A RAND NOTE

N-2595-NSF

Learning Problem-Solving Skills in Algebra

David McArthur, Cathleen Stasz, John Y. Hotta

May 1987

Prepared for
The National Science Foundation
A RAND NOTE

Learning Problem-Solving Skills in Algebra

David McArthur, Cathleen Stasz, John Y. Hotta

May 1987
The work described in this report was sponsored by the National Science Foundation under Grant No. MDR-8470342.

This Note contains an offprint of RAND research originally published in a journal or book. The text is reproduced here, with permission of the original publisher.

The RAND Publication Series: The Report is the principal publication documenting and transmitting RAND's major research findings and final research results. The RAND Note reports other outputs of sponsored research for general distribution. Publications of The RAND Corporation do not necessarily reflect the opinions or policies of the sponsors of RAND research.

Published by The RAND Corporation
1700 Main Street, P.O. Box 2138, Santa Monica, CA 90406-2138
LEARNING PROBLEM-SOLVING SKILLS IN ALGEBRA*

DAVID McARTHUR
CATHLEEN STASZ
JOHN Y. HOTTA

The RAND Corporation

ABSTRACT
In this article we describe aspects of our intelligent tutor for basic algebra. A main goal of the project is to develop a computer tutoring system whose skills and knowledge approximate those of a high-quality human tutor. We are particularly interested in exploring novel learning opportunities that can be made available to students for the first time by exploiting the reactive capabilities of such intelligent tutors. In this context, we focus here on the role of an algebra expert system embedded in the tutor. We discuss how it can be used to help students learn several nontraditional types of skill and knowledge in the context of algebra, including goal-directed reasoning skills, and debugging techniques.

For the past year, we have been developing an intelligent tutoring system to help students learn basic algebra. Our goal is to provide a computer tutor whose skills and knowledge approximate those of a high-quality human tutor. The previous generation of computer-aided instruction (CAI) programs possess little real knowledge of the subject they teach. In contrast, ours embeds several types of knowledge:

1. Knowledge of the subject — The tutor is an expert problem solver in basic algebra. It embeds an inspectable algebra expert system, capable not only of solving all the algebra problems we expect the students to solve, but also able to produce intermediate reasoning steps similar to those of an "ideal" student.

* This research is being funded by the National Science Foundation (Applications of Advanced Technologies Program).

2. **Knowledge of the student** – The tutor is being built to understand which concepts are most difficult for students, and will be able to detect and help remedy student misconceptions. This is the job of the student diagnosis module.

3. **Knowledge of teaching** – The tutor is being designed to include pedagogical techniques that make difficult concepts easier to understand. These techniques also determine what information to provide, and when to begin learning.

While this information is prerequisite to producing high-quality tutoring systems, it is only the raw material. It must be used appropriately to help students learn in the most effective ways. Our philosophy, in deciding how to deploy this knowledge, has been first to identify specific key skills that we believe students learning algebra should learn, and then to design equally specific learning tools that support the cognitive processes involved in learning the target skills.

In this article, we discuss the target skills and knowledge that our tutoring system attempts to help students learn. We also describe in detail how one of the basic knowledge sources of the system, the inspectable algebra expert, is being used as a basis for a variety of learning tools that aid the student in learning the target skills. The next section identifies the skills we are targeting for student learning. The following section gives an overview of the components of the system as it now stands, focusing especially on the expert system embedded in the tutor. The final section follows with a discussion of specific facets or tools embedded in the tutor that promote learning of the target skills.

**KNOWLEDGE GOALS FOR BASIC ALGEBRA**

The design of many aspects of our tutor has been influenced by a simple philosophy we have towards the development of educational software. Our view has two principles:

- Identify specific skills, concepts, or knowledge that you want to communicate to the student.
- Design computer-based learning tools that support the specific cognitive processes involved in learning those skills through practice.

McArthur discusses how many CAI programs fail to show these principles, and also describes in detail how these principles can help design some simple yet effective computer-based learning tools in several different domains [1]. In this article we focus specifically on the field of basic algebra.

Our first principle says to identify the specific knowledge and skills we want students to learn. Of course we want them to learn how to solve algebra
problems, but what are the specific pieces of knowledge and skills that comprise algebra competence? A standard answer to this question might be:

- Students should know the important “laws” of algebra, such as the distributive property for multiplication over addition. We refer to this as axiom-level knowledge; and
- Students should know algorithms or procedures for solving various classes of problems.

To judge from high school algebra textbooks, axiom-level knowledge and procedures are regarded as the most important for students to understand. However, our view is that while this knowledge might be sufficient for solving basic algebra problems, this is not the only important reason for learning algebra. Indeed, since computer programs for doing symbolic algebra are rapidly becoming accessible to all students, the mechanical symbol manipulation skills of algebra are becoming progressively less important to learn. Conversely, we believe that computers, when combined with ideas from artificial intelligence and cognitive psychology, open up opportunities for learning several new and important types of knowledge in the field of algebra.

The kinds of understanding we have chosen to focus on can be described as algebra problem-solving knowledge. We distinguish several types of problem-solving knowledge that students can learn. First:

Students should know goal-directed reasoning skills. Such skills enable the student to break down problems into smaller subproblems recursively, until a piece axiom-level knowledge can be selected to satisfy a particular goal.

Goal-directed reasoning skills will be discussed more precisely when we describe our inspectable algebra expert for solving equations.

On several grounds, we distinguish such knowledge from the common algorithms or procedures often taught to students in the classroom. First, such procedures are often “cookbook” methods that give the impression that algebra problem solving always involves a fixed set of steps. The goal-directed reasoning skills we refer to are true problem-solving techniques: They selectively search for patterns in questions that suggest certain goals or approaches. Different patterns will suggest a different order of operations in solving a problem. Second, cookbook procedures generate only one line of solution, often not the most efficient or elegant. Expert reasoning knowledge is flexible. It can generate many different solution lines to a given problem. Third, cookbook procedures are often of limited generality. For example, some techniques for algebraic multiplication do not readily generalize beyond binomials. The goal-directed reasoning skills we refer to consist of very general problem-solving techniques, capable of seeing the underlying similarity of problems that seem superficially different.
Goal-directed reasoning knowledge is only one kind of problem-solving skill students can learn in the context of algebra. A second principle we have adopted is:

Students should know debugging and self-diagnosis skills. Here we refer to the abilities of good learners to be able to learn from their errors—to track down the location of the error, and to determine how to find the knowledge that will remedy the error.

Our own research shows that students' debugging skills are almost nonexistent in algebra. For example, they almost never attempt to verify their answers (e.g., by substituting their answer back into the original question). Moreover, when they make a mistake, students rarely attempt to find the specific reasoning step where the error is located, nor do they try to understand the nature of the error and fix it. We believe that the problem-solving skills involved in debugging are critical. They can make the difference between students who merely practice when doing homework questions, and those who learn from practice.

One reason teachers often focus on teaching axioms and procedures is simply that these concepts are well known. Conversely, problem-solving skills may not have been taught in the past because they have not been understood. However, this is no longer true. Research in cognitive psychology has uncovered many of the problem-solving skills of human experts [2-4]. This work is complimented by efforts in artificial intelligence that have formalized and implemented knowledge-based expert systems capable of reasoning solutions to complex problems in many fields [5-7]. In the area of basic algebra, in particular, we now have a relatively complete understanding of both the axiom- and problem-solving knowledge needed to be an expert problem solver (see for example, [8, 9]).

Because this knowledge is now well-understood and explicitly represented, there is no reason it cannot be communicated to students as effectively as the more traditional skills found in current text books. One of the main goals of our project is to develop and assess computer-based tools for helping the student learn these important skills, as well as to discover other new concepts that students might effectively learn in a computer-based algebra environment. In the following sections we give an overview of the algebra tutor, then describe the current tools it embeds for helping students learn effective algebra problem-solving skills.

OVERVIEW OF AN INTELLIGENT COMPUTER TUTOR FOR BASIC ALGEBRA

We are building and testing our algebra tutor in a series of versions. The first version of the tutor runs on the Symbolics 3600 Lisp machine, and on Sun Microsystems workstations. It has been tested at The RAND Corporation, using local high school students, and we are now in the process of installing six Sun 3/50s in Santa Monica High School, where it will be more extensively tested and developed over the next several years.
Figure 1. The algebra tutor interface. Shown are the several windows and menus that comprise the tutorial interface. Students can either input algebraic expressions using a mouse and menu selection; or they can write them by hand, using a tablet and pen.
Version 1 of the tutor has no sophisticated knowledge of the student, nor does it possess intelligent tutorial policies that determine how to present concepts and administer feedback. We have focused in this version on developing an "inspectable" algebra expert system, and exploring a variety of tutorial uses of such an expert to teach algebra problem-solving skills. Parallel with the implementation of the first version of the tutor, we have been cataloging standard student misconceptions in algebra, to provide the knowledge base for the student diagnoses module [9]. In addition, we have begun to conduct extensive interviews with expert algebra tutors [10]. The ultimate goal of this research is to build a fairly extensive cognitive model of the expert, detailing many of the pedagogical tactics and strategies that appear to enhance student learning. This information is providing the basis for the pedagogical module of the tutor.

The student sees the tutor as a collection of windows and menus, as shown in Figure 1. The large window in the upper right is the DisplaySpace, where the student's reasoning is recorded and queried. Problem solving is represented here as a reasoning tree. Each branch in the tree represents an alternate solution, or line of attack on the problem. A tree representation allows easy comparison of different solutions, both the student's and tutor's. The boxed equation represents the student's current focus of attention in problem solving. To the left of the DisplaySpace are several menus, comprising a set of options that permit the user to manage his or her reasoning, as well as allowing the tutor to assist the student in reasoning in various ways.

Below the DisplaySpace, on the right, is the CommentSpace, where the tutor sends textual feedback to the student. To the left is the WorkSpace where the student creates each new line in her solution. New lines or reasoning steps can be created in two ways. First, a student can type in equations using the keyboard. A carriage return or line-feed indicates to the tutor that the student has completed a step. The tutor responds by placing the new expression in the DisplaySpace.

The second method for creating a new reasoning step is to simply write it on an electronic tablet which recognizes the strokes as characters, and passes them to the tutor where they are interpreted as mathematical expressions. The expression appears in the WorkSpace as it is written. In this input mode, a new line is signaled in this mode by clicking a button on the pen. As with keyboard input mode, a new line causes the expression in the WorkSpace to appear in the DisplaySpace. We are using a Penpad\(^1\) for input of handwritten characters, and initial results look encouraging. The Penpad recognizes approximately 95 percent of characters, and places only a few constraints on student's penmanship.

Further, if the student writes something that the system misinterprets, he or she can simply write over the character.

\(^1\) The Penpad is a product of Percept Inc.
The Inspectable Algebra Expert System

Items in the menus enable the student and tutor to talk about the student's (and tutor's) reasoning steps, and the processes underlying those steps. The algebra expert system embedded in the tutor is primarily responsible for responding to these menu requests. Since these menu items are at the heart of the tutor's ability to support the student's learning of the problem-solving knowledge, in this section we describe the algebra expert's structure in some detail. The reader interested in a more complete discussion is referred to McArthur [9].

Our goal in building the algebra expert has been to construct a system that comes as close as possible to mimicking the behavior of the "ideal" algebra student. We place several constraints on the performance of the system:

- The system should generate the right answer.
- The system should produce intermediate reasoning steps that resemble students' steps, and that are comprehensible to them.
- The system should be flexible. It should be able to answer a question in several different ways, if there are several reasonable approaches to the problem.
- The system should be inspectable. The student should be able to access the problem-solving knowledge that the system used to solve a problem, and easily relate the system's knowledge to its performance. In other words, the student should be able to understand the rationale for the system's performance.

Each of these properties contributes to the tutor's overall ability to support the student in learning the problem-solving knowledge described in the preceding section. The current algebra expert system has been relatively successful in satisfying properties. It solves all equations in first semester algebra in a flexible, inspectable fashion.

Axiom-level knowledge — In order to meet the above performance demands, the algebra system includes several levels of knowledge, each playing a different role in problem solving. The lowest level of knowledge available to the system consists of primitive axioms that describe legal (truth-preserving) transformations of algebraic expressions. These axioms are primitive in the mathematical sense that they cannot be derived from any more basic ones. Such axioms include well-known "distributive," "associative," and "commutative" rules for multiplication and addition.

Primitive axioms comprise only a small portion of the useful knowledge algebra students have for transforming algebraic expressions. Even beginning students know, for example, that you can reorder the terms in an addition any way you want—usually to bring together all terms involving a variable. Thus, students given $3+2x+5-x$ are often observed to immediately write $2x-x+3+5$. 

Of course, this transformation can be obtained by a series of applications of the primitive axioms commutative and associative axioms, since rearranging is a logically valid operation. However, this is certainly not how the students operate. In fact, they may not even remember associative and distributive laws, while rearranging terms may be completely natural. For this reason we distinguish psychologically basic cognitive axioms from primitive axioms, and represent them explicitly.

Cognitive axioms are rooted in students’ remarkable pattern recognition abilities. Rearranging is so natural because it is simple for students to scan an addition, with any number of terms, and locate terms containing a variable, wherever they are located, and however embedded. Such capabilities cannot be captured by the traditional language for specifying algebraic axioms. Thus, to represent cognitive axioms, we have had to create a powerful pattern matching language that allows us to conveniently specify the highly abstract patterns that students are capable of recognizing.

Strategy-level knowledge — Collectively, primitive and cognitive axioms represent all the information students need to be able to perform any valid algebraic transformation, but they do not represent information about how to intelligently and selectively use this knowledge to solve problems. Random applications of the axioms will eventually solve any problem, but students’ searches are usually very selective, indicating a good deal of strategic or “meta” knowledge about how to use the axioms effectively. In order to produce solutions that approximate those of an “ideal” student, we have actually had to define several distinct types of strategy-level knowledge: strategic rules, compound strategic rules, rule types, and plans.

Strategic rules give advice about what axioms would be reasonable to use under various conditions. They are typically rules of thumb that have an informal English expression. For example a good rule for isolating an unknown is: “If you have an equation with an addition, and the variable is in the addition, and the equation has only one occurrence of the variable, you should move the term not including the variable to the other side of the equation.” As with many strategic rules, this heuristic mentions (a) a set of conditions that must be true of the current expression, (b) an action that must be taken, and (c) an expected result. In terms of the concepts we have discussed, the action to be taken is the application of a particular cognitive axiom. The set of conditions usually reduces to a collection of constraints on the values of the pattern variables of the cognitive axiom, and (possibly) a set of constraints on the expression as a whole.

To learn efficient problem-solving skills in algebra, students often need to know more than strategic rules supply. Strategic rules may say what the next reasonable action is; but students often notice sequences of useful operations which “go together” so frequently in reasoning that it is worth remembering
the sequence explicitly. To capture these sequences, we define compound strategic rules. These typically encode conjunctions of strategic rules. A good example is collecting like terms. In the simplest case, this involves only one strategic action, namely, applying distributivity (e.g., transforming $3x+5x$ to $(3+5)x$). However, usually the expression needs some initial processing. First, terms in an addition may have to be rearranged to make those involving the variable contiguous (e.g., transforming $3x + 2 + 5x$ to $3x + 5x + 2$). Second, some terms involving the variable may need to have coefficients added, to put all terms in a standard form (e.g., transforming $3x + x + -x$ to $3x + 1x + (-1)x$). When students are first learning algebra, the sequence of actions comprising collection may be unfamiliar; but eventually it becomes as automatic as the application of single strategic rules.

An algebraist embedding only knowledge at the levels of cognitive axioms, and strategic rules, would generate lines of reasoning which only partially approximate those given by a competent high-school student. Students will usually execute actions of a certain abstract type (e.g., removing parentheses) all at once. Other kinds of actions, like evaluating completely separate terms, are rarely interpolated. Additionally, students tend to apply one group of rules before others in order to reduce the number of reasoning steps that will be needed to generate a solution. For example, students will often apply all rules to collect terms before applying any rules to isolate a variable. In short, students achieve a global coherence or focus to their problem-solving behavior through knowledge of abstract classes to which the various strategic rules belong, and through knowledge of different policies that should dictate how the rule classes are used. However, the axioms and strategic rules discussed so far cannot achieve such global coherence, although they result in locally reasonable steps.

To capture knowledge at the level of abstract rule classes, we simply define rule types aggregating our strategic rules. For example, all rules for removing parentheses, or evaluating arithmetic terms are collected together in separate rule classes. Now the problem solver can reason at a higher level; for example, it can decide that removing parentheses is usually a high-priority action and should be done at once, when solving an equation.

To capture knowledge about how to use abstract classes of rules to solve equations in a focused, globally coherent manner, we define plans. A plan is a list of rule types, together with control information about how the rule types are to be used in problem solving. For example, one plan we have used might be paraphrased as: "(Simple plan) First remove all parentheses around expressions, then collect all terms containing the variable to the same side of the equation. Third, collect terms containing the variable. Finally, once there is only one appearance, isolate the variable by itself, resulting in a solution."

Different plans can be created by reordering rule types and specifying different control information key words. In this way the algebra problem solver demonstrates the flexibility we required to generate many plausible, interestingly
Figure 2. The reasoning structure for a simple equation. The reasoning structure shows the decomposition of a top-level plan for solving the problem into rule types used, and the decomposition of rule types into compound rules (comps) and strategic rules (rules). The last level of decomposition is from rules to cognitive axioms (caxioms). Some low-level decompositions have been pruned out for simplicity.
different solution plans for equations. Using different plans, our problem solver can produce solution lines identical to: (a) those produced by Bundy's PRESS program [8]; (b) those recommended by "cookbook" algorithms in many basic algebra texts (e.g., Holt's Algebra 1 [9]); (c) those produced by the methods taught by the teachers in our local high school; and (d) those recommended by a "function-oriented" approach to mathematics (e.g., [10]).

Reasoning structures — Each visible algebraic expression the tutor produces when generating a solution is a consequence of several reasoning decisions on the part of the system. Collectively, they form a reasoning structure. Figure 2 shows a simplified version of the reasoning structure for a relatively easy problem. Each node in the tree represents the particular piece of knowledge responsible for generating a given inference. The hierarchical nature of the structure reflects the different levels of knowledge that contribute to a solution. As the figure indicates, higher-level nodes are generated by more abstract strategic knowledge, like plans, rule types, or compound rules. Progressively lower-level nodes reflect contributions of strategic rules, and various axioms. Overall, the reasoning structures demonstrate that the tutor does not follow a linear, "cookbook" procedure when solving problems, but rather that it deliberates in a relatively intelligent fashion. The tutor therefore fulfills a requirement we mentioned in the previous section: It understands that solving algebra problems is a goal-directed reasoning process. The reasoning structures it creates resemble goal decomposition graphs, generated by many types of problem-solving systems (e.g., Sacerdote [11]).

TOOLS FOR HELPING STUDENTS LEARN PROBLEM-SOLVING SKILLS

Having described the reasoning processes of our algebra expert system embedded in our tutor, we now discuss the ways it is being used to create tools to help students learn specific kinds of problem-solving knowledge and skills. We first discuss tools supporting debugging, then describe tools for helping the student learn about goal-directed reasoning skills.

Tools for Learning Debugging Skills

Generally, debugging skills refer to the ability to fix mistakes. In computer programming we are quite comfortable with the idea that mistakes are inevitable, and that the ability to fix them is an important skill. However, in learning mathematics we are not as tolerant. Students are only given credit for right answers, not for how effectively they find and fix mistakes. Consistent with this policy, they are not encouraged or trained to debug their answers. To the contrary, we believe that learning to debug is an essential skill. We feel that the value of practicing algebra—doing homework—is to learn from your mistakes.
Figure 3. Student using the GO BACK menu option. To go back to a previous line of a solution, the student selects GO BACK, then points to the equation to which she wants to return. The tutor branches a new solution line from that point.
A student who gets all the questions right immediately has performed well, but learned little; the student who has made many mistakes yet fixed some of them, and understood the reasons for the errors, may have performed more poorly, but has learned much more.

The first way the tutor assists learning debugging skills is simply that it encourages mistakes, and does not penalize them. If the student determines the current solution line is not correct, or just wants to investigate a new line of attack on the problem, he or she can easily try alternatives. To determine whether the answer is correct, the student uses Answer OK? to find out if the solution is acceptable. The tutor's feedback even at this level is superior to CAI programs with canned answers: it makes a distinction between answers that are wrong, and ones that are correct as far as they go, but unfinished.

Next, the student can select the menu item Go Back. The tutor will then ask which solution expression in the DisplaySpace the student wants to now be the problem focus. The student responds by using a mouse to point to any expression in the reasoning tree and selects it by clicking one of the mouse buttons. For example, in Figure 3, having been told the answer \( y = \frac{32}{3} \) is wrong, the student has just clicked on Go-Back, then selected the equation \(-6y = -15 + 17 - 9y\). The tutor then moves the box that indicates the current focus of attention to this equation or node.

Generally, the Go Back option provides a way for the student to try out problem solving strategies efficiently. The traditional pencil-and-paper medium does not encourage this exploration and learning because it exacts a high cost to trying multiple solution lines. To try a new line, students must erase the old one, which they are reluctant to do because it is slow, and because they may forget the old line, if they wish to return to it.

Debugging is a multi-faceted skill, and merely encouraging the student to make, and explore errors, will not make a student skilled at debugging. Two component subskills of debugging that need distinct support are isolating the bug, and fixing the bug, or determining which algebra knowledge should have been used to compute the correct reasoning step at a given point. Isolating an error may be problematic because the tutor's problem-solving environment encourages the student to try out many reasoning steps. To learn from his or her mistake, the student may have to wade through these steps to find the one step that hides a misconception. In the case of Figure 3, for example, how does the student know which step is the culprit? Several menu item options will help simplify the search. First, the Step Ok? menu item allows the student to ask the tutor if any step in the reasoning tree constitutes an appropriate mathematical transformation in the current context. The student selects Step Ok?, then points to any step in the reasoning tree. A step is any pair of connected nodes in the reasoning tree, and is referred to by pointing to the arc between the two nodes, as shown in Figure 4.
Figure 4. Student using the STEP OK? menu option. This menu option can be used at any time by the student to select a reasoning step, and ask the tutor whether it is valid and appropriate.
When the student has selected a reasoning step, the algebra expert system then says whether the step from the previous node to the selected one is acceptable. In critiquing the student’s step, the tutor makes a distinction between steps that are mathematically invalid, and steps that are inappropriate. Inappropriate steps are logically valid, but would not be chosen by the algebra expert, because they do not move the student closer to a solution. Thus, using Step Ok?, the student not only succeeds in isolating a faulty reasoning step, but also obtains a characterization of the misconception underlying the error. Invalid steps imply errors in knowledge of algebra transformation rules; inappropriate steps imply errors in goal-directed reasoning methods. Having successfully used Step Ok?, the student is now in a position to execute Go Back as in Figure 3.

Step Ok? has been shown here as a debugging aid, but it is actually a more general tool. The student can use the Step Ok? operation any time he or she chooses, not just after discovering the answer is wrong. This freedom allows the student to follow several problem-solving modes. For example, he or she can proceed in “careful” mode, checking the appropriateness of each step as it is made, or, at the other extreme, the student can operate in “reckless” mode, creating reasoning steps until an answer is obtained which is obviously incorrect. The student then studies the reasoning tree, regarding it somewhat like an engineer regards a malfunctioning electronic circuit. The engineer may take voltage or current measurements at any point to isolate the fault; the algebra student uses Step Ok? to take “appropriateness” measurements to isolate the misconception.

Do Next Step is similar to Step Ok?. While Step Ok? lets the student confirm the validity and appropriateness of a step, Do Next Step lets the student ask the tutor to generate the next step that an expert algebra problem solver would take from the current problem focus. One of several ways this option can be used is to give the student information about an error-full reasoning step, not by telling him or her directly about the error, as Step Ok? does, but by showing the logical consequences of continuing this line of reasoning, letting the student draw the appropriate conclusions, if the consequences appear absurd. By repeatedly applying Do Next Step, the student can ask the tutor to provide such consequences at any point in the reasoning tree.

The immediate utility of Step Ok? and Do Next Step are that they quickly allow the student to pinpoint a single reasoning error in a large tree of possibilities, and to characterize the misconception behind the faulty step. More generally, and perhaps more importantly, these activities teach the student that an important part of learning a cognitive skill is learning to study your own reasoning processes. A surprisingly few students understand that reasoning can be explicitly examined, let alone that it can be debugged or improved. For example, when students are asked why they do poorly on a mathematics test in grade
school, common attributions are “I’m dumb in math” or “I had a bad test” or “The test was unfair.” Very few identify specific knowledge that they might have lacked, or even understood that their correctable lack of knowledge might have been responsible for failure.

**Tools for Learning Goal-Directed Reasoning Skills**

The supports we provide for helping the student understand that solving algebra problems can be a goal-directed reasoning process, as opposed to cookbook procedure, divide into two classes. First we enable the student to observe the tutor’s own exemplary reasoning processes in considerable detail. Second, we provide aids for the student’s own problem solving, in the form of hints that reveal the goal-structure of algebraic problem solving. We describe these tools in turn.

*Inspecting the tutor’s exemplary reasoning* — As the reasoning structure diagrammed in Figure 2 suggests, the visible algebraic steps that the tutor presents in solving a problem represents only a fraction of the reasoning that goes into the creation of a solution. To help students understand that a goal-directed reasoning process is behind the generation of the visible steps, and to give them insight into the nature of the reasoning process behind the visible steps, we want to provide the student with ways of exposing aspects of this process. However, the presentation of the process must be organized; a complete dump of the whole reasoning structure would probably be as uninformative as just showing the bare mathematical transformations.

Several of the menu items allow the student to “open up” the tutor’s reasoning process to a depth and detail that the student tailors to his or her cognitive needs. If, in viewing the expert’s reasoning, the student does not understand how the expert went from one expression to the following one, he or she can use Elaborate Step to see more detailed intermediate steps which may illuminate the expert’s reasoning. For example, in Figure 5 the student asks for an elaboration of the expert’s step collecting like terms, and Figure 6 shows the result. The new detail explicitly shows the expert’s addition of the implicit “−1” coefficient. This may be useful for the student whose understanding of collection does not yet encompass such special cases.

While the Elaborate Step option allows the student to selectively view more of the reasoning substeps that comprised a larger step, the option Explain Step provides more textual justifications of how and why those steps were taken.²

The tutor’s explanation capability is particularly powerful because it can exploit its hierarchical reasoning structures to create multi-leveled explanations.

---

² The current explanation facility in the tutor has been programmed by Mike Pazzani, a graduate student at UCLA.
Figure 5. Student selecting a step to elaborate. To elaborate a step, the student first selects the ELABORATE STEP menu option, then picks a step in the DisplaySpace by clicking on the arc connecting two expressions.

Figure 6. Tutor elaborating a step for the student. The tutor elaborates a selected step by interpolating smaller reasoning steps between the two expressions that comprised the original step.
Figure 7. Tutor explaining a step for the student. Shown is a series of questions posed by the student to the tutor.

Figure 7 illustrates this idea. We will focus on the tutor’s explanation of the step marked in Figure 5.

At the start of Figure 7, the student selects the Explain Step menu item, and then points to the indicated step. The tutor begins by bringing up an explanation window, and, below it, an explanation menu. In the explanation window the tutor prints the basic explanation describing the generation of the step. Each explanation placed in the window is boxed, and prefixed by a goal number. Goal numbers have several uses. First, they provide unique handles for referring to different explanations. This becomes important when several explanations populate the window. Second, they suggest to the student the process of generating reasoning steps is a goal-directed one. Finally, they explicitly show how goals are connected in a hierarchy of subgoals and supergoals.

To generate a basic explanation, the tutor simply accesses the part of the reasoning structure that was directly responsible for the visible transformation. In this case, it is the compound strategic rule collect-like-terms. Once the basic explanation is displayed, the student has an opportunity to deepen the explanation and tailor it to his or her own needs using the menu items How Goal and Why Goal. The student first selects one of these items, then clicks on a goal in the explanation window by placing the mouse in the box describing a goal. A new goal description is then placed in the explanation window, and its goal number reflects its subgoal/supergoal relationships with the goal just selected.
Generating answers to "How" and "Why" questions is relatively straightforward, since, again, it exploits the reasoning structure created in answering the question. To explain how it achieved a given goal, the tutor discusses goals below that goal. In Figure 7, for example, subgoals below—collect-like-terms, which implement that goal, are rearrange-addition, add-coefficients, distribute*+, and evaluate (a goal which does "cleanup" evaluations of numeric expressions and is invoked automatically after significant algebraic transformations). Each of these subgoals is now explained in its own goal box (labeled GOAL 2.1.1, GOAL 2.1.2, GOAL 2.1.3, and GOAL 2.1.4). These new goal descriptions, in turn, are subject to further elaboration by repeated applications of How Goal.

As GOAL 2.1.2.1 and 2.1.2.1.1 show, How Goal explanations can be deepened extensively. The tutor simply traces reasoning structures, like the one shown in Figure 2, moving down from a given goal to progressively deeper subgoals, until terminal "leaves" are met.

Similarly, "Why?" questions may have multiple answers. However, instead of looking at lower goals to answer Why Goal, the tutor looks at goals in its reasoning structure that are just above the goal involved in the basic explanation. In this case, the goal above collect-like-terms was the tutor’s top-level goal of solving the problem using one of its known plans. The tutor therefore would generate a description of the overall action of the plan.

At a general level, the menu options Elaborate Step and Explain Step play two distinct roles. First, when the student has made a mistake or is stuck, he or she can use these tools to locate and understand specific pieces of algebra problem-solving knowledge that is needed at that point. Thus, these tools help the student’s incremental accretion of concrete algebra skills. Second, at a more abstract level, these tools emphasize to the student that algebra problem solving is a goal-directed reasoning activity. The students see that each visible step is a consequence of several reasoning decisions and relies on several kinds of problem-solving knowledge. In addition, they can come to understand that each visible step in the tutor’s reasoning, and their own, can be justified by appeal to the knowledge that generated it.

Goal-directed hints — The reasoning structures that are so useful in generating explanations for the student appear to have a variety of other important tutorial uses. We are just beginning to explore a few of the more obvious ways they can be exploited to help students learn algebra problem-solving techniques. One of the most important uses concerns hints. When students are stuck at a given point in solving a problem, we generate hints to try to get them back on track. Having observed expert tutors give hints, we have come to realize that generating a good hint is a complex task. Although we lack a "theory" of hints, it is apparent that an appropriate hint gives the student just enough information to get him or her started. Hints that give too little information do not help at all; hints that give too much information fail to let the student exercise his or her knowledge and may detract from his or her feelings of control and competence.
But how can we generate hints that give "just enough" information? Most CAI programs use a single type of hint, if any. A more competent tutor needs to be able to generate hints at several different levels of detail or specificity, and select one that best matches the student's needs at a given time. Using the reasoning structures the tutor produces when answering a question, we are now in a position to generate hints at many levels. For example, Figure 8 shows that hints we can now generate for the simple equation $5x + 8 = 9$, from the most general to most specific. Analogous to explanation, the student controls the specificity of hints generated by selecting MORE DETAILED HINT as needed.

The generation of hints is driven by the tutor's reasoning structures in a way that parallels its production of explanations. Goals at higher levels in the reasoning structure provide more general advice about what to do, while lower level goals can be used to generate more concrete hints that tell the student how to make the next step. The main difference between explanations and hints is simply context, not computation. Explanations exploit the tutor's reasoning structures after the tutor has demonstrated the correct reasoning; hints use the reasoning structures when the student is trying to generate the correct line of reasoning him or herself.

Like explanation, we view hints as having two distinct roles in learning. When the student is stuck, a hint gives him or her the opportunity to continue. More

---

3 While we can generate these hints automatically, we have not yet implemented techniques that determine which one is best for the student at a given time.
importantly, if the hint provides just the minimal information necessary to succeed, it enables the student to acquire or solidify an understanding of specific pieces of algebra knowledge required at that point in problem solving. Second, hints, like explanations, reveal the goal-directed nature of reasoning and problem solving in algebra. Among other things, students begin to see that larger problem-solving goals (e.g., like isolating the variable) can be systematically broken down into smaller more tractable tasks (e.g., adding \(-4\) to both sides of an equation). The skill of breaking down large tasks into smaller ones is very general, but it is often an unfamiliar notion to students.

CONCLUSIONS

In this article we have discussed the role of our algebra expert system in providing tools to help students learn several kinds of nontraditional skills in the context of algebra. While many CAI systems continue to present ways of helping students learn the types of knowledge that they have been taught for decades, we believe computers provide an opportunity to teach other new and important skills. We have focused on how an algebra expert system can help promote students' understanding of goal-directed reasoning skills and debugging techniques. As we expand the intelligence of our tutoring system to encompass pedagogical knowledge it will cease to be a passive system, in which students always have to ask for assistance using menus. It will take a more active role in structuring the learning situation for the student. Nevertheless, the theme of investigating novel learning situations for students will remain a constant in the project.

REFERENCES


