Int. J. Man-Machine Studies (1985) **23**, 1-25

The design of a rule-based CAI tutorial

JESSE M. HEINES

Dept. of Computer Science, University of Lowell, MA 01854, *USA*

AND

TIM O'SHEA

The Open University, Milton Keynes, England

(*Received 31 August 1983, and in revised form 20 March 1984*)

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.

The paper presents the design of a CAI course employing a rule-based tutorial strategy. This design has not undergone the test of full implementation; the paper presents a conceptual design rather than a programming blueprint. One of the unique features of the course design described here is that it deals with the domain of computer graphics. The precise 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 inter-relationships, discusses how program control might be expressed in the form of production rules, and presents a program that demonstrates one facet of the intended course: the ability to parse student input in such a way that rules can be used to update a dynamic student model.

1. 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 the four components listed below. (These components are described in more detail in the sections that follow.)

(1) 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.

(2) 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.

0020-7373/85/070001+25$03.00/0

(3) 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.

(4) 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.)

2. 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

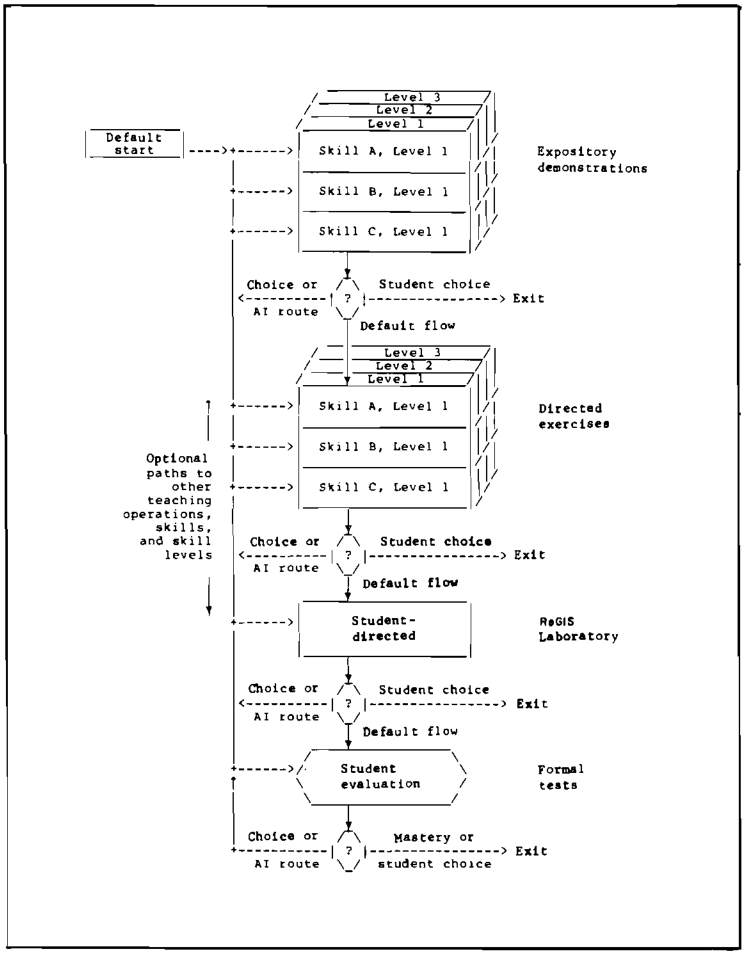


FIG. 1. Relationship between teaching operations.

first three ReGIS instructions (Position, Vector, and Curve), screen addressing in absolute, relative, and default modes, and the use of the terminal's address stack. The course employs four teaching operations: expository demonstrations, directed exercises, a ReGIS "laboratory," and formal tests. The inter-relationships between these operations are shown in Fig. 1, and their characteristics are described in the sections that follow.

2.1. 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.

2.2. DIRECTED EXERCISES

Exercises provide either computer-generated or prestored problems for students to solve. Student responses to exercises in this course will usually take the form of ReGIS command strings. The CAI program will parse these responses and provide extensive error checking with detailed feedback.

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.

2.3. ReGIS LABORATORY

The third teaching operation demonstrates the greatest amount of response-sensitivity 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 will be implemented in a dual screen format with the ReGIS code appearing on one side and the graphical output on the other (see section 6).

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 & Brown (1982) and Shrager & Finin (1982). Even if the course does not attempt to offer any real "coaching" à la West (Burton & Brown, 1982), it could still provide detailed error messages for syntactic and simple theoretical errors similar to those in the directed exercises. A program demonstrating some of these capabilities in a prototype ReGIS Laboratory is presented in section 6.

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 will therefore be coupled with more conventional directed exercises.

2.4 FORMAL TESTS

When a student demonstrates a particular skill in either the directed exercises or ReGIS laboratory, the probability with which she/he actually possesses that skill is somewhat less than 1·0. 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 she/he has just demonstrated.

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

3. The task representation

3.1. 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

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	$[x] \rightarrow [x, yc]$
11	$[, y] \rightarrow [xc, y]$
12	$[\,] \rightarrow [xc, yc]$
13	Understands relative addresses.
	Given the current cursor position as $[xc, yc]$, knows the meaning of:
14	$[\pm x, +y] \rightarrow [xc \pm x, yc \pm y]$
15	$[\pm x] \rightarrow [xc \pm x, yc]$
16	$[, \pm y] \rightarrow [xc, yc \pm y]$
17	$[\pm x, y] \rightarrow [xc \pm x, y]$
18	$[x, \pm y] \rightarrow [x, yc \pm y]$
19	Can use the basic P command to position the cursor.
20	Knows the current cursor position after execution of a P command.

TABLE 1 (*cont.*)

Number	Description
21	Can use all addressing schemes in P commands.
22	Can work with P command abbreviations such as P[*x*1, *y*1] [*x*2, *y*2] . . . [*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.
28	Can draw multiple vectors with a single V command.
29	Can draw closed polygons using (B) and (E) in V commands.
30	Knows the terms "open" and "closed" as applied to figures.
31	Can use the basic C command to draw circles.
32	Knows the current cursor position after drawing a circle with the basic C command.
33	Can use the C(C) command to draw circles.
34	Knows the current cursor position after drawing a circle with the C(C) command.
35	Understands angle measure in degrees.
36	Can translate angle measures into screen angles.
37	Can translate screen angles into angle measures.
38	Can use the C(A⟨degrees⟩) [X, Y] command to draw arcs starting at [X, Y] and using the current cursor position as the center.
39	Knows the current cursor position after drawing an arc with the C (A⟨degrees⟩) [X, Y] command.
40	Can use the C(A⟨degrees⟩C) [X, Y] command to draw arcs starting at the current cursor position and using [X, Y] as the center.
41	Knows the current cursor position after drawing an arc with the C(A⟨degrees⟩C) [X, Y] command.
42	Can draw open curves with the C(S) command.
43	Understands interpolation.
44	Can use [ ] at the front of a C(S) command.
45	Can use [ ] at the end of a C(S) command.
46	Knows the current cursor position after execution of a C(S) command.
47	Can draw closed curves with the C(B) command.
48	Can use [ ] at the front of a C(B) command.
49	Can use [ ] at the end of a C(B) command.
50	Knows the current cursor position after execution of a C(B) command.

will 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 will 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 will be presented together the first time for the sake of continuity and organizational convenience.

Skills 35 and 43 will not actually be taught in the course. These skills involve understanding angle measure in degrees and interpolation, respectively, and are what Mager & 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.

3.2. 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.

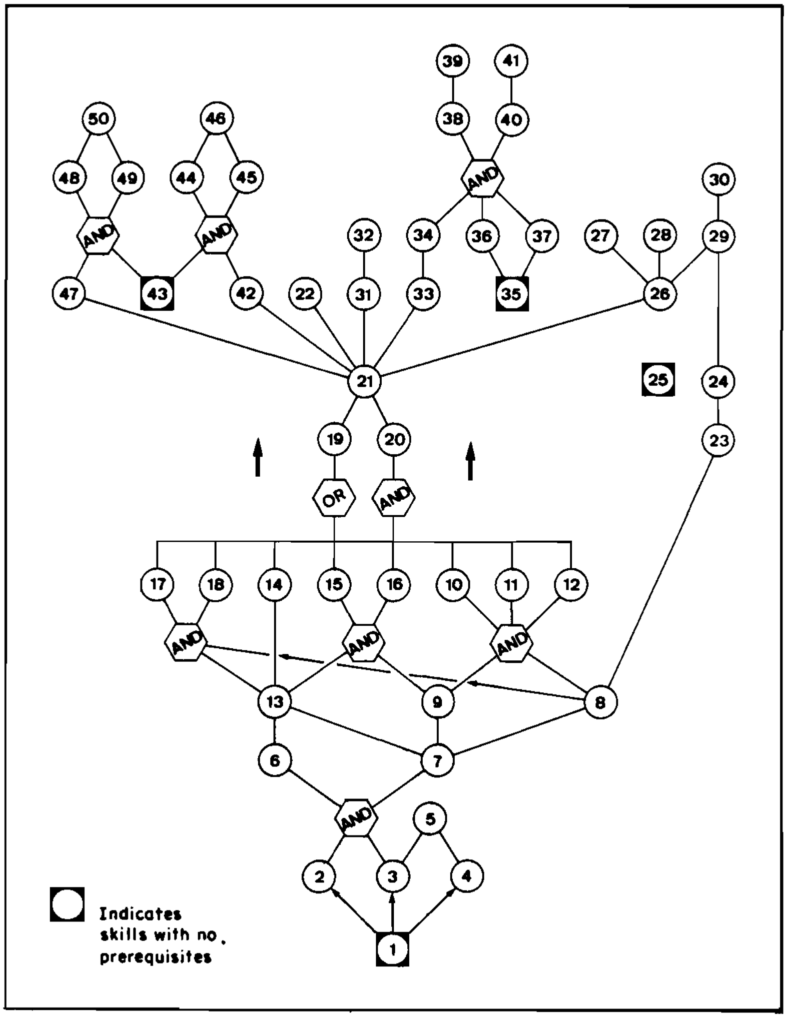


FIG. 2. Task model represented as a directed graph.

This graph should be interpreted as follows:

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

3.3. PREREQUISITE RELATIONSHIPS: PRODUCTION RULES

The directed graph presented in the previous section is equivalent to the set of facts listed in Table 2.

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 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

TABLE 2
Task model represented as production rules

If the student has mastered these skills . . .	She/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

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 Fig. 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.
- 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 in sections 3.2 and 3.3.

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, she/he may have actually only studied Skill 12. By virtue of demonstrating mastery on that skill, the system's AI component will 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.

4. The student model

4.1. 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 will use the following seven values:

−3 *NON-MASTERY DEMONSTRATED on a test.* The student has demonstrated that she/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 she/he used the skill incorrectly in either the ReGIS laboratory or the directed exercises.

−1 *NON-MASTERY ASSUMED due to incorrect usage of a prerequisite skill.* The student is assumed not to possess the skill because she/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.

0 *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 she/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 she/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 0 when the student registers. As she/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 discussed in section 3, 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.

4.2. 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 will govern the speed, depth, and amount of repetition and reinforcement in initial presentations of new material. Fast students will receive fast presentations extending to considerable depth before going into the directed exercises as reinforcers. Slower students will be presented with more detailed introductions to new material at lower levels, and will find more repetition in the presentations as well as more frequent reinforcement via the directed exercises. Learning rate might be expressed as one of five values: VERY-FAST, FAST, AVERAGE, SLOW, and 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 finese is also feasible: simply ask students which style they prefer. Students will be allowed to change their learning style preference as the course progresses, as well as override the default selection for any particiar module. The system might monitor the number of overrides and change the default learning style when this number becomes significant.

5. 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 re-entering a module she/he has already studied and done *well* on, query him or her as to what skills she/he wishes to study and in what learning style. (The response to these queries will be recorded in the student history as "choices.")
3. If the student is re-entering a module she/he has already studied but done *poorly* 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 will force more repetition and reinforcement.)
4. If the student demonstrates non-mastery on a specific skill and the student model indicates that there are *no* prerequisites for that skill on which mastery has *not* 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. (Secondary teaching operations are ones that are designed for remediation only, no initial presentation.) 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* prerequisite for that skill on which mastery has not been demonstrated or assumed, apply Rule 1 to the module containing

the prerequisite she/he is "most likely" lacking. (The system will determine which skill is "most likely" lacking by analyzing the student model values for other skills with the same prerequisites.)

5.1. REPRESENTATION OF LEFT-HAND SIDES

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

```
1. ((STATUS-OF-MODULE-CHOSEN EQUAL 0)
     (ALL-SUBSKILL-STATUSES EQUAL 0)).
2. ((STATUS-OF-MODULE-CHOSEN (NOT EQUAL) 0)
     (AVERAGE-SUBSKILL-STATUS GREATER-THAN 0)).
3. ((STATUS-OF-MODULE-CHOSEN (NOT EQUAL) 0)
     (AVERAGE-SUBSKILL-STATUS LESS-THAN 0)).
```

The function calls (STATUS-OF-MODULE-CHOSEN EQUAL 0) and (STATUS-OF-MODULE-CHOSEN (NOT EQUAL) 0) would return TRUE if the status of the module chosen is equal to or not equal to 0, respectively. The function call (ALL-SUBSKILL-STATUSES EQUAL 0) would operate on sets of skills, and return TRUE if each of those skills has a status value equal to 0. Likewise, the function calls (AVERAGE-SUBSKILL-STATUS GREATER-THAN 0) and (AVERAGE-SUBSKILL-STATUS LESS-THAN 0) would return TRUE if the average subskill status value is greater than or less than 0, 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 0)
     (NO-PREREQ-SKILL-STATUSES LESS-THAN 0)).
5. ((SKILL-STATUS LESS-THAN 0)
     (ONLY-ONE-PREREQ-SKILL-STATUS LESS-THAN 0)).
6. ((SKILL-STATUS LESS-THAN 0)
     (MORE-THAN-ONE-PREREQ-SKILL-STATUS LESS-THAN 0)).
```

5.2. REPRESENTATION OF RIGHT-HAND SIDES

The right-hand sides (RHSs) of the rules (the THEN parts) can 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.
- 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:

```
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) 0)
           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 asterisk is a wild card that matches any value of the function argument list in the corresponding position.

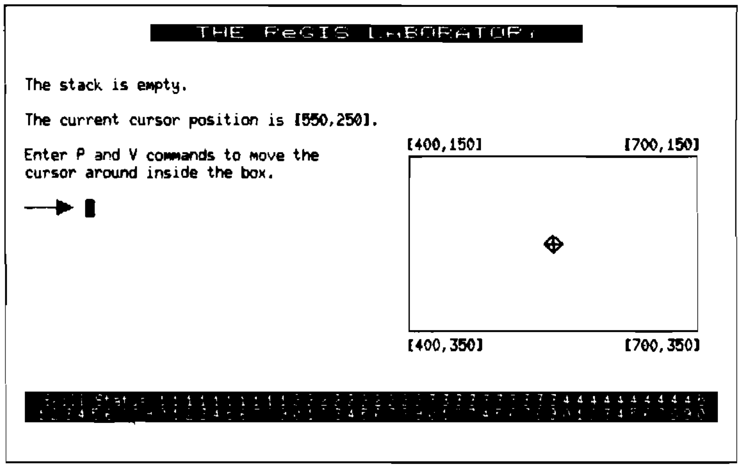
The RHS for Rule 4 has two TEACH functions to cover the case in which no secondary teaching operations exist for a specific skill. This representation makes the rules more complex, but provides explicit definition of what to do if a TEACH function request cannot be filled. Another way to tackle this 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.

6. A prototype ReGIS laboratory

We have implemented a prototype of the ReGIS Laboratory to test some of the concepts presented in this paper and to demonstrate the use of these techniques in a graphics domain. This section describes selected reproductions of sample screens from an actual run of the prototype program.

6.1. SCREEN LAYOUT

Figure 3 shows the basic screen layout. The screen includes a number of functional areas (Heines, 1984), but the main feature is that ReGIS code entered by the student



```
THE ReGIS LABORATORY

The stack is empty.

The current cursor position is [550,250].

Enter P and V commands to move the
cursor around inside the box.
```

FIG. 3. The initial state of the ReGIS laboratory for new students. Note that no values have yet been asigned to the student model.

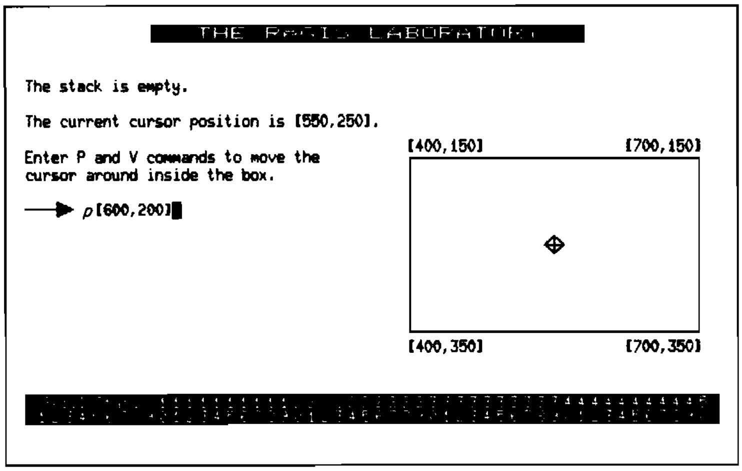
appears on the left-hand side and graphical output for correct ReGIS commands appears on the right. Other functional areas include messages informing the student of the contents of the stack and the current cursor position. The area reserved for displaying the results of entered ReGIS commands is of course only a portion of the screen's addressable area, but students are requested only to enter commands that keep the cursor within this area. Commands that move the cursor outside the reserved area are not executed.

The Skill Status display at the bottom of the screen is for demonstration purposes only. This display would not appear in a real version of the course, although it might be advisable to make it available to students if they so request. It has little meaning, however, without a clear understanding of the course's internal task representation, so it is debatable whether it should be made available to students at all. In any event, the display as it appears here is always shown in the demonstration software so that the system's student model building processes can be observed.

The value assigned to each skill (see section 5.1 above) is displayed below the skill number, but 0 values are suppressed. The sample student has just registered at the point shown in Fig. 3, so the value assigned to each skill is 0 and no values are displayed.

6.2. PROCESSING A SIMPLE STUDENT ENTRY

Figure 4 shows a simple student entry to the ReGIS Lab: *p*[600, 200]. This is a ReGIS Position command and should move the graphics cursor to the point with an X coordinate of 600 and a Y coordinate of 200.



```
THE REGIS LABORATORY

The stack is empty.

The current cursor position is [550,250].

Enter P and V commands to move the
cursor around inside the box.

——▶ p[600,200]█
```

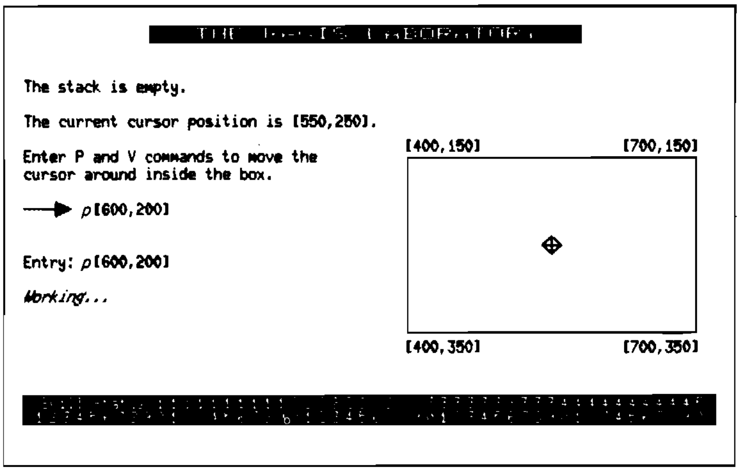
FIG. 4. A simple REGIS "position" command entry.

The system begins parsing the student's entry in Fig. 5. The entered string is repeated, and the system prints "Working..." to indicate that it has begun parsing. The parsing process is rather slow, especially on a loaded system, so the need for some type of "I'm busy" message was critical.

Figure 6 shows the state of the Laboratory after parsing of the student's entry is complete.

The display has changed from Fig. 5 in the following ways:

- The "Working..." message has been erased to indicate that parsing of the entire entry is complete.
- An arrowhead has been positioned under the character at which parsing stopped.
- "OK" has been printed under the arrowhead to indicate that no errors were found during parsing.



THE REGIS LABORATORY

The stack is empty.

The current cursor position is [550,250].

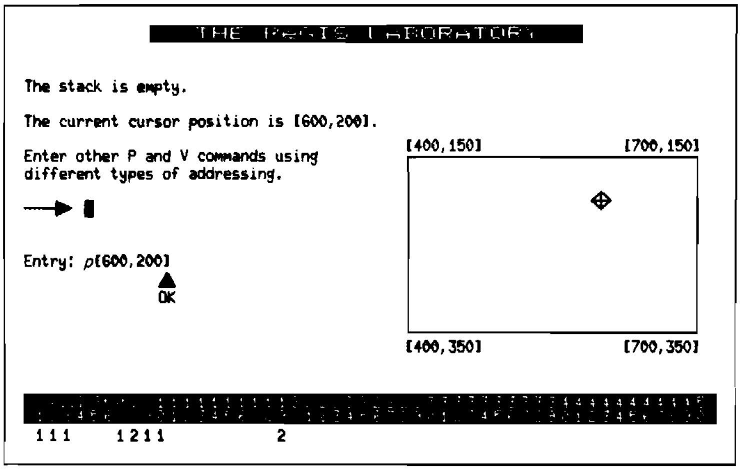
Enter P and V commands to move the cursor around inside the box.

→ p[600,200]

Entry: p[600,200]

Working...

FIG. 5. Parsing begins. The system repeats the command and prints "working . . . " to indicate that it has begun parsing.



THE REGIS LABORATORY

The stack is empty.

The current cursor position is [600,200].

Enter other P and V commands using different types of addressing.

→ ▌

Entry: p[600,200]
▲
OK

```
111    1211        2
```

FIG. 6. Parsing complete. The system displays an up arrow where parsing terminated, executes the student's command in the ReGIS window, renews the stack and position messages, updates the student model, prints "OK" to complete the problem, and displays new directions.

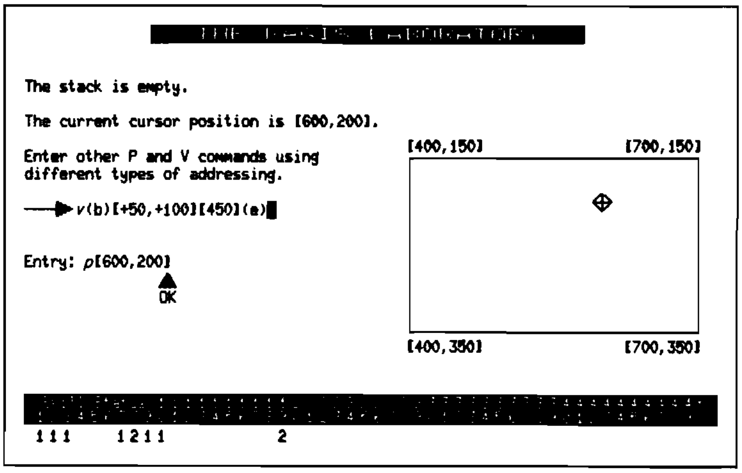
- The student's command has been executed in the graphic area at the right of the screen, moving the graphic cursor to position [600, 200].
- The current cursor position message at the top left of the screen has changed from [550, 250] to [600, 200].
- The wording in the directions has changed slightly since a command has already been processed.
- The student's previous entry has been erased from after the arrow prompt to make room for a new entry.
- The student model vector at the bottom of the screen has been updated.

Look more carefully at the student model vector at the bottom of the screen. A value of +2 has been assigned to Skills 8 and 19 because the student's entry demonstrates correct use of these two skills. A value of +1 has been assigned to Skills 1, 2, 3, 7, 9, and 10 because these skills are prerequisite to Skills 8 and 19.

These actions reflect the skill hierarchy shown in Fig. 2. When viewing these actions in context, however, it appears questionable that the system should make decisions concerning Skills 9 and 10 for this student entry. This situation clearly demonstrates the power and flexibility of the rule-based approach, because it indicates that the student model is in need of "fine tuning" to reflect the task analysis more accurately. Since all student model updates are goverened by production rules, the fine tuning can be easily accomplished without requiring substantial reprogramming.

6.3. PROCESSING A COMPLEX STUDENT ENTRY

Figure 7 shows a complex student entry to the ReGIS Lab: *v*(b)[+50,+100] [450] (e).



```
[illegible]

The stack is empty.

The current cursor position is [600,200].

Enter other P and V commands using        [400,150]           [700,150]
different types of addressing.

——▶ v(b)[+50,+100][450](e)█


Entry: p[600,200]
          ▲
          OK
                                          [400,350]           [700,350]

 1 1 1     1 2 1 1          2
```

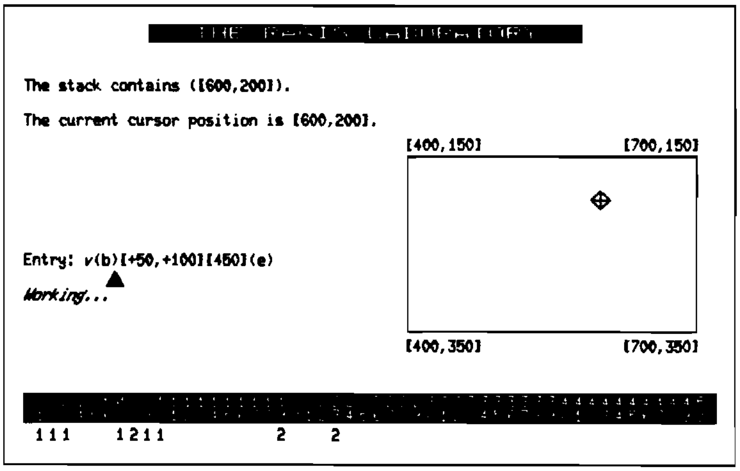
FIG. 7. A complex REGIS "vector" command entry.

This is a four-part ReGIS Vector command:

(b)	pushes the current cursor position onto the stack.
[+50, +100]	draws a vector from the current cursor position to the position 50 pixels to the right and 100 pixels down.
[450]	draws a vector from the current cursor position to the position with an X coordinate of 450 and a Y coordinate the same as that of the current cursor position.
(e)	pops an address off the stack and draws a vector from the current cursor position to that address.

Parsing begins when the student presses RETURN, and Fig. 8 shows the screen after the first component has been parsed:

- The student's entry has been copied and the original entry and directions erased.
- The "Working..." message has been printed to indicate that parsing is in progress.



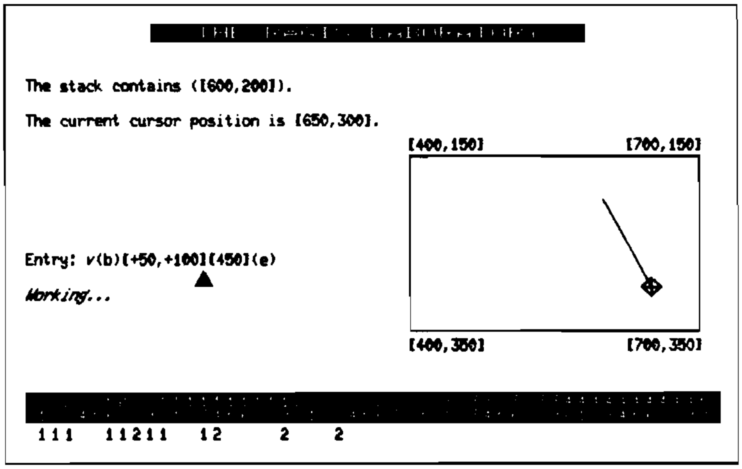
```
The stack contains ([600,200]).
The current cursor position is [600,200].

Entry: v(b)[+50,+100][450](e)
         ▲
Working...

 111   1211      2  2
```

FIG. 8. Stack push. The system displays an up arrow where parsing terminated, renews the stack and position messages, updates the student model, and continues parsing.

- An arrowhead has been positioned under the character to which parsing proceeded thus far.
- The stack contents message at the top left of the screen has updated to indicate that [600, 200] has been pushed onto the stack.
- A value of +2 has been assigned to Skill 23 ("can store addresses on the stack with (B)") at the bottom of the screen.

Parsing continues in Fig. 9:

- The arrowhead has been moved over to indicate that parsing has proceeded through the second address in the command.

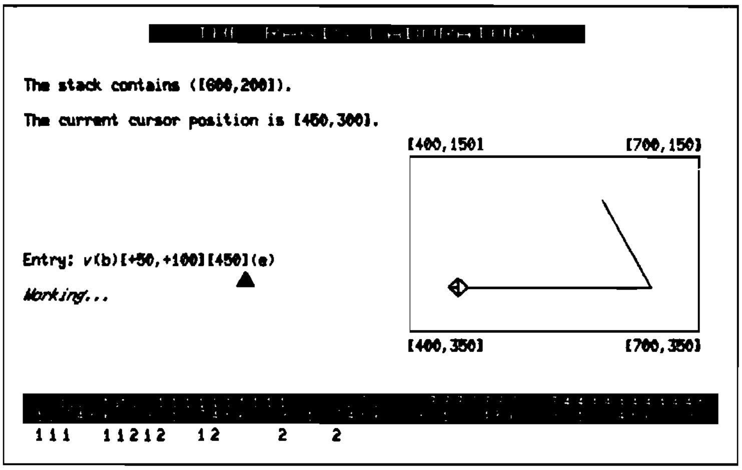


```
The stack contains ([600,200]).
The current cursor position is [650,300].

Entry: v(b)[+50,+100][450](e)
                    ▲
Working...

 111  11211  12    2  2
```

FIG. 9. Relative X and Y addressing. The system displays an up arrow where parsing terminated, executes the student's command in the REGIS window, renews the stack and position meessages, updates the student model, and continues parsing.

- The results of executing the command so far are shown in the graphic area at the right of the screen, drawing a vector from the current cursor position ([600, 200]) to a position 50 pixels to the right and 100 pixels down ([650, 300]).
- The current cursor position message at the top left of the screen has been updated.
- A value of +2 has been assigned to Skill 14 ("given the current cursor position as [*xc*, *yc*], knows the meaning of [$\pm x$, $\pm y$]$\rightarrow$[$xc\pm x$, $yc+y$]") and a value of +1 has been assigned to Skill 13 ("understands relative addresses").

Parsing continues in Fig. 10:



```
The stack contains ([600,200]).

The current cursor position is [450,300].

Entry: v(b)[+50,+100][450](e)
                            ▲
Working...
```

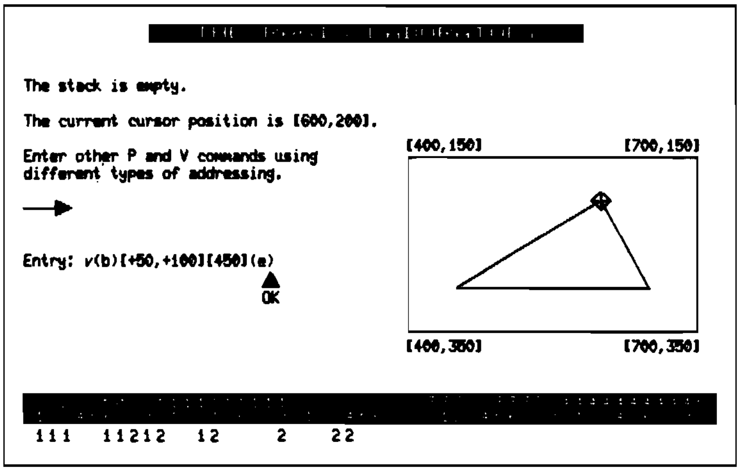
FIG. 10. Absolute X and default Y addressing. The system displays an up arrow where parsing terminated, executes the student's command in the REGIS window, renews the stack and position messages, updates the student model, and continues parsing.

- The arrowhead has been moved over to indicate that parsing has proceeded through the third address in the command.
- The results of executing the third part of the command are shown in the graphic area, drawing a vector from the current cursor position ([650, 300]) to the position with an X coordinate of 450 and a Y coordinate the same as that of the current cursor position ([450, 300]).
- The current cursor position message at the top left of the screen has been updated.
- A value of +2 has been assigned to Skill 10 ("given the current cursor position as [*xc*, *yc*], knows the meaning of [*x*]$\rightarrow$[*x*, *yc*]").

  Note the difference between assigning a value of +2 to Skill 10 here versus assigning a value of +1 to Skill 10 in Fig. 5. Here defaults are used explicitly, yielding greater confidence that the student knows how to use defaults. In the previous example, a value of +1 was assigned because the student used a higher level skill for which Skill 10 is a prerequisite.

Parsing continues in Fig. 11:

- The arrowhead has been moved over to indicate that parsing has proceeded through the fourth address in the command.



```
The stack is empty.

The current cursor position is [600,200].

Enter other P and V commands using
different types of addressing.

Entry: v(b)[+50,+100][450](e)
                              ▲
                              OK
```

FIG. 11. Stack pop. The system displays an up arrow where parsing terminated, executes the student's command in the REGIS window, renews the stack and position messages, updates the student model, prints "ok" to complete the problem, and displays new directions.

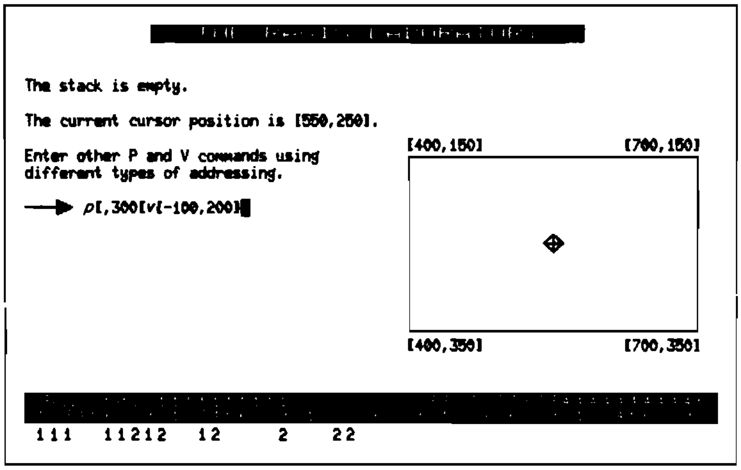
- The results of executing the fourth part of the command are shown in the graphic area, drawing a vector from the current cursor position ([450, 300]) to the position popped off the top of the stack ([600, 200]).
- The stack contents message at the top left of the screen has updated to indicate that [600, 200] has been popped off the stack.
- The current cursor position message at the top left of the screen has been updated.
- New directions are printed to indicate that the system is ready for the next student entry.
- A value of +2 has been assigned to Skill 24 ("can pop addresses off the stack with (E)") at the bottom of the screen.

6.4. PROCESSING AN INCORRECT STUDENT ENTRY

The screen has been cleared and the student now enters the incorrect entry shown in Fig. 12. The problem with this entry is that the "["in position 7 of the entry should be a"]".

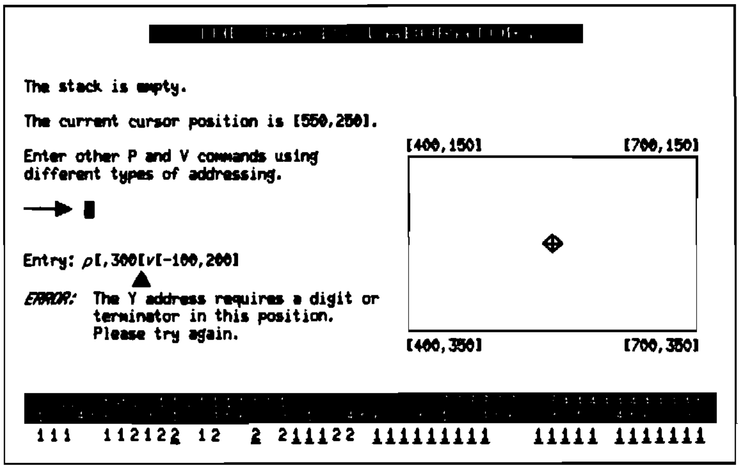
The error is detected in Fig. 13:

- The arrowhead indicates the point at which the error was found and parsing stopped.
- The word "ERROR:" is displayed to indicate that an error has been detected, and an explanatory error message is printed.
- The student's original entry is erased to make room for him to enter a new command.
- A value of −2 is assigned to Skill 11 ("given the current cursor position as [xc, yc], knows the meaning of [, y]→[xc, y]") and Skill 17 ("given the current cursor position as [xc, yc], knows the meaning of [±x, y]→[$xc+x$, y]") because the student's entry indicates incorrect usage of these skills.



```
The stack is empty.

The current cursor position is [550,250].

Enter other P and V commands using
different types of addressing.

——▶ p[,300[v[-100,200]

[400,150]      [700,150]

[400,350]      [700,350]

111  11212  12    2   22
```

FIG. 12. A combination "position" and "vector" command that includes an error. The "[" in positin 7 should be a "]".

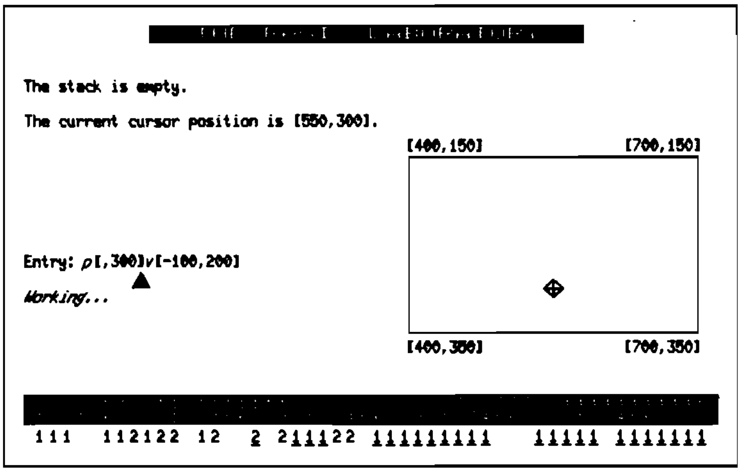


```
The stack is empty.

The current cursor position is [550,250].

Enter other P and V commands using
different types of addressing.

——▶

Entry: p[,300[v[-100,200]
             ▲
ERROR:  The Y address requires a digit or
        terminator in this position.
        Please try again.

[400,150]      [700,150]

[400,350]      [700,350]

111  112122 12   2 211122 111111111   11111 1111111
```

FIG. 13. Error detected. The system displays an up arrow where the error occurred, prints an error message, updates the student model, and displays new directions. Underlined values in the student model are negative.

- A value of −1 is assigned to all skills that require either of these skills as prerequisites. (Negative skill values are indicated in the Skill Status display by underlining due to space limitations.)

The student re-enters the command correctly. Figure 14 shows the state of the screen after the first command component is parsed. In addition to the overall screen updates identified for previous Lab displays, note that the value of Skill 11 has been changed from −2 to +2, but that the −1 values assigned to all postrequisite skills remain unchanged.

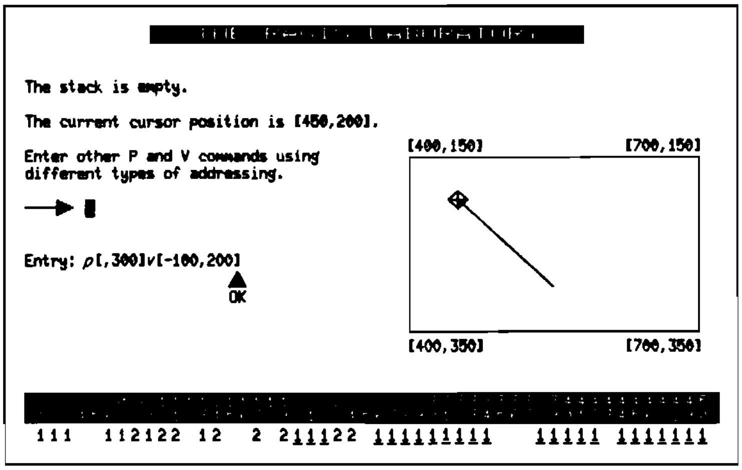


```
The stack is empty.

The current cursor position is [550,300].
                                          [400,150]          [700,150]

Entry: p[,300]v[-100,200]
           ▲
Working...
                                          [400,350]          [700,350]

111  112122  12  2  211122  111111111   11111  1111111
```

FIG. 14. Default X and absolute Y addressing. The system displays an up arrow where parsing terminated, executes the student's command in the REGIS window, renews the stack and position messages, updates the student model, and continues parsing. Note that the value of skill 17 has been changed from −2 to +2.

Figure 15 shows the state of the screen after the second command component is parsed. Note that the value of Skill 17 has been changed from −2 to +2.

This two-part command reverses *part* of the damage done by the incorrect command entry in Fig. 12, but all of the −1 values assigned to postrequisite skills remain. This feature allows the student model to "zero in" on the student's precise skill level. As before, all of these actions are governed by easily changed rules, so fine tuning the courseware to reflect the "real" skill hierarchy more accurately is not difficult.



```
The stack is empty.

The current cursor position is [450,200].
                                          [400,150]          [700,150]
Enter other P and V commands using
different types of addressing.

──▶ ▮

Entry: p[,300]v[-100,200]
                        ▲
                        OK
                                          [400,350]          [700,350]

111  112122  12  2  211122  111111111   11111  1111111
```

FIG. 15. Relative X and default Y addressing. The system displays an up arrow where parsing terminatd, executes the student's command in the REGIS window, renews the stack and position messages, updates the student model, prints "ok" to complete the problem, and displays new directions. Note that the value of skill 17 has been changed from −2 to +2.

6.5. IMPLEMENTATION NOTES

Implementation of the prototype in MacLISP on a DECsystem-20 showed that while the production rule formalism is highly efficient for expressing the prerequisite relationships, 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 was solved by computing all of the prerequisites and postrequisites when the course is installed and storing these as lists in a simple array. This approach allowed the student model to be updated much more quickly without sacrificing the elegance of the rule-based strategy.

The prototype also showed that the production rule approach can be applied efficiently to interpreting student responses. For example, the ReGIS command parser returned specific patterns related to the skills in the task model. For the ReGIS command "p[250, 100]", the parser returned:

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

This result was 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 was a perfect subset of the result returned by the parser, the program updated 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 can 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.

7. Critique and conclusions

The rule-based tutorial described in this paper has not undergone the test of full 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 will probably have to change to ensure computational efficiency and to maintain reasonable response time in the particular interactive computer environment adopted.

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 the 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.

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.

References cited and related readings

BURTON, R. R. & J. S. BROWN (1982). *An Investigation of Computer Coaching for Informal Learning Activities.* In Sleeman, D. & Brown, J. S. Ed., *Intelligent Tutoring Systems*, pp. 79-98. New York: Academic Press.

HARLEY, J. R. (1973). The design and evaluation of an adaptive teaching system. *International Journal of Man-Machine Studies*, **5**(2).

HEINES, J. M. (1983). Basic concepts in knowledge-based systems. *Machine-Mediated Learning*, **1**(1), 65-96.

HEINES, J. M. (1984). *Screen Design Strategies for Computer Assisted Instruction.* MA.: Digital Press, Bedford.

MAGER, R. F. & PIPE, P. (1974). *Criterion-Referenced Instructional: Analysis, Design, and Implementation.* Los Altos Hills, CA: Mager Associates.

O'SHEA, T. (1979). *Self-Improving Teaching Systems.* Ph.D. Dissertation, University of Leeds. Basel, Boston, Stuttgart: Birkhauser.

O'SHEA, T., BORNAT, R., DU BOULAY, B., EISENSTADT, M. & PAGE, I. (1983). Tools for designing intelligent computer tutors. In A. Elithorn & R. Banerjii, Ed., *Human and Artificial Intelligence.* London: North-Holland.

SELF, J. (1974). Student models in computer-aided instruction. *International Journal of Man-Machine Studies*, **6**, 261-276.

SHRAGER, J. & FININ, T. (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.