TR 270

AN INSTRUCTIONAL INTERPRETER FOR BASIC

Avron Barr and Marian Beard

Institute for Mathematical Studies in the Social Sciences
Stanford University
Stanford, California

The BASIC Instructional Program (BIP) was developed to investigate tutorial modes of interaction in computer-assisted instruction (CAI). BIP is a problem-solving laboratory that helps students while they are solving introductory programming problems in the BASIC language. The problems are presented in an individualized sequence based on a representation of the structure of the curriculum and a model of the student's state of knowledge. This paper describes the BIP system, with emphasis on recently developed features. The goal of the tutorial laboratory is informative interaction with the student, which is provided by an instructional BASIC interpreter, information on BASIC syntax cross-referenced with the BIP student manual, and debugging aids. The system also has access through the curriculum representation to features that the student may use to help her solve her current problem. These features include hints, easier "subtasks," a stored solution that can itself be executed, and an interactive flow chart representation of the solution. The nature of the student-BIP interaction is captured in an annotated student dialogue of a typical session.

I. Background

Computers are now used in a wide variety of applications in education and training, including curriculum presentation and drill, information retrieval, and simulation of complex systems. The research reported here deals with an additional application: the use of the computer as a problem solving laboratory. In the computer-based laboratory environment, the student attempts to solve problems on-line with the guidance of the instructional system. The system plays the role of interactive tutor, giving hints, correcting errors, and evaluating progress. The full power of the computer as calculator and simulator is available to the student, providing the motivational effects of learning by working on real problems. The main focus of our work in the Complex Instructional Strategies research group at the Institute for Mathematical Studies in the Social Sciences at Stanford University is the individualization of the sequence of instruction presented in computer-assisted instruction (CAI). This paper describes a computer-based "programming laboratory" which we have used in our research and which we believe is an excellent example of an effective CAI program, a learning environment adaptively suited to each student at her own level of development.

In 1970 the Institute's Complex Instructional Strategies group developed a large CAI curriculum for a course to teach the AID programming language at the 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). However, it is a linear, "frame-oriented" CAI program and cannot provide individualized instruction during the problem-solving activity itself. After working through lesson segments on such topics as variables, output, and expressions, the student is assigned a problem to solve in AID. She must then call up a separate AID interpreter, perform the required programming task, and return to the instructional program with an answer. As she develops her program directly with AID, her only source of assistance is the minimally informative one-line error messages provided by the interpreter. Because of the limitations of the AID course, we undertook development of a new course with a completely different instructional design, based in part on earlier work by Paul Lorton (Lorton & Slimick, 1969).

The BASIC Instructional Program (BIP) is a stand-alone, fully self-contained course in BASIC programming at the high school/college level developed over the past two years with the assistance of over 300 undergraduates who have taken the course at DeAnza College, the University of San Francisco, and Stanford. BIP's major features are:

1. A monitored BASIC interpreter, written in SAIL (Van Lehn, 1973) which allows the instructional system maximal knowledge about student errors.

2. A curriculum consisting of approximately 100 programming problems at widely varying levels of difficulty.

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

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

5. A complete student manual that includes a general introduction to programming and detailed reference information on BASIC statements and structures.

Figure 1 is a schematic representation of the tutorial programming laboratory environment supported by BIP, described fully in Barr, Beard, and Atkinson (1975b).

Computer programming, like many other procedural subjects, is better learned through experience than through direct instruction, especially if that experience can be paced at a speed suited to the individual student. Throughout the BIP course, the primary emphasis is placed on the solution of programming problems, called "tasks." BIP does not present a sequence of instructional statements followed by questions. Instead, a problem is described and the student is expected to write her own BASIC problem to solve it. As she develops her BASIC program for each task, the student is directed to appropriate sections of the student manual for full explanations of BASIC statements, programming structures, etc. She is also encouraged to use the numerous student-oriented features, such as an interactive debugging facility and various "help" options described below.

Figure 2 shows all the curriculum elements that describe each task. The text states the requirements of the task to the student, and suggests any prerequisite reading in the BIP student manual. The hints present additional information at the student's request, and subtasks isolate a part of the "main" problem as a smaller programming problem which she is to solve, helping her reduce the main task to separately soluble parts. The skills are the specific programming elements required in the solution. The model solution is a BASIC program that solves the problem presented in the task, and is accessible to the student if she cannot reach her own solution. The model also contains coded test input data that is used to compare the results produced by the student's program against those of the model. The "must follow" tasks (if any) will follow the main task automatically, and require extensions of the student's original solution. The "required operators" are BASIC statements that must be included in the student's program before she is allowed to pass out of the current task; the "disabled operators" are BASIC statements that, for pedagogical reasons, are not to be used in her solution program.

The sequence of events that occur as the student works on a task is shown in Figure 3. When she has finished the task by successfully running her program, the student proceeds by requesting "MORE". Her progress is evaluated after each task. In the "Post Task Interview" she is asked to indicate whether or not she needs more work on the skills required by the task, which are listed separately for her. This information is used to update BIP's model of the student's knowledge, which is then used in the selection of an "optimal" next task. The process of individualized task selection is fully described in Barr, Beard, and Atkinson (1975a).

BIP has undergone several major revisions; a number of significant improvements have been made to existing features, and a major graphic assistance routine has been added. Section II describes these modifications briefly, and Section III presents an annotated dialogue of a typical session, illustrating the system in action. Table 1 lists the BIP commands available to the student grouped by their functions, as an overview of the system.

II. BIP's Interpreter: An Instructional Laboratory

Because the BIP course is aimed at students with no previous programming experience, the error messages are designed to contain more information than those supplied by "standard" BASIC systems, and they are carefully worded in non-computer-oriented terms to minimize confusion. Since BIP runs in an interactive environment, the student receives immediate feedback about her syntax errors, and information about other errors as soon as they are detected. These features keep the student from going too far down the wrong track without some warning. BIP's interpreter is built right into the instructional program so that the student can request additional explanation if she is still confused; that is, the instructional system knows something about the programming errors.

BIP's error detection capabilities cover four different areas: syntax errors, execution time errors, program structure errors detectable before execution but involving more than the syntax of one line, and errors related to the current task (making an otherwise correct program an unacceptable solution to the problem). Typical student errors and BIP's responses to them are illustrated in Section III.

Debugging. A common difficulty among beginning students is an inadequate or inaccurate understanding of the process of program execution. In most systems, variable assignments, logical decisions, and program branching are all invisible to the user, and there is no way for the student to see the flow of execution of her program. Interactive graphic debugging systems are important instructional tools that can greatly assist the student's conceptualization of program execution, as well as teach her useful debugging techniques. BIP makes available two such facilities, which have proved to be valuable both to the students and to the system's designers.

BIP's TRACE option allows the student to see exactly how execution of her program proceeds, and to identify the point at which it begins to stray from what she intended. As each line of her program is executed, the line number is displayed on her teletype or display terminal. Any variable assignments performed in that line are also indicated, as well as any input or output.

FLOW is a more sophisticated program tracing aid designed for CRT displays. The main program

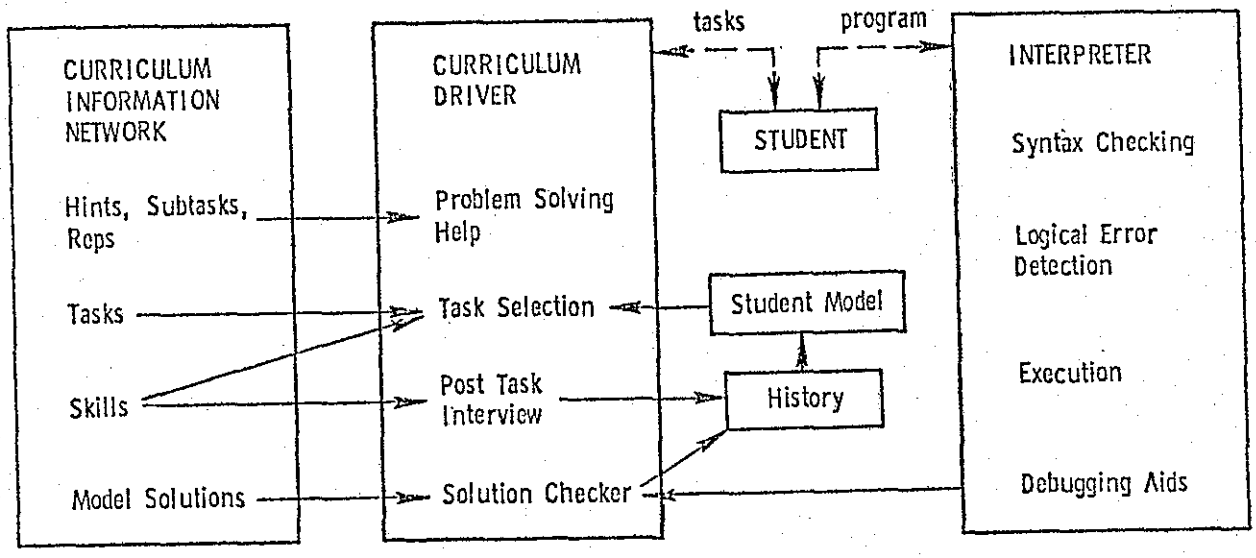

Figure 1. A schematic representation of the tutorial programming laboratory.

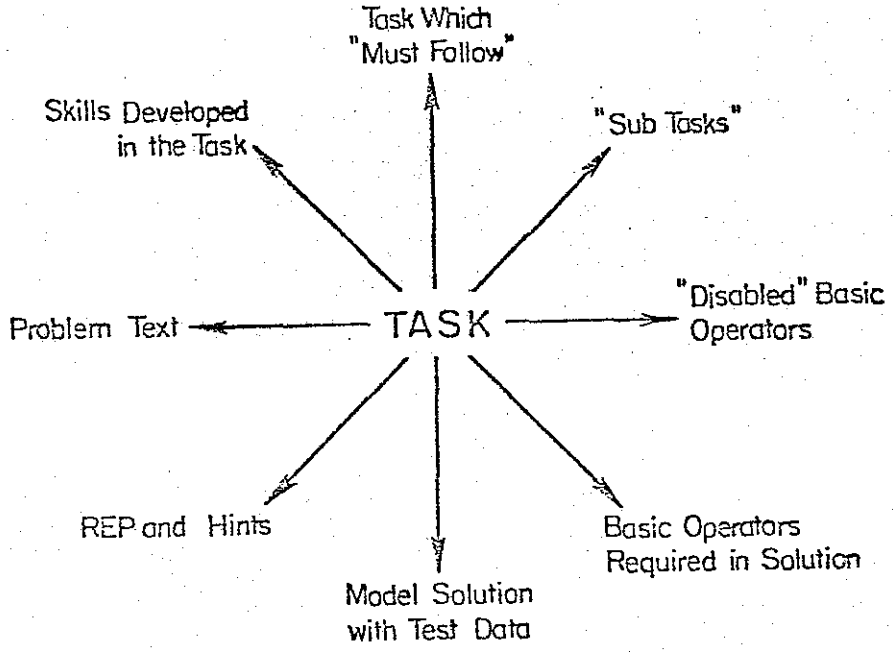

Figure 2. Elements that describe a task.

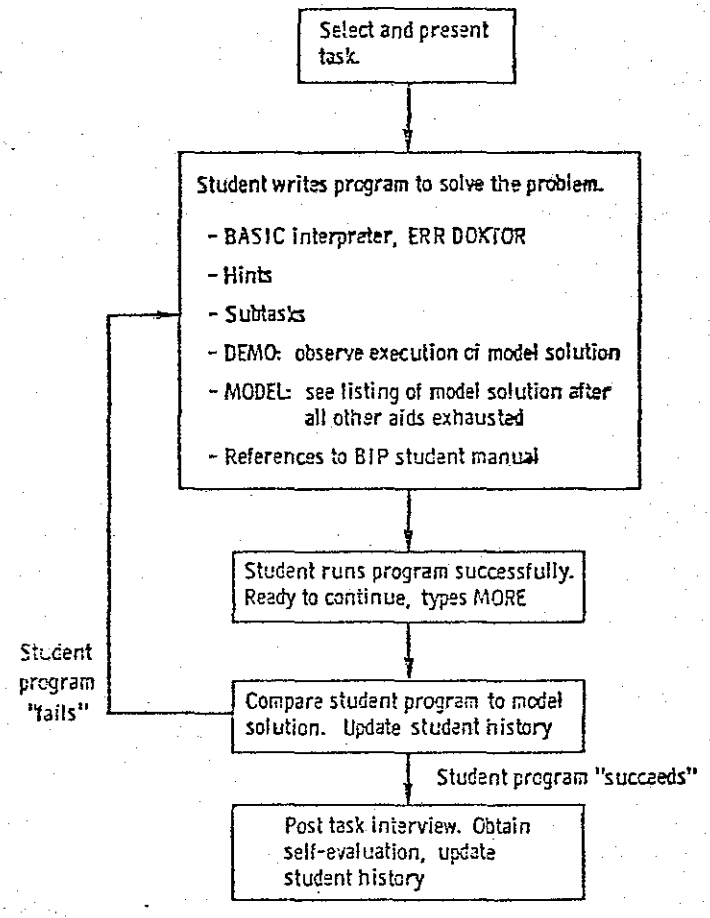

Figure 3. Working through a task.

Table 1

BIP's Student Commands

INFORMATION	INSTRUCTION
Who (is at this terminal)	Task
What (task am I doing)	More
When (is it now)	Reset (exit all tasks)
Gripe (to Stanford)	Enough (exit current task)
Calculator	
Hardcopy	
PROBLEM SOLVING AIDS	**DEBUGGING AIDS**
Rep	Flow
Hint	Trace
Subtask	
Demo	
Model	
INTERPRETER COMMANDS	**FILE SYSTEM**
Run	Files (to see directory)
Scratch	Save
Sequence (renumber lines)	Get
List	Merge
Edit (a line)	Kill

is displayed on the terminal, with the text of all subroutines removed. Each time the student presses the CR key, one line of her program is executed, and its line number blinks on the screen display. When an IF or GOTO statement is executed, an arrow is drawn on the screen to indicate the transfer of control. The student may also request that up to six variables be traced. The current values of all traced variables are shown at the top of the screen. If an array is traced, the value of the most recently assigned array element is shown. (See Barr et al. (1975a) for an illustration of FLOW.)

Problem solving aids and solution checking. Another graphic feature is the REP system, which is part of the problem solving assistance supplied by the instructional laboratory. (Other problem solving aids are described in Barr, Beard, & Atkinson, 1975b.) The student can request a flow chart-like representation of the solution of each programming problem and probe the flow chart to expand parts in detail.

In using REP the student is allowed to probe the representation in both breadth and depth and in any sequence. If she probes in the breadth dimension she may first look at control structure information and find that the program requires a loop. Next she may look at INPUT/OUTPUT or other key information. Thus, once she has established that the program requires a loop, more information might be requested on control structure until, finally, she is shown the actual BASIC code. An illustration of REP appears in Barr et al. (1975a).

Since the BIP course runs without human graders, a simple "solution checker" is built in to determine whether the student's program is a solution to the problem she was to solve. The checker works by running the model solution and comparing its output to the output of the student's program. While this approach does not catch all possible bugs in a student's program, it is the method used by most human instructors in grading programming assignments; it is quick and sufficient.

The solution checker executes the stored model solution invisibly, saving its output, and then executes the student's program in the same way. Each time the student's program produces output, it is compared to the list stored from the execution of the model, and any matching element in that list is flagged. If any unflagged output remains when execution is completed, the student is told that her program "doesn't seem to solve the problem." If all outputs of the model have been matched, she is told that her program "looks ok." If the student's program is found to be inadequate, she is advised to run the DEMO, which executes the model solution as an illustration. For interactive programs, the test values on which the student's program failed are suggested as appropriate input to the DEMO so that she can see how the model solution handles those values. Frequently, the student's program will fail to deal with the test values, and the failure is evident from the correct example provided by the DEMO. In these cases, the solution checker gives instructive help in just those areas that cannot be identified by the other error detection facilities.

Though simple and obviously limited in some ways, BIP's solution checker is an effective tool, not only in acting as a grader that governs progress through the curriculum, but also as an additional source of instruction. Particularly in the more demanding interactive tasks, the test values used by the checker and suggested to the student add to the beneficial learning effects of hands-on experience and manipulation of her own BASIC programs.

III. An Annotated Dialogue

The task shown here is one of the more difficult tasks in the curriculum. As in most tasks, there is no explicit instruction in the syntax of BASIC. The student is expected to refer to the manual and to request assistance as she requires it.

The student/BIP interaction is shown on the left, with all student input preceded by an asterisk. Our explanatory annotation is indented to the right.

```
*TASK

 BEGIN TASK: XMAS
 ON THE FIRST DAY OF CHRISTMAS, SOMEONE'S TRUE LOVE
 SENT HIM/HER A PARTRIDGE IN A PEAR TREE (ONE GIFT
 ON THE FIRST DAY). ON THE SECOND DAY, THE TRUE
 LOVE SENT TWO TURTLE DOVES IN ADDITION TO ANOTHER
 PARTRIDGE (THREE GIFTS ON THE SECOND DAY). THIS
 CONTINUED THROUGH THE 12TH DAY, WHEN THE TRUE LOVE
 SENT 12 LORDS, 11 LADIES, 10 DRUMMERS, . . . ALL
 THE WAY TO YET ANOTHER PARTRIDGE.

 WRITE A PROGRAM THAT COMPUTES AND PRINTS THE
 NUMBER OF GIFTS SENT ON THAT TWELFTH DAY. (THIS
 IS NOT THE SAME AS THE TOTAL NUMBER OF GIFTS SENT
 THROUGHOUT ALL 12 DAYS -- JUST THE NUMBER SENT ON
 THAT SINGLE 12TH DAY.)
```

This is a fairly difficult problem, and several problem solving aids are available. In the previous section we discussed the REP feature which helps students break down problems with flow charts. (Its graphic nature cannot be demonstrated here.)

Additional text material is also available, either as a clarification of the task itself or, as in this case, to help the student construct her own algorithm.

```
*HINT

 HINT #1
 THIS PROGRAM REQUIRES A LOOP. EACH EXECUTION OF
 THE LOOP INVOLVES ACCUMULATING THE VALUE OF THE
 INDEX INTO A TOTAL.

*HINT

 HINT #2
 FINDING A TOTAL OR SUM ALMOST ALWAYS MEANS TWO
 THINGS:
 1. SETTING A VARIABLE EQUAL TO ZERO OUTSIDE A
    LOOP.
```

2. ACCUMULATING VALUES INTO THAT VARIABLE WITHIN THE LOOP. IN WORDS, TOTAL EQUALS OLD TOTAL PLUS ANOTHER VALUE.

In some cases, the task involves a number of complications, some of which can be isolated and presented as smaller "subtasks," as shown here. This subtask directs the student's attention to the problem of writing a simple loop using a FOR. .NEXT statement. The student always has the option of completing the subtask, or of returning to the "main" task as soon as she has gained the information she thinks she needs to solve the original problem.

```
*SUB

 BEGIN TASK: S:FORLOOP
 WRITE A PROGRAM WITH A LITTLE -FOR . . NEXT- LOOP.

 THE WORK OF THE LOOP IS JUST TO PRINT THE VALUE OF
 THE LOOP'S INDEX (THE VARIABLE THAT KEEPS TRACK OF
 THE NUMBER OF TIMES THE LOOP HAS BEEN EXECUTED).

 WHEN YOU RUN THE PROGRAM, IT SHOULD LOOK LIKE IT
 IS COUNTING FROM 1 TO 10.
```

The subtask is treated as a regular task by BIP, and can have hints and subtasks of its own. Here the student makes several attempts at a solution, illustrating the error detection facilities.

First, although the problem specifically required a FOR. .NEXT loop, the student will try to construct a "hand-made" loop of her own.

```
*10 LET I = 1

*20 PRINT "THE INDEX IS; I
           ↑
 SYNTAX ERROR: UNMATCHED QUOTE MARKS -- FOUND NEAR
 '"THE INDEX IS'
 LINE NOT ACCEPTED   (TYPE ? FOR HELP)
```

Here the student has made a common error by omitting the closing quote marks in her PRINT statement. BIP is able to indicate the point at which it detected the error, and to give further information specifically related to this particular incorrect BASIC statement:

```
*?
 '"THE INDEX IS' HAS AN ODD NUMBER OF QUOTE MARKS.
 REMEMBER THAT ALL STRINGS MUST HAVE A QUOTE AT THE
 BEGINNING AND END.

 TYPE ? FOR MORE HELP

*?REF
 II.7, II.8
```

BIP can always present an appropriate manual reference for all errors.

This student gives up the fancy PRINT statement and includes a simpler but adequate one. Unfortunately, she is soon to find that her loop construction is unacceptable for this task:

```
*20 PRINT I

*30 IF I = 9 THEN 99
 FOR THE PURPOSES OF THIS TASK, DON'T USE THE 'IF'
 STATEMENT.
```

Her original plan for a loop including a conditional branch will have to be abandoned. She first deletes the lines she has entered so far, then continues, undaunted, to try to solve the problem without a loop.

```
*SCR

*10 PRINT 1

*20 PRINT 2

*30 PRINT 3

        . . .
```

After entering all of the PRINT statements, she lists her program, and then runs it.

```
*LIST

 10 PRINT 1
 20 PRINT 2
 30 PRINT 3
 40 PRINT 4
 50 PRINT 5
 60 PRINT 6
 70 PRINT 7
 80 PRINT 8
 90 PRINT 9
 99 END

*RUN

 EXECUTION OF YOUR PROGRAM

 1
 2
 3
 4
 5
 6
 7
 8
 9

 EXECUTION COMPLETED AT LINE 99
```

The program runs without execution errors. When the student considers her program to be an adequate solution to the problem posed in the task, she requests BIP's evaluation of it by typing MORE, indicating that she wants to proceed to another task. Here, the student seems to be satisfied, and decides to continue.

```
*MORE

 WAIT.  SOMETHING IS MISSING.  FOR THIS TASK, YOUR
 PROGRAM SHOULD ALSO INCLUDE THE FOLLOWING BASIC
 STATEMENT(S):

          FOR

 ADD TO YOUR PROGRAM AND -RUN- IT AGAIN BEFORE YOU
 TYPE 'MORE'
```

BIP has recognized that the essential FOR. . NEXT loop is missing from the student's program, and refuses to evaluate it further. Finally she sees the light, deleting her last effort and trying again, this time with a FOR. .NEXT loop.

```
*SCR

*10 FOR I = 1 TO 9

*20 PRINT I

*99 END
```

This three-line program is syntactically incorrect, as there is an open loop. The ERR DOKTOR routine checks the structure of complete programs before they are run. Otherwise, detection of such errors would be delayed until they appear later as less informative execution-time errors.

```
*RUN

 ERR DOKTOR DIAGNOSIS: 'FOR' STATEMENT WITHOUT
 MATCHING 'NEXT'

 LOOK AT LINE NUMBER: 10
 TYPE ? FOR HELP
```

As always, more help is available after the original error message.

```
*?
 YOU ARE MISSING THE NEXT STATEMENT CORRESPONDING
 TO THE -FOR- ON THIS LINE

 TYPE ? FOR MORE HELP
```

After listing her program, the student sees and corrects the loop error, and the program executes successfully.

```
*LIST

 10 FOR I = 1 TO 9
 20 PRINT I
 99 END

*30 NEXT I

*RUN

 EXECUTION OF YOUR PROGRAM

 1
 2
 3
 4
 5
 6
 7
 8
 9

 EXECUTION COMPLETED AT LINE 99
```

Again, the student is ready for MORE. BIP evaluates her solution by invisibly executing both the model solution and her program on the same "test" data, then comparing their output.

```
*MORE

 EXECUTION OF THE MODEL

 EXECUTION OF YOUR PROGRAM

 YOUR PROGRAM DOESN'T SEEM TO SOLVE THE PROBLEM.

 IF YOU ARE COMPLETELY CONFUSED OR LOST, YOU CAN
 LEAVE THIS TASK RIGHT NOW.

 DO YOU WANT TO STAY IN THE TASK?  TYPE  Y  OR  N
                                                *Y

 O.K.  TRY AGAIN.  YOU MIGHT LOOK AT A -DEMO- OF
 THE MODEL SOLUTION AND COMPARE ITS OUTPUT TO THE
 OUTPUT OF YOUR PROGRAM.
```

In this case, the model produced output different from that of the student's program, so she is told that her program doesn't seem correct. She elects to remain in the task to try again. The DEMO feature allows the student to execute the stored model solution to see the kind of interaction required. In this task, the DEMO serves to remind the student that her program was supposed to count to 10, not 9.

```
*DEMO

 EXECUTION OF THE MODEL

 THE INDEX IS 1
 THE INDEX IS 2
 THE INDEX IS 3
 THE INDEX IS 4
 THE INDEX IS 5
 THE INDEX IS 6
 THE INDEX IS 7
 THE INDEX IS 8
 THE INDEX IS 9
 THE INDEX IS 10
 WE ARE NOW OUT OF THE LOOP.

 EXECUTION COMPLETED AT LINE 99
 DEMO ENDED
```

And she corrects the first line. BIP reminds the student that the line has been changed.

```
*10 FOR I = 1 TO 10
 WAS: 10 FOR I = 1 TO 9

*RUN
 EXECUTION OF YOUR PROGRAM

 1
 2
 3
 4
 5
 6
 7
 8
 9
 10

 EXECUTION COMPLETED AT LINE 99
```

Again, satisfied, the student asks for MORE.

```
*MORE
 EXECUTION OF THE MODEL

 EXECUTION OF YOUR PROGRAM

 LOOKS OK
```

Following the successful comparison, BIP presents the "post task interview," in which the student's own evaluation of her understanding and skills is acquired and stored.

First the model solution is displayed, and the student is asked whether or not she understands how it works.

```
POST TASK INTERVIEW

HERE IS A TYPICAL SOLUTION TO THIS TASK:
        10 FOR I = 1 TO 10
        20 PRINT "THE INDEX IS"; I
        30 NEXT I
        40 PRINT "WE ARE NOW OUT OF THE LOOP."
        99 END

DO YOU UNDERSTAND THE SOLUTION?  (Y/N)        *Y
```

Then, for each of the specific skills required in the construction of the program for this task, the student is asked to indicate whether or not she feels confident in her use of that skill. These questions are a major source of information for BIP's task selection process, by which the next task will be chosen at a level appropriate for this student. Those skills to which the student responds "Y" will not be sought in her next task, since she feels that she has had enough work on them. Those to which she responds "N", on the other hand, will be looked for explicitly. Unless she has exhausted a portion of the curriculum, some of those "N" skills will definitely be required in her next task, providing her with the opportunity to use those skills again in a new context.

```
THINK ABOUT THE SKILLS USED IN THIS TASK.  FOR EACH
SKILL,
        TYPE  Y  IF YOU HAVE HAD ENOUGH WORK WITH
          THAT SKILL.
        TYPE  N  IF YOU THINK YOU NEED MORE WORK
          ON IT.

FOR . . NEXT LOOPS WITH LITERAL AS FINAL VALUE OF
  INDEX * Y
MULTIPLE PRINT [STRING LITERAL, NUMERIC VARIABLE]
  * N
```

Since she did not use the "multiple print" statement shown in line 20 of the model, our student indicates that that skill would be appropriate in her next problem.

BIP informs her that she has returned to the larger task at hand, and allows her to have its text re-displayed.

```
RETURNING FROM A SUBTASK.

YOU ARE IN TASK XMAS.

DO YOU WANT THE TEXT PRINTED OUT?  TYPE Y OR N.
      *N

*
```

Since it has access to the curriculum representation, the interpreter knows something about the semantics of the programs the student is to write. Thus, without attempting formal program analysis or verification, BIP is able to offer meaningful tutorial assistance that goes far beyond the error detection supplied by most students' first programming environment.

BIBLIOGRAPHY

Barr, A., Beard, M., & Atkinson, R. C. The computer as a tutorial laboratory: The Stanford BIP project (Tech. Rep. 260). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975. (a)

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)

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.

Brooks, R. A model of cognitive behavior in writing code for computer programs. Fourth International Joint Conference on Artificial Intelligence, 1975, 878-884.

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.

Danielson, R., & Nievergelt, J. An automatic tutor for introductory programming students. SIGSCE Bulletin, 7, 1975.

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.

Goldberg, A. Design of a computer-tutor for elementary mathematical logic. Paper presented at the IFIP Congress, Stockholm, August 1974.

Hart, R. O., & Koffman, E. B. A student oriented natural language environment for learning LISP. Fourth International Conference on Artificial Intelligence, 1975, 391-396.

Laubsch, J. H. Some thoughts about representing knowledge in instructional systems. Fourth International Conference on Artificial Intelligence, 1975, 122-125.

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.

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

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

Suppes, P., Smith, R., & Beard, M. University-level computer-assisted instruction at Stanford: 1975 (Tech. Rep. 265). Stanford, Calif.: Institute for Mathematical Studies in the Social Sciences, Stanford University, 1975.

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

ACKNOWLEDGMENT

This research was sponsored by Personnel Training and Research Programs, Office of Naval Research. Continuing work is sponsored by the Office of Naval Research and the Navy Personnel Research and Development Center. The authors thank Oliver Buckley, Richard Kahler, Jay Lindsay, and William Swartout for their contributions to BIP.