seen some 20 months ago are described below:

A. PLATO (**P**rogrammed **L**ogic for **A**utomatic **T**eaching **O**peration), a program developed by a study team at University of Illinois.[2] PLATO III uses a CDC 1604 computer (48-bit words, 32K words core). Most impressive was PLATO's speed of operation and its on-line editing capability. PLATO's slide-oriented system allows for very fast interactive speed. The access time to any of a possible total of 240 slides is 1/10 second. The slides are ordinary transparencies mounted on transparent plastic sheets, 120 slides to a sheet. The system can take a maximum of two such sheets in the central slide selector. Through CATO (a language developed for writing courses to run under the PLATO system) it is possible to write sophisticated instructional logic.

The system, however, has some features that made it unacceptable for our purpose. The limit of 240 slides imposes a serious restriction on the number and combination of courses going on at the same time without previous notice. Furthermore, though on-line editing is easy for minor changes, any substantial modification of the lesson requires a recompilation of the course. Also, while CATO allows complete freedom for structuring complex instructional logic, the language (FORTRAN-like) probably requires a good deal of training and experience for a user to become proficient in it.

B. COURSEWRITER, an IBM program in use at Pennsylvania State University, Florida State University, University of California at Irvine, UCLA, and University of Michigan.[3] A number of COURSEWRITER systems have been used throughout the country. The systems vary in internal design from interpreter to compiler and employ various computers including the 1410-7740, 1401, 1440, 1460, and 1500 operating systems.

COURSEWRITER, which is being marketed by IBM, is by far the largest operating system in existence. It was designed to be used in a "large" sense, i.e., for many courses simultaneously by many students. The system provides a set of routines for accommodating and evaluating typed responses. This feature is a step toward the analysis of constructed responses, and is ideally suited to lengthy sequences for shaping spelling behavior. (The incorrectly spelled input word is echoed by the system with blanks in place of incorrect characters.) The basic language, COURSEWRITER, is fairly easy to use for lesson preparation, but it has no real calculation capability; and any deviation from the basic language set for special functions not already included requires a knowledge of AUTOCODER.

C. SOCRATES, developed at the Training Research Laboratory, University of Illinois.[4] The program ran under an IBM 1620 computer and IBM 1710 control system dedicated to evaluating multiple-choice responses. This was a small system with no general-purpose software language for author input. This program, to the best of the author's knowledge, is no longer being used.

D. MENTOR, a computer language for programmed discourse developed by Bolt Beranek and Newman Inc., of Cambridge, Massachusetts. It is, in this author's opinion, one of the best languages designed for interactive discourse. For our purposes, however, we needed a sophisticated calculation capability that could operate on student records of past performance and could also be used by students in working exercises.

E. BASIC,[5] developed at Dartmouth, and TELCOMP,[6] developed at Bolt Beranek and Newman Inc. These programs are similar in nature: Both are unsuitable for use by nonprogrammers, since they employ high-level language. They are used to program completed routines allowing the student to explore the computing potentials of these languages.

Failing to find a program that satisfied our particular needs, we designed our own. This eventually became PLANIT.

Description of PLANIT

The initial design started in January of 1966; by June PLANIT was operational. It was written in JOVIAL and used the IBM AN/FSQ-32V computer via an interactive console under the SDC time-sharing system.

Our original purpose was to have a language that would enable a user to build instructional material in

statistics. However, since computer-based instruction in statistics requires so much flexibility and power, we discovered when we were through that we had more than we expected: We had a language designed to assist an author in building not only mathematical and related material, but such courses as history, spelling, psychology, etc. We have since added a few additional features, so that we now have a multipurpose language designed for CHI, of which lesson building is a subset.

PLANIT comprises not only a language but also a program, developed for time-shared use. The system operates in four modes: Lesson building, editing, execution, and calculation. The first two modes permit the teacher to construct and edit lesson frames in various formats and store them in designated sequences for later presentation to the student in the execution mode. The calculation mode is particularly oriented to mathematical subject matter and can be used as a calculation aid for the teacher (when building the lesson) or the student (when performing the lesson). While the student has access only to modes EX (execution) and CALC (calculations), the teacher or lesson designer (LD) may use all four modes.

- *Calculation Capability.* PLANIT has an on-line calculation capability that allows either the LD or the student to perform calculations involving trigonometric functions, elementary algebraic functions, and matrix declarations. In addition, the calculation functions of PLANIT can be tied in with the lesson: The LD can request the student to compute some data and can specify that the student's answer be compared with the results of evaluating a previously defined function. The LD can use the CALC mode to define the function when preparing the lesson; during lesson execution the CALC mode can be used again to "test" the student's answer.
- *Criterion Branching.* The PLANIT language allows the LD to specify conditions for branching based on the student's performance over any portion of the lesson.

  Conditions for branching may include:
  1. Response latency on any one answer or group of answers.
  2. Number of errors made on any group of questions.
  3. Help received from the CALC mode (functions used or not used).
  4. The actual path taken through the lesson up to this point.
  5. Any combination of the above four points.
- *Service Routines.* PLANIT also provides the following service functions for evaluating student answers that depart from the anticipated responses specified by the LD:
  1. PHONETIC Comparison. This routine gives the student credit for his answer even though it is spelled incorrectly as long as it is phonetically equivalent to one of the LD's answers. For example, PLANET and FONETIC are acceptable wrong spellings for PLANIT and PHONETIC.
  2. KEYWORD Match. This routine instructs PLANIT to search for a set of words in the student's response as the KEYWORDS of his answer. Furthermore, if the lesson designer wishes, he may specify that the answer will be evaluated as correct only if these keywords appear in a prescribed order.
  3. FORMULA Equivalent. This routine will allow the student credit for his answer as long as it is one of a subset of expressions algebraically equivalent to any one of the LD's answers. For example, if one of the LD's anticipated answers is 5/9* (F-32), then (5*F-160)/9 or 5*(F/9)-160/9 or (−32*(5)+5-*F)/9 would be equivalent and therefore acceptable. The LD can have any combination of these three routines turned off and on during lesson execution (even between actual occurrences of anticipated answers during student interaction with the lesson).

Illustrations

A lesson is composed of a set of frames; frames are composed of groups; groups are composed of lines of information such as textual material, questions, anticipated answers, actions, etc. There are five frame types: Problem, Question, Multiple Choice, Decision, and Copy. The Q (Question) frame will be illustrated below. The P, M, D, and C frames will be discussed briefly to point out their capabilities.* In this illustration (and in practically all PLANIT dialog) data entered by the user follow an asterik typed out by PLANIT. (In our examples, lines exceeding text width are shown indented.) All data the entered interactively via teletype.

```
(Q) THE QUESTION FRAME
FRAME 1.φφ LABEL=*HIST
2. SQ.
*?
2. SPECIFY QUESTION.
*WHO INVENTED THE ELECTRIC LIGHT?
*
3. SA.
*A+THOMAS EDISON
```

*See Reference 7 for a detailed discussion of all frames.

*B ALEXANDER BELL
*
4. SAT.
*A F:THATS VERY GOOD B:3
*B R:HE INVENTED THE TELEPHONE, TRY AGAIN . . .
*-R:
*-C:

Explanation of Frame 1

First line. All frames are, automatically, serially numbered by the program. Here the LD chooses to label the frame HIST. (If he chooses not to label the frame, he will merely strike the space bar and the carriage return and pass on to Group 2. Group 1 consists merely of the frame label.)

Group 2: SQ. The LD, not sure what SQ means, types in the question mark (?). PLANIT immediately repeats the group number, 2, and elaborates. The LD then types in his question WHO INVENTED THE ELECTRIC LIGHT? PLANIT returns with an asterisk, waiting for more lines of input. PLANIT has no way of knowing when the LD is through with the question group and so returns with an asterisk for each next line. The LD ends the group by striking the space bar once and then the carriage return key. There is no theoretical limit to the number of lines that can be entered in each group. We do, however, have a practical limit of 63 lines for the entire frame.

Group 3: SA. PLANIT is asking the user to SPECIFY ANSWER. The LD now enters all the anticipated answers, tagging the first one A and the second B, etc. The plus sign (+) next to the A indicates to PLANIT that this is the correct answer. The LD then indicates the end of the group by striking a space bar and carriage return.

Group 4: SAT. User is requested to SPECIFY ACTIONS to be TAKEN, depending upon which answer the student gives.

There are four types of commands in the action group:

F: What follows is the feedback message that is to be presented to the student. If no message follows F:, PLANIT will choose one randomly from its stored list of feedback messages, according to whether the student's answer was correct or incorrect. The LD can thus enter F: by itself, knowing that the student will not get a monotonous YES or NO when he enters an answer. (Responses are usually one-word messages such as FINE, YES, CORRECT, NO, WRONG.)

R: This command instructs PLANIT to wait for another answer without printing the question again. It can be entered along with an appropriate message such as in the above example. In this case the student receives the feedback: HE INVENTED THE TELEPHONE, TRY AGAIN; and PLANIT then waits for another answer. R: by itself instructs PLANIT to print out a fixed message (WRONG, TRY AGAIN) and wait.

C: This command used alone instructs PLANIT to print out the fixed message: THE CORRECT ANSWER IS, followed by the correct answer (indicated by the plus sign in Group 3). C: can be followed only by another command—F:, R:, B: or a CALC statement. For example, C: COUNT=COUNT*1 or C:X =FACT(3). This puts PLANIT in the CALC mode. COUNT is an item in CALC, as is X. Here COUNT is to be incremented by one and X is set equal to FACT (3), i.e., the factorial of 3.

B: This instructs PLANIT to branch (B:3 means BRANCH TO FRAME 3). All frames are numbered; they can also be labeled, and a branch can be made to any numbered or labeled frame. In addition, B:LSNAM (where LSNAM is the name of another lesson) means that the lesson LSNAM will now be brought into PLANIT and executed as a part of the current lesson; the student will never know the difference. Upon completion of the lesson LSNAM, PLANIT will continue with the next frame in the original lesson. B: by itself causes PLANIT to return to the calling lesson. For example, if during lesson AA the command B:BC is encountered (where BC is a lesson name), lesson BC will be called and run until the command B:, in lesson BC, automatically returns PLANIT to lesson AA. Similarly, B:PROGM means branch to the program whose name is PROGM. When PROGM relinquishes control, execution continues with the next frame in the calling lesson.

The first user input in Group 4 is read as follows: If the student gave answer A (THOMAS EDISON), print out the message: THATS VERY GOOD and branch to Frame 3.

The second input is interpreted as follows: If the student gave answer B, print out: HE INVENTED THE TELEPHONE, TRY AGAIN . . . and wait for another answer.

The third input, prefaced by a minus sign, indicates an action to be performed if the student's answer did not correspond with either A or B—namely, for any unanticipated response, wait for another answer. (In this case, since nothing appears after the R:, PLANIT then prints out the fixed message: WRONG, TRY AGAIN, and waits for another answer.)

The LD terminates the group and thereby the frame by striking the space bar and the carriage return key.

The space bar and CR alone on any line will terminate the group. The dollar sign ($) and CR alone on any line will terminate the frame. If, for example, the LD (in the middle of Group 3) decided to end the frame and proceed to the next frame, he would

only have to enter the $ and CR alone on a line and PLANIT would respond with:

```
P/Q/M/D/C.
```

User may then, to give another illustration, build a new lesson as follows:

```
*Q
FRAME 2.φφ LABEL=*MATH
2. SQ.
*LETS SEE WHAT YOU REMEMBER ABOUT
  TEMPERATURE. USING F FOR DEGREES
*FAHRENHEIT AND C FOR DEGREES CEN-
  TIGRADE, WRITE THE FORMULA FOR
*CONVERTING FROM DEGREES FAHREN-
  HEIT TO DEGREES CENTRIGRADE.\
*HINT:  F=9*C/5+32 CONVERTS FROM
  CENTIGRADE TO FAHRENHEIT.
*
3. SA.
*φ FORMULAS ON
*A+C=(5/9)*(F—32)
*B F=9*C/5+32
*C C=(5/9)*F—32
*
4. SAT.
*A F:  B:7
*B R:YOUR ANSWER IS THE SAME AS THE
  ONE I GAVE YOU, TRY AGAIN . . .
*A F: NOW YOU'VE GOT IT. B:15
*B R:YOU'RE STILL CONVERTING FROM
  CENTIGRADE TO FAHRENHEIT, TRY
  AGAIN . . .
*BC F:NOTE THE DIFFERENCE. C: B:OUT
*—R:
*—C:
*
```

Explanation of Frame 2

First line. The LD labels this frame MATH.

Group 2. SQ. The LD enters his question. Notice the back slash (\) after CENTIGRADE on the third line. This instructs PLANIT to skip a line after printing out CENTIGRADE—to set off the following HINT. For example:

```
LETS SEE WHAT YOU REMEMBER ABOUT
  TEMPERATURE. USING F FOR DEGREES
FAHRENHEIT AND C FOR DEGREES CENTI-
  GRADE, WRITE THE FORMULA FOR
CONVERTING FROM DEGREES FAHRENHEIT
  TO DEGREES CENTIGRADE.

HINT: F=9*C/5+32 CONVERTS FROM CEN-
  TIGRADE TO FAHRENHEIT.
```

The "\" can be used anywhere in Group 2.

Group 3. SA. This group illustrates the algebraic matching ability of PLANIT (activated by the expression: φ FORMULAS ON). The student can type in any equivalent algebraic form of the correct answer and get full credit for it; e.g., C=(F—32)*(5/9), C=5*(F—32)/9, C=(5*F—160)/9, etc., are all equivalent and therefore acceptable forms. If FORMULAS is not turned on (or is deactivated by the expression: φ FORMULAS OFF), then only the exact form as typed in by the LD would be looked for in the answer. This, of course, is not true symbol manipulation; we merely employ a technique which (in part) includes performing algebra on the student's answer. The matching technique is not restricted to correct answers alone but works for all anticipated answers in Group 3.

Group 4. SAT. This group illustrates repeated use of a frame. The *first* time through this frame, if the student's answer corresponds to:

A—he receives a (randomly selected) feedback message followed by a branch to Frame 7.

B—he receives the feedback message: YOUR ANSWER IS THE SAME AS THE HINT I GAVE YOU, TRY AGAIN. . .

C—he receives: NOTE THE DIFFERENCE. THE CORRECT ANSWER IS: C=(5/9)*(F—32) . . . followed by a branch to the frame whose label is OUT.

If he gives an unanticipated answer (neither A, B, nor C), he receives the message: WRONG, TRY AGAIN.

The second time through the frame, if the student's answer corresponds to:

A—he receives: NOW YOU'VE GOT IT. PLANIT then branches to Frame 15.

B—he receives: YOU'RE STILL CONVERTING FROM CENTIGRADE TO FAHRENHEIT.
TRY AGAIN. . .

C—he receives the same message as in C above.

If no match (second unanticipated answer), he receives: THE CORRECT ANSWER IS C=(5/9)*(F—32). PLANIT will then go on to the next frame in the sequence.

If the student goes through the frame a third time, or more, and gives an answer corresponding to:

A—he receives the same feedback as for A the second time through.

B or C—he receives the same feedback as for C above.

Unanticipated answer—he receives the same feedback as for unanticipated answers the second time through.

Note: Several commands (F: C: R: B:) occurring on one line are performed in the order of their appearance from left to right.

If there is no Group 3, then all commands in Group 4 will be executed, and there will be no pause for the student's answer.

Let's go on to another example frame illustrating the PHONETIC and KEYWORD routines.

```
P/Q/M/D/C.
*Q
FRAME 3.φφ LABEL=*PRES
2. SQ.
*WHO WAS THE FIRST PRESIDENT OF THE
  USA?
*
3. SA.
*φ PHONETIC ON
*φ KEYWORD ON
*A+GEORGE WASHINGTON
*B ABE LINCOLN
*C+G. WASHINGTON
*
4. SAT.
*A F: B:SOMEPLAC
*B R:HE WASN'T THE FIRST, TRY AGAIN...
  C: COUNT=COUNT+1
*C R:SPELL HIS FIRST NAME.
*
```

Explanation of Frame 3

First line. The LD labels it PRES.

Group 3. SA. Two different correct answers are designated above by the letters A and C. This is perfectly acceptable to PLANIT. However, when the command C: is used, the correct answer printed out will always be the last one with the plus sign. This group also illustrates the use of the phonetic and keyword matchers. The phonetic matcher encodes all answers into their phonetic equivalent. The keyword function looks for the answer or answers designated by the LD anywhere in the student's response. Both phonetic and keyword are turned on, as shown in Group 3. The zero in front of PHONETIC ON and KEYWORD ON tells PLANIT to perform this function before any answers are matched. Their combined use would cause PLANIT to accept the following answer as correct: I THINK IT WAS JEORGE WASHINGTUN. KEYWORD does not accept the answer in reverse order; i.e., WASHINGTON GEORGE would not be accepted. However, if FORMULAS was turned on too, then either order would be accepted.

(M) THE MULTIPLE CHOICE FRAME

The M frame is built exactly the same as the Q frame. The only difference is that during the execution of the lesson, the choice of answers (in Group 3) is printed out. The plus sign, if any, is omitted of course.

(P) THE PROBLEM FRAME

The use of this frame is rather specialized. Three kinds of information may be inserted here by the LD:

- Probability distribution parameters for generating data in the form of random samples, e.g., means, variances, and correlation coefficient for a bivariate Gaussian (normal) distribution. The random data would actually be generated for student use during the execution, for example, of a statistics lesson (the LD can also specify headings and format for the actual printing of the data).
- Steps to the solution of a problem in which the random data will be used (the student receives one step at a time in response to his request "STEPS").
- Controls over the mathematical functions which shall (or shall not) be made available to the student when he attempts to solve a particular problem.

In addition there is another less specialized use for this frame. The LD may insert here the names of files (data bases) that can be used by PLANIT to search for answers to questions posed by students.

(C) THE COPY FRAME

The Copy frame is more of a building aid than a frame in its own right. It allows one to copy and modify any frame previously built in the same lesson and to include it in the frame presently being built.

(D) THE DECISION FRAME

As previously illustrated, all branching decisions are made as a function of what actions have taken place during execution of the frame. The Decision frame affords the LD the opportunity to consider branching decisions (and other forms of program behavior) as a function of the student's past performance, that is, as a function of results that have taken place during execution of a set of frames. Since our goal is to provide the user with a language for handling— quickly, naturally, and easily—the kinds of problems that one encounters in CHI, we devised a "language" for describing patterns of past performance. This language takes two basic forms.

The first form, called the pattern form, allows one to inquire if the student took a particular path through the material. For example, let us imagine the student went through Frame 1, answered A or C, and followed it by Frame 5 (in which he responded incorrectly), and then followed that by Frames 10 through 15, where the student was correct. An inquiry concerning whether or not the student went through that pattern exactly with no deviations can actually be written in the language pretty much as it has been stated. For example:

```
IF 1,AC 5,— 10-15,+
```

This, then, is the form of the pattern question. If the query is answered affirmatively, one can then use any combination of the three action commands:

F:, C:, and B:.

The second form permits queries about summarized student performance over a set of frames. For example,

one may want to know if the student got less than or equal to five right out of Frames 10 through 22 and Frames 30 and 33. This can also be written, almost as stated, in the following manner: IF LQ 5 RIGHT 10-22,30,33. Similarly, if these conditions are satisfied, any of the three commands can then be executed. In place of RIGHT, one can substitute WRONG, SEEN, MINUTES, USED. With USED, one can have any function like FACT (factorial), SIN, COS, etc. For example:

IF GQ 5 MINUTES 10-15 B:10
IF USED SIN 6 B:7

means: If the student spent at least five minutes on Frames 10 through 15, then branch to Frame 10. The second IF statement reads: If he used the function SIN in Frame 6, then branch to Frame 7. Also, in place of GQ one can substitute LS for less than, GR for greater than or other relationals such as LQ, NQ, EQ. Finally one can use IF statements to query the contents of items set in the calculation mode. IF IQ LS 50 B:WORK means: If the item whose name is "IQ" contains a value less than 50, branch to the frame whose label is "WORK."

All these forms can be connected as one large statement via the connectives

AND and OR, e.g.:

IF 1,AC 5,— 10-15,+
OR GQ 5 MINUTES 10-15 AND USED SIN 6
AND IQ LS 50 B:WORK

The above examples illustrate only a few uses of PLANIT. In the calculation (CALC) mode, these same capabilities can be turned to mathematical subject matter. CALC provides a powerful computing capability; arithmetic expressions can be instantly evaluated, mathematical functions can be defined, and a number of stored functions (such as the generation of pseudorandom numbers) and primitives (e.g., FACT(N) = N!) are available to teacher and student. If, for example, the student is operating in the execution (EX) mode and working on a lesson frame that requires him to perform computation, he may enter the CALC mode by typing a left arrow, instruct the machine to perform the desired operation, and receive an immediate answer, as illustrated:

	Dialogue	*Explanation*
User:	*FUNCTION A(X,Y) = X*Y	Defining of function A(X,Y) to be equal to the product of X and Y.
System:	IN	
User:	*A(5,4)	Using the function with arguments 5 and 4.
System:	20.0	
User:	*A(7,FACT(3))	Using the function with arguments 7 and the factorial of 3.
System:	42	

Current and future uses

Materials produced via PLANIT are: Instructional sequences in statistics (aimed at students of the social sciences enrolled in a first course in statistics); spelling and vocabulary (for children three to eight years); economics (for undergraduates); introduction to computer programming (for persons having some knowledge of algebra). Agencies who have used or are using PLANIT are: System Development Corporation, Southwest Regional Laboratory, University of California at Los Angeles, University of Southern California, University of California at Irvine, United States Naval Personnel Research Activity (San Diego, California), New England Educational Data System, and Lackland Air Force Base.

In view of PLANIT's interactive and evaluative capabilities, one can readily employ the system for other applications. For example, it is being currently used in the development of a computer-based vocational counseling program, and is being adapted for use in administrative planning and management. Its usefulness in this area will be greatly enriched by the CALC capability.

The PLANIT system, now operating on SDC's Q-32 time-sharing system, will soon be available for use on the IBM 360/65. In the near future we hope to make conversion to other smaller computers such as the IBM 360/30.

ACKNOWLEDGMENT

The author wishes to acknowledge the work of Dr. Charles Frye of System Development Corporation who was co-developer of the system.

REFERENCES

1 M H PERSTEIN
Grammar and lexicon for basic JOVIAL
System Development Corporation Santa Monica California Technical Memorandum TM-555/005/00 10 May 1966

2 D L BITZER J A EASLEY
PLATO: A computer-controlled teaching system
In M A Sass and W D Wilkinson Eds Symposium on computer augmentation of human reasoning Spartan Books Washington D C pp89-103 1965

3 IBM 1401 1440 or 1460 operating system computer assisted instruction
IBM Reference Publication C24-3253-1 Rev International Business Machines Corporation 1964

4 L M STOLUROW
Systems approach to instruction
University of Illinois Training Research Laboratory Technical Report no 7 July 1965

5 *A manual for BASIC, the elementary algebraic language designed for use with the Dartmouth Time Sharing System* Dartmouth College Hanover New Hampshire 1 January 1965

6 J M NEWTON
ONR conference on CAI languages
Report of material presented at conference March 2-3 1966 Cambridge Massachusetts ENTELEK Incorporated Newburyport Massachusetts unnumbered technical report

7 S L FEINGOLD C H FRYE
User's guide to PLANIT
System Development Corporation Santa Monica California Tech Memo TM-3055/000/01 17 October 1966

A formal system for the specification of the syntax and translation of computer languages*

by JOHN J. DONOVAN and HENRY F. LEDGARD
Massachusetts Institute of Technology
Cambridge, Massachusetts

INTRODUCTION

This paper presents two basic results: the use of established methods of recursive definition to present a single method to

1. specify the syntax of computer languages (including contextsensitive requirements, such as the restrictions implied by declaration statements),
2. specify the translation of programs in one computer language into programs in another language.

The method can be used to write one specification for both the syntax of a language (e.g. a source language) and its translation into a target language (e.g. an assembler language). A syntactically legal program and its translation into the target language can then be generated using the specification. If the target language is understood, the semantics of the first language is specified.

The paper develops the method of recursive definition in conjunction with an example specifying the syntax of a limited subset of PL/I (including declaration, arithmetic, and conditional statements) and its translation into IBM System/260 assembler language.

Background

Our objective is to present a single method for expressing the syntax and translation of computer languages. The method was presented and first applied to specify only the syntax of computer languages in an earlier work by Donovan.[4]

The objective to develop methods for specifying either the syntax or the translation of computer languages is not new. In response to the demand for numerous problem-oriented computer languages to meet the needs of diverse fields, there has been considerable activity to ease the effort required to define and implement a language.

Much of this activity has led to the development of methods for specifying, at least in part, the syntax of computer languages.[9-14] The methods for specifying syntax have facilitated the description of computer languages to members of the computing field and have led to the development of syntax-directed translators.[25-28] However, most of the methods for specifying syntax have been shown to be equivalent to context-free phrase structure grammars and hence inadequate[13,23,24] for completely characterizing the syntax of computer languages. For example, some programming languages require that all statement labels in a program be different, that reference labels refer to existing statement labels, that the arithmetic type of a variable be declared, or that the dimensions of an array be declared before referring to an element within the array. In a Fortran program, for instance, the statement "GO TO 20" is not legal unless "20" occurs as a statement label in the same program. Restrictions like these cannot be specified by a context-free grammar. We consider these restrictions to be syntactic in that programs violating these restrictions are never translated, but are rejected solely on their form. Debate as to whether these restrictions are syntactic or semantic is immaterial when we wish to specify both the syntax and translation of a language, because then all these restrictions must be satisfied.

Other activity has been directed to developing table-driven compilers[27,28] and programming languages for expressing string transformations.[20,21] The table-driven compilers have generally been limited to a particular type of target language and have required excessive detail in writing the specifications to fill

*Work reported herein was supported (in part) by Project MAC, an MIT research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01).