REPRESENTING AND USING PROCEDURAL BUGS FOR EDUCATIONAL PURPOSES(1)

John Seely Brown, Richard R. Burton, Kathy M. Larkin

Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

"If you can both listen to children and accept their answers not as things to just be judged right or wrong but as pieces of information which may reveal what the child is thinking you will have taken a giant step toward becoming a master teacher rather than merely a disseminator of information." ---J.A. Easley, Jr. & Russell E. Zwoyer

1. Preface

Until recently efforts in constructing "intelligent" knowledge-based instructional systems (ICAI) have been primarily focussed on endowing computers with sufficient expertise to answer a student's questions, critique his behavior, and in some cases, help him debug his own ideas.(2) Although such expertise is necessary for sophisticated tutorial systems, it is by no means the whole story. Master teachers have skills that transcend their particular field of expertise. One of their greatest talents is the artful synthesis of an accurate "picture" of a student's misconceptions from the meager manifestations reflected in his errors. An accurate picture of a student's capabilities is a prerequisite to any attempt at direct individual remediation. The pictures of students that teachers develop (in whatever form) are often called "models". The form, use and induction of such models for procedural skills is the topic of this research. In particular we shall describe some initial efforts in the development and use of a representational technique called "procedural networks" as the framework for constructing diagnostic models of procedural skills. A diagnostic model attempts to capture a student's common misconceptions or faulty behavior as simple changes to (or mistakes in) a correct model.

This paper consists of three sections. The first describes a domain of application and provides examples of the problems which must be faced with a diagnostic model. The second introduces procedural networks as a general framework for representing procedural knowledge underlying a skill in such a way as to facilitate discovering or inferring misconceptions or bugs existing in a particular student's encoding of this knowledge. The third discusses pedagogical issues that emerge from the use of diagnostic models of procedural skills. This discussion is framed in the context of a computer-based tutoring/gaming system that was developed to explore the characteristics of our models.

2. Problems for a Diagnostic Model of Procedural Skills

In our research we have been investigating the procedural skills necessary to perform high school algebra. This includes not only the generally recognized rules of algebra, but also such normally implicit skills as reading formulas, parsing of expressions and determining of which rules to apply next. [Brown and Burton 1975, Brown and Collins 1977] For this paper, however, we will limit our discussion to examples encompassing arithmetic skills. This will allow us to concentrate on the critical ideas of diagnosis without the need for a large number of algebraic rules. Limiting our examples to arithmetic also provides a compelling demonstration of how much more difficult it is to diagnose what is wrong with a student's method of performing a task (i.e. to form a diagnostic model) than it is to simply perform the task itself. In particular it seems fair to assume that our readers find it no great challenge to add or subtract two numbers. Let us consider some examples of attempts to use this competency to diagnose what is wrong with the internalized representations of these arithmetic skills (procedures) of some students. We shall start with a case study in which we examine five snap shots of a student's performance during addition (as might be seen on a homework assignment). The task is to discover the student's misconception or bug.

(1) This research was supported in part, by the Advanced Research Projects Agency, Air Force Human Resources Laboratory, Army Research Institute for Behavioral and Social Sciences, and Navy Personnel Research and Development Center under Contract No. MDA903-76-C-0108.

(2) Some examples of such systems are: SOPHIE [Brown and Burton 1975], SCHOLAR [Carbonell and Collins 1973], BIP [Barr et al 1974], and MYCROFT [Goldstein 1974].

Sample of the student's work:

41	328	989	66	216
+9	+917	+52	+887	+13
50	1345	1141	1053	229

Once you have discovered the bug, try testing your hypothesis by "simulating" that bug and predicting the results on the following two test problems.

446	201
+815	+399

The bug is really quite simple. In computer terms, the student, after determining the carry, forgets to reset the "carry register" and hence the amount carried is accumulated across the columns. For example, in the second problem, 8+7=15 so he writes 5 and carries 1. 2+1=3 plus the one carry is 4. Lastly 3+9=12 but that one carry from the first column is still there -- it hasn't been reset -- so adding it in to this column gives 13. If this is the bug, then the answers to the test problems will be 1361 and 700. This "bug" is not so absurd when one considers that a child might use his fingers to remember the carry and forget to bend back his fingers, or counters, after each carry is added.

A common assumption among teachers is that students do not follow procedures well and that erratic behavior is the primary cause of a student's inability to perform each individual step correctly. Our experience has been that students are remarkably able procedure followers, but that they often follow the wrong procedures. One case encountered last year is of special interest in this regard. The student proceeded through a good portion of the school year with his teacher thinking that he was exhibiting random behavior in his performance of arithmetic. As far as the teacher was concerned there was no systematic explanation for his errors; and, we must admit that before we had "discovered" his bug we, too, thought that he was erratic. Here is a sample of his work:

7	9	8	6	8	9	17	19
8	+5	+3	+7	+8	+9	+8	+4
15	14	11	13	16	18	25	23

87	365	679
+93	+574	+794
11	819	111

923	27,493	797
+481	+1,509	+48,632
114	28,991	48,119

There is a clue to the nature of his bug in the number of ones in his answers. Every time the addition of a column involves a carry, a one mysteriously appears in that column; he is simply writing down the carry digit and forgetting about the units digit! One might be misled by 17+8 which normally involves a carry yet is added correctly. It would seem that he is able to do simple additions by a completely different procedure -- possibly by counting up from the larger number on his fingers.

The manifestation of this student's simple bug carries over to other types of problems which involve addition as a subskill. What answer would he give for the following?

A family has traveled 2975 miles on a tour of the U.S. They have 1828 miles to go. How many miles will they have traveled at the end of their tour?

He correctly solved the word problem to obtain the addition problem 2975 + 1875 to which he answered 3191. Since his work was done on a scratch sheet, the teacher only saw the answer which is, of course, wrong. As a result, the teacher assumed that he had trouble with word problems as well as arithmetic.

When we studied this same student's work in other arithmetic procedures, we discovered a recurrence of the same bug. Here is a sample of his work in multiplication:

68	734	543
x46	x37	x206
24	792	141

758	2764
x296	x53
144	2731

There are really several bugs manifested here; the most severe one being that his multiplication algorithm mimics his addition algorithm. But notice that the bug in his addition algorithm above is also present in his multiplication procedure. The "carry unit" subprocedure bug shows up in both his multiplication and addition. For example, to do 68x46, in the first column he performs 8x6, gets 48 and then writes down the "carry" which in this case is 4, ignoring the units digit. Then he multiplies 6x4 to get 2 for the second column. All along he has a complete and consistent procedure for doing arithmetic. His answers throughout all of his arithmetic work are far from random. In fact they display near perfection with respect to his way of doing it.

3. A First Approximation to Representing Procedural Skills

In order to build a computer system capable of diagnosing aberrant behavior such as the above, the skill being taught must be represented in a form amenable to modelling incorrect as well as correct procedures. Additionally, the model should break the skill down into shared sub-skills in order to account for the recurrence of similar errors in different skills. We use the term diagnostic model to mean a representation that depicts a student's internalization of a skill as a variant of a correct version of the skill. For a representation of a correct skill to be useful as a basis for a diagnostic model, it must make explicit much of the tacit knowledge underlying the skill. In particular, it must contain all of the knowledge that can possibly be misunderstood by a student performing the skill or else some student misconceptions will be beyond the diagnostic modelling capabilities of the system. For example, if the model of addition doesn't include

the transcription of the problem, the system would never be able to diagnose a student whose bug was to write 9's which he later misread as 7's.

The technique we use to represent diagnostic models is a procedural network.(3) A procedural network consists of a collection of procedures (with annotations) in which the calling relationships between procedures are made explicit by appropriate links in the network. Each procedure node has two main parts: a conceptual part representing the intent of the procedure, and an operational part consisting of methods for carrying out that intent. The methods (also called implementations) are programs that define how the results of other procedures are combined to satisfy the intent of a particular procedure.(4) Any procedure can have more than one implementation which provides a way to model different methods for performing the same procedure (skill). For most skills, the network representation takes the form of a lattice. Figure 1 presents an example of the partial breakdown of a part of the addition process into a procedural network. Conceptual procedures are enclosed in ellipses. The top procedure in the lattice is addition.(5) Two of the possible algorithms for doing addition are presented as alternative methods. In method 1, (the standard algorithm) the columns are added from right to left with any carries being written above (and included in the column sum of) the next column to the left. In method 2, the columns are added from left to right with any carries being written below the answer in the next column to the left. If there are any carries, they must be added in a second addition. Notice that these two methods share the common procedures for calculating a column sum and writing a digit in the answer, but differ in the procedure they use when carrying is necessary. One structural aspect of the network is to make explicit any subprocedures that can be potentially shared by several higher level procedures.

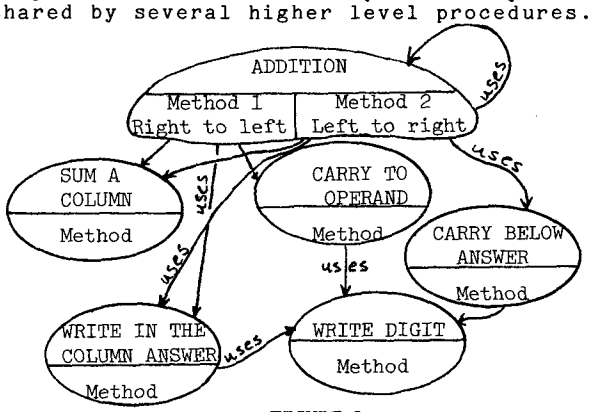

FIGURE 1
A Simplified Piece of a
Procedural Network for Addition

The decomposition of a complex skill into all of its conceptual procedures terminates in some set of primitives that reflect assumed elements of an underlying computational model. For addition, typical primitives are recognizing a digit; being able to write a digit; and knowing the concepts of right, left, etc. The complete procedure network (explicitly specifying all the subprocedures of a skill) can be evaluated or "executed", thereby simulating the skill for any given set of inputs. By itself, this merely provides a computational machine which performs the skill and is not of particular import. However, the possible "misconceptions" of this skill are represented in the network by "buggy" implementations associated with procedures in the decomposition. Each buggy version contains incorrect actions taken in place of the correct ones. An extension to the network evaluator enables the switching in of a buggy version of a procedure thereby allowing the network to simulate the behavior of that buggy subskill. This provides a computational method for determining the external behavior of the underlying bugs.

4. Inferring a Diagnostic Model of the Student

The problem of diagnosing a deep structure failure in a student's knowledge of a procedural skill can now be accomplished, at least theoretically, in a straightforward manner. Suppose, as in the examples on page 3, we are provided with several surface manifestations of a deep structure misconception or bug in the student's addition procedure. To uncover which possible subprocedures are at fault, we use the network to simulate the behavior of buggy subprocedures over the set of problems and note those which

(3) This term has been used by Earl Sacerdoti [1975] to describe an interesting modelling technique for a partially ordered sequence of annotated steps in a problem solving "plan". Our use of procedural nets differs from and is less developed than his.

(4) The language we have used is LISP. The particular programming language is unimportant from a theoretical standpoint because an implementation is non-introspectable. The modelling aspects of the network must occur at the conceptual procedure level. For example, the implementation of the subtraction facts table look up procedure in the computer is necessarily different from that in the student. However, the conceptual properties of the facts table procedure are the same in both. Those aspects which are the same (e.g., the invoking of other procedures, the values returned, the relevant side effects) are included in the network, while the implementation details which may differ are "swept under the rug" into the program. This is not a limitation, as any "implementational issue" can be elevated to the conceptual level by creating a new conceptual procedure in between the existing ones. The distinction between conceptual and implementation details also allows skills to be modelled efficiently at different levels.

(5) This is a simplified representation intended only to demonstrate those features of the procedural network particularly relevant to the diagnostic task. The actual breakdown into subprocedures may be different in a particular network and will be considerably more detailed.

generate the same behavior as exhibited by the student. To catch a student's misconceptions that involve more than one faulty subprocedure, we must be able to simulate various combinations of bugs.(6) For example, a student may have a bug in his carrying procedure as well as believing that 8+7 is 17 (a bug in his addition facts table). To model his behavior, both buggy versions must be used together. A deep structure model of the student's errors is set of buggy subprocedures which when invoked, replicate those errors. Each buggy version has associated information such as the underlying teleology of the bug, specific remediations, explanations, examples and so on. These may be used by a tutoring system to help correct the student's problem.(7)

5. Relationship of Diagnostic Models to Other Kinds of Structural Models.

It is beyond the scope of this paper to discuss all the past and current work on structural models of students and how it relates to diagnostic models based on procedural networks. However, a few words are in order. Most of the past and current research on this subject has been focussed on the intuitively appealing notion that if one has an explicit, well formulated model of the knowledge base of an expert (for a given set of skills or problem domain) then one can model a particular student's knowledge as a contraction or simplification of the rules comprising the expert [Brown and Burton 1974, Collins, Warnock and Passafiume 1975, Burton and Brown 1976, Carr and Goldstein 1977]. Recently, Goldstein has expanded this concept in his Computer Coach research and has coined the term "overlay model" for capturing how a student's manifested knowledge of skills (rules) relates to an expert's knowledge base [Goldstein 1977].

The work reported in this paper differs in emphasis from such approaches in that the basic modelling technique focuses on viewing a structural model of the student not primarily as a simplification of the expert's rules but rather as a set of semantically meaningful deviations from an expert's knowledge base.(8) That is, each subskill of the expert is explicitly encoded, along with a set of potential misconceptions of that subskill. The task of inferring a diagnostic model then becomes one of discovering which set of variations or deviations best explains the surface behavior of the student. This view is in concert with (although more structured than) the approach taken by Self [1974] in which he models the student as a set of modified procedures taken from a procedural expert problem-solver.

6. BUGGY - An Instructional Game for Training Student Teachers (and Others)

As we saw in the first section, it is often difficult to induce a student's bug from his answers. The need for teachers to thoroughly appreciate and strategically cope with the possible range of student bugs led us to construct a game called BUGGY. BUGGY is a computerized game based on the diagnostic interactions of a tutor and a student. The computer portraits an errant student whose malady must be identified by the teacher/diagnostician (a role played by an individual or a team). The diagnostician is presented with an arithmetic homework problem that the "student" has done incorrectly. The "student's" behavior is generated using a procedural network and manifests an underlying bug in one of the arithmetic subprocedures. The diagnostician's job is to discover exactly what the underlying misconception is by providing strategic test problems for the "student" to solve. The problems given by the diagnostician are answered by the "student" using the bugged procedure. When the diagnostician indicates that he knows the bug, he is first asked to describe the bug then, to make certain that he really has found the bug, he is given a five-problem test. During the test the diagnostician must answer the problems in the same errant way that the student would do them, i.e. he must simulate the "student's" bug. When the diagnostician's concept of the bug is well formulated enough to predict the buggy answer on all five problems, he is considered to have diagnosed the student and he proceeds to a new bug.

We have found it beneficial to allow teachers to team up while playing the diagnostician's role in BUGGY. The interaction between team members encourages verbalization of the decision-making processes involved in the game. The following is a protocol of a team of teachers using BUGGY. The dialogue is interspersed with commentary about the issues that we feel are pedagogically important. Those lines typed by the team are marked with a vertical line in the left margin.

---

(6) Additional structure in the network helps resolve what combination of bugs are worth considering. In general simulating or evaluating all simple and multiple bugs takes approximately 2 cpu seconds for the addition and subtraction procedural nets.

(7) West [1971] has broken down the diagnostic teaching task into four steps: i) distinguish between conceptual and careless errors; ii) identify the exact nature of the conceptual error (bug); iii) determine the conceptual basis (cause) of the bug; and iv) perform the appropriate remediation. The system we describe has been directed towards problems (i) and (ii). The buggy implementation nodes in the network provide the proper places to attach information relevant to problems (iii) and (iv).

(8) Because these deviations are based on both the student's intended goals and underlying teleology of the subskills, we have no automatic way to generate them (as opposed to what could be done if the deviations were based on the surface syntax of the rules). However, ongoing work by Goldstein and Miller [1976], Rich and Schrobe [1976] and Burton and Brown [forthcoming] will eventually help overcome this limitation.

7. Protocol of a Team Using BUGGY

WELCOME TO BUGLAND.
I HAVE CHOSEN A BUG. HERE IS AN EXAMPLE OF THE BUG.

```
 17
+5
---
 13
```

The system presents an example in which the student doesn't get the right answer. At present these initial examples are stored in the network linked to the buggy implementation.

NOW YOU CAN GIVE ME PROBLEMS TO DETERMINE THE BUG.

```
 18      43
+6     +79
---    ---
 15     23
```

The team generates problems that are answered by the "student". Notice that the team's test problems are similar to the initial example generated by BUGGY. In general this is a poor diagnostic strategy as it may leave other symptoms of a bug undiscovered.

Got the bug!

The team forms an initial hypothesis and tells the system they have discovered the bug. The system now asks the team to describe the bug and then tests them on their understanding. Asking for a description forces the team to verbalize their sometimes fuzzy idea of what the bug is and often makes them realize how ill-defined or vague their idea actually is.(9)

PLEASE DESCRIBE THE BUG YOU THINK I HAVE.

Student adds the number to be carried into the last column worked in.

This cryptic description is typical of those we have observed. Describing bugs in English is surprisingly difficult. What the team is trying to say is that if a column adds to more than 9, the units digit and the carry digit are added together and the result is written as the answer. For example, in the first column of the example problem, when 7 and 5 are added, 1 (the carry) and 2 (the units digit) are added to get 3. The same with 8+6, 2+9, and 3+9. (This bug could easily arise if the student were using method 2 from figure 1 and mistakenly wrote the carry under the same column.) The hypothesis does account for the observed behavior. Unfortunately, it is not the student's bug. Notice the possibly humorous, probably disastrous results that would have transpired if the team had stopped at this point and tried to tutor the student on the basis of their initial belief!

HERE ARE SOME PROBLEMS TO TEST YOUR THEORY ABOUT THE BUG.
WHAT IS:

To ensure that the team has found the correct bug, the system generates several problems for the team to solve. (At present, BUGGY has a heuristic problem generation program that gives a broad range of "different" problems. These are then filtered to try to get three problems which cause symptoms and two that give correct answers. This simple routine does surprisingly well at producing critical problems. So well, in fact, that a more "intelligent" routine which determined the consistent set of bugs and then filtered the test problems to split the set was discarded as unnecessarily time-consuming.)

```
 21
+39
---
 51
```

The team's hypothesis is revealed as incorrect by the first of the five test problems. BUGGY presents its answer and then allows the team to enter more test problems.

THAT'S NOT THE BUG I HAVE.
USING MY BUG:

```
 21
+39
---
 15
```

TRY GIVING ME SOME MORE EXAMPLES.

```
  51      99      68
+17     +99     +9
---     ---     ---
  21      36      23
```

After failing the test, the team gives a greater range of problems, specifically those using bigger numbers. How is it that 51+17 can add up to 21? By broadening the scope of example problems they are able to reevaluate their initial hypothesis, dismiss it, and develop another one that is complete and consistent with respect to the new errors.

Got the bug!

PLEASE DESCRIBE THE BUG YOU THINK I HAVE.

The student is adding all the digits together.

(9) In the present system, the descriptions are saved but they are not currently used, since the understanding of these utterances is beyond the capabilities of any existing natural language system.

HERE ARE SOME PROBLEMS TO TEST YOUR THEORY ABOUT THE BUG.
WHAT IS:

33	1091	8	28	90
+99	+60	+34	+70	+6
26	17	15	17	15

This time the team's hypothesis passes the test and they are given the description of the bug stored in the procedural network.

VERY GOOD. MY DESCRIPTION OF THE BUG IS: THE STUDENT ALWAYS SUMS UP ALL THE DIGITS WITH NO REGARD TO COLUMNS.

8. Pedagogical Issues for Teachers

One application of BUGGY and the "diagnostic model" view of procedural skills is in the domain of teacher training. The realization that "random" errors are actually the surface manifestations of an underlying bug in a procedure is a major conceptual breakthrough for many student teachers. Often behavior that appears to be random, has a simple, intelligent, and complete underlying explanation. By proper diagnosis, remediation can be directed towards the specific weaknesses. The importance of this notion cannot be overstressed. Admitting the possibility of underlying bugs is critical to remediation in the classroom. Without the ability to diagnose procedural bugs, failure on a particular problem must be viewed as either carelessness or total algorithm failure. In the first case, the remediation consists of giving more problems, while in the second, it consists of going over the entire algorithm.(10) When a student's bug (which may only manifest itself occasionally) is not recognized by the teacher, the errant behavior must be explained as carelessness, laziness or worse. This causes the teacher to adapt her model of the student's capabilities, thereby mistakenly lowering her expectations. From the student's viewpoint, the situation is worse. He is following what he believes to be the correct algorithm and, seemingly at random, gets marked wrong. This situation can be exacerbated by improper diagnosis. For example, Johnnie subtracts 284 from 437 and gets 253 as an answer. Of course, says the teacher "you forgot to subtract 1 from 4 in the hundreds place when you borrowed." Unfortunately Johnnie's algorithm is to subtract the smaller digit in each column from the larger. Johnnie doesn't have any idea what the teacher is talking about (he never "borrowed"!) and feels that he must be very stupid indeed not to understand. The teacher agrees with this assessment as none of her remediation has had any effect on Johnnie's performance.

BUGGY, in its present form, presents teachers with examples of buggy behavior and provides practice in diagnosing the underlying causes of errors. Using BUGGY, the teacher gains experience in forming theories about the relationship between the symptoms of a bug and the underlying bug itself. This experience can also be cultivated to make teachers aware that there are methods or strategies that they can use to properly diagnose bugs. There are a number of strategy bugs that teachers may have in forming hypotheses about a student's misconceptions. The development of a good "troubleshooting" strategy by a teacher can avoid these pitfalls. A common mistake is to jump too quickly to one hypothesis. Prematurely focussing on one hypothesis can cause a teacher to be unaware that there are many competing hypotheses that are possibly more likely. A common psychological effect of this is that the teacher only generates problems for the student that confirm her hypothesis!

In some cases, a teacher may believe her hypothesis so strongly that she will ignore disconfirmations that exist or decide that these disconfirmations are merely random noise.(11) One way this can be avoided is by the technique of differential diagnosis [Rubin 1975] in which one always generates at least two hypotheses and then chooses test problems that separate them.

Another important issue concerns the relationship between the language used to describe a student's errors and its effect on what a teacher should do to remediate it. Is the language able to convey to the student what he is doing wrong? Should we expect teachers to be able to use language as the tool for correcting the buggy algorithms of students? Or should we only expect teachers to be able to understand what the bug is and attempt remediation the student with things like manipulative math tools? The following are quotes of student teacher hypotheses taken from protocols of BUGGY which give a good idea of how difficult it is to express procedural ideas in English. The descriptions in parentheses are BUGGY's (prestored) explanations of the bugs.

"Random errors in carryover." (Carries only when the next column in the top number is blank.)

"If there are less digits on the top than on the bottom she adds columns diagonally." (When the top number has fewer digits than the bottom number, the numbers are left-justified and then added.)

"Does not like zero in the bottom." (Zero subtracted from any number is zero.)

"Child adds first two numbers correctly then when you need to carry in the second set of digits child adds numbers carried to bottom row then adds third set of digits diagonally finally carrying over extra digits." (The carry is written in the top number to the left of the column

(10) In computer programming metaphors, this corresponds to the debugging activities of resubmitting the program because the computer must have made a mistake and of throwing the whole program away and starting over from scratch.

(11) There is, of course, some amount of "processor failure" as kids are often all too human.

being carried from and is mistaken for another digit in the top number.)

"Sum and carry all columns correctly until get to last column. Then takes furthest left digit in both columns and adds with digit of last carried amount. This is in the sum." (When there are an unequal number of digits in the two numbers, the columns that have a blank are filled with the left-most digit of that number.)

What does this say to us? Even when one knows what the bug is in terms of being able to mimic it, how is one going to explain it to the student having problems? Considering the above examples, it is clear that anyone asked to solve a set of problems using these explanations would no doubt have real trouble. One can imagine a student's frustration when the teacher offers an explanation of why he is getting problems marked wrong, and the explanation is as confused and unclear as these are. For that matter, when the correct procedure is described for the first time, could it too be coming across so unclearly?

This issue is further complicated by the existence of another important issue: there are fundamentally different bugs which cause identical behavior! In other words, there can be several distinct bugs that are logically equivalent and always generate the same "answers". For example, here is a set of problems:

38	186	298	89
+46	+254	+169	+64
174	2330	2357	243

The underlying flaw in the student's procedure (his bug) can be described as "The columns are added without carries and the left-most digit in the answer is the total number of carries required in the problem." In this case, the student views the carries as tallies to be counted and added to the left of the answer. But another equally plausible bug also exists; the student is placing the carry to the left of the next digit in the top number instead of adding it to the digit (i.e. he is actually carrying ten times the carry digit). This generates the same symptoms. So even when the teacher is able to describe clearly what she believes is the underlying bug, he may be addressing the wrong one. The student may actually have either one of these bugs!(12)

9. Pedagogical Issues More Specific to Students

We feel that all of the issues discussed above are as important for school-level students as they are for teachers. There is great value in introducing students to procedural notions. The BUGGY system provides a well controlled environment for such an introduction, as well as one that can be meaningfully related to standard curricula. The diagnostic task of a player requires studying the procedural skill per se as opposed to merely performing it. This can be especially important as students begin algebra, which is their first exposure to "parameterized" procedures.

Additionally, BUGGY can be used to explore the powerful ideas of hypothesis formation, debugging, debugging strategies, and so on. To further encourage thought along these lines, the BUGGY environment can be adapted to provide students with a specialized language for writing procedures. (Note that such an environment could provide immediate focussing on debugging strategies -- a topic usually left until the end in most secondary school programming courses.) In this environment it appears to be possible to construct a very intelligent debugging agent or programming assistant as well as a computer-based tutorial helper that can aid a student when he gets stuck. This kind of programming assistant has been impossible to provide for the open worlds usually encountered in the environments of general-purpose programming languages. Having students write their own procedures also allows use of a game developed in the SOPHIE environment [Brown, Rubinstein and Burton 1976] where one student writes a procedure introducing a bug and another student tries to discover it by presenting test problems.

Another reason for having students develop a language for talking about procedures, processes, bugs, etc. is that this language enables the student to talk about (and think about) procedures and the underlying causes of his own errors. This is important in its own right, but it also gives a student the motivation and the apparatus for stepping back and critiquing his own thinking, as well as saying something interesting and useful about his errors. This is especially important given the fact that there's been so little success in getting students to look over their own work (such as estimating answers) and to use this perusal to good advantage.

An important side effect of a student's involvement with BUGGY is exposure to the idea of role reversal.(13) In order to communicate effectively with others, children must learn not only the language itself, but the use of "social speech": speech that takes into account the knowledge and perspective of another person [Krauss and Glucksberg 1977]. Piaget uses the term "childhood egocentrism" to describe the child's inability to detach himself from his own point of view and take into consideration another's perspective. Although Krauss and Glucksberg agree that egocentrism plays a large part in very young children's speech, they believe that in older children the ability to role play only breaks down when they are faced with a demanding cognitive task. We believe that taking on the viewpoint of the errant student by analyzing another's

(12) This leads to an interesting question concerning how one can "prove" two different descriptions of bugs entail logically the same surface manifestations.

(13) This idea is due to Tim Barclay at the Cambridge Friends School who has been experimenting with various uses of Buggy with sixth through eighth graders.

mistakes can be a demanding one and that this kind of exercise can be beneficial to the development of "social speech".

10. Where From Here?

Most of what the students learned while using BUGGY they learned or discovered, in some sense, on their own. BUGGY does no explicit tutoring. It simply challenges their theories and encourages them to articulate their thoughts.(14) The rest of the learning experience occurred either through the sociology of team learning or from what a person abstracted on his own. The next step in improving the educational effectiveness of BUGGY is (i) to implement an intelligent tutor to critique the example test problems the students create, (ii) to point out interesting facets of their debugging strategies and (iii) to isolate manifested weaknesses in their strategies. Our experiences indicate that such a tutor would be very helpful for middle school and remedial students where it could keep students from getting caught in unproductive ruts and could help focus their attention on the structure of the procedures themselves.

Along these same lines the "expert" portion of the procedural net should be made "articulate" in the sense of being able to explain and justify the subprocedures it uses. This would allow a student to pose a problem to the system and obtain a running account of the relevant procedures as the "expert" solves the problem. A useful notion may be to have additional explanation or justification of each symbolic procedure (in the network) expressed in terms of a "physical" procedure using manipulative tools (such as Dienes' blocks). In this way, the execution of each symbolic procedure could cause its analogous physical procedure to be displayed on a graphics device, thereby letting the student see the numeric or abstract computation unfold in conjunction with a physical model of the computation. This directly attacks the problem of getting procedures to take on "meaning" for a student which, we believe, is accomplished by recognizing mappings or relations between the new procedures and existing procedures or experiences (reality).

Another area for extension concerns the psychological validity of the skill decomposition (and buggy variants) in the procedural network. Determining the proper functional breakdown of a skill into its subskills is critical to the psychological validity of the model and the resulting behavior of the system. If the breakdown of the skill is not correct, bugs that people would consider simple may be difficult to model while those suggested by the model may be judged by people to be "unrealistic". From the network designer's point of view this leads to the issue of choosing or constructing one structural decomposition instead of another. We are just beginning to acquire a large data base of arithmetic errors from Institute for Mathematical Studies in the Social Sciences at Stanford [Searle, B. et al. 1976] and will be testing to see how well our diagnostic model accounts for all of them. In particular, we are concerned not only with how many underlying bugs our current model captures, but also how many bugs our network predicts that never show up. A more subtle issue concerns the validity of the actual functional decomposition of the skills in the network. Measuring the "correctness" of a particular network is a problematic issue as there are no clear tests of validity, but issues such as the ease or "naturalness" of inclusion of newly discovered bugs and the appearance of combinations of bugs within a breakdown can be investigated.

We are also in need of a theory of what makes an underlying bug easy or difficult to diagnose. Simple conjectures concerning the depth of the bug from the surface don't seem to work, but more sophisticated measures might. It's hard to see how to predict the degree of difficulty in diagnosing a particular bug without a precise information processing or cognitive theory of how people actually formulate conjectures about the underlying bug or cause of an error.

Finally, we note that we have left open the entire issue of a semantic or teleological theory of how bugs are generated in the first place. The need for such a theory is important for at least two reasons. First it could provide an interesting theoretical mechanism that would account for the entire collection of empirically arrived at bugs, and second, it provides the next step in a semantically based productive theory of student modelling.

(14) As a historical footnote, BUGGY was originally developed to explore the psychological validity of the procedural network model for complex procedural skills. During that investigation we realized the pedagogical potential of even this simple version of BUGGY as an instructional medium.

References

Barr, A. et al. A rationale and description of the basic instructional program. Psychology and Education Series, Stanford University, Technical Report 228, April 1974.

Brown, J.S. et al. Structural models of a student's knowledge and inferential processes. BBN Proposal No. P74-CSC-10, Bolt Beranek and Newman, Inc., Cambridge, Massachusetts, April 1974.

Brown, J.S. & Burton, R.R. Systematic understanding: Synthesis, analysis, and contingent knowledge in specialized understanding systems. In D. Bobrow and A. Collins (Eds.), Representation and Understanding: Studies in Cognitive Science, New York: Academic Press, 1975.

Brown, J.S., Collins, A. & Harris, G. Artificial intelligence and learning strategies. To appear in H.F. O'Neil (Ed.), Learning strategies. New York: Academic Press, 1978, in press.

Brown, J.S., Rubinstein, R. & Burton, R.R. Reactive learning environment for computer assisted electronics instruction. BBN Report No. 3314, A.I. Report No. 1, Bolt Beranek and Newman Inc., Cambridge, Massachusetts, October 1976.

Burton, R.R. & Brown, J.S. A tutoring and student modelling paradigm for gaming environments. In Proceedings for the Symposium on Computer Science and Education, Anaheim, California, February 1976.

Carbonell, J. & Collins, A. Natural semantics in artificial intelligence. In Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, 1973.

Carr, B. & Goldstein, I. Overlays: A theory of modelling for computer aided instruction. Massachusetts Institute of Technology, AI Memo 406, February 1977.

Collins, A., Warnock, E. & Passafiume, J. Analysis and synthesis of tutorial dialogues. In G.H. Bower (Ed.), Advances in Learning and Motivation, Vol. 9, 1975.

Easley, J.A., Jr. & Zwoyer, R.E. Teaching by listening - toward a new day in math classes. Contemporary Education, Fall 1975, 47(1), 19-25.

Goldstein, I. The computer as coach: An athletic paradigm for intellectual education. Massachusetts Institute of Technology, AI Memo 389, January 1977.

Goldstein, I. Understanding simple picture programs. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Technical Report 294, September 1974.

Krauss, R.M. & Glucksberg, S. Social and nonsocial speech. Scientific American, 1977, 236(2), 100-105.

Miller, M. & Goldstein, I. Overview of a linguistic theory of design. Massachusetts Institute of Technology, AI Memo 383, December 1976.

Rich, C. & Shrobe, H.E. Initial report on a LISP programmer's apprentice. Massachusetts Institute of Technology, AI-TR-354, December 1976.

Rubin, A. Hypothesis formation and evaluation in medical diagnosis. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, AI-TR-316, January 1975.

Sacerdoti, E. A structure for plans and behavior. Stanford Research Institute, Artificial Intelligence Center Technical Note 109, August 1975.

Searle, B., Friend, J. & Suppes, P. The Radio Mathematics Project: Nicaragua 1974-1975. Institute for Mathematical Studies in the Social Sciences, Stanford University, 1976.

Self, J.A. Student models in computer-aided instruction. Int. J. Man-Machine Stud., 1974, 6, 261-276.

West, T. Diagnosing pupil errors: looking for patterns. The Arithmetic Teacher, November 1971.