An Intelligent Ruletutor CBI System for Diagnostic Testing and Instruction

JOSEPH M. SCANDURA, DON C. STONE, and ALICE B. SCANDURA

University of Pennsylvania

High level authoring languages are often used for preparing course material for computer-assisted instruction (CAI), but authoring effective CAI requires considerable expertise in computer programming, the subject matter and instructional design. This combination of skills is not easy to find in one individual. Recently, attention has been focused on author/driver systems. The author part of such a system allows a subject matter specialist to prepare course material without having to program; using it is generally somewhat like using a word processor. The driver part of the author/driver system treats the entered course material as data, presenting it in a sequence determined by the material itself and student inputs. However, presently available author/driver systems lack serious use of available knowledge concerning cognitive processes and instructional systems. Hence this article addresses the question: can a conceptual approach which uses available knowledge of cognitive processes provide an explicit basis for engineering current and future intelligent CAI systems?

To answer this question, we:

1. review a conceptual framework in structural learning and show how it provides an explicit basis for engineering intelligent CAI systems;

2. detail a design for an intelligent "RuleTutor" system based on these conceptualizations; this RuleTutor is a driver designed to teach rules, and thus the subject matter data which is authored for it will contain a representation of the three components of a rule (in structural learning theory): a domain, a range, and a procedure;
3. describe a concrete implementation of the RuleTutor system in whole number arithmetic; this implementation includes diagnostic pretesting, instruction, mastery testing, and retention post-testing;  
4. discuss the strengths and limitations of this implementation and suggest ways in which the RuleTutor design can be enhanced; the major advantages of the RuleTutor are that it functions effectively and efficiently in testing and instruction and that it requires only the rule representation as input; it does not, for example, require an author to anticipate all answers that students might give or to work out branching for various responses.

INTRODUCTION AND STATEMENT OF THE PROBLEM

As increasingly reliable and low cost microcomputer systems become available, a major impediment to effective use of computer-aided instruction (CAI) in elementary and secondary schools is the scarcity of good software and good courseware — good CAI systems that meet demonstrable and recognized educational needs. Developing effective and efficient CAI, however, is not an easy or an inexpensive task. It requires considerable knowledge of computers or programming, intimate familiarity with the subject matter involved, and knowledge about how to present the subject matter effectively to students. It also requires familiarity with the educational context in which the software is to be used — not to mention the need for an adequate distribution system for getting software into schools.

Thus, simply knowing the subject matter, having experience in teaching it and knowing how to program is not generally sufficient for developing good CAI. Because of the many decisions which must be made in implementing any computer program, explicit conceptual guidelines can be very helpful. During the past two decades a considerable amount in this regard has been learned both about cognitive processes and about how to design effective and efficient instruction based on such knowledge. To date, however, this knowledge has had little impact on CAI development.

To be sure, experience and techniques developed in preparing a CAI system for one body of content make it easier to develop new systems. But, even under the best of conditions, CAI development, if it is to be educationally sound, is a difficult and time-consuming, for example, than writing a successful textbook.

To aid the process, considerable attention has been given to the development of general purpose CAI-oriented programming languages and authoring systems. Among the more widely used CAI programming languages have been Coursewriter, TUTOR, PILOT and MUMPS. These are relatively easy to learn and have been designed to facilitate the author's preparation of course and test material and its subsequent presentation to students who interact with the system. While designed to be general purpose, these languages are nonetheless better suited for CAI development in some areas than in others. In effect, although high level CAI authoring languages facilitate CAI authoring, the process still requires considerable familiarity with computer programming, with subject matter and with instructional design — a combination of knowledge and skills not easy to find in individuals. This fact almost certainly has led as much as anything to the uneven quality of current CAI systems.

In order to ameliorate this problem, a growing number of CAI specialists have pursued the course of developing authoring languages in combination with general purpose "drivers". The authoring systems allow authors to prepare the course content using English and possibly some easily learned codes. This material is usually entered into the authoring system in response to specific prompts; the use of such a system by a course author is often similar to the use of a word processor. Treating the resulting coded course material as data, the driver program, in turn, operates on the coded material, presenting it in a sequence determined by the material itself and the student inputs (e.g., responses to questions).

In order for authoring/driver systems to function properly, various restrictions must be placed on the data (course material). It is necessary, for example, to put some limit on the range of student responses that the driver can analyze appropriately. These restrictions typically limit either the variety of subject matter that can successfully be implemented and/or the instructional effectiveness of the implementation (i.e., while most subject matter can be "forced" into a given format, the resulting instruction may be less than optimal).

This concept of authoring/driver system has promise in the sense that it minimizes demands on authors and allows for the development of quality, yet cost-effective CAI for microcomputers.
Nonetheless, all authoring/driver systems that we are aware of have a major limitation: they do not make serious use of presently available knowledge concerning cognitive processes and instructional systems — knowledge held by the instructional design community but not as yet made widely available to the CAI field. This limitation is probably responsible, for example, for Bork's (1984) criticism of authoring systems.

The purpose of the research discussed herein is to introduce a conceptual approach to CAI development which may help to alleviate some of the aforementioned problems. More specifically, we shall:

(a) review a conceptual framework in structural learning and show how it provides an explicit basis for engineering intelligent CAI systems (Chapter 2);
(b) detail a design for an intelligent "RuleTutor" based on these conceptualizations (Chapter 3);
(c) describe a concrete implementation of the RuleTutor in the context of whole number arithmetic (Chapter 4); and finally,
(d) determine the strengths and limitations of this implementation and suggest ways in which the RuleTutor design can be enhanced (Chapter 5).

To state the problem concisely: can a conceptual approach which uses available knowledge of cognitive processes provide an explicit basis for designing current and future intelligent CAI systems?

THEORETICAL BACKGROUND AND REVIEW OF THE LITERATURE

Over the past 15 years, structural learning theories have been used in a wide variety of contexts, ranging from basic behavioral research involving complex content to the design of course materials (e.g., the critical reading workbooks of Scandura, Lowerre, and Scandura, 1973, 1978, discussed in Scandura, 1972a, 1972b, and Ehrenpreis and Scandura, 1974) and instructional systems. One clear implication of this research is recognition of the importance for instruction of prior analysis of content, all the more so in the case of computer implementations.

To utilize structural learning principles in designing instruction, therefore, the essential first step is to identify: (1) the educational goals — what the learner is to be able to do after instruction and (2) prototypic cognitive processes or rules — what the learner must learn if he is to successfully perform tasks associated with the educational goals.

Consider column subtraction problems as a simple prototype. In this case assume the educational goal is to find the difference. The prototypic rule or cognitive process in this case refers essentially to what the learner must master in order to subtract numbers.

A number of approaches have been used in the cognitive sciences for representing knowledge. Some formulations take the declarative approach, emphasizing facts about objects and events (and their relationships), while other formulations have a procedural orientation. Some knowledge representation schemes of current importance are production systems (Newell & Simon, 1972; Waterman & Hayes-Roth, 1978), semantic nets (Findler, 1979), and frames (Minsky, 1975). (A competent survey of knowledge representation appears in volume I of Barr and Feigenbaum, 1981.)

The present research is based on what Scandura (e.g., 1970, 1977) has called a “rule.” The rule construct plays a central role in the class of structural learning theories (e.g., Scandura, 1973, 1977, 1980), which was specifically formulated with the requirements of (i.e., the constraints imposed by) instructional systems in mind and, hence, represents a viable candidate for present purposes. The rule in structural learning theories combines the declarative and procedural approaches; it consists of a hierarchical (top-down) representation of both a procedure and its domain and range. Formally, a rule is defined (e.g., in Scandura, 1982) as a triple whose components are a domain, range, and a restricted type of procedure. The domain specifies the set of elements to which the rule can be applied; the range specifies the set of possible results of applying the rule; and the procedure is a specification of the algorithm. (Since each of these elements is itself represented as an ordered set, we have a hierarchical representation of arbitrary depth.) The procedure part of a rule is restricted by not permitting it to both generate and apply a new (sub)rule; a separate control mechanism is responsible in structural learning theories for testing a rule for applicability (by comparing its domain with the problem to be solved) and then executing the rule if appropriate.

"The structural learning theory" provides a general method of analysis, called Structural Analysis, by which the rules to be learned can be derived from suitably operationalized educational goals (e.g.,
Ehrenpreis and Scandura, 1974; Scandura, Durnin and Wulfek, 1974; and Scandura and Durnin, 1977). While there are many details still to be completely objectified, the method is relatively systematic and has been applied successfully in analyzing a wide variety of content. Qualitative advances in this direction have recently been proposed by Scandura (1984a, 1984b).

According to Scandura (1982), the first step in structural analysis involves selecting a representative sample of problems associated with the goal in question. This sample should be representative in the sense that a person's being able to solve them would suggest (strongly) can ability to solve arbitrary other problems associated with the goal. In the case of simple subtraction, this might include problems such as:

\[
\begin{array}{cccc}
9 & 879 & 432 & 402 \\
-5 & -325 & -129 & -129 \\
? & ??? & ??? & ???
\end{array}
\]

The second step in structural analysis involves identifying rules which make it possible to solve each of the selected problems. Identifying such rules involves several identifiable substeps:

(a) Assumptions must be made regarding the minimal encoding and decoding capabilities of the students in the target population. In the case of second graders, for example, the teacher/analyst would normally assume that all students are able to distinguish "the minus sign", the individual digits 0, 1, ..., 9, the columns, the rows, and that all are able to write the individual digits in desired locations. The present illustration builds on this assumption. Consequently, the remainder of the analysis will be inadequate just to the extent that these assumptions are in error for students in any given target population.

(b) The analyst must decide the scope of each of the representative problems. This scope effectively defines the domain of the rule associated with the prototype. The problem,

\[
\begin{array}{cc}
432 & \\
-129 & \\
?? &
\end{array}
\]

for example, might be held prototypic of the entire class of column subtraction problems, namely those formed by varying the individual digits 0, 1, ..., 9 and/or the number of columns. Indeed, in the present case, each of the selected representative problems is prototypic of this same domain. Consequently, in the present case, it is reasonable to assume that there is only one domain, the domain of column subtraction problems (with non-negative results).

(c) Next, the analyst must identify the steps (operations and decisions) involved in solving each of the representative problems. These operations and decisions must be sufficiently simple that using them refers only to abilities that are assumed available to all students in the target population (i.e., encoding/decoding capabilities). The operations and decisions also must be atomic in the sense that, for each student in the target population, the ability to correctly use an operation once is indicative of uniform success and, conversely for failure.

The flow diagram in Figure 1 depicts the procedural portion of a column subtraction rule. In this rule, it is implicitly assumed that each operation acts only on digits, rows and columns - consequently, the previously referred to need to assume certain minimal encoding/decoding abilities. The decisions of this procedure (e.g., Is top digit >= bottom digit?) constitute additional assumptions concerning minimal cognitive ability. According to structural learning principles only to the extent that these assumptions are met will this subtraction rule provide a useful and operationally precise basis for designing efficient and effective instructional strategies.

Most manipulations of concrete objects, language arts, and grammar (e.g., adding "ing" to verbs) are areas which readily lend themselves to structural analysis. Ordinary arithmetic algorithms certainly fall into this group. Relatively complete algorithmic (rule-based) analyses of geometry construction problems (Scandura, Durnin, & Wulfek, 1974; Wulfek & Scandura, 1977), algebraic proofs (Scandura & Durnin, 1977), geometry proofs (Landa, 1976; Greeno, 1978), use of microscope (Reigeluth, in press), and even Piagetian conservation (Scandura and Scandura, 1980) also have been published. Included in these analyses have been rule representations of what have been called higher order rules or general cognitive strategies. Analyses along these lines have even been attempted in such areas as the theory of suppositional proof or natural deduction (Corcoran, 1976), the foundations of mathematics.
(Scandura, 1973), and the use of verbal rules in problem solving (Chaiklin, 1984).

More could be said about the actual processes by which such rules are identified but this is the topic of active inquiry (e.g., Scandura, 1982, 1984a, 1984b) and going more deeply into it here would detract from the main concerns. The essential thing to emphasize is that the use of structural learning theories for purposes of designing instruction necessarily begins with a rule-based analysis of the subject matter (broadly conceived) in question. The previous examples deal with task domains (e.g., column subtraction) solvable via single rules, since the RuleTutor described in the next chapter is concerned solely with simple domains of this type. It should be noted, however, that structural analysis is a general process applicable to arbitrarily complex domains where the role of higher order rules becomes paramount (e.g., Scandura, Durnin and Wulfeck, 1974; Scandura, 1977, 1984).

Once an analysis has been completed, designing an effective instructional strategy follows directly and precisely from the theory. Specifically, once an analysis has been completed, one knows (a) what the student is to be able to do once he has achieved the educational objective (e.g., solve arbitrary column subtraction problems) and (b) what the student must learn in order to be able to do that (i.e., the column subtraction rule).

Given this information, the first thing to do in designing an effective instructional strategy is to determine what each student already knows. Specifically, which part of what the student knows is directly relevant to what the student is to learn? The process by which this is accomplished has been detailed in the literature (e.g., Scandura, 1971b, 1973, 1977, 1982) and will not be considered here. It is sufficient for present purposes to observe that solving a particular subtraction problem involves following one and only one path through the subtraction rule. In effect, there is a unique class of problems associated with each path through the rule. (Note: There are a finite number of paths associated with any given rule.)

A basic principle in structural learning theories is that rules must be represented in terms of operations and decisions that are atomic; they are either totally available or unavailable to any given learner in the target population (i.e., the population the analyst had in mind in performing the structural analysis). The existence of such a representation can always be guaranteed (e.g., Scandura, 1971c, 1973; Suppes, 1969). For example, if the operation of subtract column (using basic facts) were not atomic (which it probably would not be for a group of first graders), one might break down the operation into a more detailed subprocedure that explicitly represented each subtraction fact. In general, the less sophisticated a group of students, the more detail is required. Conversely, the more sophisticated the students, the larger the components may be. Hence, complexity of a rule representation is a relative notion, which does not depend solely on the apparent task complexity.

Collectively, the paths of the procedure partition the domain of column subtraction problems into types, or equivalence classes, or problems. For each path there is an associated type and vice versa. In effect, success or failure on any one problem associated with a class of problems provides complete information as to the availability to the student of the corresponding path. For example, the problem

\[
\begin{array}{c}
\text{Go to right-most column (Step 1)} \\
\text{WHILE more (full) columns to left} \\
\text{DO} \\
\text{IF top digit } \geq \text{ bottom digit} \\
\text{THEN} \\
\text{Subtract column (using basic facts) (Step 2)} \\
\text{ELSE} \\
\text{Remember starting column and go to next column to left (Step 3)} \\
\text{WHILE 0 is top digit} \\
\text{DO} \\
\text{Go to next column to left (Step 4)} \\
\text{REPEAT} \\
\text{Borrow 1 from current column and regroup in column to right (Step 5)} \\
\text{UNTIL column is starting column} \\
\text{Subtract column (Step 6)} \\
\text{Go to next column to left (Step 7)} \\
\text{Subtract column (using basic facts) (Step 8)} \\
\end{array}
\]

Figure 1. Procedure for column subtraction rule.
is solved with the algorithm of Figure 1 by following the path defined by the following sequence of steps: 1, 3, 5, 6, 7, 2, 7, 8.

By testing on a small, finite set of problems, it is possible to identify precisely and unambiguously which parts of the subtraction rule any given individual knows and which parts the student does not know. Such testing, in effect, defines the student's entering level: it determines which paths the student knows and which he or she does not know.

A significant amount of supportive data has been accumulated over the past two decades, with respect to a wide variety of mathematical tasks, with subjects ranging from preschoolers to Ph. D. candidates. Given a class of tasks, the general form of each study went as follows:

1. One or more rules were identified which were both adequate for generating solutions to each of the tasks and compatible with the way a knowledgeable or idealized member of the target population might be expected to solve them.

2. These rules singly and/or collectively applied were used to partition the class of tasks into equivalence classes.

3. Subjects in the target population were tested on two items (tasks) from each equivalence class (type of item).

4. Performance on an item from each equivalence class was used as a basis for predicting success or failure on the other (second) item.

With highly structured tasks run under carefully prescribed laboratory conditions, and given performance on initial items, it has been possible to predict performance on new (second) items with over 96% accuracy (Scandura, 1970, 1973; Scandura & Durnin, 1978). When testing took place under ordinary classroom conditions, with the subjects run as a group, the predictions were accurate in about 84% of the cases (Durnin & Scandura, 1973).

In the latter study, which involved subtraction directly, the equivalence classes determined via the underlying rules were compared with the item forms (types of items) identified by Hively, et al. (1968) and Ferguson (1969). Whereas the levels of prediction on success items were approximately the same, the rule based approach was both more reliable and more efficient (requiring only half as many test items). Moreover, there is a hierarchical relationship among the paths of any rule (with paths lower in the hierarchy being included in those at higher levels).

Such hierarchies provide a theoretically derived and empirically verified partial ordering of selected test items, according to difficulty. This difficulty hierarchy has been utilized to provide even more efficient assessment. Once a student has failed at a given level, for example, there is no need to test on more difficult tasks in the hierarchy. Conversely, there is no need to test on tasks at levels below where the student has already succeeded. In this regard, more can be said about such things as testing in situations where more than one rule is involved and about increasing efficiency via sequential testing (e.g., Scandura, 1971b, 1971c, 1973, 1977). An increasing amount of similar research is being generated by others and with similar, positive results. Thus, for example, Klahr (1978) has used similar assessment procedures to ascertain developmental level with respect to the balance beam task. Brown and Burton (1978) extended aspects of this work, first by including what Scandura (1977) has called "error" rules, or what Brown and Burton call "bugs"— incorrect ways of performing given tasks, and second by implementing the diagnostic method on their computer. Recent work by Tennyson (1984) and Park (1984) combines a diagnostic/adaptive structure based on Bayesian statistics within a microcomputer-based instructional system.

Prescribing instruction, then, follows directly from what the student knows. All one needs to do is to identify the missing portions of the desired subtraction rule and to present them to the student. The structural learning theory is neutral on whether this information should be presented, say, in an expository or a discovery manner. Thus, deciding on the appropriate method of presentation depends on secondary objectives that the instructional designer may have in mind (e.g., to help students learn how to detect regularities). The important question about subtraction performance is simply can the student follow the rule.

If a student's knowledge of a rule is complete except for one step, the computerized instructional system would need only to make sure that the student knows, at the appropriate point, how to carry out that step. Where the student knows less, of course, one would start with the simpler prototypes (partial rules representing what the student knows) and gradually "elaborate" (cf. Merrill, 1980, 1984), or add increasing detail until the student has mastered the entire rule.
To summarize, it must be emphasized that this illustration of prescriptive aspects of the structural learning theories constitutes only a simple prototype. It "epitomizes" the instructional aspects of the theory. The theoretical system itself provides a far more generalized basis for instructional prescription — which in principle, may be used with any subject matter (or educational goal) that might be of interest.

More generally, all theories which satisfy the requirements of Structural Learning Theories (e.g., Scandura, 1980), include two major components: (1) the cognitive procedures corresponding to the domain of discourse (e.g., what is to be learned) and represented as rules derived by the process of Structural Analysis and (2) the individuals (e.g., teacher and/or learner) participating in the discourse. (At this very high level, SLT's are similar to Pask's (1975) Conversation Theory.) In SLT's, however, individual knowledge is represented quite differently (e.g., rules are strictly modular) and explicit attention is given to basic psychological characteristics of the learner.

In their simplest form, SLT's may be represented schematically as shown in Figure 2. In this case, one individual is interacting with his environment (consisting of the problems associated with the individual’s goals and inherent in his environment). The goal-directed individual is viewed as attempting to solve “problems” that are presented to him (or to achieve desired results in given environmental situations) (down arrow). The individual’s responses (up arrow) are generated via his specific rules of available knowledge and the cognitive universals governing their use. Individual responses also provide potential information regarding the specific rules available to the individual. As described briefly above and more fully by, e.g., Scandura (1977), these individual rules are operationally defined in terms of the prototypic rules associated with the problem domain.

A more general form, shown in Figure 3, represents a dialogue between two (or more) individuals. In interpreting this schema, it should be noted that neither individual has perfect knowledge of the domain or perfect diagnostic and/or teaching knowledge. Hence, there is no a priori guarantee that a participant can either accurately assess what the other participant knows (or can do) or influence that participant in theoretically optimal ways. In general, these inferencing and influencing capabilities will be partial.
and can recognize and/or generate arbitrary problems in the Problem Domain. In addition, this idealized teacher has built into it all of the theoretically optimal machinery for diagnosing learner difficulties and for providing optimally efficient remediation.

GENERAL DESIGN PRINCIPLES FOR A RULETUTOR

From the previous discussion, we may conclude that any CAI tutorial system based on the Structural Learning Theory must as a minimum include: the content specific rules (procedures) to be taught and a general purpose RuleTutor (driver program) which takes the content specific rules and uses them within a general interactive, instructional test/management system. In Chapter 2, the theoretical rationale of structural analysis (by which the content specific rules/procedures are defined for a particular subject area) was explained and illustrated. In this chapter, a design based on those conceptualizations for an intelligent "RuleTutor" will be described.

An effective and practicable instructional system should have at least five distinct instructional capabilities as well as appropriate student performance data management capabilities. The instructional capabilities should include:

1. Diagnostic Pretest administration;
2. Prescriptive (adaptable) instructional sequencing based on pretest results;
3. Capability for incorporating additional instructional objectives such as "meaning," short cuts, verbalization of procedures, etc.;
4. Systematic Mastery Testing of simpler paths to confirm learning of basic procedures necessary for later instruction;
5. Efficient Post-test administration to check overall retention.

The management system should, at a minimum, record pretest, instructional and post-test results for individual students. Additional management capabilities might include: ability to override pretest prescriptions, assignment of an individual to any place in the sequence, adaptation of criteria for pretest, lesson and post-test progression, analysis of an individual's capabilities in areas such as "meaning," verbalization, "timed" practice, etc. meaning, "metacognition" (or verbal awareness of what one knows) and short-cuts commonly achieved by experts.

Since the RuleTutor is a driver program, it needs content for its completion. This content takes the form of software for generating problems (tasks) and procedures for solving these tasks. The RuleTutor can utilize these capabilities in deciding which problems to present during testing and which instruction to provide during training. The basic system is quite general and could in principle be used with arbitrary cognitive procedural tasks.

(Clearly, there are some domains of tasks which cannot be solved by any single rule. In practice, however, a wide variety of (even most) tasks that humans are expected to perform may be solved by means of such rules, and we have called such tasks "cognitive procedural tasks").

Let us describe in more detail how this RuleTutor functions. Given the content-specific information, the diagnostic testing portion of the system efficiently determines a student's entering level, as described above. More specifically, it stores a "checklist" for the current student (in the student records disk file) of the known and not-yet-known paths. This checklist is read and updated by the instructional portion of the system as it teaches the student in turn each of the not-yet-known paths.
The instruction on each path has a number of components or instructional levels. For example, the RuleTutor:

1. teaches the meaning of the process,
2. teaches the relationship between this meaning and the process itself,
3. teaches the process itself, providing help where necessary,
4. helps the student to verbalize the cognitive processes he or she has learned by having the student name the processes used or observed, and
5. helps the student to automate the process (once the rule has been learned), thereby increasing his degree of skill.

Within each of the above instructional components or levels, the RuleTutor system also can vary the difficulty of the material and can adapt to the student by increasing problem difficulty (or type) at a rate depending on the student’s prior learning efficiency. For students who are currently learning very efficiently the difficulty level will increase relatively rapidly, while the difficulty level will increase relatively slowly for students who are currently not learning as efficiently as they might. Furthermore, learners may skip some of the instructional levels mentioned above if warranted by their learning efficiency on previous paths in the domain.

The instructional portion of the RuleTutor system stores detailed information about a student’s performance in the student records disk file. This information can be accessed via the management portion of the system, and the parameters representing, for example, the rate at which problem difficulty is increased for a given student or the paths on which that student should be given instruction can be explicitly altered by an instructor.

In the following sections, we describe the RuleTutor design more explicitly. Anticipating the arithmetic implementation in Chapter 4, many of the ideas are illustrated in this context.

Diagnostic Pretests

Each Pretest simultaneously evaluates the student in two ways:
1. It determines which types of problems the student can and cannot solve.
2. Independently, it determines whether the student has adequately mastered the prerequisites (e.g., basic facts).

The problem types used in any particular implementation will be determined via structural analysis, which, as discussed in Chapter 2, will yield types which are homogeneous in the sense that all problems of given type require the same cognitive processes for solution.

For purposes of pretesting, these problem types are arranged in a partial ordering. For example, consider the following lattice:

```
  7
 / \      6
|   |      |
5   3 --> 4
|   |      |
2   1
```

Figure 5. Prototypic Testing Lattice

In this lattice the seven nodes refer to problem types. Notice that these nodes are arranged at five different levels. Type 7 is more difficult than Types 5 and 6, which are prerequisite to it. Conversely, Types 5 and 6 must be mastered before Type 7. On the other hand, the relative difficulties of Types 3 and 4, or for that matter Types 3 and 6 or 5 and 6, cannot be determined from the lattice.

In general, diagnostic pretesting begins at or near the middle level (here Level III). In our example, the student first would be tested at a relatively simple problem of Type 3 (i.e., on a problem involving the smallest numbers of digits for that type). If the student fails, the system would infer that the student would be unable to solve more complex problems (i.e., of Types 5 or 7) which are above Type 3 in the hierarchy; testing then would progress to Type 4 problems. If the student succeeds, the next problem presented would be the most complex of Type 3 (that can be displayed on the screen). Success here would cause the system to infer mastery not only of Type 3 problems but also of all prerequisites of Type 3 problems, specifically those of Types 2 and 1. Then, testing would progress to Type 4. Note that the pretest system may automatically infer capabilities (or lack thereof) on more than one type at a time. It is this capability which makes testing so efficient.
The above simple, then complex, problem pattern is used with all problem types that have not at any given point been determined to be mastered or not mastered.

Suppose, for example, that a student has mastered Type 3 and hence also Types 1 and 2, but that he or she fails Type 4 (implying also a lack of mastery on Types 5, 6, and 7). In this case as few as three problems are sufficient to determine mastery on all problem types and, hence, that the student should begin instruction at Type 4 before moving on to Types 5, 6, and 7.

If testing had indicated mastery of both Types 3 and 4, then testing would have continued at Level IV (Types 5 and 6). In general, testing will move to higher or lower levels in the hierarchy as determined by the student's performance — just until the system has been able to infer which types the student has and has not mastered.

During pretests (as well as lessons and post-tests) the type of problem, along with the number of numbers (e.g., rows in addition), the number of digits in each row and the time allowed for each response, can be determined directly from the screen. In the upper right hand corner of the screen, "T" for Type is set equal to the problem type number; "D" for Difficulty Level is set equal to the number of numbers (e.g., rows in an addition problem), followed by the number of digits in each number; and "S" for Speed is set equal to the allowed number of seconds for student responses. (On Lessons, "I" for Instruction-Level indicates the type of instruction.)

At the same time the system is assessing problem types, it also is determining how adequately the prerequisites (e.g. basic facts) have been mastered.

As soon as pretesting has been completed, the system displays for the student the types of problems, including prerequisites, on which instruction is needed.

Lessons

Lessons are organized according to Problem Types, Instruction Levels and Difficulty Levels (of problems).

Whole number subtraction problems, for example, may be partitioned into the seven problem types illustrated in the lattice of Figure 5:

1. Single Digit Facts
2. Facts From 10 to 19
3. No Regrouping
4. All Regrouping
5. Mixed Regrouping
6. Regrouping Across 0
7. Mixed Regrouping Across 0

There are up to nine Instruction Levels (I-Levels) for each Problem Type.

1 — Meaning
2 — Meaning/Rule Relationship
3 — Directions/Rule
4 — Rule-Remedial
5 — Describing Rule Processes
6 — Shortcut Instruction
7 — Shortcut (Mental) Calculation
8 — Timed Practice
9 — Mastery test

In general (e.g., unless the instructor decides otherwise or a given problem type does not require that I-level), instruction begins, first, with the MEANING (I = 1) of the process (or RULE) as applied to the given problem type.

Note that the term RULE refers to the basic cognitive skill under consideration — for example, the column subtraction algorithm, MEANING refers to a process corresponding to the RULE which relates more directly to its meaning — for example, in the case of subtraction the process is related to numbers in expanded notation (e.g., 346 = 300 + 40 + 6) which in turn is equivalent to the use of Dienes' blocks, sticks grouped by 1's, 10's, 100's, etc.

Instruction at the meaning level is extremely detailed. To minimize the amount of keyboard entry that is required of the learner, inessential parts of the required responses are generated by the computer automatically. For example, in expanded notation problems, the 0's are generated automatically as soon as the learner types the corresponding digit (e.g., the "3" in "300"). Various corrective feedback occurs at any incorrect input during the lessons.
1. **Meaning**
   Show all work.
   
   \[
   \begin{align*}
   200 + 20 + 8 & \quad \text{T=4, I=1} \\
   & \quad \text{D=3,2} \\
   \hline
   & \quad \text{90} - 9 \\
   \hline
   & \quad \text{228} \\
   & \quad \text{99} \\
   \end{align*}
   \]

   Now you do it, Name!
   At the second I-Level, the system visually shows the relationship between the meaningful process and the targeted RULE. Throughout, the student must interact with the system.

2. **Meaning/Rule**
   Show all work.
   
   \[
   \begin{align*}
   300 + 10 + 0 & \quad \text{T=4, I=2} \\
   & \quad \text{D=3,3} \\
   \hline
   & \quad \text{310} \\
   \hline
   & \quad \text{242} \\
   \end{align*}
   \]

   Third, the student is taken through the rule step by step. A description of the step is displayed on the screen and the student is required to carry it out.

3. **Directions/Rule**
   Show all work.
   
   \[
   \begin{align*}
   7717 & \quad \text{T=4, I=3} \\
   & \quad \text{D=4,4} \\
   \hline
   & \quad \text{3959} \\
   \end{align*}
   \]

   Fourth, the student must go through the process on his or her own for the first time. If errors are made the system automatically reviews the meaning of the process up to the point where the error was made.

4. **Rule/Remedial**
   Show all work.
   
   \[
   \begin{align*}
   82 & \quad \text{T=4, I=4} \\
   & \quad \text{D=2,1} \\
   \hline
   & \quad \text{5} \\
   \end{align*}
   \]

   Fifth, having mastered the RULE as a process, the student is next required to achieve a higher level of verbal awareness by describing steps which are shown to him or her.

5. **Describing Rule**
   Describe the step shown.
   
   \[
   \begin{align*}
   & \quad \text{T=4, I=5} \\
   & \quad \text{D=3,2} \\
   \hline
   & \quad \text{228} \\
   & \quad \text{99} \\
   \end{align*}
   \]

   \(A\) Move left
   \(B\) Subtract column
   \(C\) Start at right hand column
   At the sixth I-level, the student is taught to perform the rule mentally without going through the steps overtly. The student is shown the detailed steps on one display of the problem, up to just before each digit in the answer; the student must input the correct digit in the second display.

6. **Shortcut Instruction**
   Compute in your head.
   
   \[
   \begin{align*}
   7303 & \quad \text{T=4, I=6} \\
   & \quad \text{D=4,4} \\
   \hline
   & \quad \text{7413} \\
   \hline
   \text{3429} & \quad \text{3429} \\
   \hline
   & \quad \text{84} \\
   & \quad \text{84} \\
   \end{align*}
   \]

   Rename in your head, then subtract.
   Seventh, mental practice (i.e., drill and practice) is given and eighth, where the problem type is deemed especially important, further practice is provided under graded time pressures.

7. **Shortcut (Mental) Calculations**
   Compute in your head.
   
   \[
   \begin{align*}
   & \quad \text{T=4, I=7} \\
   & \quad \text{D=4,4} \\
   \hline
   & \quad \text{5150} \\
   \hline
   & \quad \text{2963} \\
   \end{align*}
   \]

8. **Timed Practice**
   Compute in your head.
   
   \[
   \begin{align*}
   & \quad \text{T=4, I=8} \\
   \end{align*}
   \]

   Try to beat the beep.
   
   \[
   \begin{align*}
   & \quad \text{D=4,4} \\
   \hline
   & \quad \text{S=5} \\
   \hline
   & \quad \text{4631} \\
   \hline
   & \quad \text{2695} \\
   \end{align*}
   \]
Students are signed off the system automatically as soon as they complete all (normally seven or eight) levels of instruction on a given problem type (except for prerequisites where only 1-level 8 is used). (This is to encourage students to start the corresponding Mastery Test when fresh.) The next time such a student signs on, he or she automatically will be assigned to a mastery test.

In general, Difficulty Level refers to the number of numbers (e.g., being added), the number of digits per number, and the times allowed for response. There is a range of possible difficulty levels and response times for each problem type/instruction level combination. (In a few cases, screen size prohibits the presentation of certain problem/type levels. When this happens, the system makes the needed adjustments automatically.)

Lesson instruction is individualized at three basically different levels. First, the pretest automatically determines the type of problems on which the learner should begin.

Second, the learner progresses through the material at his or her own rate. Mastery is required at each difficulty level, instruction level and problem type before advancing to the next level. Conversely, poor performance results in the learner's moving back levels. If a student fails a Mastery Test, then he or she is required to start over at Instruction Level 7, except for Prerequisite Types or Facts where all practice is timed (i.e., at Instruction Level 8). (All steps on a given problem must be correct during instruction in order for that problem to count toward advancement. Two or more errors on a problem count toward moving back; one error counts for neither but just results in another problem being presented.)

Third, and perhaps most subtly, the system automatically adjusts "Step-Size," either increasing it or decreasing it, according to how efficiently the learner has learned previous material.

Two types of step-size are used, one involving difficulty level and the other instructional level. The smallest Difficulty-Level-Step-Size (0) requires the learner to go through and master all difficulty levels associated with the given problem type/instruction (I-level) combination. If a learner masters a given 1-level efficiently (say by missing fewer than 10% of the questions asked during instruction), then he moves to the intermediate step-size (I). Here, the learner must first master the minimum difficulty level and then the maximum before progressing. The learner skips all intermediate levels. If a learner does poorly (say by missing more than 25% of the instructional questions), then, as you might expect, the step-size will be decreased — unless the learner is already at the smallest step-size.

At the highest Difficulty-Level-Step-Size (2), the learner only receives instruction at the highest difficulty level for each problem type/instruction level combination. Instruction in this case can progress quite rapidly.

There also are three levels of Instruction-Level-Step-Size. The smallest Instruction-Level-Step-Size (0) requires the learner to go through and master all instruction levels for each problem type. The intermediate size (1) allows the learner to skip Instruction Levels 1 (Meaning) and 3 (Directions/Rule) on the assumption that the missed material is readily inferred from Instruction Levels 2 (Meaning/Rule Relationship) and 4 (Rule Remedial), respectively. When at the largest step-size (2), instruction begins with levels 4 and 5 (Descriptive). Level 6 (Shortcut Instruction) also is skipped based on the assumption that fast learners will be able to make the necessary adjustments independently. As with difficulty levels, transitions from one Instruction-level step-size to another are determined by learning efficiency — this time based on performance on previous problem types.

Optional criteria for determining both difficulty and instruction level step sizes may be assigned by the instructor in Option 8 of the Management System. Option 5 further allows the teacher to override the other two individualization features and to assign students to whatever problem types and instruction levels might be desired.

**Mastery Tests**

Mastery Tests are to consist of a predetermined number of problem steps or questions (between 50 and 75 are recommended). The first and other odd numbered problems in each mastery test require the learner to perform the problems mentally (under timed or untimed conditions, depending on whether or not instruction normally includes timed Instruction Level 8 for the given problem type).

The even numbered problems on Mastery Tests are randomly selected from the following kinds:

Meaning Problems where the learner must perform all steps in the meaningful process;
Rule Problems where the learner must perform all steps in the rule;  
Descriptive Problems where the learner must choose the correct descriptions for given steps;  
Mental Problems where the learner must solve the problems mentally;  
Timed Problems where the learner must solve the problems mentally under time pressures;  
Review Problems where the learner must mentally solve review problems of the current problem type minus one. (If the review type is a prerequisite or other key problem type, the review problems are timed; otherwise not.)

No direct feedback is given during Mastery Tests. However, where answers depend on previous ones, as in working through problems step by step, the system will erase erroneous answers (after beeping) and require the learner to give the correct answer before allowing him or her to proceed. The system, of course, keeps a record of all responses, correct or otherwise.

Retention Post-Tests

Retention Post-tests are very much like pretests except that they are designed to provide more reliable estimates of performance on the various problem types. On post-tests, learners may be recycled through up to nine pretest cycles. Under ordinary test conditions, this could involve very large numbers of problems (e.g., $9 \times 16 = 144$ where there are eight problem types and two problems of each type). However, the efficiencies built into the testing system, as described in the pretest section, reduce this number to manageable proportions.

Students are required to complete a post-test at one sitting. The system is initially preset at five test cycles.

To summarize, instruction begins at the first instructional level available for the entering problem type specified by the learner’s performance on the pretest. As determined by learner responses, instruction progresses to increasingly higher difficulty and instruction levels or, conversely, back to remedial levels. In addition, difficulty and instruction level step-sizes are adjusted periodically depending on the efficiency of the student's prior learning.

Because of these adaptive features, it is almost impossible for a learner to work through the system without mastering the material. This is equally true of the beginning student and of the older student who needs remedial help.

To discourage unusually long stays at the computer, learners are automatically signed off the system whenever they demonstrate mastery of a new problem type (e.g., by passing a mastery test) or by doing so poorly that they are demoted to lower problem types. In this regard, it is worth noting that both progressions and demotions are governed by the content lattice (of problem types). On transitions between instruction levels, the learner is asked if he or she would like to continue. If the learner has additional time allocated, he or she may continue with the next level by pressing Y. (The learner is allowed to exit from the system at any time by pressing the ESC key.)

As a final step, after completing the Lessons/Mastery Testing the learner is asked to take the corresponding Retention Post-test.

TUTORIAL ARITHMETIC SYSTEM

The RuleTutor design described in Chapter 3 has been implemented on the Apple II microcomputer (using DOS 3.3, Applesoft BASIC and 6502 Assembler) by Scandura Training Systems, Inc. Specifics of this implementation, involving whole number arithmetic, are detailed in the following sections.

General Characteristics

Paralleling the preceding analysis and design principles, the RuleTutor has two modular main sections, supplemented by three supervisory or utility programs:

1. The first of the two main sections is the content specific code and data. As its name implies, there is a version of this for each content area. It includes: code for generating random problems of each problem type, code for displaying problem components in windows (called "slots") on the screen (and a file of data called a "template" which specifies the positional relationships between the slots), code for generating each solution step in the solution algorithm to be learned, a file of descriptions/verbal instructions for each step in the algorithm, and a lattice hierarchy of problem types along with an array of allowable instruction levels for each problem type. The content specific code was generated manually, although work is
currently being done on a system that would facilitate the authoring of such procedural material. All of the problem generators used until now involve random problem generation (subject, of course, to the constraints necessary to guarantee that a problem belongs to the desired problem type), but it would also be possible to use predetermined problems of essentially any sort — as might be desired, for example, in verbal or language instruction.

(2) The other main section is the common 'code', which is used with all content areas. This is essentially the driver program. It accesses the content specific code as necessary for diagnostic testing and instruction. During diagnostic testing, for example, it determines from the lattice the appropriate initial and subsequent problem types to be presented (in accordance with the theory described above). Then, it calls on the problem generator to generate a problem of the desired type and on formatting/display routines to display it on the screen. After that it gets student inputs and compares them with solution components generated by applying the content specific solution algorithm to the problem. Next, it draws inferences concerning other problem types in the lattice as described previously (i.e., types subordinate to a passed type are marked as passed; types superordinate to a failed type are marked as failed). Finally, the process is repeated until the pass/fail status of all problem types has been determined.

During instruction, the common code builds on student prescriptions determined during pretesting. Given the prescribed problem type, the common code determines the next allowable instruction (I) level for that type and calls on the problem generator and display routines. Then it executes the steps of the solution algorithm one by one, providing for each step the type of instruction called for by the current I-Level, while maintaining various kinds of information concerning student responses.

(3) The common and content specific code are embedded in or called from general supervisory management code. The management system does such things as present title or introductory screens, obtain the student's name and access or update the student records. It also maintains teacher-modifiable system characteristics, such as for mastery, and it controls access to the RuleTutor and the records transfer and data access systems mentioned below.

(4) The records transfer program is a utility made necessary by the fact that pretests, instruction and post-tests are on separate disks. The system simply transfers the records of students from pretest to instruction (or from instruction to post-test) disks.

(5) The access program is a fairly long program in its own right which provides the teacher user with access to and a means of modifying all of the student and system records stored on disk (by the management system). This code could not be integrated with the other common code because of memory limitations.

Scope and Sequence for Whole Number Addition

In whole number column addition, there are nine problem types and 2—9 instruction/mastery test levels with 1—5 difficulty levels for each, for a total of up to 405 different instructional combinations. Space is provided for maintaining the records of 30 students on each Learning/Mastery Test disk.

Problem Types

1. NC-Facts: Facts whose sum is less than 10
   (Available Instructional Levels (I-L evels) 8, 9)
2. C-Facts: Facts whose sum is between 10 and 19
   (Available I-L evels 8, 9)
3. MX-Facts: Facts whose sum is between 0 and 19
   (Available I-L evels 8, 9)
4. NC-2 rows: Problems with 2 rows (2 addend problems), no carrying and from 2 to 4 digits in each row
   (Available I-L evels 1—9)
5. C-2 rows: 2 addend problems with carrying in all columns and from 2 to 4 columns
   (Available I-L evels 1—9)
6. MX-2 rows: 2 addend problems with carrying in some columns and not in others and from 2 to 4 columns
   (Available I-L evels 1—9)
7. EXT-FCTS: One-step problems with 2 to 4 digits in top addend and 1 on bottom
   (Available I-L evels 8—9)
8. **SNGL-COL:** Problems with 3 or 4 rows of 1 digit addends and mixed carrying 
   (Available I-Levels 1–9)

9. **MULT-COL:** Problems with up to 4 addends, each with up to 
   4 digits, and mixed carrying 
   (Available I-Levels 1–9)

*Prerequisite (e.g., facts) types 
**Key (i.e., timed) types (Ordinarily, instruction on problem types, 
other than prerequisites or key types, does not include I-Level 8 
(timed) unless specifically assigned by the teacher.)

**LATTICE OF PROBLEM TYPES (FOR TESTING)**

<table>
<thead>
<tr>
<th>LEVEL VI</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL V</td>
<td></td>
</tr>
<tr>
<td>LEVEL IV</td>
<td></td>
</tr>
<tr>
<td>LEVEL III</td>
<td></td>
</tr>
<tr>
<td>LEVEL II</td>
<td></td>
</tr>
<tr>
<td>LEVEL I</td>
<td></td>
</tr>
</tbody>
</table>

Testing starts at Type 4 of LEVEL III (first at the lowest D-level, 
and then, only if successful, at the highest D-level); then testing 
proceeds to Type 5. Depending on performance at Level III, testing 
may stop or proceed to higher or lower levels. If a student succeeds 
with a given problem type in the hierarchy, then he or she can be 
assumed to know all types below it, and failure on a type implies 
failure at all higher level types for which the failed type is a pre-
requisite. Testing stops when all problem types are marked as either 
passed or failed.

Knowledge of basic facts is tested and scored simultaneously. 
Between 89%–75% (or less than 75%) accuracy on prerequisites 
(e.g., basic facts) results in a comment to the effect that the student 
may need help with facts (or that extra work on facts is recom-

**Sample Instruction Levels (Type 9: MULT-COL)**

1. **Meaning**
   Show all work. 
   T=9, I=1
   D=3; 3,3,3,0
   
   \[
   \begin{align*}
   400 + 70 + 4 &= 474 \\
   800 + 80 + 6 &= 886 \\
   + 100 + 50 + 0 &= 155 \\
   \end{align*}
   \]
   Now you do it, Name!

2. **Meaning/Rule**
   Show all work. 
   T=9, I=3
   D=3; 3,3,3,0
   
   \[
   \begin{align*}
   400 + 50 + 1 &= 451 \\
   900 + 30 + 9 &= 939 \\
   + 800 + 00 + 3 &= + 803 \\
   \end{align*}
   \]
   Start at top digit

3. **Directions/Rule**
   Show all work. 
   T=9, I=3
   D=3; 3,3,3,0
   
   \[
   \begin{align*}
   620 + 100 + 412 &= 1132 \\
   \end{align*}
   \]
   Other directions: 
   Position box for sum
   Start at the top digit
   Add
   More digits in column (Y/N)?
   Last column (Y/N)?
   Put box top next column
   Put box & carry top next column
   Position box for sum

4. **Rule/Remedial**
   Show all work. 
   T=9, I=4
   D=4; 3,3,3,3
   
   \[
   \begin{align*}
   111 + 160 &= 271 \\
   100 + 177 &= 277 \\
   \end{align*}
   \]
If the student makes an error, the program presents the “meaning” form of the problem and solves it up to the point of the error.

5. **Describing Rule**
   
   Describe the step shown. 
   
   \[ T=9, \ I=5 \]
   \[ D=3; \ 3,3,0 \]
   
   728
   
   842
   
   + 430
   
   (A) Start at top digit
   (B) Position box for sum
   (C) Add

6. **Shortcut Instruction**
   
   Compute in your head. 
   
   \[ T=9, \ I=6 \]
   \[ D=3; \ 3,3,3,0 \]
   
   813
   
   430
   
   + 123
   
   6

   Add column

7. **Shortcut (Mental) Calculations**
   
   Compute in your head. 
   
   \[ T=9, \ I=7 \]
   \[ D=4; \ 3,3,3,3 \]
   
   530
   
   502
   
   710
   
   + 821
   
   63

8. **Timed Practice**
   
   Compute in your head. 
   Try to beat the beep. 
   
   \[ T=9, \ I=8 \]
   \[ D=4; \ 3,3,3,3 \]
   \[ S=5 \]
   
   249
   
   380
   
   195
   
   + 801

---

**Scope and Sequence for Whole Number Subtraction**

In whole number subtraction, there are seven problem types and 2—9 instruction/mastery test levels with 1—5 difficulty levels for a total of up to 315 different instructional levels. Space is provided for maintaining the records of 30 students on each Learning/Mastery Test Disk.

**Problem Types**

*1. SINGL DIG: Facts subtracted from 9 to 0 (positive) (Available I-Levels 8, 9)

*2. 10 to 19: Facts subtracted from 19 to 10 (positive) (Available I-Levels 8, 9)

*3. NO REGRP: Problems requiring no borrowing (Available I-Levels 1—9)

*4. REGROUP: Problems requiring all borrowing, but not across 0 (Available I-Levels 1—9)

*5. MXD RGRP: Problems with both borrow and no borrow columns, but with no borrowing across 0 (Available I-Levels 1—9)

*6. REGRP-0: Problems of all borrowing across 0 (Available I-Levels 1—9)

*7. MX RGP-0: Problems with both borrow and no borrow columns, with some borrowing across 0 (Available I-Levels 1, 3, 5—9)

*Prerequisite (e.g., facts) types

**Key (i.e., timed) types (Ordinarily, instruction on problem types, other than Prerequisite or Key types, does not include I-Level 8 (timed) unless specifically assigned by the teacher.)

**LATTICE OF PROBLEM TYPES (FOR TESTING)**

**LEVEL V**

**LEVEL IV**

**LEVEL III** Start →

**LEVEL II**

**LEVEL I**
Testing starts at Type 3 of LEVEL III (first at the lowest D-level, and then, only if successful, at highest D-level); then testing proceeds to Type 4. Depending on performance at Level III, testing proceeds to higher or lower levels. Success/failure criteria are same as addition. Success at one level implies knowledge of prerequisite levels. For example, getting Types 3 and 4 correct implies knowledge of Types 2 and 1. Failure at a level implies failure at higher levels. The two sides of the lattice (3, 4 and 5, 6) must be tested separately before moving up or down the levels. Knowledge of the basic facts is tested and scored simultaneously with the levels being tested.

Examples of subtraction problems for the eight available Instruction Levels were given in Chapter 3.

Scope and Sequence for Whole Number Multiplication

In whole number multiplication, there are nine problem types and 2–9 instruction/mastery test levels, with 1–5 difficulty levels for a total of up to 405 different instructional combinations. Space is provided for maintaining the records of 30 students on each Learning/Mastery Test Disk.

Problem Types

*1. NC-FACTS: Facts whose product is a single digit
   (Available I-L evels 8, 9)

*2. C-FACTS: Facts single digit a \times b: a from 0 to 5 and b from 0 to 9 and product > 9
   (Available I-L evels 8, 9)

*3. C-FACTS 2: Facts single digit a \times b: a from 0 to 5 and b from 0 to 9
   (Available I-L evels 8, 9)

*4. C-FACTS 3: Facts single digit a \times b: a from 6 to 9 and b from 6 to 9
   (Available I-L evels 8, 9)

*5. MX-FACTS: Facts single digit a \times b: a from 0 to 9 and b from 0 to 9
   (Available I-L evels 8, 9)

6. NC-1 DIG: Problems with 2–4 digits on top, 1 digit below, and no carrying
   (Available I-L evels 1–9)

7. MX-1 DIG: Problems with 2–4 digits on top, 1 digit below, all with carrying
   (Available I-L evels 1–9)

8. NC-X DIG: Problems with 2–4 digits on top, 2–3 digits below, and no carrying
   (Available I-L evels 1–9)

**9. MX-X DIG: Problems with 2–4 digits on top, 2–3 digits below, mixed carrying and no carrying
   (Available I-L evels 1–9)

*Prerequisite (e.g., facts) types
**Key (i.e., timed) types (Ordinarily, instruction on problem types, other than prerequisite or key types, does not include I-Level 8 (timed) unless specifically assigned by the teacher.)

LATTICE OF PROBLEM TYPES (FOR TESTING)

LEVEL V

LEVEL IV

LEVEL III

LEVEL II

LEVEL I

Testing starts at Type 4 of Level II and proceeds in the same manner as addition and subtraction.

Sample Instruction Levels (Type 6: NC-1 DIG)

1. Meaning
   Show all work.

   \[
   \begin{align*}
   20 + 4 \\ \\
   \times 2 \\ \\
   + 8
   \end{align*}
   \]

   Multiply in your head

   \[T=6, I=1\]

   \[D=2,1\]
2. **Meaning/Rule**
   Show all work.
   
   \[
   \begin{array}{c}
   30 + 3 \\
   \times 1
   \end{array}
   \]
   \[
   \begin{array}{c}
   33 \\
   \times 1
   \end{array}
   \]
   Now you do it, Name!

3. **Directions/Rule**
   Show all work.
   
   \[
   \begin{array}{c}
   43 \\
   \times 3
   \end{array}
   \]
   Other directions:
   Multiply
   Start at right hand column
   Enter product (& carry) digit(s)
   Multiply in your head
   Last digit in multiplier (Y/N)?
   Enter zero
   Enter last carry
   Add

4. **Rule/Remedial**
   Show all work.
   
   \[
   \begin{array}{c}
   71 \\
   \times 8
   \end{array}
   \]
   If the student makes an error, the program presents the “meaning” form of the problem and solves it up to the point of the error.

5. **Description/Rule**
   Describe the step shown.
   
   \[
   \begin{array}{c}
   92 \\
   \times 3
   \end{array}
   \]
   (A) Add
   (B) Multiply in your head
   (C) Enter zero

6. **Shortcut Instruction**
   Compute in your head.
   
   \[
   \begin{array}{c}
   74 \\
   \times 2
   \end{array}
   \]
   Multiply

7. **Shortcut (Mental) Calculations**
   Compute in your head.
   
   \[
   \begin{array}{c}
   21 \\
   \times 9
   \end{array}
   \]

8. **Timed Practice**
   Compute in your head.
   Try to beat the beep.
   
   \[
   \begin{array}{c}
   33 \\
   \times 3
   \end{array}
   \]
   S=5

---

**Scope and Sequence for Whole Number Division**

In whole number division, there are seven problem types and 6 – 9 instruction/mastery test levels with 1 – 5 difficulty levels for a total of up to 315 different instruction levels. Space is provided for maintaining the records of 30 students on each Learning/Mastery Disk.

**Problem Types**

1. **NR-FACTS:**
   No remainder (NR) FACTS where 1 digit divisor goes evenly into 1 or 2 digit dividend (e.g., 9|83)
   (Available I-Levles 8, 9)

2. **FACTS+R:**
   FACTS plus Remainder, 1 digit divisor goes unevenly into 1 or 2 digit dividend leaving remainder (R) (e.g., 9|83)
   (Available I-Levles 8, 9)
3. MDIG DIV: 1 digit quotient with 2–4 digit divisor/dividend such that trial divisor (first digit in divisor) goes into first digit in dividend (e.g., 30\(\overline{64}\)) (Available I-Lvels 1–5 & 9)

4. DIV>IDVD: 1 digit quotient with one more digit in 2–4 digit dividend than in 1–3 digit divisor such that initial digit in dividend (IDVD) is less than trial divisor (e.g., 41\(\overline{369}\)) (Available I-Lvels 1–5 & 9)

5. TQUO>QUO: 1 digit quotient where trial divisor goes into trial dividend (first 1 or 2 digits) more times than divisor goes into 2–4 digit dividend (i.e., trial quotient (TQUO) is greater than quotient) (e.g., 25\(\overline{61}\)) (Available I-Lvels 1–5 & 9)

**6. 2DIG QUO:** 2 digit quotient (QUO) where trial quotient (TQUO) may or may not be greater than quotient digits (e.g., 89\(\overline{8531}\)) (Available I-Lvels 1–5 & 9)

**7. SHORT DV:** Arbitrary 1 digit divisor/3–4 digit dividend involving short division (SHORT DV) (e.g., 41\(\overline{3791}\)) (Available I-Lvels 6–9)

*Prerequisite (e.g., facts) types

**Key (i.e., timed) types. (Ordinarily, instruction on problem types, other than prerequisite or key types, does not include I-Level 8 (timed) unless specifically assigned by the teacher.)

**LATTICE OF PROBLEM TYPES (FOR TESTING)**

**LEVEL V**

LEVEL IV

LEVEL III Start \(\rightarrow\)

LEVEL II

LEVEL I

Testing starts at Type 3 of Level III and proceeds as in the other skill areas.

INTelligent RULETutor CBI SYSTEM

*Type 7 is treated independently of but subsequent to Type 6.

Sample Instruction Levels for Types 6 (2Q<TQ) OR 7 (SH-DIV)

1. Meaning
Show all work.

T=6, I=1

D=2,4

5

6

90

89\(\overline{8531}\)

8010

521

Now you do it, Name!

2. Meaning/Rule
Show all work.

T=6, I=2

D=2,4

90

89\(\overline{8531}\)

8010

521

Label remainder

3. Directions/Rule
Show all work.

T=6, I=3

D=2,4

89\(\overline{8531}\)

Other Directions:
Find divisor
(Determine the divisor then press the space bar to go on)
Find initial dividend
Divisor > initial dividend (Y/N)?
Move box for quotient
/, then lower trial quotient (Press the slash (/) key on top of the trial quotient)
Find tquo < tdivid/divisor
(Estimate the trial quotient by dividing the trial divisor into the
7. Shortcut (Mental) Calculation
Compute in your head.
T=7, I=7
D=1,4
4[3791]

8. Timed Practice
Compute in your head.
T=7, I=8
Try to beat the beep.
D=1,4
S=5
94
4[3791]

Note: 1 Level 9 refers to Mastery Tests.

LIMITATIONS AND FUTURE DIRECTIONS

The Tutorial Arithmetic implementation of the RuleTutor has been used by a large number of elementary school students with considerable success. Although no statistical study of its effectiveness has been carried out, most teachers have observed that students enjoy working with the system and learn effectively from it.

In spite of its positive features, however, certain problems with the arithmetic RuleTutor should be noted. First of all, the RuleTutor was designed to provide diagnostic testing and instruction on individual rules. Consequently, even in principle, the RuleTutor is limited to relatively restricted kinds of content.

For another thing, Tutorial Arithmetic is currently implemented in Applesoft (a version of BASIC developed for the Apple II computer) and 6502 machine language. Consequently, it is not easily transportable. Moreover, use of the BASIC language has made it more difficult to adhere strictly to the modularity (e.g., the distinction between common and content-specific code) we have strived for.

Finally, implementation in some cases did not reflect the underlying theory as accurately as possible. For example, meaning, metacognition and automation were treated in an ad hoc fashion. Whereas accurate implementation would have called for introduction of higher-order rules (rules which operate on rules), this was not done for both practical (i.e., resource limitations) and
For these reasons, in future extensions of the RuleTutor concept, it would be desirable to improve the basic design, thereby yielding a new RuleTutor more powerful and generalizable than the existing one. As a minimum, future designs should make provision for the following.

First, rule diagnosis and rule instruction in the current RuleTutor (Tutorial Arithmetic) are totally independent activities. Thus, all diagnostic testing is completed (in a sequential and highly efficient manner) before any instruction is provided. In fact, however, testing and teaching are highly interrelated both in practice and in principle. Thus, partial information from testing may provide a sufficient basis for (some) instruction. Conversely, instruction on a portion of a rule may influence test performance on other items and, hence, reduce the amount of instruction that otherwise might be prescribed.

Second, the current RuleTutor involves instruction on individual rules (i.e., cognitive procedures). Consequently, the current RuleTutor design could not readily be extended to deal with sets of lower- and higher-order rules, even in principle. Among other things, doing so would require implementation of the universal "goal switching" control mechanism proposed by Scandura (e.g., 1971, 1973, 1981).

Third, although it cannot deal with higher-order rules, the current RuleTutor was designed so as to achieve some of the benefits they would have provided. Unfortunately, the RuleTutor's ability to deal with such things as rule meaning and verbal awareness was bought at the price of some loss of extensibility.

Fourth, even though modularity and structured programming were at the forefront of our original RuleTutor development effort, our use of the Basic language (because of its broad availability on microcomputers) and memory limitations of the Apple II computer resulted in some unavoidable compromises along these lines.

Figure 6 is a flow diagram describing at a fairly high level how the idealized "teacher" discussed in Chapter 2 might perform. It provides a blueprint for an enhanced version of the RuleTutor. It assumes that this idealized teacher can utilize two kinds of data prepared in advance: (1) a set of rules (represented as ordered triples) and (2) a set of problems (or problem generators) characterizing the curriculum goals (which is equivalent to the Problem Domain). The latter might refer only to some subset of the entire Problem Domain, perhaps selected so as to be appropriate for a specific student population and with the various subgoals perhaps hierarchically arranged to reflect educational priorities.

In this general model, some processing of the rule and problem/goal data would be done prior to any instruction. The first step in this processing would be to identify (if necessary) the specific rules or parts of rules necessary for achieving the curriculum goals. The second step would be to identify the "derivation history" of the rules needed to achieve the curriculum goals (e.g., Scandura, 1977). Rule derivation plays a significant role in domains where some of the rules are derived by the learners when needed, using higher-order rules (e.g., for generalization or composition of operations) which
they have been taught. Given the derivation information and the (hierarchy of) goals, it might be possible in principle (but very difficult practically) to identify an optimal derivation sequence for the rules, given the assumed processing constraints of the learners. This optimal sequence then might be used to guide the teaching/learning process.

The idealized teacher depicted in Figure 6 would be applicable with arbitrarily complex content—not just with cognitive procedural tasks involving single rules. For example, such a system if fully implemented might be used to provide instruction on learning strategies (higher-order rules), lower-order rules (cognitive procedural tasks) and interactions between them. (It might, in addition, integrate testing and instruction, and control the interaction between the problems and the learner so as to optimize learning efficiency.)

Most steps in the above procedure correspond to steps in the current RuleTutor and are relatively unambiguous. Hence, there would be relatively little difficulty in implementing them. Nonetheless, the step pertaining to INSTRUCTION (capitalized and marked with "***" in the flow diagram) would play a central role in any future implementation and deserves elaboration. Specifically, when a student has failed a problem type and is to receive instruction, he or she may very well know some components of the rule path corresponding to that problem type. Consequently, the RuleTutor's instruction is designed to focus on those components which the student does not know. A future RuleTutor might make a conservative approximation of what components (i.e., steps) of the rule are known by first marking all steps as unknown and then marking as known those steps which are used in known problem types.

In addition, unlike the current RuleTutor, the student should be allowed some discretion in selecting types of problems for instruction and/or kinds of instruction. This option recognizes the fact that a given learner may wish to receive instruction (only) on a particular type of problem or in a particular way.

Before instruction begins, the program would have a representation of a problem with one or more wrong answers, specifically the failed test problem. These wrong answers will be erased from the buffer, and then the student and RuleTutor would work through those steps of the rule procedure which are used in solving that problem. For known steps, the student might be able to enter the answer or press the ENTER key to have the RuleTutor enter the answer. For each unknown step, the student would immediately attempt to supply the answer or ask for instruction prior to attempting to answer. Two general forms of instruction might be made available: (1) the RuleTutor displays and then erases the answer (i.e., teaches by showing) and (2) the RuleTutor provides verbal instruction/hint/description on the unknown step and then erases it. In each case, the student is expected to respond after the instruction.

If the student enters an incorrect answer for any step, the RuleTutor might "beep", erase the answer, provide backup instruction on the step where deemed appropriate and have the student try again. In addition to knowledge of results on each step, the student might request motivational support in the form of positive feedback after completing instruction on a path.

The flow diagram in Figure 7 describes this hypothetical instructional process in more detail.

In summary, the Arithmetic RuleTutor case study has shown that the structural learning theory can provide a strong foundation for the design of an intelligent CAI system. And, as the above discussion of limitations has pointed out, the theory can be used in a deeper way as the basis of a considerably more flexible and efficient CAI system than the current implementation. In the past 40 years we have seen an extraordinary rise in the amount of computational power that can be purchased by a given amount of money, and we have also witnessed considerable progress in the understanding of how people process information. The combination of these two trends is barely beginning to have a large-scale impact on education, and we are eager to continue our involvement in this revolution.

Bibliography

Brown, J. S. and R. R. Burton. Diagnostic models for procedural bugs in basic
Determine unknown (and known) components of current type.

Erase current wrong answers to current problem.

WHILE solution to current problem is not complete

DO

IF current step involves a known component of rule

THEN

Student chooses to do one of the following:

(1) Enter answer for current (known) step.

(2) Press ENTER key to get RuleTutor to generate answer for current step.

ELSE

Student chooses to do one of the following:

(1) Enter answer for current (unknown) step.

(2) Press ENTER key in order to choose to do one of the following:

(1) See answer generated by RuleTutor (which then is erased), then enter answer.

(2) See verbal instruction or description supplied by RuleTutor (which then is erased), then enter answer.

IF student entered incorrect answer

THEN

RuleTutor “beeps”, waits for student to press ENTER, then erases answer. (Step does not change.)

ELSE

RuleTutor advances to next step of rule for current problem (after providing knowledge of results).

Motivating feedback (where desired by student).

Figure 7. Refinement of the RuleTutor step “Provide INSTRUCTION on (missing components of) solution path associated with current type.”
INTELLIGENT RULETUTOR CII SYSTEM