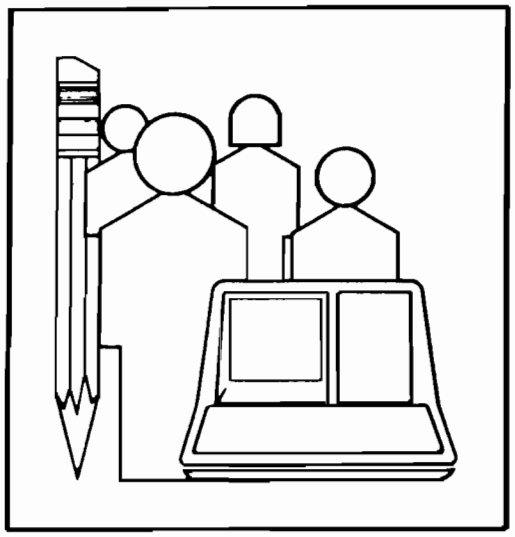

TECHNICAL REPORT

THE DESIGN OF A RULE-BASED CAI TUTORIAL

Technical Report No. 14

May 1983

DIGITAL EQUIPMENT CORPORATION
EDUCATIONAL SERVICES
12 Crosby Drive
Bedford, Massachusetts 01730

digital

THE DESIGN OF A RULE-BASED CAI TUTORIAL

Jesse M. Heines, Ed.D.
Tim O'Shea, Ph.D.

ABSTRACT

Rule-based systems are a development associated with recent research in artificial intelligence (AI). These systems express their decision-making criteria as sets of production rules, which are declarative statements relating various system states to program actions. For computer-assisted instruction (CAI) programs, system states are defined in terms of a task analysis and student model, and actions take the form of the different teaching operations that the program can perform. These components are related by a set of means-ends guidance rules that determine what the program will do next for any given state.

This paper presents the design of a CAI course employing a rule-based tutorial strategy. The course has not undergone the test of implementation; the paper presents a conceptual design rather than a programming blueprint. The subject of the course is ReGIS, the Remote Graphics Instruction Set on Digital Equipment Corporation GIGI and VT125 terminals. The paper describes the course components and their interrelationships, and discusses how program control might be expressed in the form of production rules.

Jesse M. Heines is with Digital Equipment Corporation in Bedford, Massachussetts. Tim O'Shea is with The Open University in Milton Keynes, England.

THE STRUCTURE OF A RULE-BASED COURSE

O'Shea (1979) has argued that one of the most important aspects of any CAI program is its response-sensitivity: to assert that one teaching program is more response-sensitive than another teaching program is to claim that in some sense it is more adaptive to the individual learning needs of the students being taught than the other program.

O'Shea's work achieved response-sensitivity by adopting Hartley's (1973) framework for adaptive teaching programs. This framework consists of:

- a "vocabulary" of teaching operations,
- a representation of the task,
- a model of the student, and
- a set of means-ends guidance rules.

The teaching operations are the different instructional activities that the CAI program can present. In the course design described in this document, these operations take the form of on-line presentations of new material, exercises directed at reinforcing specific instructional objectives, "laboratory" sessions in which students try out graphics commands in a controlled environment, and formal tests.

The representation of the task is a detailed task analysis listing each component skill needed to master the material being taught. It is represented as a directed graph that defines the prerequisite relationships between each skill.

The student model is a representation of the student's knowledge in terms of the task analysis and a history of the student's interactions with the program. It can be thought of as a state vector that describes the student's degree of mastery for each component skill and various other pertinent student characteristics.

The means-ends guidance rules relate states defined by the student model to sets of teaching operations. These rules determine which instructional activities the CAI program will present next given different student states. (See Heines, 1983, for a basic discussion of rule-based systems.)

Each of these components is described in detail in the sections that follow.

THE VOCABULARY OF TEACHING OPERATIONS

ReGIS, the Remote Graphics Instruction Set, allows programmers to perform a large variety of graphics operations on GIGI and VT125 terminals manufactured by Digital Equipment Corporation. The course design presented in this paper limits itself to the first three ReGIS instructions (Position, Vector, and Curve), screen addressing in absolute, relative, and default modes, and the use of the terminals' address stack. This subject matter was chosen because it is:

- sophisticated enough to yield a meaningful task analysis, yet simple enough to allow the analysis to be performed in a reasonable amount of time,
- applicable to a wide range of students at various levels so that its response-sensitivity can be evaluated,
- suitable for the type of treatment to be implemented.

The course employs four teaching operations:

- expository demonstrations,
- directed exercises,
- a ReGIS "laboratory", and
- formal tests.

The characteristics of these operations are described in the sections that follow. Their interrelationships are shown in Figure 1.

Expository Demonstrations

Expository demonstrations are basically "press-RETURN-to-continue" slide shows that introduce concepts. Of all the teaching operations, they exhibit the lowest levels of interaction and response-sensitivity. Their purpose is as much "telling" as it is "teaching", and they last about 3-5 minutes each. While one or two orientation questions may be incorporated into the presentation, the student's only real option is to interrupt the operation and call up a control menu. These sections make heavy use of graphics to relate ReGIS commands to screen actions.

Directed Exercises

Exercises provide either computer-generated or prestored problems for students to solve. In the course, student responses to these problems would usually take the form of ReGIS command strings. The CAI program would parse student responses to allow extensive

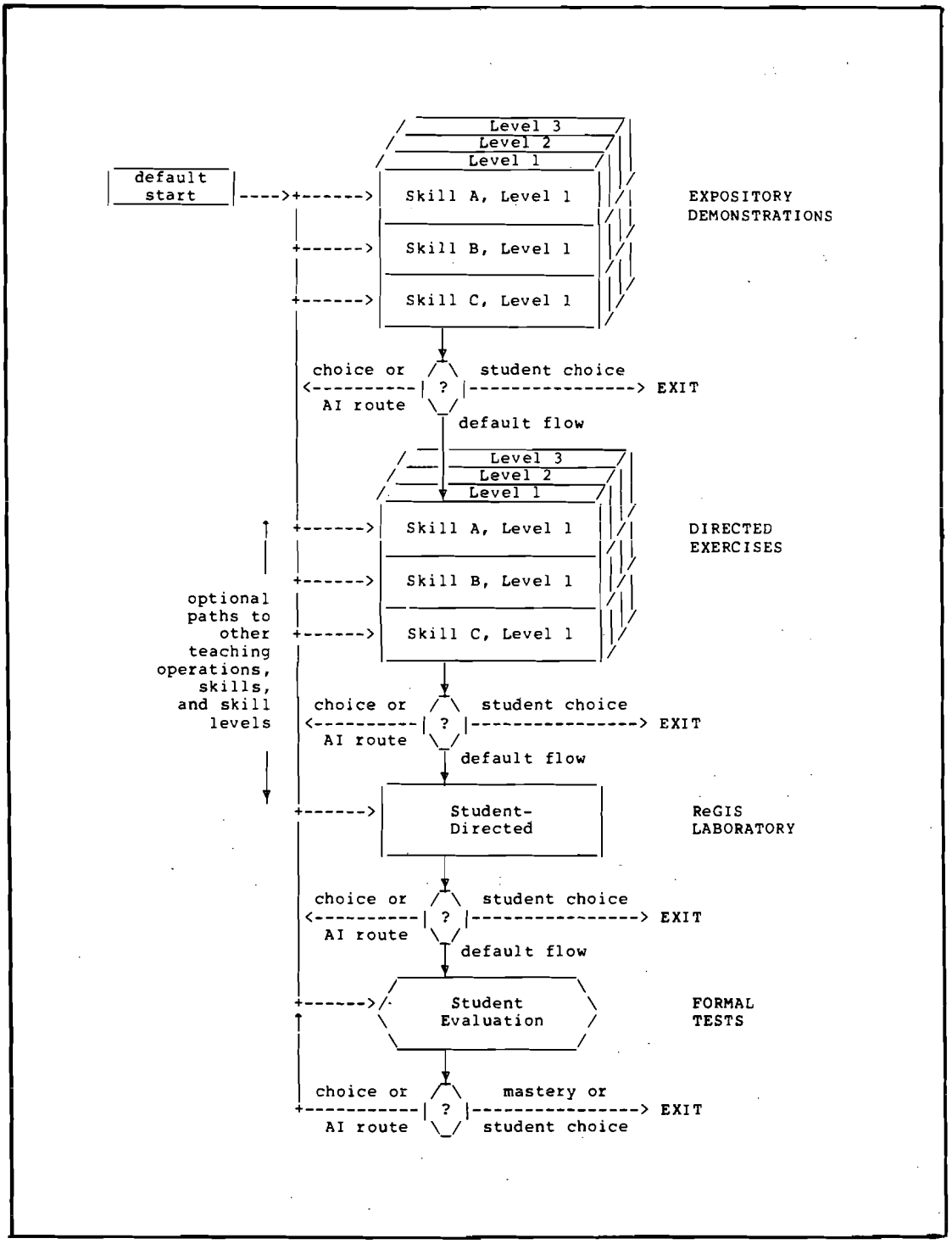

Figure 1

RELATIONSHIPS BETWEEN TEACHING OPERATIONS

error checking and provide detailed feedback.

AI comes into play in these exercises as student responses are parsed. Each skill demonstrated by the student is recorded in a response history file, and the student model is updated accordingly. This data can then be used to point out inconsistencies in the student's thinking if s/he makes an error using the same skill in a later lesson.

The "directed" nature of the exercises refers to using the student's performance data to determine the type and difficulty of problems presented. This feature contributes to the program's response-sensitivity. Students who do well will find that the problems get harder quite quickly. Weaker students will be led along more slowly, making sure that they possess each component skill before higher level skills are presented.

The initial exercise type and difficulty level depend on three factors:

(1) whether the student has gone through the expository demonstration,

(2) the reason the student is going through the exercises (by choice or as remediation caused by problems with higher level material), and

(3) the student's own statement of what s/he thinks s/he knows.

Students will normally exit an exercise section by demonstrating mastery or non-mastery of the skills being practiced. (They may also exit by pressing an EXIT key or entering EXIT in response to a question.) If they demonstrate non-mastery, one of three things may happen depending upon their state as defined by the student model:

(1) If the student has already demonstrated mastery on each of the prerequisite skills for this module, s/he would be branched to a set of secondary teaching operations (if present) on the current material. (Secondary teaching operations are ones that are designed for remediation only, not initial presentation.) These teaching operations present the material again in a more detailed manner or from another perspective. This rule implies that there might be some instructional modules that are only accessed for remedial purposes. Students who go through the course grasping concepts the first time around would never see these modules.

(2) If the student has not demonstrated mastery on <u>one</u> of the prerequisite modules, s/he would be directed to study (or restudy) that module before attempting the

current module again.

(3) If there are more than one prerequisite modules on which the student has not demonstrated mastery, the system would ask a number of questions to try to identify which prerequisites are most likely missing. This is done by analyzing the task representation in relation to the student model.

ReGIS Laboratory

The third teaching operation is intended to provide the greatest amount of response-sensitivity and demonstrate the course's most sophisticated AI aspects. These characteristics can be accomplished by providing students with a ReGIS "sketchpad" or "laboratory" environment in which they can enter ReGIS commands and see the results of these commands directly. The lab would be implemented in a dual screen format with the ReGIS code appearing on one side and the graphic output on the other.

The AI component here involves "watching" students as they enter commands, updating the student model for each skill demonstrated, and looking for cliché errors in their code. These techniques are similar to those of Burton and Brown (1982) and Shrager and Finin (1982). Even if the course did not attempt to offer any real "coaching" à la WEST (Burton and Brown, 1982), it could still provide detailed error messages for syntactic and simple theoretical errors similar to those in the directed exercises.

The difference between the directed exercises and ReGIS lab is the degree of computer control. If effectively implemented, measures of their instructional effectiveness should yield equivalent results. A major difference between the two styles, however, is that it is more difficult to identify missing prerequisites in a laboratory environment because there is no mechanism for asking direct questions. Failure to demonstrate mastery in this environment would therefore be coupled with more conventional directed exercises.

Formal Tests

When a student demonstrates a particular skill in either the directed exercises or ReGIS laboratory, the probability with which s/he actually possesses that skill is somewhat less than 1.Ø. For example, a student may "discover" defaults by leaving out an X or Y value for the P command in the ReGIS laboratory. No error message would be generated, and the student may not even realize what has happened until some time later. While this situation represents an excellent (and some would say the best) learning scenario, it is impossible to know for certain whether the student has actually internalized the concept s/he has just

demonstrated.

Evaluation of actual learning requires a formal testing situation. The administration of formal tests could be very similar to the methods employed for directed exercises, with the stipulation that part of the feedback would be eliminated. Formal testing need not be limited to multiple choice and short answer responses.

THE TASK REPRESENTATION

List of Component Skills

Table 1 lists the component skills needed to master all aspects of the ReGIS Position, Vector, and Curve commands covered in the course. Each of these skills represents the course's smallest possible instructional unit. That is, the course's AI component would be designed to identify skills that a student lacks and route him or her to the specific teaching operations on those skills.

The component skills would be grouped into modules for presentation to students going through the course for the first time. This does not mean that remedial work on one skill in a module necessitates redoing the other skills in that module. It simply means that the skills would be presented together the first time for the sake of continuity and organizational convenience. The skill groupings for each module are shown in Table 2.

Skills 35 and 43 would not actually be taught in the course. These skills involve understanding angle measure in degrees and interpolation, respectively, and are what Robert Mager and Peter Pipe (1974) call "entry level objectives". They are required for students to master higher level objectives, but students are expected to bring these skills to the course rather than learn them from the course.

Prerequisite Relationships: Directed Graph

A number of prerequisite relationships exist between the 50 component skills. Such prerequisites indicate those skills a student must possess in order to master higher level skills. Figure 2 shows the prerequisite relationships represented by a directed graph. This graph should be interpreted as follows:

- Nodes inside shaded squares represent skills with no prerequisites. These are Skills 1, 25, 35, and 43.

Table 1

LIST OF COMPONENT SKILLS

Number	Description
1	Recognizes screen as a rectangular dot matrix.
2	Can translate screen positions into (x,y) pairs.
3	Can translate (x,y) pairs into screen positions.
4	Knows absolute screen limits (767,479).
5	Knows that initial cursor position is upper left-hand corner of screen.
6	Understands the concept of current cursor position.
7	Can interpret the standard [x,y] address format.
8	Can specify absolute screen addresses in [x,y] format.
9	Understands defaults.
	Given the current cursor position as [xc,yc], knows the meaning of:
10	[x] -> [x,yc]
11	[,y] -> [xc,y]
12	[] -> [xc,yc]
13	Understands relative addresses.
	Given the current cursor position as [xc,yc], knows the meaning of:
14	[±x,±y] -> [xc±x,yc±y]
15	[±x] -> [xc±x,yc]
16	[,±y] -> [xc,yc±y]
17	[±x,y] -> [xc±x,y]
18	[x,±y] -> [x,yc±y]
19	Can use the basic P command to position the cursor.
20	Knows the current cursor position after execution of a P command.
21	Can use all addressing schemes in P commands.
22	Can work with P command aberrations such as P [x1,y1] [x2,y2] ... [xn,yn].
23	Can store addresses on the stack with (B).
24	Can pop addresses off the stack with (E).
25	Can use the S(E) command to erase the screen.
26	Can use the basic V command to draw a vector.
27	Knows the current cursor position after execution of a V command.

(continued...)

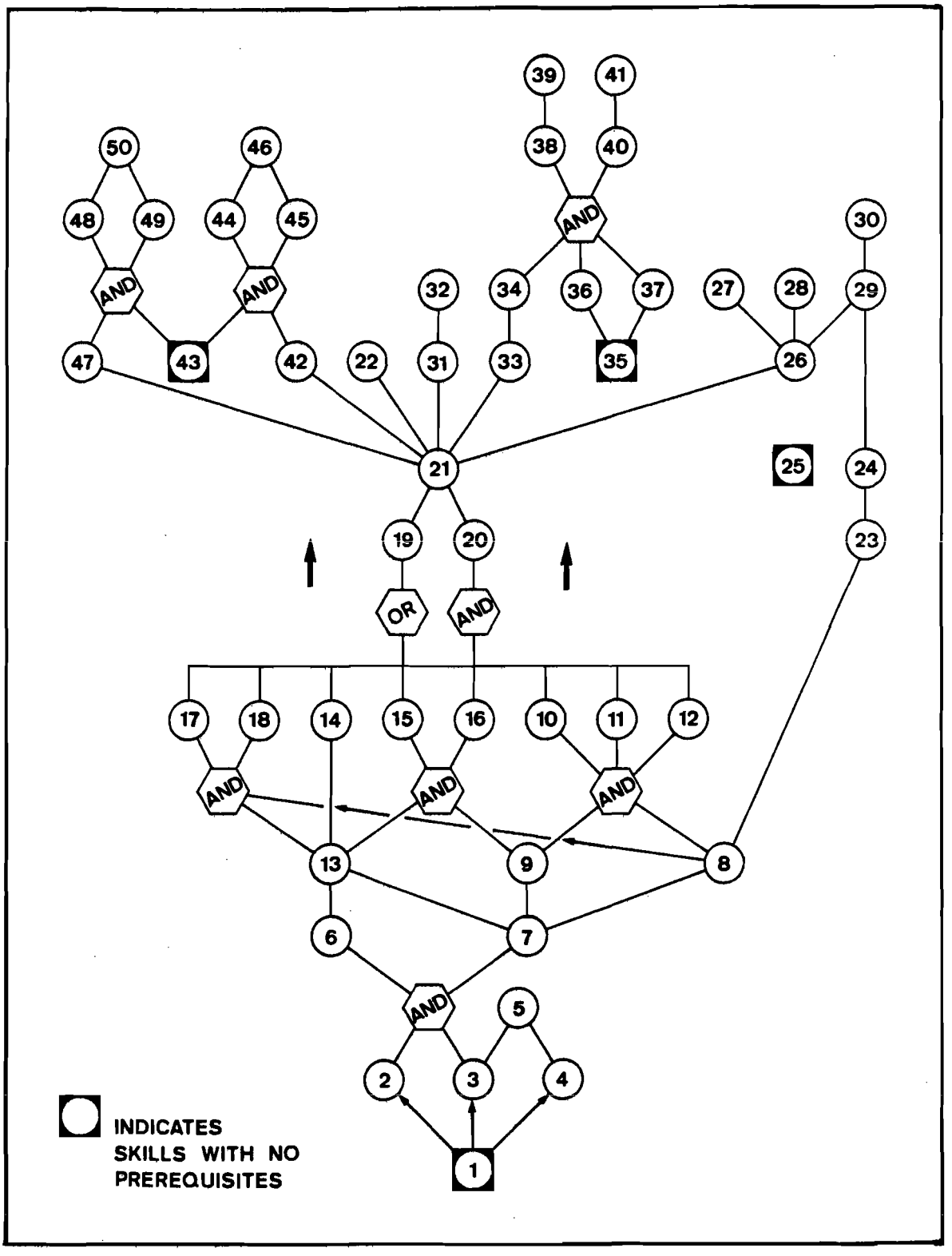

Figure 2

TASK MODEL REPRESENTED AS A DIRECTED GRAPH

- The skill hierarchy reads from bottom to top. Lines connecting two skills indicate that the higher level skill requires the lower level skill below it as a prerequisite. For example, Skill 2 requires Skill 1 as a prerequisite.

- When two or more lines lead into a node from the bottom, all of the lower level skills are required as prerequisites. For example, Skill 5 requires both Skills 3 and 4 as prerequisites.

- When two or more lines lead out of a node from the top, the lower level skill is a prerequisite for each of the higher level skills. For example, Skill 26 is a prerequisite for Skills 27, 28, and 29.

- The AND hexagon is not a node. It indicates that all of the lower level skills are required as prerequisites for each of the higher level skills. For example, Skills 6 and 7 each require both Skills 2 and 3 as prerequisites.

- The OR hexagon is not a node. It indicates that any one of the lower level skills is sufficient prerequisite for each of the higher level skills. For example, Skill 19 requires any one of Skills 10 through 18 as a prerequisite.

(The AND and OR hexagons were used to keep the graph readable by avoiding a large number of crossing lines. Note that Skill 19 requires any of Skills 10-18, while Skill 20 requires all of Skills 10-18. Skill 21, therefore, requires all of Skills 10-20.)

The prerequisite relationships of the component skills dictate the prerequisite relationships between the modules. These prerequisites are shown in Figure 3.

Prerequisite Relationships: Production Rules

The directed graph presented in the previous section is equivalent to the set of facts listed in Table 3. This table should be interpreted as follows:

- NIL in the left-hand column indicates that no prerequisites are required for the corresponding skills in the right-hand column. For example, no prerequisites are required for Skills 1, 25, 35, and 43.

- If more than one skill is listed on a single line in the left-hand column, all of those skills are required as prerequisites for the each of the skills listed in the right-hand column. (This is the AND function.) For

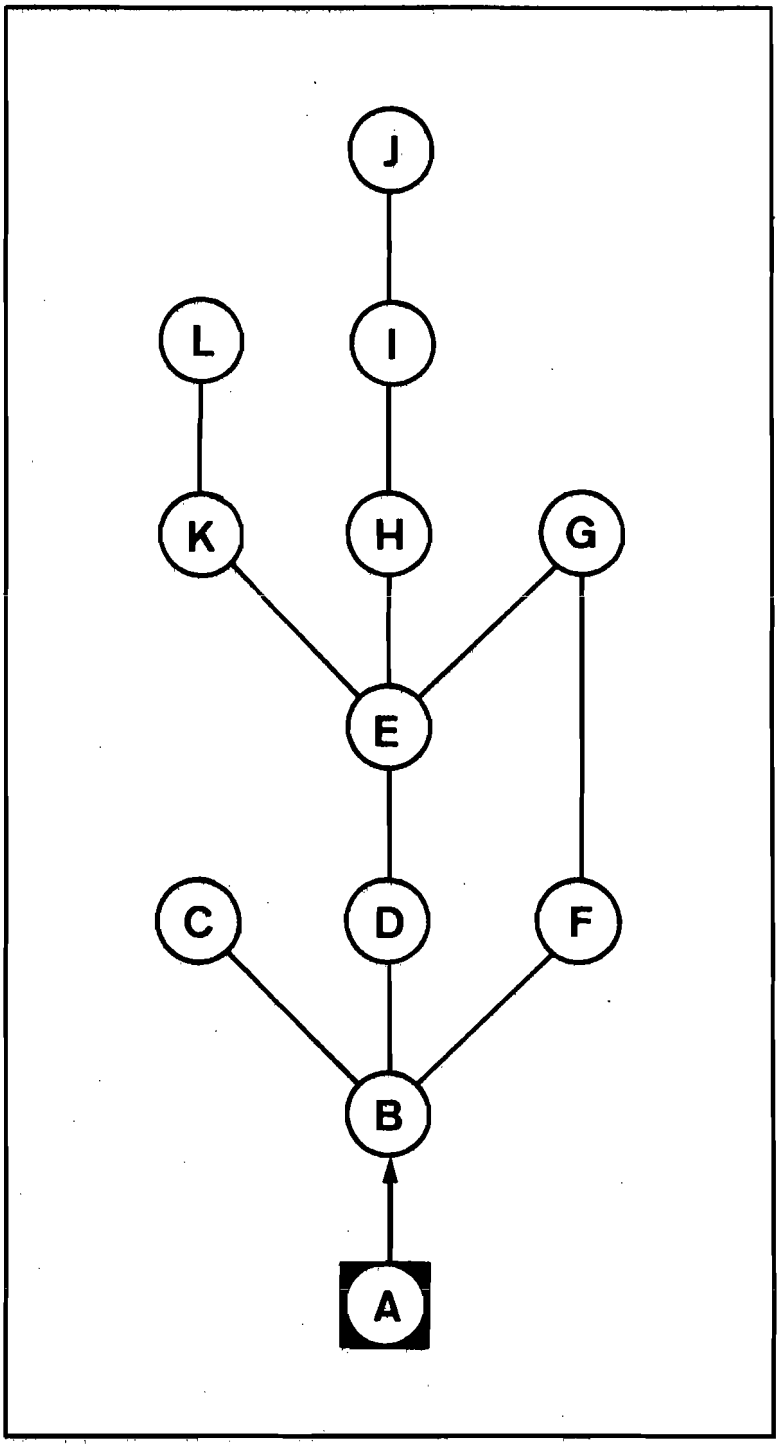

Figure 3

PREREQUISITE RELATIONSHIPS BETWEEN MODULES

Table 3

TASK MODEL REPRESENTED AS PRODUCTION RULES

if the student has mastered these skills ...	s/he has met the prerequisites for these skills ...
NIL	1 25 35 43
1	2 3 4
2 3	6 7
3 4	5
6 7	13
7	8 9
8 9	10 11 12
8 13	17 18
8	23
9 13	15 16
10 11 12 13 14 15 16 17 18	20
10	19
11	19
12	19
13	14
13	19
14	19
15	19
16	19
17	19
18	19
19 20	21
21	22 26 31 33 42 47
23	24
24 26	29
26	27 28
29	30
31	32
33	34
34 36 37	38 40
35	36 37
38	39
40	41
42 43	44 45
43 47	48 49
44 45	46
48 49	50

example, Skills 3 and 4 are both required as prerequisite for Skill 5.

- If more than one skill is listed on a single line in the right-hand column, all of the skills listed in the corresponding left-hand column are required as prerequisites for each of the skills listed in the right-hand column. For example, both Skills 6 and 7 require Skills 2 and 3 as prerequisites.

- If a skill appears on more than one line in the left-hand column, that skill is required as a prerequisite for more than one higher level skill. For example, Skill 3 is required as a prerequisite for Skills 5, 6, and 7.

- If a skill appears on more than one line in the right-hand column, any of the corresponding left-hand columns provides sufficient prerequisites for that skill. (This is the OR function.) For example, Skill 19 requires any one of Skills 10 through 18 as a prerequisite.

Using these facts, the prerequisite relationships (and thus the entire representation of the task) can be defined by production rules. The full set of production rules defines the program's task model.

The production rule formalism makes it conceptually simple to identify both the skills for which the student has met the prerequisites and the prerequisites needed to study any particular skill. For example, suppose that a rudimentary student model consists of a simple list of the skills that a particular student has mastered. Such a list might contain 1, 2, 3, 7, 8, 9, and 12. A function can then be written that steps down the list of task model rules, testing whether each left-hand side (LHS) is a perfect subset of the student model. If it is, the student has met the prerequisites for the skills listed on the right-hand side (RHS) of that rule. For the example list of skills shown above, this function would identify the Skills 4, 6, 10, 11, 19, 23, 25, 35, and 43. For a fuller understanding of why this is so, compare this list to the graph in Figure 2. The student is ready for:

- Skill 4 because the list of skills mastered includes Skill 1,

- Skill 6 because it includes both Skills 2 and 3,

- Skills 10 and 11 because it includes both Skills 8 and 9,

- Skill 19 because it includes Skill 12 (only one of Skills 10-18 is required for Skill 19, but note that the student is not ready for Skill 20, because that skill requires all Skills 10-18),

- Skill 23 because it includes Skill 8, and
- Skills 25, 35, and 43 because these have no prerequisites.

The discussion thus far has concerned moving up the directed graph to answer the question: "given a specific set of mastered skills, which skills is the student now ready to study, i.e., for which skills does the student now possess the prerequisites?" The beauty of the production rule approach is that the same representation can be used equally well to move down the directed graph and answer the converse question: "given a specific skill, which prerequisite skills must the student possess to be ready to study it?" This characteristic is crucial to achieving the AI diagnostic qualities of the directed exercises and ReGIS laboratory discussed previously.

For example, suppose that the student hadn't really studied all of the skills specified by the list 1, 2, 3, 7, 8, 9, and 12. Instead, s/he may have actually only studied Skill 12. By virtue of demonstrating mastery on that skill, the system's AI component would update its student model by marking the student's mastery of all the skills prerequisite for Skill 12 as "assumed". To determine which skills to mark, a function can be written that tests the RHS of each rule. If the skill just mastered is a member of the list of RHS skills, mastery of each of the skills listed on the LHS is assumed.

In practice, the function described above would be called with the number of the skill just mastered and a list of skills representing the student model. The function would then return a list of RHS skills that are not already members of the student model. If, for example, the student model list is empty, calling this function with "12" as an argument would return the list of skills 1, 2, 3, 7, 8, and 9. If the student model already indicates mastery on Skills 1 and 3, calling it with "12" as an argument would return the Skills 2, 7, 8, and 9.

THE STUDENT MODEL

Skill Status

The basic purpose of a student model is to represent a student's current knowledge state. In its simplest form, this state might be defined by the student's status on each of the skills in the task model. As indicated in the previous section, one rudimentary way to do this is to maintain a list (or state vector) of those skills on which the student has demonstrated mastery. The utility of this list can be greatly improved, however, by letting the status of each skill take on a number of values. The student

model employed in the course would use the following seven values:

-3 NON-MASTERY DEMONSTRATED on a test
The student has demonstrated that s/he does not possess this skill by failing a test that covered it. This is the strongest assertion of non-mastery that the system can make.

-2 NON-MASTERY ASSUMED due to incorrect usage in lab
The student is assumed not to possess the skill because s/he has failed a test that covers a skill prerequisite to the one in question.

-1 NON-MASTERY ASSUMED due to incorrect usage of a prerequisite skill
The student is assumed not to possess the skill because s/he has either demonstrated non-mastery on or used incorrectly a lower level skill for which this skill is a postrequisite. (Note that the skill in question may be more than one level removed from the lower level skill on which the student is actually working.) This is the weakest assertion of non-mastery that the system can make.

Ø NO DATA
The student has not studied this skill, has not demonstrated mastery on any skill for which it is a prerequisite, and has not demonstrated non-mastery on any skill for which it is a postrequisite.

1 MASTERY ASSUMED due to correct usage of a postrequisite skill
The student is assumed to possess the skill because s/he has either demonstrated mastery on or used a higher level skill for which this skill is a prerequisite. This is the weakest assertion of mastery that the system can make.

2 MASTERY ASSUMED due to correct usage in lab
The student is assumed to possess the skill because s/he used the skill in either the ReGIS laboratory or the directed exercises.

3 MASTERY DEMONSTRATED on a test
The student has demonstrated mastery of this skill by passing a test that covered it. This is the strongest assertion of mastery that the system can make.

The student model value for each skill is initialized to Ø when the student registers. As s/he works through the course, one of the non-zero values is assigned to each skill on which the system has or can infer data. These values add a level of complexity to

the functions introduced in the discussion of the task representation, in that analysis of the production rules cannot be done simply by testing for the presence of a skill number in a list. The complication is not extreme, however, and should present no serious implementation problems.

It is also possible to express a student model in terms of procedures rather than a state vector. See Self, 1974, for a discussion of this technique.

Learning Rate and Learning Style

In addition to a student's skill status, the student model can also maintain two simple and rudimentary representations of the student's learning rate and learning style. Learning rate is a measure of the student's ability to assimilate new material quickly. Learning style is a measure of the manner in which the student prefers new material to be presented.

The student's learning rate would govern the speed, depth, and amount of repetition and reinforcement in initial presentations of new material. Fast students would receive fast presentations extending to considerable depth before going into the directed exercises as reinforcers. Slower students would be presented with more detailed introductions to new material at lower levels, and would find more repetition in the presentations as well as more frequent reinforcement via the directed exercises. Learning rate might be expressed as one of the following five values:

- VERY-FAST
- FAST
- AVERAGE
- SLOW
- VERY-SLOW

The student's preferred learning style might be represented by one of three values:

- EXPOSITORY -- the student prefers to go through the full expository demonstration before doing exercises.

- EXERCISE -- the student prefers to dive right into the directed exercises.

- LABORATORY -- the student prefers to try things out in the ReGIS laboratory after a short explanation of pertinent concepts and commands.

A number of techniques exist for assessing learning style, but a finesse is also feasible: simply ask students which style they prefer. Students would be allowed to change their learning style preference as the course progresses, as well as override the

default selection for any particular module. The system might monitor the number of overrides and change the default learning style when this number becomes significant.

THE MEANS-ENDS GUIDANCE RULES

Means-ends guidance rules relate states defined by the student model and student history to specific teaching operations and determine which instructional activities the CAI program will present next given different student states. A rudimentary student history could be as simple a list of all responses entered by the student. In practice, this history might also flag responses to non-subject matter queries as "choices" made by the student, e.g., his or her selections when presented with a number of options on a menu.

Sample means-ends guidance rules (in plain English format) might be as follows:

1. If the student is entering a module for the first time, make the initial subject matter presentation in the form specified by the LEARNING-RATE and LEARNING-STYLE elements of the student model.

2. If the student is reentering a module s/he has already studied and done <u>well</u> on, query him or her as to what skills s/he wishes to study and in what learning style. (The response to these queries would be recorded in the student history as "choices".)

3. If the student is reentering a module s/he has already studied but done <u>poorly</u> on, make subject matter presentations in EXPOSITORY style on all Skills for which the student model indicates NO-DATA or NON-MASTERY, and make these presentations as if the value of LEARNING-RATE was SLOW or VERY-SLOW. (This would force more repetition and reinforcement.)

4. If the student demonstrates non-mastery on a specific skill and the student model indicates that there are <u>no</u> prerequisites for that skill on which mastery has <u>not</u> been demonstrated or assumed (that is, mastery has been demonstrated or assumed on all the prerequisites), branch to a secondary teaching operation for that skill if one exists. If no secondary teaching operations exist for the skill in question, apply Rule 3 above. (The bulky negative wording in the IF clause was used to make this rule consistent with Rules 5 and 6.)

5. If the student demonstrates non-mastery on a specific skill and the student model indicates that there is only one prerequisite for that skill on which mastery has not been demonstrated or assumed, apply Rule 1 to the module containing that prerequisite skill.

6. If the student demonstrates non-mastery on a specific skill and the student model indicates that there are more than one prerequisites for that skill on which mastery has not been demonstrated or assumed, apply Rule 1 to the module containing the prerequisite s/he is "most likely" lacking. (The system would determine which skill is "most likely" lacking by analyzing the student model values for other skills with the same prerequisites.)

Representation of Left-Hand Sides

The left-hand sides (LHSs) of these rules (the IF parts) represent specific patterns to be matched against data derived from the student model and student history. For Rules 1, 2, and 3, these data would include the status of the module the student has chosen to study and the statuses of each of that module's subskills. The LHSs of these rules might therefore take the form:

1. ((STATUS-OF-MODULE-CHOSEN EQUAL Ø)
(ALL-SUBSKILL-STATUSES EQUAL Ø)))

2. ((STATUS-OF-MODULE-CHOSEN (NOT EQUAL) Ø)
(AVERAGE-SUBSKILL-STATUS GREATER-THAN Ø)))

3. ((STATUS-OF-MODULE-CHOSEN (NOT EQUAL) Ø)
(AVERAGE-SUBSKILL-STATUS LESS-THAN Ø)))

The function calls (STATUS-OF-MODULE-CHOSEN EQUAL Ø) and (STATUS-OF-MODULE-CHOSEN (NOT EQUAL) Ø) would return TRUE if the status of the module chosen is equal to or not equal to Ø, respectively. The function call (ALL-SUBSKILL-STATUSES EQUAL Ø) would operate on sets of skills, and return TRUE if each of those skills has a status value equal to Ø. Likewise, the function calls (AVERAGE-SUBSKILL-STATUS GREATER-THAN Ø) and (AVERAGE-SUBSKILL-STATUS LESS-THAN Ø) would return TRUE if the average subskill status value is greater than or less than Ø, respectively. When the values of all function calls on the LHS of a rule are TRUE, that rule fires.

For Rules 4, 5, and 6, the patterns to be matched would include the status of the particular skill just studied and the statuses of each of that skill's prerequisite skills. The LHSs of these rules might therefore take the form:

```
4.  ((SKILL-STATUS LESS-THAN Ø)
         (NO-PREREQ-SKILL-STATUSES LESS-THAN Ø))

5.  ((SKILL-STATUS LESS-THAN Ø)
         (ONLY-ONE-PREREQ-SKILL-STATUS LESS-THAN Ø))

6.  ((SKILL-STATUS LESS-THAN Ø)
         (MORE-THAN-ONE-PREREQ-SKILL-STATUS LESS-THAN Ø))
```

Representation of Right-Hand Sides

The right-hand sides (RHSs) of the rules (the THEN parts) could be expressed as a TEACH function with the form:

```
(TEACH (MODULE-ID
        SKILL-ID
        LEARNING-RATE
        LEARNING-STYLE
        SEARCH-STRATEGY))
```

where

- MODULE-ID represents the module to be entered, as identified in Table 2.
- SKILL-ID represents the skill to be taught, as identified in Table 1.
- LEARNING-RATE represents the amount of repetition and reinforcement to use in presentations.
- LEARNING-STYLE represents which of the three types of teaching operations to use.
- SEARCH-STRATEGY represents the manner in which this teaching operation was selected, e.g., REMEDIAL or MOST-NEEDED.

Using this format, the RHSs of the sample rules could be expressed as follows (the asterisk is a wild card that matches any value of the function argument list in the corresponding position):

```
1.  (TEACH (MODULE-CHOSEN
            *
            LEARNING-RATE
            LEARNING-STYLE
            *))
```

```
2.  (TEACH (MODULE-CHOSEN
            QUERY
            FAST
            QUERY
            *))

3.  (TEACH (MODULE-CHOSEN
            (SKILLS-WITH-STATUSES (LESS-THAN OR EQUAL) Ø)
            SLOW
            EXPOSITORY
            *))

4.  (TEACH (MODULE-CONTAINING-SKILL
            SKILL
            LEARNING-RATE
            *
            REMEDIAL))
    OR
    (TEACH (MODULE-CONTAINING-SKILL
            SKILL
            SLOW
            EXPOSITORY
            *))

5.  (TEACH (MODULE-CONTAINING-PREREQUISITE-SKILL
            *
            LEARNING-RATE
            LEARNING-STYLE
            *))

6.  (TEACH (MODULE-CONTAINING-PREREQUISITE-SKILL
            *
            LEARNING-RATE
            LEARNING-STYLE
            MOST-NEEDED))
```

The RHS for Rule 4 has two TEACH functions to cover the case in which no secondary teaching operations exist for a specific skill. This representation makes the rules more complex, but provides explicit definition of what to do if a TEACH function request cannot be filled. Another way to tackle this problem is to use only one TEACH function in each rule but check whether the rule succeeds after the RHS fires. If the rule does not succeed, the system must continue looking for another rule whose LHS matches the data in the student model and student history.

CRITIQUE AND CONCLUSIONS

The rule-based tutorial described in this paper has not undergone the test of implementation. However, the approach described here

is a conceptually clean extension of a working computer tutor that uses production rules in the teaching of quadratic equations (O'Shea, 1979). On implementation, some of the details of the formalism described here would probably have to change to ensure computational efficiency and to maintain reasonable response time in the particular interactive computer environment adopted.

For example, test implementations in MacLISP on a DECsystem-20 have shown that while the production rule formalism is highly efficient for expressing the prerequisite relationships as shown in Table 3, use of this formalism to update the student model after each response is relatively inefficient. The main inefficiency stems from processing the rules repeatedly to find all of the prerequisites or postrequisites for the skill on which the student is currently working. This problem can be attacked by computing all of the prerequisites and postrequisites when the course is installed and storing these as lists in a simple array. This approach allows the student model to be updated much more quickly without sacrificing the elegance of the rule-based strategy.

Test implementations have also shown that the production rule approach can be applied to interpreting student responses. The parser that interprets command strings entered by the student in the ReGIS laboratory can be made to return specific patterns, which can be related to the skills in the task model via production rules. For example, consider the command:

```
P[250,100]
```

to position the cursor 250 pixels from the left-hand margin and 100 pixels down from its current position. Given this command, the parser returns:

```
((COMMAND POSITION) (TYPE POINT)
 (X-VALUE-TYPE ABSOLUTE) (X-VALUE 250)
 (Y-VALUE-TYPE RELATIVE) (Y-VALUE-SUBTYPE +)
 (Y-VALUE 100))
```

This result can be related to the skills in the task model with the following rules (these are only a subset of the full set of address diagnostic interpretation rules):

```
(((X-VALUE-TYPE ABSOLUTE) (Y-VALUE-TYPE ABSOLUTE))  8 19)
(((X-VALUE-TYPE ABSOLUTE) (Y-VALUE-TYPE DEFAULT))  10 19)
(((X-VALUE-TYPE DEFAULT)  (Y-VALUE-TYPE ABSOLUTE)) 11 19)
(((X-VALUE-TYPE DEFAULT)  (Y-VALUE-TYPE DEFAULT))  12 19)
(((X-VALUE-TYPE RELATIVE) (Y-VALUE-TYPE RELATIVE)) 14 19)
(((X-VALUE-TYPE RELATIVE) (Y-VALUE-TYPE DEFAULT))  15 19)
(((X-VALUE-TYPE DEFAULT)  (Y-VALUE-TYPE RELATIVE)) 16 19)
(((X-VALUE-TYPE RELATIVE) (Y-VALUE-TYPE ABSOLUTE)) 17 19)
(((X-VALUE-TYPE ABSOLUTE) (Y-VALUE-TYPE RELATIVE)) 18 19)
```

For each rule whose LHS is a perfect subset of the result returned by the parser, the program updates the student model by:

(1) assigning a value of +2 to each the skill listed on the rule's RHS, and then

(2) assigning a value of +1 to each of the prerequisites for each of those skills.

Thus the rule-based tutorial might employ several sets of rules for different functions. The advantage of this approach is that all such sets are easily changed to "tune" the course and enhance its response-sensitivity.

Production rule programming is not a trivial task, and one line of development being pursued is a rule-based authoring system which makes it easy for educational designers without substantial programming skills to enter and change sets of course related production rules (O'Shea et al., 1983). However, we contend that even without access to such a system it is still more effective to build CAI programs by identifying tutorial rules than to use conventional CAI authoring languages, because latter typically exhibit restrictive orientations toward automating programmed learning texts via the clumsy apparatus of frames, branches, and multiple choice questions.

In conclusion, we have shown how the rule-based approach can be usefully applied to the design of a course that includes exposition, directed exercises, a simulated laboratory, and tests. In the resulting course, the various teaching operations are modular and distinct, as are the production rules used in the student model for response-sensitivity and as means-ends guidance rules for scheduling the presentation of teaching operations. It is therefore possible, for example, to integrate new teaching operations into the course while maintaining the general level of response-sensitivity by adding new production rules. Likewise, any increase in response-sensitivity achieved in the student model will be applied to all teaching operations. We believe that these rule-based techniques represent an efficient and elegant approach to the task of designing and implementing CAI tutorials.

REFERENCES CITED AND RELATED READINGS

Burton, Richard R., and John Seely Brown, 1982. "An Investigation of Computer Coaching for Informal Learning Activities." In Intelligent Tutoring Systems, ed. D. Sleeman and J.S. Brown, pp. 79-98. New York: Academic Press.

Hartley, J.R., 1973. "The Design and Evaluation of an Adaptive Teaching System," International Journal of Man-Machine Studies, 5(2).

Heines, Jesse M., 1983. "Basic Concepts in Knowledge-Based Systems," Machine-Mediated Learning, 1(1):65-96.

Mager, Robert F., and Peter Pipe, 1974. Criterion-Referenced Instruction: Analysis, Design, and Implementation. Mager Associates, Los Altos Hills, CA.

O'Shea, Tim, 1979. "Self-Improving Teaching Systems." Ph.D. Dissertation, University of Leeds. Basel, Boston, Stuttgart: Birkhauser.

O'Shea, Tim, R. Bornat, B. du Boulay, M. Eisenstadt, and I. Page, 1983. "Tools for Designing Intelligent Computer Tutors," in Human and Artificial Intelligence, ed. A. Elithorn and R. Banerjii, London: North-Holland.

Self, John, 1974. "Student Models in Computer-Aided Instruction," International Journal of Man-Machine Studies, 6:261-276.

Shrager, Jeff, and Tim Finin, 1982. "An Expert System that Volunteers Advice." Proceedings of the 1982 National Conference on Artificial Intelligence (sponsored by the American Association for Artificial Intelligence), Pittsburgh, Pennsylvania, August 18-20, 1982, pp. 339-340. Los Altos, CA: William Kaufmann, Inc.

ACKNOWLEDGMENTS

This work was performed while the first author (J.H.) was a Visiting Researcher with the Computer Assisted Learning Research Group at The Open University. He is grateful to Tony Hasemer and Rick Evertsz for their help in programming test implementations of these ideas in LISP, and to the staff of the Academic Computing Service for making computer equipment and time available for this study.

The second author (T.O'S.) is grateful to Richard Bornat of Queen Mary College for his substantial help in developing production rule formalisms for computer tutors.