11/ The TEACHER'S APPRENTICE: Designing an Intelligent Authoring System for High School Mathematics

MATTHEW W. LEWIS,
ROBERT MILSON, and
JOHN R. ANDERSON

Introduction

The goals of the TEACHER'S APPRENTICE project overlap with the goals of other authors appearing in this collection: Improve education by making intelligent computer-assisted instruction more effective, available, and realistically usable. At the same time that we are developing tutoring software, we, like others, are also deliberately exploring the epistemological issues about the nature of the knowledge that is being tutored and how that knowledge can be learned. In our project, the design of tutoring software and the refinement of cognitive theory go hand in hand and affect one another (Anderson, 1985). The constraint that an operational tutoring system must, by definition, be well defined presents the opportunity to examine the system's successes and failures in terms of the underlying epistemological and pedagogical decisions made during its design and implementation.

This work is supported by grant MDR 8470337 from the Science and Engineering Education division of the National Science Foundation. The ideas in this chapter owe a great deal to interactions and research done with James Greeno of the University of California, Berkeley. Many thanks go to Ms. Margaret Shields, Dr. John R. Young, and Dr. Paul G. LeMahieu of the Pittsburgh city schools for their generosity and cooperation. We feel that comments made by Patrick Thompson, Bret Wallach, and Derek Sleeman on an earlier draft have substantially improved this chapter. This work also benefited from fruitful interactions with Frank Boyle and with members of the algebra research group: Julie Epelboim, Steve Garlick, Kate McGilly, Kevin Singley, Joe Thompson, and Ik Yoo.

The TEACHER'S APPRENTICE project is so named because the goal of our research is to produce a system that has many of the attributes of a human TEACHER'S APPRENTICE, or student teacher. This system has knowledge of the domain (it can solve all the problems it teaches), it has knowledge about patterns of errors that occur in its domain of expertise, it has knowledge about teaching strategies, and it has the intelligence to emulate a human teacher's language and tutoring strategies.

The first domain for which the TEACHER'S APPRENTICE is being developed is a first-year course in high school algebra. Basic algebra was chosen because there has been other relevant research in the areas of algebra problem solving (Davis, 1975; Davis, Jockusch, & McKnight, 1978; Lewis, 1981; Wagner, Rachlin, & Jensen, 1984), errors (Matz, 1982; Sleeman, 1982; Sleeman, 1984), simulation (Bundy, 1983; Bundy & Welham, 1981), and tutoring (Brown, 1983; McArthur, 1985; Sleeman, 1982). In addition, basic algebra is fundamental to later work in much of mathematics and in most technical fields.

Relative to the other intelligent tutoring research in progress at Carnegie-Mellon University, the TEACHER'S APPRENTICE project is still in its early stages. Other local projects have had the goals of constructing intelligent tutoring systems for LISP (Anderson & Reiser, 1985) and for high school geometry (Anderson, Boyle, & Yost, 1985). These systems, while successful, have led us to investigate the domain-independent parts of tutoring and how to speed up the task of building such systems. The TEACHER'S APPRENTICE project is an exploration of these questions.

The TEACHER'S APPRENTICE project has aspects which are aimed at each of three general goals:

- *Make intelligent tutors "effective": design, implement, and evaluate an intelligent tutor that emulates effective human tutoring and is based on specific predictions of a learning theory.* The goal of making intelligent tutors "effective" translates into having them emulate what good human tutors do when tutoring. The teaching effectiveness of individual tutoring over standard classroom instruction has been documented (Bloom, 1984) and has been referred to as the "two-sigma" effect: children being tutored perform two standard deviations above children learning the same materials in a standard classroom setting. This size of an effect is quite impressive in the world of instructional interventions. We also wish to keep as many of the pedagogical decisions as possible grounded predictions of a learning theory.
- *Make intelligent tutoring systems "available": cut development time and cost through design modularity and get intelligent tutors to run in real time.* To make intelligent tutoring software available means being able to produce it at a reasonable cost. Producing an intelligent authoring system as a tutoring architecture for a domain like high school mathematics would greatly reduce the time needed to bring up lesson materials in similar subdomains. If a tutoring component could be designed that was relatively independent of the exact expertise it was teaching, then the possibility of using a single architecture across the subdomains of Algebra I, Algebra II, and high school calculus becomes possible. There are also the important issues of how to make the software run in real time on machines that will be affordable to schools five years from now.
- *Make intelligent tutors "usable": design systems which will be amenable to different problem solving and teaching styles and fit into the culture of classrooms.* The reality of teachers and students in a dynamic classroom environment presents another goal for our project: Make this system usable for real teachers in real classrooms. Different teachers might teach subtly different algorithms and use different language for math entities and concepts. We are pursuing the development of a system that tailors its tutorial style to the style of the individual teacher using it. Individual teaching style is one part of a larger "culture of the classroom" into which this teaching aid must be integrated. The tutor's design might capitalize on features of the culture to maximize the probability of the tutor's eventual acceptance and effective use.

It is appropriate to also state what is *not* one of our goals: It is *not* our goal to replace high school mathematics teachers. Mathematics teachers have other roles besides transmitting skills. There are many activities that are part of a well-rounded mathematics education that teachers often complain they never get a chance to do: they are too busy teaching the required skills to get into activities often put under the title of "enrichment and motivation." This is where the "art" of mathematics can be conveyed as well as conveying a broader appreciation for the structure and functionality of mathematics. Motivating algebra learners seems to be a large part of the battle in the "normal level" classrooms we've observed. High school students want better answers to such questions as "Why are we learning to factor?" than the standard responses of "You'll need it later" or "It will be on the SATs." These terse answers are necessitated by the constraints of have 30 students in a classroom and not being able to digress to answer such questions. By giving teachers intelligent teaching tools that are more effective at transmitting problem-solving skills than a normal classroom environment we may be able to restructure classrooms into groups that effectively reduce the student-teacher ratio. This allows more opportunities for teachers to provide enrichment and motivation as well as supplying time to attend to the needs of individual learners.

We do foresee situations where the tutor might be practical without the aid of a human teacher. These are settings with adults involved in remedial mathematics or settings where human teachers are not available. These cases are discussed at the end of the chapter.

We will first briefly describe the theoretical foundations of the project and how those theoretical foundations translate into the design of intelligent

tutoring systems. We then describe the components of the system and their interactions. Following this description of the system we discuss some possible interesting side effects of developing intelligent authoring systems for ICAI and how it might affect teacher training and teacher participation in learning research. We also address how the system differs from quality CAI: how the surface similarity of the system's output to the output of CAI belies the flexibility of the underlying mechanisms in the TEACHER'S APPRENTICE that generate that output. We close with a summary of how the TEACHER'S APPRENTICE project attempts to achieve the general goals stated above. Our experiences with pilot studies and preliminary results with algebra-naive learners are interspersed throughout the sections.

Theoretical Foundations of the TEACHER'S APPRENTICE Project

Fundamental to our tutoring work is the claim that we can construct models of how students actually perform the skills that are to be tutored and how these skills are acquired. Our cognitive model of performance and learning is the ACT* model (Anderson, 1983) of cognition. Skills are represented as sets of productions which must be acquired through practice. Learning in ACT* is strongly a learning-by-doing model of skill acquisition with a clear distinction between declarative and procedural knowledge. Students can encode (often incorrectly or incompletely) the description of a skill from a text or from examples into some declarative form. The declarative encoding is then run interpretively by general productions in the system. As the set of declaratively represented productions are applied in the service of problem solving they are automatically compiled into a set of faster, more smoothly operating problem-solving actions through the mechanisms of composition and proceduralization: problem-solving steps are collapsed and the productions are generalized (Anderson, 1982). Strengthening and tuning of productions takes place through repeated application, with the end product being the automatic application of problem-solving skill.

Components of the TEACHER'S APPRENTICE System

A theory of instruction should be maximally effective when it is based on a theory of learning. Based on the ACT* theory of problem-solving performance and skill acquisition, a set of design principles for intelligent tutoring systems has been enumerated and the connections between the principles and the learning theory have been argued (Anderson, Boyle, Farrell, & Reiser, in press). These principles, presented in Fig. 11.1, have driven the design decisions of the TEACHER'S APPRENTICE. What follows is a description of the components of the TEACHER'S APPRENTICE system: the student model, consisting of the "ideal"

1. Represent the student as a set of productions.
2. Communicate the goal structure underlying problem solving.
3. Provide instruction in the problem-solving context.
4. Promote an abstract understanding of the problem-solving knowledge.
5. Minimize working memory load.
6. Provide immediate feedback on errors.
7. Adjust the grain size of instruction with learning.
8. Facilitate successive approximations to the target skill.

Figure 11.1 Design Principles for Intelligent Tutoring Systems, Based on Predictions of the ACT* Model of Cognition (from Anderson, Boyle, Farrell, & Reiser, in press)

and errorful or "buggy" models of student performance (Brown & Burton, 1978; Sleeman, 1982), the interface, the tutorial component, and the rule induction component for learning tutoring rules.

The Ideal Model of the Student

Having a simulation of the ideal performance of a student is required for the "model tracing" methodology of tutoring (Reiser, Anderson, & Farrell, 1985). The human learner's problem-solving behavior is compared with the performance of the ideal model. The instruction provided to the learner at each point is a function of our interpretation of the learner's internal state in the problem solving. This can be inferred by matching the learner's output to problem states generated by various ideal and "buggy" rules. Creating sets of production rules that simulate the problem-solving processes of an ideal student is a sizable but largely achievable task. As in the other tutoring projects at Carnegie-Mellon University, this is accomplished through iterations of writing rules followed by observation of student performance. The observation allows assessment of the completeness of the rule set, relative to productions inherent in the performance of human learners.

Figure 11.2 contains an example of one possible set of productions from an ideal model for doing distribution. These rules would be invoked if an expression such as $3(7x + 5)$ appeared somewhere in the equation being simplified. The first rule recognizes that distribution is applicable to this equation and sets the subgoal to distribute (strategic knowledge). The second rule

```
P1: IF    the equation to be solved contains a subexpression of
            the form num(term1 + term2)
    THEN  set as a subgoal to distribute num over term1 and term2

P2: IF    the goal is to distribute num over term1 and term2
    THEN  set the subgoal to multiply num times term1
      AND set the subgoal to multiply num times term 2
      AND set the subgoal to combine the previous results with +

P3: IF    the goal is to multiply num times term
    THEN  write the product of num and term

P4: IF    the goal is to combine term1 and term2 with a +
    THEN  write term1 + term2
```

Figure 11.2 Examples of Ideal Model Rules for Doing Distribution

sets up three subgoals: to multiply $7x$ by 3, to multiply 5 by 3, and to add these intermediate results together. The third rule evaluates multiplication subgoals. The fourth rule evaluates add-together subgoals. Rules 2, 3, and 4 are instances of axiomatic knowledge in this rule set.

These rules highlight two features that are critical in generating rule sets for the student model:

1. The grain size of rules is critical when simulating students.
2. Both the strategic and axiomatic knowledge must be simulated.

If the ideal model does not contain rules for a problem-solving action, then that action cannot be independently recognized or tutored by the system. If strategic knowledge is represented independently from the axiomatic knowledge of a domain, this leaves the opportunity to tutor each independently. Having the ability to focus a student's attention solely on strategic decision making at some points and solely on axiomatic knowledge application at other points is a tutoring strategy we have observed some teachers using. The load on working memory can be lightened by varying this division of knowledge between student and tutor during a tutorial session. Results of some informal studies of novices learning to apply the transformations of algebra imply that simply learning how the axioms manipulate symbols may be easier than learning when to apply that axiom in service of problem solving. These strategic and axiomatic components of a skill must both be well learned if the skill is to be applied successfully in problem solving.

But what lower bound do we set on simulating the ideal student? Do we simulate the addition of positive integers as a set of productions that simulate counting on fingers? The answer is no, but how does one make decisions about grain size of ideal model rules in a principled way? There seems to be no clear-cut answer, but the general heuristic is that if you don't *ever* want to tutor a subskill, then represent it as a function call to LISP instead of an ideal model rule that has to fire. This saves time because you don't have to write the rule originally nor does the rule need to be matched during the ideal model's problem solving.

Simulation of Errorful or "Buggy" Problem Solving When a student makes a problem-solving action that is nonoptimal (perhaps a legal but strategically nonuseful transformation) or incorrect (the application of the axiom is carried out incorrectly in some way), a good human tutor would have some idea of the underlying cognitive cause of the problem. Error-recognition productions of this sort were hand-coded into the LISP tutoring system (Reiser, Anderson, & Farrell, 1985). A goal for the TEACHER'S APPRENTICE is to simulate and recognize these errors on line, in the manner of Brown and van Lehn (1980). Very nice work on the processes underlying algebra errors has been done by Matz (1982). However, from observing classrooms and test performance, there appear to be other classes of errors not covered in the set of generative bugs mapped out by Matz. Errors in factoring that we have observed provide an illustration of these classes.

An amazing variety of errors occurred when an Algebra I class that had studied factoring of both two- and three-term expressions was asked to factor an expression such as $3x + 6y + 12$. Some consistent errors fit into classes covered by Matz, for example, $3(x + 2y) + 12$. This is the failure to repeat the application of an iterative procedure. It is possible that the student is still using a nongeneral, two-term form of the factoring procedure. Also note that this is not a "true error" (a transformation that does not preserve equivalence), but an incomplete application of a transformation.*

However, there seem to be other classes of errors that appear to be amenable to simulation. An example is what we call "form errors." Students gave answers to factoring problems that had the proper form (they looked like an answer to a factoring problem) but had none of the operations of factoring. Examples we saw included $3x(6y + 12)$ and $3(x + 6y + 12)$. Both responses conform to the informal rule that you "pull something out of the terms and leave something in parentheses." Both answers have the *form*

* Indeed, there are times when you need to refrain from applying the distributive property to an entire expression in an equation to solve that equation in a simple manner. However, there is also the ability to correctly factor out the greatest common factor from all terms of an isolated expression.

of a correct solution. The latter is different from a "failure-to-iterate" error in that the terms that have not been factored are included in the parentheses. This is a "true error," unlike the incomplete iteration bug above.

In work done with James Greeno, we interviewed students who make such form errors to acquire data regarding how they are misparsing transformations. An example is the student who applied the associative law—presented as $A + (B + C) = (A + B) + C$ to the student—to the arithmetic expression $3 + (4 + 5)$ and got the expression $(4 + 5) + 3$. This is a legal application of the commutative law, but not a legal application of the associative law. When asked how she did this she responded, "You take the part that is sticking out and flip it around to the other side of the parentheses." This response shows no encoding of the fact that the variables have to have the same bindings (i.e., A has to keep the value of 3) or the goal of the application of the associative law: change the grouping of terms. However, the form of the answer was correct. We are working on algorithms that will relax the conditions of some productions to generate buggy rules, as in the work of Brown and Burton (1978). At the same time, we are investigating procedures that will wrongly induce the transformation action embodied in a correct algebra transformation rule.

A study of errors in factoring (Seigler & McGilly, in preparation) is being carried out in parallel with our research. This study investigates the performance of students on factoring problems given different amounts of partial solutions. One of the goals of the pencil-and-paper tests is to decompose the skill of factoring into psychologically meaningful subprocedures. The classroom of study was a nonaccelerated Algebra I class who had not restudied factoring for several weeks, and the problems were presented in the context of a normal class examination. For the problem of factoring $3x + 6y + 12$, students received one of the following frames to fill in answers:

_____	(write out the answer)
___(___ + ___ + ___)	(fill in the blanks)
3(___ + ___ + ___)	(fill in the blanks)
___(x + 2y + 4)	(fill in the blank)

The first is the standard, unaided answer, the second provides the form of the answer and prompts for at least three terms inside the parentheses, the third has the greatest common factor already factored out, and the fourth has the terms with factors removed, leaving only the identification of the greatest common factor. The general result to date is that the more of the subprocedures that were done for the student, the better the performance. Correct responses were most prevalent in the third (88%) and fourth (100%) cases and least prevalent in the first case (58%). Correct performance with the second case was

intermediate (71%). Interestingly, there were significantly more correct answers in the second case than the first. Prompting with the form of the answer provided greater support for the correct procedure than no prompt at all. This result fits in with an observation of Matz's (1982) that the first hurdle of the student is knowing what the goal is, that is, inferring the features of the desired result from the instruction. Apparently once the goal is clear, performance improves significantly. This result also corroborates the reports of teachers who identify a class of students who are able to solve sets of problems they are competent to solve only after an example is done for them. This worked example apparently provides the form or goal of the problem, and the students can then apply their skill.

There are also special cases where students with generally correct procedures make errors. An example is the case of factoring an expression where the greatest common factor is an integer term in the expression, for example, $24x + 12$. Some students we observed will not write out 1 as a term in the factored expression. Erroneous responses to the expression given by such students include $12(2x + 12)$ or $12(x)$. These errors might be manifestations of an operator like:

$$X/X \rightarrow X$$

or

$$X/X \rightarrow 0$$

An alternative explanation, offered by a reviewer, is that many students think they are acting *on the symbols per se*. When they are factoring the expression $24x + 12$, they "take the twelve out," leaving a blank space, not zero. This would produce answers like $12(x)$ or $12(2x)$.

These special-case errors can be simulated explicitly and can be candidates for matching buggy responses in situations other than the context of factoring problems. If the error occurred in a division problem, for example, $12/12 = X$, then possible simulated buggy responses from the erroneous rules above would be $0 = X$ and $12 = X$. Siegler and McGilly plan on varying this special case of errors within the factoring experiment mentioned above.

The Interface

Previous experience with developing intelligent tutors has shown that the design of the interface can make or break the effectiveness of a tutor, regardless of the clever design that went into the guts of the underlying system. Our experience points to five critical features of an effective interface for tutoring. It should:

1. Be as easy to use as possible. This means it should minimize the number of actions (e.g., keystrokes, mouse-clicks) needed to communicate with the tutor.
2. Have a structure or representation that is as congruent as possible to the underlying structure of the problems to be solved.
3. Be highly interactive and provide as much information about intermediate problem-solving states of the learner as possible. Given that the feedback received by the student is meaningful, Malone (1981) has pointed to interaction as a motivator in computer-based activities.
4. Have the ability to notice low-level errors as they occur; that is, it should continuously monitor the student's input.
5. Have the ability to vary working memory load. This can be accomplished by giving the student ready access to problem-relevant information and minimizing the number of parts of the screen that have to be attended to and integrated.

A sample of the current interface screen, in reduced form, is presented in Fig. 11.3. Our development machine is the Xerox 1109 DandeTiger.

The TEACHER'S APPRENTICE interface component is a software module that gives both the teacher/curriculum designer and learner access to the input/output facilities of the underlying hardware. It turns the computer's high-resolution display into a combination notebook, scratch pad, and blackboard that is used to echo tutoring interactions and store a record of the student's intermediate work. This design allows the mouse to be used as the primary input device: students can select parts of expressions or menu options with the mouse.

At the current time we have the following windows:

- The Tutoring Window (made to resemble a blackboard), where tutorial interactions are printed.
- The Scratch Pad, which has keypads for generating algebraic expressions needed for responses to the tutoring interactions.
- The Current Equation Window, which displays the current state of the student's equation transformation process.
- The History Window (made to resemble a notebook), where a trace of the problem-solving interaction is recorded for on-line review and later printing by the student.

Our goal was to make the interactions with the interface as easy as using pencil and paper. When generating responses to the tutorial dialogue, students need not type anything on the keyboard. In fact, in most cases, a response can be generated by pointing to parts of the existing expressions in the current

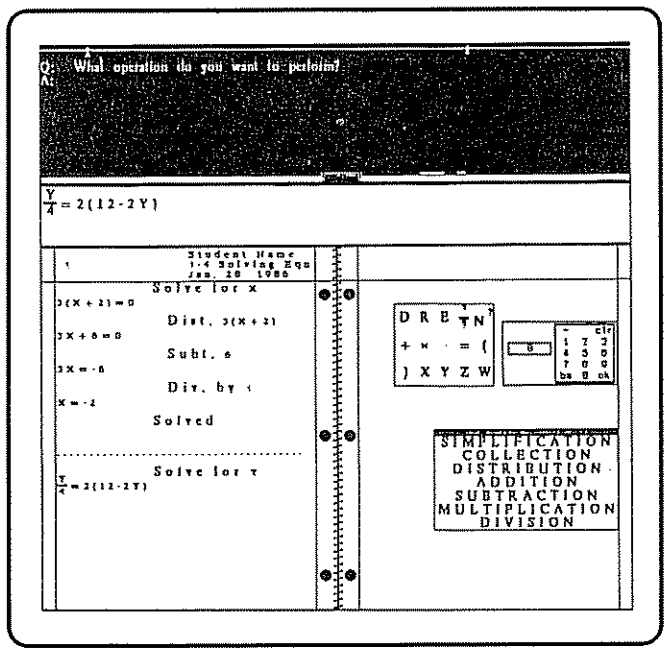

Figure 11.3 Interface of the TEACHER'S APPRENTICE with a Problem Trace and Prompt for Initial Strategic Decision for a New Problem

equation window. By using pointing instead of requiring students to regenerate complete expressions when only part might be changed, we may eliminate the majority of simple slips we've seen in the early stages of learning algebra. Accidentally dropping a negative sign or forgetting to bring down a term are common errors when moving from one equation to the newly transformed equation. By having students point to expressions and having the system bring down the untransformed parts of the equation intact, we can minimize the chance that these low-level errors would interfere with the goal of the lesson, for example, practicing the distributive law in equation solving.

However, this also raises the "training-wheels" issue: what happens when the student leaves this environment that supports his or her problem solving and has to face the realities of pencil and paper? The purpose of training wheels in this context is to unburden working memory with extra tasks until the subskill can be refined and strengthened. Then the burden of working memory required by an unsupported environment can be added in. For example, after bringing down all the untransformed terms and symbols in an equation the system could require the student to bring down individual terms and then eventually end with having the learner enter the entire expression, just as in pencil-and-paper work. We have experience with a PC-based version of a tutor for solving linear equations* that at one point required students to type in entire expressions. Observing high school algebra novices interact with this system made it clear that even the action of typing was, for many students, a large burden. Many errors occurred due to slips and mis-strikes of keys. Often the cause of the system's responses to a simple low-level typing error was misattributed by the learners to a misunderstanding on their part. We want to reduce these often nonconstructive or counterproductive errors in the early stages of skill acquisition.

The interface also has the capability to represent some of the structure of the nested-goal nature of the domain. Figure 11.4 shows a trace of a fine-grained tutorial interaction for factoring. The subskills involved in factoring include the ability to find the greatest common factor of a set of terms. This subgoal must be set at some point, either explicitly or implicitly, depending on the grain size of your rule system at the time of tutoring. Since these are subgoals to the top goal of factoring an expression (which in itself might be in service of some higher-level goal of isolating a variable), we represent the setting of this subgoal as an indented expression in the history window. As subgoals are stacked, the indentation increases, representing the nested structure of the problem solving.

One area where we currently do not represent the structure of the domain with our interface is the idea of search during algebra problem solving. In the geometry tutor (Anderson, Boyle, & Yost, 1985), the search of applying inference rules to generate parts of a proof is represented graphically. The to-be-proven statement is at the top of the screen and the givens of the problem are along the bottom. As the search proceeds, a tree structure is drawn on the screen connecting the entities used in the inferences and generated by the inferences. A claim of the geometry tutoring project is that this graphic representation of the proof-solving process as a tree provides the students with an understanding of what it means to "prove" something in a formal system. A similar tack is being taken in algebra tutoring work by Brown (1983) and McArthur (1985). They represent the search of applying algebraic transformations to expressions as a

*This system currently runs on an IBM PC in under 192K of memory, contains an expert system for solving linear equations, and allows multiple solution paths.

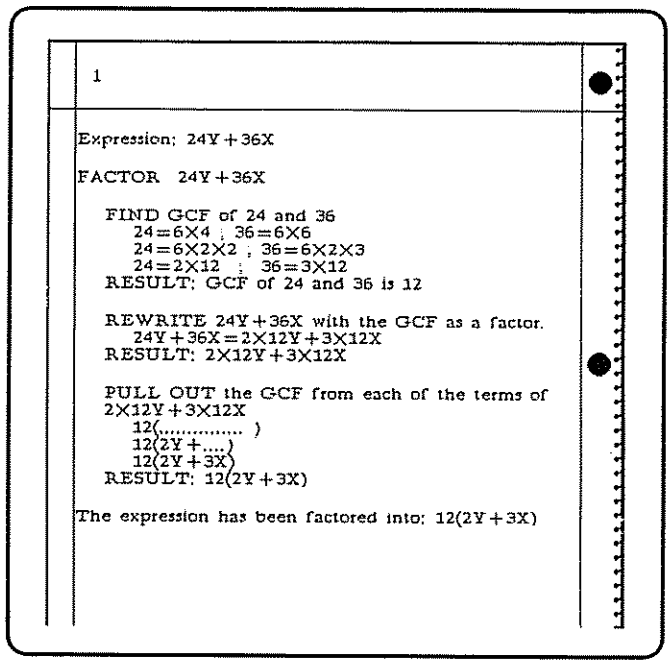

Figure 11.4 An example of the problem-solving trace which appears in the history window during the course of factoring an expression. This trace shows the use of indention to represent subgoals.

tree structure. The reason that this has not been incorporated into the TEACHER'S APPRENTICE project is that the model-tracing methodology currently keeps the learner on one of the correct solution paths in the problem. The possibility of going off one of the correct solution paths is not currently allowed. When an attempt is made at a strategically nonoptimal move the student is told that it is nonoptimal, given a justification in terms of reducing differences in the expression, and prompted to choose again. The efficacy of letting students search off of correct or optimal problem-solving paths is very much an open question that invites some incisive research.

Although the graphic displays of tutors have been explored, a dimension of an interface that we have not yet explored is the use of synthesized speech for the tutorial dialogues. Use of speech would embody part of the fourth goal of effective interface design enumerated above: try to reduce the number of parts of the screen to be attended to and integrated. The interactions of good human tutors with students of algebra contain a great deal of pointing while talking, in most cases students never take their eyes off of the expressions being manipulated as the tutor points to parts of that expression. We feel that extra pedagogical leverage might be gained through the use of highlighting expressions while presenting synthesized speech.

The Tutoring Component

The third component of the system, and the one that interacts with both the student models and the interface, is the tutoring component. The tutoring component determines the exact nature of the interactions between the tutoring system and student by taking cues from the state of the student model simulation and by utilizing input/output facilities provided by the interface so that appropriate advice and feedback can be given to the learner.

The tutoring component implements a specific tutoring strategy that is, roughly speaking, a set of rules describing the sort of questions to present, the answers to expect, and the nature of feedback during the student interactions. Thus the same problem-solving task is amenable to several tutoring strategies. For instance, if the tutoring domain is the solution of linear equations in one variable, two strategies are conceivable.

The first possible strategy is to guide the student through the solution path a step at a time and be very explicit regarding the nature of responses when asking questions of the student. Thus, for the example $3(x + 2) = 14$, the system might provide the very simple dialogue given in Fig. 11.5. The tutorial interactions we present are intentionally impoverished and simplified due to space constraints and belie the flexibility of the system. As mentioned earlier, students can solve a single problem in a variety of ways, and errors can be detected and handled intelligently. The goal of presenting examples of interactions is simply to illustrate the operation of the system. The richness of the tutorial interaction is in the hands of the implementor.

The dialogue in Fig. 11.5:

- Presents the student with a menu of possible operations that can be performed on the equation, accepting only DISTRIBUTION as a correct choice.
- Makes the student indicate the term to be distributed (3) and the expression over which to distribute it.
- Guides the student through the distribution process one term at a time (3 over $x \rightarrow 3x$ and then 3 over $2 \rightarrow 6$).

CURRENT EXPRESSION: 3 (x + 2) = 14

Q: What operation do you want to perform?

Operations	
COLLECT LIKE TERMS	SUBTRACT FROM BOTH SIDES
DISTRIBUTE	MULTIPLY BOTH SIDES
ADD TO BOTH SIDES	DIVIDE BOTH SIDES

A: DISTRIBUTE [student moused the DISTRIBUTE menu option]
→ Correct

Q: To what expression do you want to apply the distributive law?
A: 3 (x + 2) [student has boxed this expression with the mouse]
→ Correct

Q: What term do you want to distribute?
A: 3 [boxed in the current expression]
→ Correct

Q: What expression do you want to distribute 3 over?
A: (x + 2) [boxed in the current expression]
→ Correct

Q: What part of the expression of x + 2 do you want to distribute over?
A: x [boxed in the current expression]
→ Correct

Q: What is 3 * x?
A: 3x [boxed in the current expression, as two individual entities]
→ Correct

Q: What part of the expression of + 2 do you want to distribute over?
A: 2 [boxed in the current expression]
→ Correct

Q: What is 3 * 2?
A: 6 [generated from a keypad menu on the screen]
→ Correct

Q: What is the new expression after distribution?
A: 3x + 6 [generated from the current expression and the keypad menu]
→ Correct

CURRENT EXPRESSION: 3x + 6 = 14

Figure 11.5 An Example of an Extended Tutorial Dialogue for the Application of the Distributive Law in the Service of Simplifying an Equation

- Requires the student to assemble the new expression and then rewrites the rest of the equation.

A second possible tutoring strategy would be to allow the student to reach the solution as independently as possible. This strategy would be implemented by allowing the student to type in any equation that is closer to the final solution than the equation considered previously. Feedback is given when the student enters an equation that is not closer to the final solution. The exact content of the feedback depends on the problem-solving context and the student's actual error. For example, the erroneous transformation:

$$3(x + 2) = 14 \quad \rightarrow \quad 3x + 2 = 14$$

generates feedback such as "You only distributed the 3 over the x and not over the 2. When distributing, the distributing term has to multiply each term in the distributed expression." Students are then allowed to correct the mistake once; if they fail, they are simply given the next equation on the solution path and allowed to continue independently.

The two tutoring strategies outlined above have very different consequences in regard to the problem-solving processes. The first strategy will greatly restrict the kinds of errors that the student may make by taking a very strong-handed or structured approach to the tutoring process. The second strategy is much less restricted, allowing more freedom for learners to collapse steps. We are working towards a general and tailorable tutoring component to allow precisely this sort of difference: both of the strategies may be implemented on a system with the same student model of equation solving and the same interface capabilities. The issue of the tutoring strategy is thus separable from the specific nature of the student model and the tutoring interface. In other words, it should be possible to design the student model and the tutoring interface without an exact specification of the tutoring strategy.

The Operation of the Tutoring Component The control of the tutoring component is determined by a production systemlike construct called the tutoring rule (Clancey, 1982) controller. The basic unit of the controller is the tutoring rule (t-rule) whose execution determines the exact behavior of the system. The selection of which t-rule will fire is a consequence of:

1. The state of the ideal model as it computes a solution to the given problem.
2. The content of the "active set" of t-rules that restricts the full set of t-rules known to the system down to the few that are applicable to the given state of the ideal model's execution.

Assuming, for the simple case, that the solution path of the given problem is strictly linear (i.e., at each production firing there is only one production that will be appropriate), the action of the student model will be a sequence of

production firings. Each t-rule specifies a single student model production that will trigger it. The t-rule controller will start from the current point on the solution path and scan downward until the first occurrence of a student model production that is specified by one of the t-rules in the active set. This allows the instructional designer to exclude any number of intermediate problem-solving steps from the tutorial dialogue. Subsequent cycles of the controller will then proceed from that production.

The execution of a t-rule has the effect of:

1. Formatting and sending output or soliciting input via the interface.
2. Determining the active set of t-rules for the next system cycle.

In the case of an output t-rule, the active set is specified unconditionally while an input t-rule determines the active set based on the nature of the learner's response. When the input t-rule fires, the input from the student is fitted into any of several t-rule categories. Each category has a set of t-rules associated with it and that set becomes the active set for the following cycle of the t-rule controller. Typically, one of these categories will correspond to correct inputs, another to requests for help from the student, and the rest to the various kinds of errors that are predicted by the student model at that particular point on the solution path. As an example of this process, take the current equation to be 3(X + 2) = 14. If the ideal model productions are the four productions in Fig. 11.2, then the following sequence of firings will result:

Working memory: 3(X + 2) = 14
  P1: num binds to 3, term1 binds to X, term2 binds to 2
Working memory: Distribute 3 over X and 2
  P2: bindings stay the same
Working memory: Multiply X by 3
  P3: num binds to 3, term binds to X
Working memory: Multiply 2 by 3
  P3: num binds to 3, term binds to 2
Working memory: Combine 3X and 6 with a +
  P4: term1 binds to 3X, term2 binds to 6
Working memory: 3X + 6 = 14

The t-rules in Fig. 11.6 specify a possible tutoring strategy for distribution, working with the ideal model productions in Fig. 11.2. To start this example, the active set of t-rules must be initialized as {T1}. Given the current expression, the current ideal model production to apply is P1. (This simple example contains only one solution path and no buggy rules. Only a single ideal model rule is in the conflict set at any point.) The tutor will go through the following states:

Name: T1
Production: P1
Action Type: Output
Action Body: Print "What operation do you want to perform?"
Active Set Transition: {T2}

Name: T2
Production: P1
Action Type: Input
Action Body: Bring up a menu of choices.
Active Set Transition: if student input is
DISTRIBUTION → {T3}
anything else → {T4}

Name: T3
Production: P2
Action Type: Output
Action Body: Print "Correct."
Active Set Transition: {T5}

Name: T4
Production: P1
Action Type: Output
Action Body: Print "Wrong. The right answer was DISTRIBUTION."
Active Set Transition: {T5}

Figure 11.6 T-rules to Be Used with the Ideal Model Rules in Fig. 11.2

Since the current production is P1 and the active set includes T1, and T1 is associated with P1, T1 will fire:

The tutor types "What operation do you want to perform?"

Active Set becomes {T2}, T2 is associated with P1, so now T2 fires.

The tutor brings up a menu of choices and waits for the student to select one of them. Assume that the student chooses DISTRIBUTION.

Active Set becomes {T3}. P1 finishes firing and P2 becomes the current production. Then T3 is selected as the next tutoring rule, and fires.

Name: T5
Production: P2
Action Type: Output
Action Body: Print "What is the result of distributing "num " over
"term1 + term 2"?"
Active Set Transition: {T6}

Name: T6
Production: P4
Action Type: Input
Action Body: Make the student enter an algebraic expression.
Active Set Transition: if student input is
term1 + term2 → {T7}
anything else → {T8}

Name: T7
Production: P4
Action Type: Output
Action Body: Print "Correct."
Active Set Transition: Empty set (End tutoring session)

Name: T8
Production: P4
Action Type: Output
Action Body: Print "Wrong. The correct answer was" term1 + term2
Active Set Transition: Empty set (End tutoring session)

Figure 11.6 T-rules to Be Used with the Ideal Model Rules in Fig. 11.2 (cont.)

The tutor types "Correct."

Active Set becomes {T5}, T5 is associated with P2, so the tutor will fire ideal model productions until the current production is P2 and then T5 will fire.

The tutor types "What is the result of distributing 3 over X + 2?"

Active Set becomes {T6}, T6 is associated with P4 which is now the current ideal model production so that T6 fires.

The tutor brings up the scratch pad and lets the student enter an algebraic expression.

Suppose the student enters 3X + 2 (an error). Active Set becomes {T8}. Since T8 (the "catchall" specified in T6) is associated with P4, T8 will now fire.

The tutor types "Wrong. The correct answer was 3X + 6."

Active Set is now empty and the tutoring interaction concludes.

What happens when more than one ideal model rule applies at the same time? What sort of tutoring strategies are possible in such a case? How must the t-rule controller be configured to implement them? Assume that we have the equation $3(X + 2) = -2(4 - 2X)$. The student model may allow either of the two distributions to be performed first. A function of any tutor must be to find out which operation the student wants to perform first. One can accomplish this goal by posing the appropriate question to the student and using the student's response to specify the branching in the ideal model production solution path. Before the exact path determination is made, the controller has to execute a parallel sequence of t-rule firings for each branch in the solution path. Moreover, the output produced by each parallel sequence of t-rules has to be the same, or the tutor would be faced with a dilemma: which of the t-rules being executed in parallel should determine the behavior of the system?

Here is what would happen if we used the ideal model and the t-rules of the previous example on an equation with two places to perform distribution:

Current equation is 3(X + 2) = 2(4X + 2).

BRANCH-1: distribute 3(X + 2)

The tutor types "What operation do you want to perform?"

The tutor brings up a menu of choices and waits for the student to select one of them. Assume that the student chooses DISTRIBUTION.

The tutor types "Correct."

The tutor types "What is the result of distributing 3 over X + 2?"

The tutor brings up the scratch pad and lets the student enter an algebraic expression. Suppose the student enters 3X + 2 (an error).

The tutor types "Wrong. The correct answer was 3X + 6."

BRANCH-2: distribute 2(4X + 2)

The tutor types "What operation do you want to perform?"

The tutor brings up a menu of choices and waits for the student to select one of them. Assume that the student chooses DISTRIBUTION.

The tutor types "Correct."

The tutor types "What is the result of distributing 2 over 4X + 2?"

The tutor brings up the scratch pad and lets the student enter an algebraic expression. Suppose the student enters 4X + 2 (an error).

The tutor types "Wrong. The correct answer was 8X + 4."

Note that the first three t-rule firings result in identical tutor behavior, so that it is impossible to distinguish which path the student is currently pursuing. At the fourth t-rule firing the tutor is faced with a dilemma: it is not clear which ideal model production path to follow and the tutoring strategy is giving ambiguous instructions. This means that the tutoring strategy outlined above is somehow deficient, that is, it won't handle a multiple distributions situation. One could enrich it by adding the t-rules shown in Fig. 11.7 that make the student specify the part of the equation that should be distributed. With the addition of such t-rules the tutoring interaction would be:

The tutor types "What operation do you want to perform?"

The tutor brings up a menu of choices and waits for the student to select one of them. Assume that the student chooses DISTRIBUTION.

The tutor types "Correct."

The tutor types "What part of the equation do you want to distribute?"

The tutor brings up the scratch pad and lets the student enter an algebraic expression. Suppose the student enters 2(4X + 2).

At this point the tutor knows that the student is pursuing branch 2 of the solution path and no subsequent ambiguity will arise.

The tutor types "Correct."

The tutor types "What is the result of distributing 2 over 4X + 2?"

The tutor brings up the scratch pad and lets the student enter an algebraic expression. Suppose the student enters 4X + 2 (an error).

The tutor types "Wrong. The correct answer was 8X + 4."

So, all in all, the t-rule controller has the flavor of a very simple programming language that has input and output capabilities but no ability to manipulate data. It has access to the information generated by the student model in the form of the specific production firings and the information gained by the binding of student model production system variables during the execution of the production system. However, it cannot do things like bind new variables, increment counters, and so forth. Thus control of flow is similar to the control execution for a finite state automaton where there are a finite number of transitions from any particular state of the system. The equivalent of "state" for the t-rule Controller is each t-rule which is capable of transitioning to certain other t-rules, the exact successor being determined by the productions fired by the student model and the behavior of the tutored student.

Name: T4-A
Production: P1
Action Type: Output
Action Body: Print "What part of the equation do you want to distribute?"
Active Set Transition: {T4-B}

Name: T4-B
Production: P1
Action Type: Input
Action Body: Make the student enter an algebraic expression.
Active Set Transition: if student input is
num (term1 + term2) → {T5}
anything else → {T4-C}

Name: T4-C
Production: P1
Action Type: Output
Action Body: Print "Wrong. " num(term1 + term2) " is the expression to apply distribution to."
Active Set Transition: {T5}

Figure 11.7 Additional t-rules to those in Fig. 11.6 which would have the student specify the part of the equation that should be distributed. These t-rules would be inserted by changing the active set transition pointer in T3 and T4 from T5 to T4-A. These are to be used with the ideal model rules in Fig. 11.2.

We have not yet considered the exact method for entering t-rules into the system. The system modularity/efficiency constraint requires that the t-rule controller, the interface, and the student model be specified independently from any given tutoring strategy and be able to implement a large number of different strategies. It is also important to have the t-rule entry process be as simple as possible. To avoid the necessity of a special syntax and editor for entering tutoring rules, we have taken a "learning-by-example" approach: The human instructional designer/teacher determines the tutoring interactions by specifying concrete examples in the act of tutoring simulated students on the system. This is detailed in the next section.

The Rule Induction Component: Induction of Tutoring Strategy by Example

The goal of being able to implement new tutoring strategies for a skill quickly and simply is best met when you minimize the diversity and specificity of the knowledge needed by the instructional designer or teacher (referred to hereafter as the implementor) who is implementing that strategy. This means that programming skills should not be requisite, nor should the implementor have to be familiar with the syntax and the internal details of the production system that underlies the student model. Another feature that should speed up the implementation process is an interface that programs the tutoring component interactively rather than via an indirect process such as the edit/compile/execute cycle that one finds in more conventional programming. These two goals may be achieved by making a graphics-based authoring system available to the implementor that, in an uncomplicated and straightforward manner, allows the implementor to determine the tutor's action at any specific instant in the tutoring process.

Our attempt at reducing the load on the implementor involves building an authoring system that depends on having the implementor play out the roles of the tutor and an ideal student (one that gets every answer correct) within the context of a particular problem. The authoring system translates the implementor's surface behavior into a set of general tutoring rules that will serve to specify a tutoring strategy for all problems utilizing the same student model productions as the original problem. The induced t-rules should faithfully reproduce the behavior that the implementor gave as an example.

Tutoring behavior can be divided into the categories of output and input. An output t-rule must specify the content of the output as well as its destination (prompt window or history window) and its form (font, background shade, and other appearance features). Input of arbitrary algebraic expressions and equations occurs via the scratch pad. Multiple choice decisions can be presented and choices input via menus. The content of a particular tutoring interaction can be further subdivided into the problem-independent and problem-dependent categories. If the system is tutoring on the solution of $2(X + 1) = 4$ and the tutor prints the question: "What is 2 distributed over X + 1?" the "2" and the "X + 1" are problem dependent, while the form of the question (the exact wording used) is problem independent; that is, it will be the same for any tutored problem solved in the same way as the "$2(X + 1) = 4$" problem.

Since the tutor deals with algebraic problems, the problem-dependent components will be algebraic expressions (algebraic fields of t-rules). In specifying these algebraic fields the t-rule controller utilizes the names of production system variables that are bound during the student model's solution of the problem. The implementor has access to the ideal student model and its variable bindings via a set of protocols/comments that the system generates as it runs

the production system simulation. These protocols contain an English explanation of what each production firing accomplished and lists the algebraic fields that the production utilized. The implementor must choose the ideal model rule that corresponds to that particular interaction before specifying any sample behavior. The implementor does this by indicating to the authoring system one of the protocol steps that narrate the problem solution. For instance, if the implementor wants the tutor to ask the question "What is the result of multiplying 1 by 2?" he or she should associate that question with a protocol that describes a production that instantiates this operation. When the time comes to give the actual content of a tutoring interaction, the algebraic fields detailed in the selected protocol can be used by the implementor to specify algebraic expressions to include in the tutoring interaction. Thus, in the example above, "2" and "1" would be expressions that the implementor could select from the body of the protocol rather than specifying them arbitrarily. Such a selection indicates to the authoring system that these expressions are problem dependent and should be simulated by production system variables bound during the execution of the ideal model.

Another feature of this process of associating tutoring interactions with ideal model rules is that not every ideal model rule has to be tutored. In fact, by skipping intermediate steps the implementor has control over the grain size of the tutoring strategy that is being implemented.

If there are multiple solution paths for the given problem, then there will be multiple protocol traces of these production firings. Implementors can specify the path that they want to follow as they detail the tutoring strategy for the current problem. In particular, this means that if there are multiple algorithms for solving a problem, implementors have to indicate to the system which of these algorithms they are assuming to underly the current tutoring strategy.

The following is an example of the induction process operating on a portion of one set of ideal student productions that deal with solution of linear equations. N and N1 are variables that match any positive integer. V is a variable that matches any algebraic variable. T1 and T2 match to any term, and SIGN matches either + or −

```
P1: N (T1 SIGN T2) --> COMBINE (+ DISTRIBUTE (N , T1) ,
                                SIGN DISTRIBUTE (N T2)
                                )
P2: N (- T1 SIGN T2) --> COMBINE (- DISTRIBUTE (N , T1) ,
                                  SIGN DISTRIBUTE (N T2)
                                  )
P3: DISTRIBUTE (N , N1) --> (TIMES N N1)
P4: DISTRIBUTE (N , V) --> N V
```



```
P5: DISTRIBUTE (N , N1 V) --> (TIMES N N1) V
P6: DISTRIBUTE (N , - N1) --> - (TIMES N N1)
P7: DISTRIBUTE (N , - V) --> - N V
P8: DISTRIBUTE (N , - N1 V) --> - (TIMES N N1) V
P9: COMBINE (+ T1 , + T2) --> T1 + T2
P10: COMBINE (+ T1 , - T2) --> T1 - T2
P11: COMBINE (- T1 , + T2) --> - T1 + T2
P12: COMBINE (- T1 , - T2) --> - T1 - T2
```

The current problem is 2(X + 1) = 4, and it is up to the implementor to demonstrate a strategy for distributing a positive number over a two-term expression. What follows is the sequence of ideal model firings and the resultant modifications to working memory.

```
Equation 0:    2 (X + 1) = 4
P1 fires       N (T1 SIGN T2) --> COMBINE (+ DISTRIBUTE (N , T1) ,
                                           SIGN DISTRIBUTE (N , T2))
   Bindings:   N = 2, T1 = X, SIGN = +, T2 = 1
Equation 1:    COMBINE (+ DISTRIBUTE (2 , X) , +
                        DISTRIBUTE (2 , 1)) = 4
P3 fires       DISTRIBUTE (N , V) --> N V
   Bindings:   N = 2, V = X
Equation 2:    COMBINE (+ 2 X , DISTRIBUTE (2 , 1) ) = 4
P4 fires       DISTRIBUTE (N , N1) --> (TIMES N N1)
   Bindings:   N = 2, N1 = 1
Equation 3:    COMBINE (+ 2 X , + 2) = 4
P9 fires       COMBINE (+ T1 , + T2) --> T1 + T2
   Bindings:   T1 = 2 X, T2 = 2
Equation 4:    2 X + 2 = 4
```

Below are samples of the protocols for these productions that the system would present to the implementor.

```
P1 Fires:  (Annotation)
    Solving 2(X + 1) = 4 necessitates a distribution operation.
    Set the goal to distribute 2 over X.
    Set the goal to distribute 2 over 1.
    Combine the resulting terms, both of which will be positive.
    Algebraic fields available: 2 , X , 1 , +
```

P2 Fires: (Annotation)
Distributing 2 over X.
The result is 2 X.
Algebraic fields available: 2, X.

P3 Fires: (Annotation)
Distributing 2 over 1.
The result is 2.
Algebraic fields available: 2, 1, 2.

(Note: the second "2" is specified as the result of the multiplication. N * N1 will not always equal N, so this apparent redundancy is no redundancy at all.)

P9 Fires: (Annotation)
Combining + 2 X and + 2
The result is 2 X + 2
The equation is now 2 X + 2 = 4.
Algebraic fields available: 2X, 2.

Let us assume that the implementor wishes to specify a strategy that forces the student to take three steps in three interactions:

1. Establish the operation to be performed.
2. Specify the multiplier and the two terms.
3. Specify the answer.

For the first of these interactions the implementor would indicate the first protocol, since it is at that point that the distribution operation is selected. The implementor would then have the tutor print the question "What operation should you perform now?" on the blackboard and then specify a menu that contained the label "DISTRIBUTION" along with several erroneous distractor menu items. The implementor would indicate that the answer "DISTRIBUTION" was the correct one.

The tutoring interactions that asked for the two terms would also be associated with the first production. The answer to "What are the terms of the expression?" would be answered by X and 1. The implementor could specify the "X" and the "1" by selecting appropriately from the list of algebraic fields available in the protocol of the first production.

The last production would be chosen to correspond to the third interaction. In response to the question (which the implementor specifies to appear on the blackboard) "What is the overall result of the distribution?" the tutor will expect the correct answer 2X + 2. The implementor could indicate "2X + 2" to the tutor by using the algebraic fields of the fourth protocol (they are 2X and 2) and one plus sign, which is actually invariant because the production, COMBINE {+T1 , +T2} → T1 + T2, fires only when positive terms are combined.

After entering the strategy for the ideal case, the implementor has to indicate the tutoring behavior for the case of incorrect student responses. This is done in much the same way as the specification for the correct strategy. For each production in the solution path there may be several bugs known to the system (each of these buggy productions is available as an improper variant of the correct production) that may fire at that point. The buggy productions would have a protocol associated with them. The role of the implementor would be to select the buggy production's protocol, to indicate the response made by a buggy student, and to specify the tutoring strategy to remediate such an error.

The following is an example of a buggy production that is a variant of P1 from the immediately previous set of ideal model productions.

P1* (buggy version)
N (T1 SIGN T2) → COMBINE (+ DISTRIBUTE (N , T1) , SIGN T2)

The buggy production simulates a student who fails to distribute the multiplier over the second term (the failure-to-iterate bug, Matz, 1982). In the context of our previous example—solving 2 (X + 1) = 4—the student would derive 2X + 1 = 4. A student who followed the bug would behave exactly as an ideal student in the example interactions outlined above, except for the final question ("What is the final result of this distribution operation?") to which the student would respond with "2X + 1." The implementor would simulate the bug for the system by selecting the protocol that corresponds to the buggy production, and then enter a desired remediation strategy after the student errorfully enters "2X + 1" in response to the last question of the distribution interaction.

The implementor also has to indicate the tutor's strategy for the case when the student makes an error that is not predicted by the error model, or the case when the student asks for help. There are several such strategies: give the right answer and go on, give a couple of hints before continuing, loop *N* times and then continue, and so on. This is left to the implementor's choice.

A working induction system promises to be a real boon to developers of tutors. Once the cognitive skills and their attendant errors have been simulated via a student model, building a tutor becomes the simple task of giving example interactions for a set of problems (in the case of our work the size of this set is on the order of 100 problems).† This is a process of days and not months, the time previously required to implement an intelligent tutor. The process by which the savings are gained is analogous to the savings granted by a state-of-the-art authoring system to the designer of CAI curricula: much of the work is done for the implementor. The induction component of the TEACHER'S APPRENTICE is an authoring system for intelligent tutors.

† We foresee such a system being delivered to the teacher or district with complete set of tutoring strategies encoded. These would then be amenable to change and refinement as the district or teacher saw fit.

The Pragmatics of Integrating ICAI into the Classroom

Our regular observations of an algebra class coupled with interviews of both geometry and algebra teachers have provided valuable data regarding the pragmatics of integrating intelligent tutoring systems into public school classrooms. A brief account of the insights gained by Frank Boyle and the geometry tutoring project as they took the final steps of getting the geometry tutoring system out of the lab and into the classroom is illustrative of this point.*

A tutoring system for geometry proofs has been developed in the lab and has been both piloted (Anderson, Boyle, & Yost, 1985) and used in a controlled study. The results were very encouraging: high school students were learning not only how to solve proofs but also what it means to prove a statement. They also felt positively about the experience of learning with the tutoring system. The next step was to put the tutors into a classroom. What was learned about how geometry is taught in the classroom is that proofs are very rarely assigned as classwork! Proof problems are assigned as homework. Enter the pragmatics of the classroom: If students in a standard class are given the task of solving some proof problems, there will be some fairly large proportion who will get stuck very early in the problem. These students, being stuck and frustrated, will be likely to disturb others and generally get into trouble. Hence teachers solve examples on the board and lecture in class but reportedly seldom assign proof problems.

In order for the geometry system to be useful as a tutor of problem-solving skills, teachers must change their style of teaching and assign problems during class. We hope the pedagogical effectiveness and motivational aspects of quality intelligent tutoring systems outweigh the cost to the teachers involved in changing their teaching style. This is also a nice illustration of how interactive teaching systems can accomplish multiple goals of tutoring problem-solving skill on line in the classroom and reducing the effective student/teacher ratio. If the systems can engage the students and keep them working productively (as would a good human tutor) then teachers have more options for alternative classroom structures. They would have to spend less time policing behavior.

However, the idea of changing the structure of a high school algebra or geometry class is a radical change to the normal culture of mathematics classes. A case in point is that the algebra class we have been observing contains a commercial CAI system that can accommodate eight students at a time. How is such a limited resource to be used? Should teachers rotate students through the system in three groups? What happens to the time it takes to switch students in and out of the work stations? And if you have students grouped, how do you take best advantage of the reduced student/teacher ratio? Many high school teachers apparently are not specifically trained in techniques of creating and

* The geometry tutoring system went into a classroom in the Pittsburgh public schools in January 1986 and is currently under evaluation.

using groups in classrooms or in techniques of using centers and minimizing transition time. However, many elementary school teachers are trained in such techniques. With the advent of intelligent tutors in the classroom there may need to be additional training of existing teachers and possibly additions or changes to the skills taught in high school teacher education programs.

We currently envision that a one-to-one ratio of machines to students will be necessary in classrooms. This is the ratio at the current site of the geometry tutor test classroom. However, use of intelligent tutors may fit more effectively into a "learning laboratory" type setting. Students have time allotted to go to the room where the machines are and use them at their own pace as an activity separate from the classroom. There is a precedent for this type of structure in existing reading and language laboratories in high schools.

Interesting Possible Side Effects from Developing ICAI

Possible interesting side effects of developing ICAI for mathematics may come in the area of educating teachers. A system that can emulate a teacher's style allows the possibility for a teacher to watch this emulation teach either simulated or real students and reflect on their personal pedagogical decisions and strategies. Observation and reflection is different from watching a videotape of oneself teaching. Although observing oneself teaching can be quite informative and constructive, it is difficult to step out of the context one was in when the taped events were taking place. This biases the actors' observations. It is often necessary to have another person watch the tape and make observations or ask questions to get other perspectives on the interaction.

Being able to step back and watch one's personal strategies being applied faithfully to new examples might allow the distance necessary to see incongruities in one's style, such as treating certain cases of problem solving differently than other equivalent cases. A simple example is how distribution might be handled. It is possible that some teachers might want to teach separate rules for how to do distribution of a negative sign over a parenthesized expression, $-(x + 3)$, and for how to do distribution of an integer over a parenthesized expression, $-2(x + 3)$. Because teachers might want these as separate cases, we can put in separate rules for these cases. The first case might be tutored as "change the signs of all the terms inside the parentheses" and the second case might be tutored with the iterative procedure for doing distribution. It is possible that a teacher might observe these two cases being tutored differently and decide that they might be more effectively tutored as cases of the same iterative algorithm.

This is also a nonthreatening way to observe another teacher's teaching style and possibly gain some insights, language, strategies, or tricks. One could imagine groups of teachers cooperating to build different tutoring models that could then be tested on students in their classrooms. In this way teachers could

be included in the loop of instructional research: their classroom-earned expertise could be collected and shared with other teachers and researchers. An interesting research question involves the possibility of collecting sets of tutoring rules that have been developed by teachers and have proved effective in classrooms. These sets could be the object of research to deduce why such sets are effective. There is clearly more potential for ICAI than just the benefits to end-user learners in the classroom.

Settings Where Math ICAI Might Be Effective outside the Standard High School Environment

Remedial math students are another potentially large population for whom mathematics ICAI might be useful. A local university reports that 40% of the incoming first-year students take some course in remedial algebra. The military also has an extensive need to do rapid remedial math training for incoming recruits. This older population may have different motivations for pursuing mathematics, and the interaction with a human teacher may be less critical than we perceive it to be in a standard high school class. Our experiences with a population of adults taking remedial algebra at a local junior college suggest that there is a fairly large subset of these students that don't want to have humans around to tutor them. They would rather, in many cases, keep their mistakes to themselves. The other population where intelligent tutors might be recommended for use without a human teacher in attendance would be children in isolated environments due either to health or to geographical constraints.

How the TEACHER'S APPRENTICE Differs from Traditional CAI

It is important to ask the question of how various ICAI systems differ from or improve upon the technology of quality computer-assisted instruction; that is, to perform a rational comparison, free of "straw men." Where is the "intelligence"? We acknowledge the observation that the surface output of the TEACHER'S APPRENTICE strongly resembles that of standard CAI: prompts, responses that are matched, and branching to new prompts. What this surface similarity does not reveal is the major difference that underlies the surface behavior: generative ability. In fact, it is possible to hand-program CAI that would exactly match any behavior of the TEACHER'S APPRENTICE. However, it would take huge amounts of programming time to map out the problem-solving path in any detail and specify the responses at each point for every problem in a reasonably large problem library. In the TEACHER'S APPRENTICE, the ideal model's trace and a tutorial interaction can be generated on-line for arbitrary problems: there is not a library of problems, but a library of knowledge. In addition, the ability to generate errors relieves the need to pre-establish all specific errors and all the contexts in which they will occur. So, if there is "intelligence" in the system, it lies in the ability to generate tutorial instruction.

Conclusions

This chapter began with a set of very general goals for ICAI in general, and for the TEACHER'S APPRENTICE project specifically. After describing the various components of our system and their interaction, it is now time to recap how the system attempts to fulfill the goals laid out initially.

Goal 1: Make Intelligent Tutors Effective

We have designed and are implementing an intelligent tutor based on a learning theory: ACT*. Based on the learning theory, certain principles for how to do effective tutoring have been enumerated. These principles are embodied in the design of the interface, the construction of the ideal and buggy models of the skill, and in both the timing and content of feedback students receive when errors are made.

Goal 2: Make Intelligent Tutoring Systems Available

We have attempted to cut the development time and cost of bringing up new tutors by attempting to keep the components of the system modular and independent. By maintaining modularity, it should be possible to create new tutors for related domains of mathematics such as Algebra II and calculus with reduced effort. A new student model and set of tutoring rules would need to be developed, but the tutoring control, induction, and interface components would theoretically not need to be altered a great deal.

We are also experimenting with techniques to speed up the execution of the ideal model. Letting the ideal model solve the problems before the tutorial interaction begins allows the generation of a search space for the correct solutions of the problem which can then be quickly searched "on-line" during the interactions. The only rules that would have to run in real-time would be the buggy rules.

One counterpoint to the "availability" issue is that the system currently runs on a class of very powerful and very expensive machines. We are targeting this system for the machines of the early 1990s. Reports from hardware companies and the general trend of the hardware industry support the belief that machines capable of running our software will be in the affordable range for schools by the early 1990s.

Goal 3: Make Intelligent Tutors Usable

We are building a system that will be amenable to different problem-solving and teaching styles by emulating individual teachers' language and grain size of instruction. We have taken the position that we cannot second-guess what teachers currently want, nor what they will want in the near future. We also do not want to dictate how tutoring interactions should take place: different teachers might teach subtly different algorithms and use different language for math entities and concepts because of regional differences or differences in training. By giving teachers a flexible tool we hope to maximize the probability that this new technology ends up being integrated into the teaching style of the individual teachers and hence better fits into the culture of the classroom.

References

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review, 89,* 369–403.

Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Anderson, J. R. (1985). Production systems, learning, and tutoring. In D. K. Klahr, P. W. Langley, & R. Neches (Eds.), *Self-modifying production systems: Models of learning and development.* Cambridge, MA: Bradfor Books/MIT.

Anderson, J. R., Boyle, C. F., Farrell, R. G., & Reiser, B. J. (In press). Cognitive principles in the design of computer tutors. In P. Morris (Ed.), *Modelling cognition.* New York: Wiley.

Anderson, J. R., Boyle, C. F., & Yost, G. (1985). The geometry tutor. *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 1–7). Los Angeles, CA.

Anderson, J. R., & Reiser, B. J. (1985). The LISP Tutor. *Byte,* 159–175.

Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher, 13,* 3–16.

Brown, J. S. (1983). Process versus product: A perspective on tools from communal and informal electronic learning. In *Report from the Learning Lab: Education in the Electronic Age.* New York: WNET Educational Broadcasting Corporation.

Brown, J. S., & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science, 2,* 155–192.

Brown, J. S., & van Lehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science, 4,* 379–426.

Bundy, A. (1983). *The computer modelling of mathematical reasoning.* London: Academic Press.

Bundy, A., & Welham, B. (1981). Using meta-level inference for selective application of multiple rewrite rule sets in algebraic manipulation. *Artificial Intelligence, 16,* 189–212.

Clancey, W. J. (1982). Tutoring rules for guiding a case method dialogue. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 201–225). New York: Academic Press.

Davis, R. B. (1975). Cognitive processes involved in solving simple algebraic equations. *Journal of Children's Mathematical Behaviour, 1*(3), 7–35.

Davis, R. B., Jockusch, E., & McKnight, C. (1978). Cognitive processes in learning algebra. *Journal of Children's Mathematical Behaviour, 2*(1), 10–20.

Lewis, C. (1981). Skill in algebra. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 85–110). Hillsdale, NJ: Lawrence Erlbaum Associates.

Malone, T. W. (1981). Toward a theory of intrinsically motivating instruction. *Cognitive Science, 4,* 333–369.

Matz, M. (1982). Towards a process model for high school algebra errors. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 25–50). New York: Academic Press.

McArthur, D. (1985). Developing computer tools to support performing and learning complex cognitive skills. In D. Berger, K. Pedzek, & W. Bankes (Eds.), *Applications of cognitive psychology: Computing and education.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Reiser, B. J., Anderson, J. R., & Farrell, R. G. (1985). Dynamic student modelling in an intelligent tutor for LISP programming. *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 8–14). Los Angeles, CA.

Siegler, R. S., & McGilly, K. (In preparation). Cognitive processes in factoring algebraic expressions. Carnegie-Mellon University.

Sleeman, D. (1982). Assessing aspects of competence in basic algebra. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 185–199). New York: Academic Press.

Sleeman, D. (1984). An attempt to understand students' understanding of basic algebra. *Cognitive Science, 8,* 387–412.

Wagner, S., Rachlin, S. L., & Jensen, R. J. (1984). *Algebra learning project final report.* Athens, GA: Department of Mathematics, University of Georgia.