KNOWLEDGE SOURCES FOR AN INTELLIGENT ALGEBRA TUTOR

William S. Bregar
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

Arthur M. Farley
Department of Computer and Information Science
University of Oregon
Eugene, Oregon 97403

Garland Bayley
Department of Mathematics
University of Portland
Portland, Oregon 97203

TR 86-30-7

Knowledge Sources for an Intelligent Algebra Tutor

William S. Bregar[*]

Department of Computer Science
Oregon State University

Arthur M. Farley

Department of Computer and Information Science
University of Oregon

Garland Bayley

Department of Mathematics
University of Portland

[*] Present address: Computer Research Laboratory, Tektronix Inc. Beaverton, Oregon

Knowledge Sources for an Intelligent Algebra Tutor

ABSTRACT

The focus of this paper is on the underlying knowledge base for an intelligent tutorial system for high school algebra problems. We present a model of problem solving flexible enough to account for a variety of problem solving behaviors and general enough to allow new problem domains to be defined easily. The model is based upon the analysis of protocols between students and expert tutors. We show how student errors can be monitored and remediated using the model, and we provide an approach to understanding problem difficulty that can be used to generate challenging problems and also provides a mechanism for planning their solution.

Introduction

What do we expect from an intelligent CAI system? What kind of behavior should an ICAI system exhibit? In general we would want the system to approximate the behavior of an expert human tutor. It should be adaptive to aspects of the student's behavior rather than respond according to a fixed regimen. The system should be capable of following a student's approach to a problem, furthering it when that approach is reasonable and providing appropriate guidance when errors occur. The system must be able to determine what knowledge is required to solve a problem and be able to articulate that knowledge at the right time.

Several systems have begun to demonstrate the potential for providing high quality, in-depth, tutorial interaction. The SOPHIE system (Brown and Burton, 1974) monitored student behavior on electrical circuit troubleshooting problems. It recognized when a student's hypothesis could not be supported by prior observations. In that event SOPHIE would take one of several actions ranging from noting that the student was drawing a conclusion from insufficient data to providing a set of alternatives to pursue. Thus, SOPHIE instilled proper problem solving skills by constraining the student to considering a set of hypotheses consistent with previous observations.

Clancey's Guidon (Clancey, 1979) emphasized representation and utilization of pedagogical strategies in tutoring medical diagnostic problem solving. What kind of tutorial information a student was given depended on a number of criteria, such as the the student's familiarity with certain factors, elements apparently not considered in the current problem, and the type of goal being discussed. Based on this analysis, Guidon might state relevant factors to consider or generate a question for the student that focuses on these factors.

Brown and VanLehn have been concerned with cognitive-based explanations of errors made by students on subtraction problems. They proposed *repair theory* as a fundamental cognitive framework that explained a large number of observed errors. (Brown and Vanlehn, 1980). The theory suggests that when a subject reaches an impasse in a problem, he creates a new procedure by heuristically altering the one he has been trying to use. The new procedure fails, however, when neither it nor the old procedure contains the fragment of the correct skill necessary to solve the problem.

Each of these systems relies on significant knowledge of the problem domain as a basis for intelligent tutorial guidance. Brown and Burton implemented a complete simulation of the electronic circuit used in SOPHIE. Clancey utilized the MYCIN knowledge base (Shortliffe, 1976) in GUIDON, and Brown and VanLehn implemented their cognitive model in a rule based system.

A model for algebra word problems

This paper presents a model of problem solving designed to serve as the underlying knowledge base for an intelligent tutorial system for algebra word problems. Our hypothesis is that a central component of the knowledge base for a tutorial system in mathematics must be a problem solver that performs as a student does, both correctly and incorrectly. This knowledge base should be designed so that the information it contains — strategic, procedural, factual — can be extracted and presented to a student as necessary. We argue further that the problem solving model should be based on observed human performance. We see this approach reflected in our previously cited examples by an increasing emphasis on cognitive models for their domains.

The focus in this paper is on the development of our problem solving system. It is based on tutorial protocols, from which we derive an information processing model. The model is shown to approximate human problem solvers and account for several classes of errors. In particular, the model emphasizes the nature of *qualitative* processing (Larkin, 1977), that aspect of problem solving behavior concerned with the establishment and subsequent utilization of a symbolic representation of a problem from which a solution can be derived. General, strategic knowledge is separated from domain specific computational knowledge; both are implemented as rule based systems which allow declarative access, while providing control over the level of knowledge detail the system represents.

The model operates on a structured, symbolic representation that is the result of reading a problem statement, classifying the problem into one of a set of known types, instantiating predefined schemata associated with that problem type, and incorporating relevant information from the problem statement. The problem classification phase has been observed and discussed previously by Hinsley, Hayes, and Simon (1977). Our schemata are structured data types such as those employed by Novak (1976) in the Isaac system in Mecho (Luger, 1981).

From the point at which problem entities have been defined and instances of schemata created to represent them, the system executes in a rule-based environment where rules match against aspects of the schemata. Two sets of rules interact to solve problems. These sets are referred to as *computation* rules and *strategic* rules. The former reflect typical expressions of formulas for a problem domain; the latter articulate strategies observed in our student-tutor protocols. This separation of knowledge provides generality of behavior as well as opportunity to tutor on domain-specific or general problem solving topics independently.

Goals for the model

The design criteria for our model derive from considerations of problem solving abilities and tutorial requirements. Our system must be able to solve the problems about which it is attempting to teach and to provide a basis for monitoring and correcting students during a problem solving session. Thus, a major goal for the system is that it be a model of *student* problem solving. The system must account for observed problem solving behaviors, both correct and erroneous, at appropriate levels of expertise and detail. At any level of expertise, the *granularity*, that is, the explicitness and "size" of the steps in the solution of a problem, must be of sufficient detail to provide a reasonable explanation of the process while being general enough to track a student's solution attempts.

Other important goals for the model include an ability to provide a description of the process and effects of learning, a theory of problem difficulty, and generality sufficient for application across different problem domains. Learning in our system can be construed as incremental alterations or additions to the rule sets. Thus, partially specified rules or incomplete rule sets stand as indicators of unlearned information relevant to a problem domain. Incorrect rules indicate mislearned content. Problem difficulty is a function of which information is included in the problem statement and the relative complexity and number of rules needed to solve the problem. Generality is accomplished through the separation of strategic from domain-specific computation rules. A set of computation rules for a domain can be defined independently from the strategic rule base, thereby allowing the problem solver to determine solutions for a new domain by acquiring only domain knowledge.

Tutorial protocols and their anaylsis

The basis for our model of problem solving is an analysis of think-aloud protocols of students interacting with tutors while solving several representative mixture problems. We concentrated our analysis on protocols for the following problem:

> A vat contains 10 gallons of a 20% alcohol solution. If pure alcohol is added at the rate of one gallon per hour, how long will it take to produce a 70% alcohol solution?

This problem is representative of those found in a typical algebra text book. Students tend to find this problem somewhat harder than one which asks for the result of combining two solutions where the amounts of the two solutions are known. This phenomenon was noted in several of our protocols, and we attribute it to the "indirectness" of the goal of the sample problem. Most of the student subjects (volunteers from first-year, college-level algebra classes at Oregon State University) found this mixture problem difficult or impossible to solve. They all recognized that they had to solve a mixture problem to solve the time problem.

Solutions to the problem reflected several approaches ranging from setting up and manipulating a single proportion to determining values for subgoals as generated by means-ends. We observed behavior ranging from that reflecting a total inability to understand or set up the problem to that which produced a solution in one step. The latter would correspond closely to the behavior associated with compiled rules as described by Anderson et.al (1981). Between these extremes, we were able to identify particular behaviors throughout the qualitative phase of processing. These behaviors result in one or more expressions relating quantities in the problem statement which can subsequently be solved algebraically to produce a correct solution.

Unlike many studies of problem solving behavior, which utilize uninterrupted protocols of novice or expert subjects, our protocols consist of interactions between tutors and novice or intermediate subjects. There are several advantages to be gained from observing tutors interacting with comparatively naive subjects. First, by observing tutors and novices, we were able to see details of behavior *on the part of the tutors* not generally seen in expert protocols. Specifically, a tutor's reactions to students' errors and misconceptions afford a deeper look at elements of the tutor's knowledge base. This is especially true as the tutor attempts to fill in information apparently missing from a student's knowledge base. Corrections to a student's erroneous behavior provide evidence for the identification of expert strategies and necessary elements of domain knowledge. At the same time, the protocols yield a variety of novice errors, as will be discussed below. Errors can provide significant insights into the elements and structure of expertise. Understanding the nature and causes of these errors is important from the standpoint of modeling cognition as well as for its implications for instruction (Brown and VanLehn,1980; Brown and Burton,1978; Matz,1982).

Our protocols strongly supported the observation of Hinsley, Hayes and Simon (1977) that *problem type recognition* occurs as a primary reaction to reading the problem statement. Other types of problem solving behaviors we noted include *entity construction* — the development of a representation for problem elements and their attributes, *instantiation of entity attributes* — where known and unknown values (variables) are bound to entity attributes, *goal determination* — the identification of the unknown whose value will be a solution to the given problem (including subgoal determination), *value computation* — combining known values according to domain based relations among attribute values, and *equation creation* — the generation of an algebraic relation among selected problem attributes. Evidence for these behaviors is presented in the form of annotated protocol fragments in figure 1.

Following are fragments of transcriptions of two of the student-tutor protocols analyzed in this study. S.n and T.n refer to student and tutor remarks, respectively.

This first segment deals with constructing appropriate entity representations: of initial and final solutions. Note filling in of directly computable attribute values (8, 2 and later 30%) in S.1 and S.3. We refer to this as **value computation.**

S.1 OK, the faucet ... 1 gallon per hour ... 10 gallons ... 20% ... You want to get to 70% solution. OK, 10 gallons, 20% solution would mean that 8 gallons are water and 2 gallons are alcohol.

T.1 That's right.

S.2 OK, if you want to get to the point where 70% ... 2 gallons ... if you add a gallon ... OK, that'd be 3 to 10, no 3 to 8 so you have to keep adding until you get to the point where you have 70%.

T.2 Can you express that somehow - use some sort of a relation. Maybe I might suggest that you might draw a picture showing what you have at the beginning and what you have at the end.

S.3 OK. OK, this is 8 gallons of water 2 gallons of alcohol total of 10 gallons and at the end, now, 70% alcohol and we get 30% water. OK, so 8 to 2.

T.3 See if you can relate the quantities in this solution to the quantities in this one, somehow.

This segment deals with goal determination and creating a variable reference for that unknown quantity. (S.13)

T.9 Now go back to the diagram for a second. You started out with something and you're gonna end up with something. What takes place in between the time that you're here and you're here.

S.10 The addition of 100% alcohol.

T.10 How much?

S.11 1 gallon per hour

T.11 But how much precisely. Do you know?

S.12 No, that's what I can't figure.

T.12 Well, maybe that tells you, suggests to you what you might want to use as an unknown in this problem.

S.13 OK, So we use x as the amount of alcohol added.

This segment deals with establishing an algebraic expression for the computation rule about percentages of solutions when the amounts involve unknowns.

S.23 OK. Then you're going to add alcohol to this.

T.23 OK, how much? You don't know.

S.24 To get the part where you have 7 ... So you're going to end up with more than 10 gallons. You don't know how many gallons yet.

T.24 But you do know what that "how many gallons" is don't you? You have a ... well go ahead. Don't let me confuse you. Go ahead and write what you were going to start to write.

S.25 OK. So this is the same as the ... no if you ... would be 7, 70%.

T.25 Of what?

S.26 Of 10. No, we don't know. 70% of the total.

T.26 Which is what? What is that total?

S.27 $10 + x$

T.27 Ah hah! That's exactly right!

S.28 OK.

T.28 Now let's go back to this part again. I'm not sure about this 2/10 over here. What do you think it should be?

S.29 OK, it would be 20%.

T.29 Of?

S.30 (20% of) 10 plus x = 70% of $10 + x$

New protocol: This first segment deals with subgoaling — needing the alcohol amount to get time. Note use of example in T.2 and S.2.

T.1 That's the idea. Right. So in theory, that can happen. It seems reasonable that if we let it run long enough that solution would become more and more alcohol so that at some point you'd reach 70%. And what we want to know is (what are we asked to find here) how long will it take? Suppose you knew how many gallons it took?

S.2 Well we know it's one gallon per hour of pure alcohol. Soooo...

T.2 Suppose you had to add 25 gallons to reach the 70% solution.

S.3 It would take 25 hours if you had to add 25 gallons of alcohol.

T.3 In addition to what's already in there you ...

S.4 You want to make it 20%.

T.4 Would that tell you how many hours?

S.5 Yeah, you could figure it out.

This is dealing with computation rule knowledge for computing amount given percent and quantity.

T.8 Of the 10 gallons in there at the start, do you know many gallons of alcohol there are? There are 10 gallons of a 20% solution.

S.9 Well, there's 20% of 10.

T.9 OK.

S.10 Which would be .02 times 10 which would be .2.

T.10 How much would 50% be? How much would 10% be? Let me ask you that.

S.10 One .. oh two.

T.11 OK, so that means there are two gallons there at the beginning.

S.12 So there's two gallons. Seems like I could set up a proportion.

Establishing unknown as goal (S.26) and variable reference to it (T.32, S.33).

S.26 Well somehow we have to figure out how many gallons of alcohol there are in a 70% solution.

T.26 UmHm, that's right, that'd help. Well how much ... let's see now what's in the solution when we're all done.? There's eight gallons of water and there's a certain amount of alcohol. So that would be the total volume.

S.27 So we want to find 70% of 10, because, Oh no.

T.27 Well there are going to be more than 10 gallons aren't there. All right we don't know precisely what that amount is do we?

S.28 No.

T.28 All right. Let's use a variable name for something here. There are only two things in the solution, water and alcohol.

S.29 OK

T.29 At the start there are eight gallons of water and two gallons of alcohol. Now we're going to add some more alcohol. And that's all that's being added. That's the only other thing being added.

S.30 Like if alcohol was x

T.30 x is the

S.31 Alcohol to begin with. The two gallons.

T.31 All right, now, would that be a good name for a variable? Sorry, would that be a good variable quantity? It's not a variable, is it? The amount of alcohol to start is a constant, it's two. The amount that you add is some unknown.

S.32 OK.

T.32 Suppose you let x be the amount that you're going to add.

S.33 OK, alcohol added. So then you have two plus x.

This segment deals with creating the necessary algebraic equation for the relationships in the final solutions.

T.42 Well tell me what you do know at this point, what do you know about the percent of alcohol in the total solution?

S.43 That's it .7.

T.43 Right, so the amount of alcohol ... is 70% of the total solution.

S.44 Oh, so 2 + x is the same as .7 ...

T.44 Well you have a proportion here that the alcohol, the number of gallons of alcohol to the total is equal to .7 over something and we're not sure what you do there. Well tell me in words what you want. What do you want to have in words here? What's the idea of this equation?

S.45 OK, this is the amount of alcohol and this is the total solution. So, the amount of alcohol is the proportion of the amount of alcohol to the total solution is the same as the percent of the alcohol to the total which would be ...

T.45 As a percent though, right? It's the percentage of alcohol, which is 70%, to the total solution, which would be 100% right? So it's 100 as a decimal ... OK, fine.

S.46 I could put 70 as 100

Figure 1. Protocol Fragments

Subjects with previous knowledge of the domain could be expected to already have a general representation of problem entities that they could begin to instantiate. Similarly, equation building could also involve the instantiation of previously defined relations known to be relevant to the problem type. Incomplete entity representations or failure to find correct, consistent instantiations of the attributes of these representations led to one class of student errors.

We interpreted a pattern of behavioral actions as a problem solving strategy. Sequences dominated by computation from known values reflect a *forward-directed* strategy most often employed by inexperienced problem solvers. Forward-directed strategies are based upon the computation of new values from known values until the value for the goal attribute has been determined. Another weak method we observed was *hill-climbing*, wherein a student tried a succession of values for the goal attribute, searching for a value that would make a domain relation involving the goal attribute and other, known attribute values consistent. Sequences were observed which involved substantial goal setting and subgoal generation in a backward-directed search from the original goal; such sequences are referred to as *means-ends* strategies (Newell and Simon, 1972). Finally, sequences consisting of the selection and instantiation of a relevant equation whose solution directly solves the problem are said to be *expert-level* strategies similar to those noted by Larkin, McDermott, and Simon (1980).

It should be noted that in a given problem solution, more than one problem solving strategy is often identified. Mixtures of strategies, are more likely to be observed at the novice to intermediate level. Thus, a model capable of generating or monitoring problem solving behavior in an ICAI system must represent multiple strategies and account for their co-occurrence in problem solving sessions.

System Architecture

In this section we describe the architecture of the problem solving component of our system. We assume an initial translation phase that results in the creation of an internal representation which is then utilized in problem solving.

Our model of algebra problem solving consists of two cooperating rule-based systems matching elements of a structured problem representation. These systems consist of *strategic rules* and *computation rules*, as mentioned earlier. They represent, in an explicit manner, the knowledge sources we observed in protocols of subjects solving algebra word problems. A third system of *algebraic rules* solves equations produced during the qualitative processing phase for specified variables. A detailed description of the algebraic rule system can be found in Lantz, Bregar, and Farley (1983).

The strategic rules represent the observed general problem solving strategies and levels of expertise. The computation rules represent domain-specific knowledge of relationships among problem entity attribute values; they are defined as part of a *problem-type frame*. In addition to computation rules, the problem-type frame contains *entity schemata, relational schemata*, and *linguistic features*. Entity schemata are structured representations of objects found in a given class of problems. Each schema provides a set of attribute slots for value and scale. Unfilled slots represent unknowns, some of which will become goals for the problem solver. Relational schemata indicate how problem entities are interrelated according to semantically meaningful relations of the domain.

For example, in distance-rate-time problems, one expects to find one or two moving objects with attributes such as rate, direction of travel, and time constraints. There are several forms or sub-classes of distance-rate-time problems such as those in which one object overtakes a second or in which an object is traveling in a moving substance. The relational schemata specify the correspondence between objects in a problem statement to entity schemata and state the key relationship between them. Again, in a DRT problem, two objects, 500 miles apart, traveling in opposite directions, would be entered into an "opposite-direction-relation" as

```
(type oppposite-direction-relation
 first-object          car_1
 second-object         car_2
 distance-apart        500)
```

In mixture problems we define three types of entities — substances, solutions, and composites. Solutions have parts which are substances, while composites have parts which are solutions. We restrict ourselves to discussions of solutions and composites that have only two parts. Substances are atomic (i.e., have no parts). For example, the schemata for substances have the form

```
substance
name:                <string>
proportion-value:    <decimal>
amount:              <decimal>
type-value:          <substance-type>
```

The schemata for solutions include pointers to the substances they contain, as well as specifying the measurement units:

```
solution
name:                <string>
amount:              <decimal>
units:               <measure>
part-sub1:           <substance name>
part-sub2:           <substance name>
```

Attribute values are directly available from the problem statement, can be computed from the problem statement, or represent (current) unknowns. Unknown attribute values become goals and subgoals for the problem solver. The relational schema for mixture problems is called *equiv* and identifies two solutions which are equivalent, represented by what we call a *composite-solution*. The composite-solution is a solution which can be viewed as being comprised of two solutions or, alternatively, as two substances. Note that this implies that solutions can be parts of composites and substances can be parts of solutions. This establishes multiple perspectives on the same problem element, and thus the possibility of setting up multiple computations involving the same quantity. The problem solver can then apply computation rules in appropriate combinations to solve the problem in a variety of ways.

The linguistic features in the problem-type frame consist of a domain dependent lexicon of words and phrases common to the problem type. The lexicon is employed by an augmented transition network parser which interprets natural language statements of problems and translates them into the schemata (Rapp, 1986). We will not discuss this component further in this paper, focusing our discussion on the interaction of entity representations with strategic and computation rules in the production of the variety of problem solving behavior necessary for tutorial systems.

Computation rules

A computation rule represents how the unknown values of an entity attribute can be computed in terms of the values of other appropriately related attributes. Constraints on the attributes provide important matching information to the problem solving system as it attempts to bind entity attributes to computation rule variables. Appendix A contains the set of computation rules used for mixture problems. Each rule has five parts: structure, constraints, inputs, outputs, and computation. The structure part describes a subset of the relational schemata associations; the structure and constraints are, in effect, the conditional aspects of the rule. Constraints specify restrictions on bindings of attributes to variables named in the structure part. Inputs are attributes of the entities whose values are involved in the computation; the resultant value is that of the output entity. The actual computation given in the computation part of a rule can involve arbitrary constants, as well. For example, computation rule CR5 computes the proportion of one part of a composite, given the proportion of the other part.

Strategic rules

The strategic rules provide the basic control facilities in the system and reflect our observations of problem solvers, ranging from novice to expert. Four sets of strategic rules encompass the range of approaches to problem solving we observed; these rule sets are presented in Appendix B.

The least expert level is referred to as *forward-directed* (FD) and implements an undirected search from known values. Only FD1 has a notion of a goal to be solved. Otherwise, the mode of operation is to compute a value if possible (FD2) or to select a computation rule having at least one known (numerical) value for one of its inputs (FD3) which is then considered by FD2.

A somewhat more intelligent strategy is embodied in the *hill-climbing* rules (HC). This strategy can be characterized as a heuristic search over the value space of the goal. The key aspect of this strategy is the interaction between rules HC2 and HC6. HC2 will never be invoked unless HC6 has been, previously. Rule HC6 looks for a computation rule such that there is an unknown goal variable on the right-hand side, and all other attributes have known values. Rather than solving for the goal variable algebraically, the rule stipulates that a guess be made for the goal attribute's value and the computation of the rule is then performed. If the guess is correct, HC1 fires and the goal is known. Otherwise, rule HC2 will continue to fire until an answer is found (or time runs out). The adjustment of value produces the hill-climbing effect, increasing or decreasing the value as appropriate to move the value closer to the goal value.

At the next level are the *means-ends* (ME) rules (Newell and Simon, 1972). In general, these rules solve problems through recursive application of subgoaling, working backward from the original goal to known attribute values. Rule ME1 determines if a solution has been found for the current (sub)goal. Failing that, rules ME2 and ME3 look for a computation rule that either computes the quantity in question or incorporates it in the computation of another quantity, respectively. These are referred to as *direct* or *indirect* computations, respectively. The order of the rules reflects the observed difficulty of these solution strategies. Rule ME3 fires when a computation rule has been fully instantiated. Means-ends is incorporated into rule ME4 which posts unknown variables in a computation rule as the new subgoals for the system and an implicit "push" operation to a new problem occurs. Rule ME2 picks a subgoal to process following an application of ME4. Finally, rules ME6-ME8 invoke the algebraic subsystem to create and solve equations representing computation rules which could not be fired because they were not entirely instantiated. This corresponds to an observed behavior in which subjects consciously transformed a relation into an equation before attempting to find a value for the goal variable.

The highest level strategy is manifested by the *expert* rules (EX). EX2 captures the notion that an expert can generally move directly to a known algebraic relation relevant to the problem and its variables and solve that equation. If the equation cannot be solved with the information available, subgoals must be created to provide the additional values necessary (rule EX3).

An example

To clarify the concepts presented thus far, we give an annotated example of the system solving the mixture problem used in the earlier described protocols:

> A vat contains 10 gallons of a 20% alcohol solution. How many gallons of pure alcohol must be added to produce a 70% solution?

We then compare this with a student-tutor protocol of the same problem.

The complete example is presented in figure 3. Figure 2a shows a representation of the stated mixture problem. Attributes whose values are represented by question marks are those for which no values are given directly in the problem statement. These may later become goals, based on decisions by the strategic component. Starred attributes mark those which are computed by applications of computation rules as the schema is being built. We have referred to this filling in of immediately computable values as *spontaneous computation*. Such behavior occurred repeatedly in the tutorial protocols. Figure 2b gives a diagram showing the relationships among the entities, substructures of which are utilized as matching criteria for the computation rules. Individual rules utilized in the example will be presented and explained as they are used.

solution	solution	solution
name: START	name: ADDED	name: RESULT
amount: 10	amount: GOAL	amount: ?
units: gallons	units: gallons	units: gallons
part-sub1: START.P1	part-sub1: ADDED.P1	part-sub1: RESULT.P1
sub2: START.P2	sub2: ADDED.P2	sub2: RESULT.P2
substance	substance	substance
name: START.P1	name: ADDED.P1	name: RESULT.P1
prop-val: 0.2	prop-val: 1.0	prop-val: 0.7
amount: 2*	amount: ?	amount: ?
type: alcohol	type: alcohol	type: alcohol
substance	substance	substance
name: START.P2	name: ADDED.P2	name: RESULT.P2
prop-val: 0.8*	prop-val: 0	prop-val: 0.3*
amount: 8*	amount: 0	amount: ?
type: water	type: water	type: water

equiv
name: RESULT'
sol_1: START
sol_2: ADDED
comp: RESULT

Figure 2a. Entity schemata for a mixture problem.

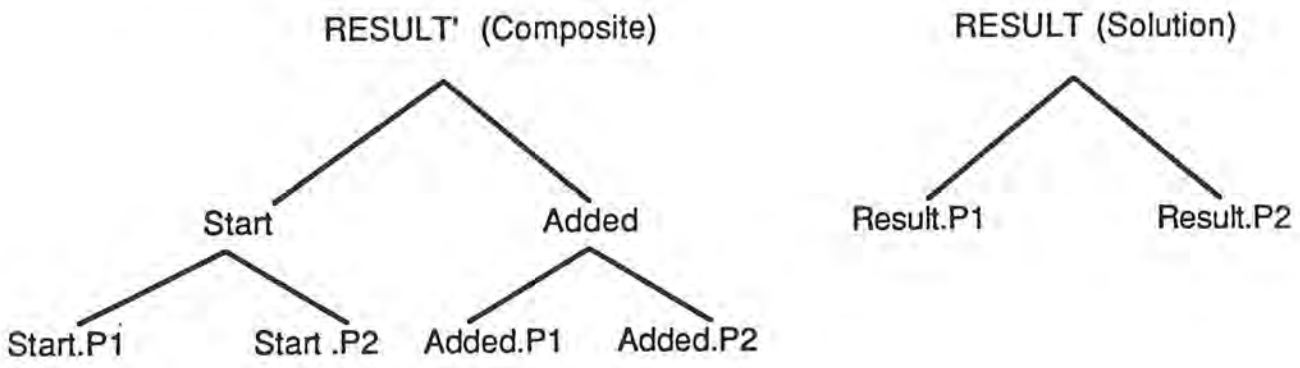

Figure 3. Annotated example.

Figure 2b. Structural representation of entity schemata. Note that RESULT' and RESULT are in the *equiv* relation.

1.	Rule ME2:	Given a goal entity, select a rule computing its value as output and determine consistent entity bindings.
	Inputs:	amount(ADDED)

	Outputs:	Select CR2
		P2 = amount(ADDED)
		P1 = amount(START)
		E = amount(RESULT')

	Comment:	The goal is amount(ADDED), which is part of the resultant solution. CR2 is selected because it computes the amount of a part. Entities are bound to CR2 rule arguments as shown. The goal, amount(ADDED) is recognized as being a part of something. The only entity descriptor in which ADDED is listed as a part is RESULT', the final solution viewed as a composite. Its other part is START. Hence, the bindings for CR2 are assigned. The computation is amount(ADDED) = amount(RESULT') - amount(START).
2.	Rule ME4:	Given a bound computation rule with unknown attribute value inputs, make the unknown input attributes goals.
	Inputs:	CR2 with bindings as above
	Outputs:	amount(RESULT')
	Comment:	CR2 is fully bound, but not all entities have values. In particular, amount(START) is known, but amount(RESULT') is not. Therefore, amount(RESULT') becomes the new subgoal.
3.	Rule ME2:	Given a goal entity, select a rule computing its value as output and determine consistent entity bindings.
	Inputs:	amount(RESULT')
	Outputs:	Select CR8 amount(RESULT') = amount(RESULT)
	Comment:	CR8 states, in effect, that if a composite and a solution are equivalent, then their amounts are equal. This will ultimately allow the system to pursue two different ways of computing the same quantity.
4.	Rule ME4	Given a bound computation rule with unknown attribute value inputs, make the unknown input attributes goals.
	Inputs:	CR2 bound to....amount(RESULT') = amount(RESULT)
	Outputs:	goal = amount(RESULT)
	Comment:	Again, a means-ends approach is employed to establish a new subgoal.
5.	Rule ME2:	Given a goal attribute, select a rule computing its value as output and determine consistent entity bindings.
	Inputs:	amount(RESULT)

	Outputs:	select CR4
		E = RESULT (the 70% solution)
		P = RESULT.P2 (the 30% part of E)
		Proportion(P) = .30

	Comment:	The current goal is amount(RESULT). CR4 states that given a solution or composite E having a part P, the amount of P, and the proportion of E that P represents, then the amount of E can be computed as amount(E) = amount(P) / proportion(P) Thus the equation amount(RESULT) = amount(RESULT.P2) / .3 is produced; amount(RESULT.P2) is unknown. Note that the bindings could have been: P = RESULT.P1 (the 70% part of E) Proportion(P) = .70
6.	Rule ME4:	Given a bound computation rule with unknown attribute value inputs, make the unknown input attributes goals.
	Inputs:	CR4, bound as above

Outputs: subgoal = amount(RESULT.P2)
Comment: Obvious

7. Rule ME2: Given a goal attribute, select a rule computing its value as output and determine consistent entity bindings.
Inputs: amount(RESULT.P2)
Outputs: select CR7
C = RESULT'
S = RESULT
PC1 = START
PC2 = ADDED
PART_OF_PC1 = START.P2
PART_OF_PC2 = ADDED.P2
Comment: CR7 is a crucial computation rule. It says that, given a composite composed of two solutions, and a solution equivalent to the composite, the amount of a substance in the solution/composite is equal to the sum of the amounts of that substance in each of the two solutions making up the composite. Obtaining consistent bindings for this rule means insuring that the resulting computation operates on the correct substances. In this instance, the instantiated rule produces the computation:

amount(RESULT.P2) = amount(START.P2) + amount(ADDED.P2)

Note that values for the quantities on the right hand side are known; they are 8 and 0 respectively, from the entity descriptors.

CR3 could have been selected here, which, with the proper instantiation would compute

amount(RESULT.P2) = .30(amount(RESULT)).

However, this is, effectively, the same computation as in step 5, which would lead to a loop. An expert will avoid the loop; a novice might not. A meta-rule can be inferred from this: If a set of bindings yields a computation equivalent to one previously generated, do not use it.

8. Rule ME3: If all inputs for a bound computation rule have known values, execute the computation rule.
Inputs: CR7 with bindings as above
Outputs: amount(RESULT.P2) = 8
Comment: Firing CR8 whose right hand side is 8 + 0 gives 8. The amount(RESULT.P2) is now known.

9. Rule ME1: Given a goal attribute with a known value, note solution and utilize answer (report result or propogate up goal tree).
Inputs: Previously bound CR5, amount(RESULT.P2) = 8
Outputs: amount(RESULT) = amount(RESULT.P2) / .3
= 8 / .3
= 26.666

10. Successive iterations of ME3 and ME1 produce the remaining actions as follows:
Fire CR8. amount(RESULT') = amount(RESULT)
= 26.666

Fire CR2. amount(ADDED) = amount(RESULT') - amount(START)
= 26.666 - 10
= 16.666

Although the example is largely self-explanatory, we point out that it was restricted to operate using only a means-ends strategy. In fact, nothing prevents the system from invoking any of the other strategic rule systems. The capability for mixing strategies is especially useful for monitoring problem solving in a tutorial session. Furthermore, the system is not limited to the solution shown. The selection of computation rules in the example is arbitrary, and any of a number of different sequences could have been executed. Again, this flexibility is important in a system designed to emulate the performance of students in a problem solving situation.

Comparison with a live protocol

Figure 4 contains the transcript of a tutorial session wherein the tutor (T) and the student (S) communicated through a pair of computer terminals; all of their interactions were recorded on line. We were interested in observing communication over a channel that more closely approximated that available in a typical intelligent CAI context.

T.1 The problem you are to solve is as follows: A vat contains 10 gallons of a 20% alcohol solution. How many gallons of pure alcohol must be added to obtain a 70% solution?

S.1 I have a 20 % solution and I must add another complete solution to it to obtain the 70% solution. Is it important to know how much in gallons is in the 70% solution.?

T.2 Yes, although that is not what you are to find to answer the problem it certainly can be used to find the answer.

S.2 I have twenty percent or two gallons of alcohol in the first solution. I have 100% alcohol in the second solution, and 70% in the third. 100% equals one. Help!

T.3 You told me you have 2 gallons of alcohol in the first solution. You also told me you have 100% alcohol in the second solution. How many gallons of alcohol do you have in the second solution?

S.3 If I have 70% sol. Then I have .7 times X number of gals. giving me the number of gallons of alcohol. I know there are more than two. There are 8 gals. of water total. 30% equals 8 gals. Now I have .3 times X number of gallons giving me 8 gallons. So 8 divided by .3 should give me the total solution.

T.4 Which is how many gallons?

S.4 26.6gals.

T.5 Is that the answer to he problem or would you like to continue? To remind you, the problem is *– A vat contains 10 gallons of a 20% alcohol solution. How many gallons of pure alcohol must be added to obtain a 70% solution.

S.5 I want to go on. The problem (sic) I just gave was only the total number of gallons in the third solution. Now I can multiply .7 times 26.6 giving me 18.62 . That is the amount of pure alcohol. 18.62 gals.

T.6 18.62 gallons of pure alcohol in which solution?

S.6 I forgot to subtract the two from the first solution. This would make the answer 16.62 gals. of pure alcohol.

T.7 Correct, you did a great job.

Figure 4. Transcript of tutorial session

Comparing the transcript to the example, one can see that at S.1 the student pursues a subgoal to find the amount of 70% solution. This corresponds to step 2 in the example. At step S.3 the student sees that the number of gallons of alcohol in the 70% solution is .7 of the total amount. She then notes that there are 8 gallons of water (corresponding to example step 8), that the water constitutes 30% of the solution, and that the amount of solution, therefore, is 8/.3 (example step 9). She then computes the total, computes the amount of alcohol in it, and subtracts the initial amount of alcohol to obtain her final answer. Our example computes these amounts somewhat differently because it uses the fact that the amount of the original solution was 10 gallons. The principle, however, is similar, and the system easily could have chosen computation rules (namely, CR3 and CR2) which would have yielded the same computation.

Our rules actually explain and provide more detail than is explicitly stated by the student subject. For example, she states that there are 8 gallons of water in the 70% solution, whereas our example actually shows the computation that must have taken place. Luger (1981) noticed a similar phenomenon in comparing traces of the Mecho system to human protocols.

Entities represented by descriptors in our system are referred to as nominals by the subject. In S.1 she mentions the "20% solution", "another solution" (to be added), and "the 70% solution". These are now part of her problem solving context, although they are not given specific names. Clearly, she has associated attributes with these entities not unlike those ascribed to them in the entity descriptors we define.

Finally, she applies computation rules without making the transition (formally) to equations. This corresponds to our example which obtains a solution by simply "reasoning" about the relations stated in the appropriate computation rules. Our system thus can derive solutions in terms of the entities and their relationships (part-whole, solution-component, etc.) without formally declaring variable names or utilizing internal names. Internal names are viewed as pointers to the objects involved and further to the structures representing associations among them.

Levels of expertise

Our model supports the widely held assumption that experts solve problems requiring multiple step solutions by utilizing some kind of planning mechanism (the strategic rules). Novices, on the other hand, either have no plan at all or attempt means-ends only to encounter a snag and flounder. The protocols show the expert tutors teaching and reinforcing a means-ends strategy. This would appear to stand somewhat in contrast to Larkin et. al. (1980) who claim that experts tend to utilize a forward-directed approach to problem solving. We would explain this by noting that tutors would or should be attempting to support a student's line of reasoning, if it were relevant. Furthermore, it would make sense to explain the derivation of a problem solving step by reasoning backwards from the next attainable step in order to save explaining why all the unchosen forward-directed steps were eliminated from consideration. Expert-level strategies also require a familiarity with domain computation rules not yet realized by novice student, and probably reflect the compilation of prior means-ends determined solutions into complex computation rules.

Our protocols further show that students who have no plan at all or who do not fully understand the relationships among problem entities have a tendency to fall back on familiar computations, even if they are incorrect or irrelevant. The student takes whatever quantities have been stated in the problem and attempts to compute whatever he can (as in the FD strategy) even if the computation is inappropriate or the quantities have not been correctly bound. This would imply that the student does have at least a partial set of computation rules, but his strategies and structural constraints on computation rule invocation are not fully specified.

Another feature which differentiates expert from novice behavior is the facility with which experts apply algebraic manipulations to instantiated rules to derive relationships between an isolated unknown and other attributes. Novices, on the other hand, do not appear to notice the relationship between a rule and the algebraic expression of that rule. Our system, therefore, distinguishes between *instantiation*, the binding of problem entities to arguments in computation rules, *invocation*, the performing of the computation part of an instantiated rule, and *equation formation*, the changing of the dynamic creation of value expressed by a computation rule into the static relation among values expressed by an equation. An explicit operation effects the transformation of a rule to an equation; this is accomplished through the application of strategic rule ME6. Evidence for this activity is observed repeatedly in the protocols, as subjects create variables and equations involving those variables.

Equation formation need not result in any substantive change in the meaning of a rule. It is intended more as a change of perspective — viewing the rule as an object manipulable by the rules of algebra. In keeping with this philosophy, the system does not allow a computation rule to be manipulated directly. When manipulation is necessary, the algebraic rules are invoked to create the equation coresponding to a particular rule, then to manipulate it or possibly combine it with other similarly created equations to solve for a specific variable. Note that this allows marking an aspect of a computation rule as having a variable name associated with it so that this same name can be used consistently in other equations.

Misconceptions and errors

While modeling correct problem solving may help to explain a large part of problem solving behavior, it does not always account for the difficulties people often have in solving problems. It is clear that until valid procedures are well establised, a variety of erroneous procedures can be invoked and executed, which will, ultimately, have to be corrected.

The identification and remediation of errors in problem solving has been the subject of significant research efforts. Stevens, Collins, and Goldin (1979) augmented original work on misconceptions from the Scholar system (Carbonell, 1970) with a taxonomy of misconceptions to be treated in a Socratic dialogue. The Buggy system (Brown and Burton, 1978; Burton, 1982) emulated errors ("bugs") in subtraction through a procedural network of correct and incorrect subskills. They further derived mechanisms for recognizing single and multiple bugs. Brown and VanLehn (1980) proposed a theory (repair theory) to explain the derivation of erroneous procedures and Matz (1982) gives a process model for the development of errors in algebra. At the procedural level Genesereth (1982) employs a plan recognition scheme to debug user's misconceptions using a symbolic algebra system.

We are currently interested in two classes of erroneous behavior evident in processing algebra word problems. We refer to these as *interpretation* errors and *procedural* errors. We are also investigating aspects of problem difficulty and how they affect translation and procedure.

Interpretation errors

Our tutor-student protocols have indicated several kinds of difficulties incurred by problem solvers in the translation process. Some general characteristics of these include

- difficulties in identifying problem entities and their attributes
- failure to determine and establish correct relationships among problem entities
- inability to translate qualitative information to quantitative information

These errors are semantic in nature. The first can be attributed to a basic lack of knowledge about the problem domain. Of course, partial knowledge might allow a subject to identify entities but not all their attributes. On the other hand, students shouldn't be expected to process problems in a domain about which they have absolutely no knowledge. Clearly, the entity schemata in our system provides adequate information for monitoring a student's ability to identify all the attributes necessary to instantiate them.

Establishing correct relationships among problem entities involves a process by which problem entities are bound to (associated with) the corresponding abstract entity in a computation rule. In our computation rules, there is a structural constraint which must be satisfied before problem entities can be associated with rule entities. Additional tests such as equivalence relationships among entities in question, algebraic constraints, and simple dimensional constraints can ensure that proper quantities are being combined or operated upon.

Finally, the inability to translate qualitative to quantitative information involves processes such as those for deducing the correct mathematical operator from the text of the problem or assigning propr and consistent units to entities in an expression. Many of these correspond to the conversions described in STUDENT (Bobrow, 1968). As part of the mechanical translation process (Rapp, 1986), our system can flag elements of the problem statement and associate with them the correct operation. These flagged elements can be used in hints, when necessary.

Procedural errors

Procedural errors appear to be related to an improper use or lack of application of strategic processes. In particular, we note errors deriving from

- failure to recognize or establish problem goals
- failure to establish subgoals where appropriate
- application of incorrect rules

Each of these procedural errors can be related to a phase in the problem solving process. Establishment of problem goals should occur during the intial translation phase in the reading of the problem. There are well-known cues in problem statements that can be pointed out to a student having difficulty determining a problem goal.

Failure to deal with subgoals is not surprising in light of studies such as that by the National Assessment of Educational Progress (NAEP, 1977) which show that teen-agers have difficulty solving problems that require more than one processing step. One hypothesis for this phenomenon is that students are unaware that a subproblem can be processed exactly the same way as the main problem and the result can then be propagated upward. Our system, with its emphasis on a means-ends strategy, can easily suggest a subgoal (an unknown from an entity descriptor) and proceed to establish a context within which the subgoal can be solved.

The application of incorrect rules is generally results when a student attempts to use a rule for which he has only a partial understanding. This most often occurs with computation rules, where operations are confused, or subcomputations within the rules are not complete. For example, several students were observed to compute proportions by adding percentages. They did not multiply the quantities of which the percentages were taken.

We account for errors of this type by associating with each computation rule, a set of *deviation rules* which represent common misinterpretations of the base rule. The concept of deviation rules is taken from Goldstein (1979). A given deviation rule may be associated with several computation rules.

An example

In figure 5, we present a partial protocol of a student (S) interacting with a tutor (T) on a representative story problem, and in the following paragraphs we show how our representation could be employed in an automated system for identifying student errors.

T1: OK, You should be getting this message. Here goes. The problem you are to solve is as follows: A vat contains 10 gallons of a 20% alcohol solution. How many gallons of pure alcohol must be added to obtain a 70% solution?

S1: Help

T2: I was beginning to expect that! Well, what is it you are trying to find?

S2: Does a 20% solution imply that there are 8 gallons of water and 2 gallons of alcohol?

T3: Yes. Now, can you tell me what you are trying to find?

S3: I'm just hunting around rather randomly for how much more alcohol needs to be added to the solution. I am then doing some dividing to try to come up with the new percent. But I don't come out with an even addition amount.

T4: Can you identify the different quantities involved in this problem?

S4: What do you mean by quantities?

T5: The various things that are being combined and produced. For example, can you state what the particular goal of this problem is. That is, what does it want you to figure out?

S5: How many gallons of alcohol must be added. I need an equation because I'm not coming up with it by trying to use logic. What do I start with as usable variables to begin with?

T6: OK, You know what you are looking for - how many gallons of alcohol must be added. You probably will need an equation. How does the amount of alcohol you are going to add relate (algebraically) to the amount you are starting with?

S6: 20% + 50% = 70%. Is that what you mean?

T7: here are a couple of things here, let's do them one at a time. First, the 20% refers to a twenty percent solution. What is the amount that it is 20% of?

S7: The 10 gallons.

T8: OK. If you want to refer to 20% of some quantity like 10 gallons, how would you write that?

S8: .20 water or 8 gallons. I'm not sure what you mean.

T9: Suppose you didn't know that there were 10 gallons of solution. If you were to express that unknown quantity and refer to the amount of alcohol in it, how would you do it?

S9: .20 X where X is the unknown.

T10: Right. So if you substituted 10 for X you'd have written what I meant. So, now let's look at the rest of the relation you wrote earlier (.20 + .50 = .70). What does the 50% refer to?

S10: .5 is how much more alcohol is added to bring the total to 70. But that isn't correct because we're talking dilution. The amount of solution is also increasing. Help.

Figure 5. Student-Tutor Protocol

Interaction S6 could be flagged as an incorrect instantiation of computation rule 1, caused by using only percents in a rule that requires amounts. When at an impasse in solving a problem students often interpolate from a known rule to allow them to continue (Brown and VanLehn, 1980, Matz, 1982) which would account for such an incorrect instantiation.

As can be seen at interactions S6 and S10 the student has missed or ignored the fact that pure alcohol is a 100% alcohol solution. In fact, the student's equation in S6 states the starting and ending proportions and a proportion (.50) which completes the equality. He fails to account for the pure alcohol and assumes that percents can be summed even when they may represent proportions of different quantities. Assuming that the student was attempting to establish

$$.20(10) + .50(x) = .70(10+x)$$

and that there exists a deviation link in the genetic graph for the quantities rule indicating that the percent coefficient be written and summed independently, the error can be discovered by the system in this case and noted to the student.

If the deviation link did not exist, a search for whether one of the correct rules had been used incorrectly could find that

$$.20(10) + 1.0(x) = .70(10+x)$$

is very close to the relation above. The actual interaction (S6) could be flagged as an incorrect instantiation of the rule for quantities made by using only percents in a rule that requires amounts.

Problem difficulty

Problem difficulty can be partially attributed to structural aspects of the problem statement such as the number of variables involved, the order in which they are presented, the number of unit conversions, and the syntactic depth of sentences (Loftus and Suppes, 1972). In addition, there appears to be a relationship between the form of a "familiar" computation and the form in which it must be used to solve the stated problem. We refer to the unfamiliar form as a *derived* rule.

We have categorized some additional features which we believe affect problem difficulty. Problems whose solutions require more than one computation rule to effect a solution appear to be more difficult than those which do not. If one or more of these rules is a derived rule, difficulty also increases. Finally, rules can be composed, which further increases the difficulty of the problem. Figure 5 illustrates these concepts with some example mixture problems.

Basic Rule B1: The sum of the parts equals the whole.

Derived Rule D1: To compute the amount of a part, subtract the amount of the other part from the whole.

Basic Rule B2: The fraction of a substance in a solution times the total amount of solution gives the amount of the substance.

Derived Rule D2: The total amount of a solutio is the amount of a substance divided by the proportion of that substance in the solution.

Problem 1 (B1)

Ten gallons of water added to a barrel containing 20 gallons of water gives how many gallons of water?

Problem 2 (D1)

How many gallons of water must be added to a barrel containing 20 gallons of water to obtain 30 gallons of water?

Problem 3 (B1 + B2)

If 10 gallons of a 20% alcohol solution are added to 4 gallons of a 50% alcohol solution, how much alcohol will be in the final solution?

Problem 4 (B1 + D2)

If a 20% alcohol solution containing 2 gallons of alcohol is added to a 50% alcohol solution containing 2 gallons of alcohol, how many total gallons of solution will be produced?

Problem 5 (B1 + B2 + addition unknown)

How many gallons of a 50% alcohol solution must be added to 10 gallons of a 20% alcohol solution to produce a 40% solution?

Figure 5. Derived rules and the evolving difficulty of problems.

The system can utilize this model of problem difficulty to assess a student's skills as well as a source of evidence for debugging student errors. The more finely problem difficulty can be defined, the greater the ability of the system to diagnose and remediate student errors will be.

Finally, the ability to generate problems at various levels of difficulty using derived rules can be employed to provide tutorials for a forward directed problem solving strategy. Namely, rules used to generate a problem are the very computation rules which should be employed to solve the problem. We are exploring methods for parsing story problems and identifying the rules and derived rules which would be most appropriate for solving them. These could then be used in an expert problem solving strategy.

Discussion

We have presented a model of problem solving for algebra problems that is designed to serve as a knowledge base for an intelligent tutorial system. The system is more than just a straightforward problem solver. It is important to note that it is based on a model of human problem solving behavior. Because it accounts for a range of strategies, the system can be used to monitor students who exhibit any one or a combination of them. Because it can solve problems in its domain of expertise, it can be used to provide logical "next steps" in keeping with a particular student's approach to a problem.

The separation of knowledge sources — entity schemata, computation rules, and strategic rules — makes the system general, as well as flexible. Generality resides in the strategic rules which are distinct from the domain-specific computation rules. In fact, we have defined a second knowledge base for distance-rate-time problems which required no modifications to the strategic rules.

Certain classes of errors are easily recognized as manifestations of missing or partially known rules, deviations from existing rules, incorrect binding of problem entity attributes to rule variables, and unfilled attribute slots in entity schemata.

Finally, our theory of problem difficulty provides the system with a mechanism for generating problem sets at appropriate levels as well as to be used in planning problem solutions based on the combination of rules encompassed in the problem statement.

The rule-based problem solver described in this paper is implemented in YAPS (Allen, 1983) under the Maryland version of Franz Lisp.

References

Allen, L. (1983) YAPS: Yet another production system. TR 1146, Computer Science Dept. University of Maryland, College Park.

Anderson, J. R., Greeno, J. G., Kline, P. J., and Neves, D. M. (1981) Acquisition of problem solving skill. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition.* Hillsdale, N. J.: Lawrence Erlbaum and Associates.

Bobrow, D. G. (1968) Natural language input for a computer problem solving system. In Minsky, M. (Ed.), *Semantic Information Processing.* Cambridge, MA: MIT Press.

Brown, J. S., Burton, R. R. (1974) SOPHIE — A pragmatic use of artificial intelligence. *Proceedings of the Annual Conference of the Association for Computing Machinery.* San Diego, CA, November, 1974, 572-579.

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

Brown, J. S., and VanLehn, K. (1980) Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science.* 1980, *4*, 379-426.

Burton, R. R. (1982) Diagnosing bugs in a simple procedural skill. In Sleeman, D. and Brown, J. S. (Eds.), *Intelligent Tutoring Systems.* London: Academic Press.

Clancey, W. J. (1979) Tutoring rules for guiding a case method dialogue. *International Journal of Man-Machine Studies,* 1979, *11*, 25-49.

Genesereth, M. R. (1982) The role of plans in intelligent teaching systems. In Sleeman, D. and Brown, J. S. (Eds.), *Intelligent Tutoring Systems.* London: Academic Press.

Goldstein, I. P. (1979) The genetic graph: A representation for the evolution of procedural knowledge. *International Journal of Man Machine Studies.* 1979, *11*, 51-77.

Hinsley, D. A., Hayes, J. R., and Simon, H. A. (1977) From words to equations: Meaning and representation in algebra word problems. In M. A. Just and P. A. Carpenter (Eds.), *Cognitive processes in comprehension.* Hillsdale, N. J.: Lawrence Erlbaum Associates.

Lantz, B. S., Bregar, W. S., and Farley, A. M. (1983) An intelligent CAI system for teaching equation solving. *Journal of Computer-Based Instruction.* 1983, *10*, 35-42.

Larkin, J. H. (1977) Skilled problem solving in physics: A hierarchical planning model. Unpublished Manuscript, University of California, Berkeley, September, 1977.

Larkin, J. H., McDermott, J., Simon, D. P., and Simon, H. A. (1981) Expert and novice performance in solving physics problems. *Science*, June, 1980, *208*, 1335-1342.

Luger, G. F. (1981) Mathematical model building in the solution of mechanics problems: Human protocols and the MECHO trace. *Cognitive Science*, 1981, *1*, 55-77.

Loftus, E. F., and Suppes, P. (1972) Structural variables that determine problem-solving difficulty in computer-assisted instruction. *Journal of Educational Psychology*. 1972, *6*, 531-542.

Matz, M. (1982) Towards a process model for high school algebra errors. In Sleeman D. and Brown, J. S. (Eds.), *Intelligent Tutoring Systems*, London: Academic Press.

NAEP Newsletter. (1977) Study traces achievement profiles. *National Assessment of Educational Progress Newsletter*, April, 1977.

Newell, A., and Simon, H. A. (1972) *Human Problem Solving*. Englewood Cliffs, NJ: Prentice-Hall Inc.

Novak, G. S. (1976) Computer understanding of physics problems stated in natural language. *American Journal of Computational Linguistics*. Microfiche 53, 1976.

Rapp, C. ALGEBRA READER: An expert reader for algebra word problems. Technical Report (in progress). Department of Computer Science, Oregon State University, Corvallis, OR.

Shortliffe, E. H. (1976) *Computer-Based Medical Consultations: Mycin*. New York: American Elsevior.

Stevens, A., Collins, A., and Goldin, S. E. (1979) Misconceptions in student's understanding. *Internation Journal of Man-Machine Studies*. 1979, *11*, 145-156.

Appendix A

Computation Rules (Mixture Problems)

CR1: The amount of a solution is equal to the sum of the amounts of its parts.

Structure:	Solution or composite E with parts P1 and P2
Constraints:	P1 .NE. P2
Input values:	amount(P1), amount(P2)
Output value:	amount(E)
Computation:	amount(P1) + amount(P2)

CR2: The amount of one part of a solution is equal to the amount of solution minus the amount of the other part.

Structure:	Solution or composite E with parts P1 and P2
Constraints:	P1 .NE. P2
Input values:	amount(E), amount(P1)
Output values:	amount(P2)
Computation:	amount(E) - amount(P1)

CR3: The amount of a part of a solution is the amount of solution times the proportion of the part.

Structure:	Solution or composite E with part P1
Constraints:	
Input values:	amount(E), proportion(P1)
Output value:	amount(P1)
Computation:	proportion(P1) * amount(E)

CR4: The amount of a solution is equal to the amount of a part divided by the proportion of the part.

Structure:	Solution or composite E with part P
Constraints:	
Input values:	amount(P), proportion(P)
Output value:	amount(E)
Computation:	amount(P) / proportion(P)

CR5: The proportion of a part is equal to the amount of the part divided by the amount of the solution or composite of which it is a part.

Structure:	Solution or composite E with part P
Constraints:	
Input values:	amount(E), amount(P)
Output value:	proportion(P)
Computation:	amount(P) / amount(E)

CR6: The proportion of one part of a solution is equal to 1.0 minus the proportion of the other part.

Structure:	Solution or composite E with parts P1 and P2
Constraints:	P1 .NE. P2
Input values:	proportion(P1)
Output value:	proportion(P2)
Computation:	1.0 - proportion(P1)

CR7: The amount of a substance in a composite is the sum of the amounts of that substance in each of the solutions comprising the composite.

Structure:	Composite C and its equivalent solution S. C has parts PC1 and PC2. PC1 has part PPC1, PC2 has part PPC2, and S has part PS.
Constraints:	PC1 .NE. PC2, name(PPC1) = name(PPC2) = name(PS), C .SAME_AS. S
Input values:	amount(PPC1), amount(PPC2)
Output:	amount(PS)
Computation:	amount(PPC1) + amount(PPC2)

CR8: If a composite and a solution are equivalent then their amounts are equal.

Structure:	Composite C and solution S
Constraints:	C and S are in the *equiv* relation
Inputs:	amount(C) or amount(S)
Output value:	amount(C) or amount(S)
Computation:	if input(amount(C)) then amount(S) else amount(C)

Appendix B: Strategic Rules

Means-ends rules

1. Given a goal attribute with a known value, note solution and utilize answer (report result or propagate upward in goal tree).
2. Given a goal attribute, select a rule computing its value as output and determine consistent entity bindings.
3. If all input attributes of a bound computation rule have known values, perform the computation rule.
4. Given a bound computation rule with unknown attribute value inputs, make the unknown input attributes goals (if not already).
5. Given a goal attribute, select a rule having it as input and determine consistent entity bindings.
6. Given a bound computation rule, activate the algebraic rules to create an equation corresponding to the rule.
7. Given a goal attribute and an equation with its corresponding variable, activate the algebraic rules to solve the equation for that variable (ie. isolate it on the left hand side).
8. Given two equations having a non-goal variable in common, activate the algebraic rules to combine equations, eliminating the common, non-goal variable.

Forward-Directed rules

1. If the goal attribute has a known value, note the solution and utilize the answer.
2. If all input attributes of a bound computation rule have known values, perform the computation rule.
3. Given an attribute with a known value, select a rule with it as input and determine consistent entity bindings.

Hill-Climbing rules

1. Given a goal attribute with correct value, note solution and utilize answer.
2. Given a goal attribute with an incorrect value as determined relative to a bound computation rule, adjust the value of the goal variable, perform the computation rule, and check the correctness of the goal attribute value (see rule HC6).
3. Given a goal attribute, select a rule computing its value as output and determine consistent entity bindings.
4. Given a bound computation rule with input attributes all having correct, known values, perform the computation rule to produce a correct output value.
5. Given a goal attribute, select a rule having it as input and determine consistent entity binding.
6. Given a bound computation rule with unknown goal attribute input and all other attribute values known, assign the goal attribute a reasonable value, perform the

computation rule, and check to see if the assignment is correct (i.e., the result is equal to the known output value).

Expert rules

1. Given a goal attribute with known value, note result and utilize answer.
2. Given a new goal attribute, select a known algebraic relation among it and the set of known attribute values and solve for the goal attribute with algebraic rules.
3. Given a goal attribute, employ means-ends rules to generate new goal attribute(s).