association with likely misconceptions à la MENO-II (see Section 11.4), and an object-oriented approach to student modeling reminiscent of the bite-sized architecture. One of the project's main theme is the effective integration of these components, with a view to implementing a global pedagogical strategy; this interest in pedagogical principles extends the research line of WEST. In this regard, SPIRIT's most distinctive feature is the integration of two didactic subcomponents, the "tutor" and the "mentor," which enable SPIRIT to vary its didactic strategy on the basis of its student model, switching between unobtrusive coaching and directive tutoring.

Chapter 8

# Bugs in procedural skills: the 'buggy repair step' story

Like WEST, the study described in this chapter was spawned in the context of the SOPHIE project. Again, the choice of domain is motivated by its simplicity compared to electronics, but this study investigates diagnosis rather than coaching strategies. Recall that WEST's differential student model, like WUSOR-II's overlay, views the student's knowledge state as a subset of issues comprising the correct skill. There is no provision for defining incorrectly encoded issues that the student may have acquired. To make this provision possible, the study covered here concentrates on the notion of *bug*. This term, borrowed from computer programming jargon, has become a buzzword among people interested in student models; it refers to errors internalized by the student that are explicitly represented in the student model. Bugs are usually procedural or localized errors rather than deep, pervasive misconceptions. The study

of the bugs commonly encountered by a student population in a domain is often called a *theory of bugs*.

This chapter covers the evolution of a long project that started with BUGGY, a purely descriptive theory of bugs whereby the model enumerates observable bugs. We also present DEBUGGY, which extends this model into a diagnostic system. Thereafter, we discuss REPAIR and STEP theories, which attempt to provide "generative bug stories" to explain the genesis of observed bugs in terms of underlying cognitive models.

## 8.1 BUGGY: an enumerative theory of bugs

The development of BUGGY began with the analysis of protocols produced by students who were solving problems requiring algebraic manipulations (Brown et al., 1975b). Later, to avoid obscuring the analysis with algebra's numerous rules and to take advantage of available data, Brown and Burton (1978a) decided to explore their modeling paradigm in the domain of arithmetic skills. More specifically, the BUGGY project has come to be generally associated with place-value subtraction.

### 8.1.1 A modular representation of subskills

The preliminary studies of students performing algebra had revealed the need for a special representational scheme for procedural skills such that any constituent of the skill that can be mislearned is represented independently. This scheme was to support the design of a *diagnostic model*: that is, a model of the student's current skill that would reflect its exact composition of correct and incorrect elementary subskills. These specifications led to the adoption of the *procedural network* as a representational mechanism for diagnostic models (Brown et al., 1977b). A procedural network for a skill is a decomposition of that skill into subprocedures which are linked together in a lattice of subgoals. Figure 8.1 shows a procedural network for subtraction. The links stand for the procedure-calling relation, down to a set of primitive actions. A procedural network is executable, and can thus be run on a set of problems to model the skill that it represents. Its structure is also inspectable in that, unlike a flat set of rules, it uses well-defined subprocedure calls.

A diagnostic modeling scheme based on a procedural network contains all the necessary subskills for the global skill, as well as all the possible buggy variants (or *bugs*) of each subskill. It can replace an



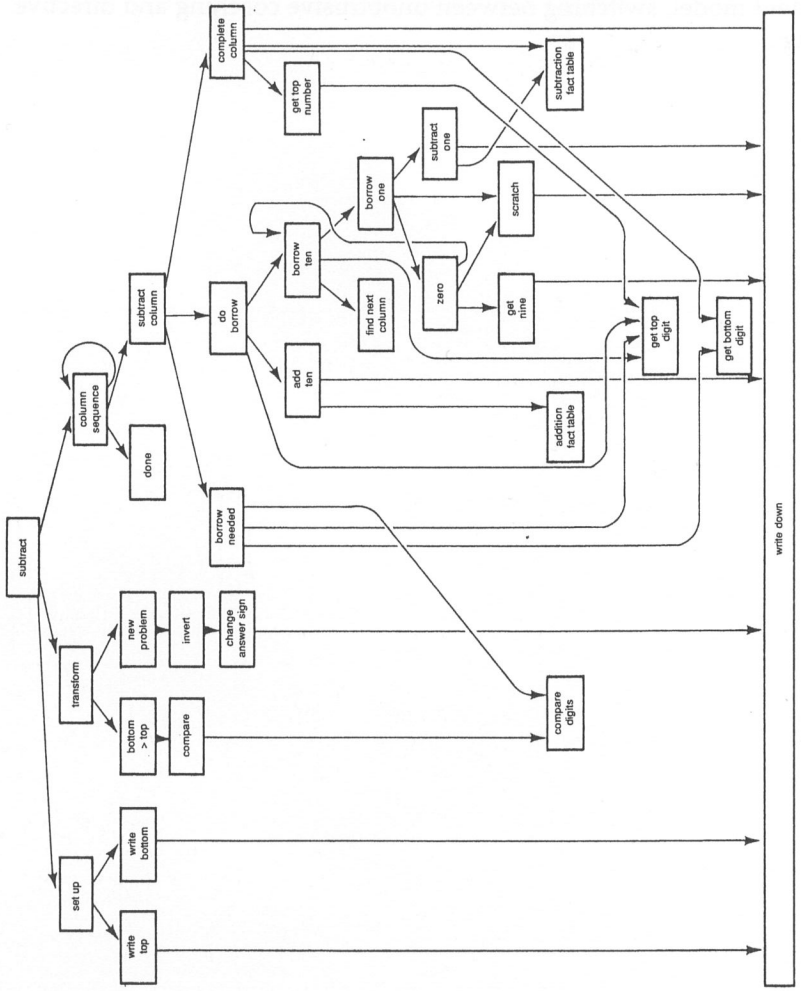


Figure 8.1 A procedural network for subtraction (after Brown and Burton, 1978a).

| | |
|---|---|
| 143<br>-28<br>125 | The student subtracts the smaller digit in each column from the larger digit regardless of which is on top. |
| 143<br>-28<br>125 | When the student needs to borrow, he adds 10 to the top digit of the current column without subtracting 1 from the next column to the left. |
| 1300<br>-522<br>878 | When borrowing from a column whose top digit is 0, the student writes 9 but does not continue borrowing from the column to the left of the 0. |
| 140<br>-21<br>121 | Whenever the top digit in a column is 0, the student writes the bottom digit in the answer, i.e., 0-*N*=*N*. |

**Figure 8.2 Examples of bugs in subtraction procedures (Brown and Burton, 1978a).**

individual subskill in the procedural network by one of its bugs, and thus attempt to reproduce a student's incorrect behavior. If such a faulted network does obtain the same answers as the student on a sufficient set of problems, the bugs that have replaced their correct siblings in the network are then posited as being those possessed by the student. Such an instantiation of the subprocedure calls, which actually models the student in a *deterministic* fashion, is considered a *deep-structure model*: it explains the student's incorrect behavior in terms of a set of exact internalized errors, rather than of, say, a probabilistic estimate of correct and incorrect answers such as is typical of stochastic models. When presenting the examples of bugs in subtraction procedures shown in Figure 8.2, Brown and Burton actually invite readers to try to discover the bugs themselves first, in order better to appreciate the difficulty of the task. Note, for instance, that the first two lines show two different bugs that produce the same answer.

### Limitations of enumerative theories of bugs

With this deterministic approach to accounting for errors, the precise information about the nature of internalized bugs contained in the deep structure is very helpful for pedagogical purposes. In this sense, BUGGY's diagnostic model is a real milestone in the research on student modeling. However, we should mention two important limitations of the paradigm. First, it is clear that the modeling capability of the network hinges on a proper decomposition of the skill to a granularity level at which each single bug can be isolated as a variant of a separate procedure. In general, this will require in-depth analysis both of the domain and of actual performances by large numbers of students. Even then, the fixed library of bugs will limit the modeling power of the system to student errors it knows about.



The second limitation is perhaps more fundamental because of its repercussions for remediation. While BUGGY's procedural network is able to single out bugs in the student's procedure, it does not attempt to explain them. In the representational language of procedural networks, any buggy subprocedure is only related to the correct one by virtue of carrying out the same subgoal, that is, of being called in the same circumstances. Apart from that, they are two individual pieces of code: the language has no facility for explicitly representing the semantic nature of the deviation a bug stands for, let alone explaining how it was generated. After describing some implemented systems based on the BUGGY paradigm, we will turn to some research that was undertaken to address these two limitations.

## 8.1.2 The nature of bugs in procedural skills

The idea of a diagnostic model assumes that, in problems that require procedural skills, mistakes are due to the correct execution of an incorrect procedure, and not to an inability to follow an otherwise correct procedure. In fact, in the process of decomposing arithmetic skills and defining libraries of bugs to account for existing data, Brown and Burton found that many students manifest a nearly perfect consistency in following incorrect procedures. Since this discovery confirms the intuition underlying the notion of diagnostic model—and contradicts the widely held belief that students who make mistakes are simply not good at following procedures—a game was devised on the BUGGY model to train teachers in analyzing student errors.

In this game, the program instantiates a simple bug in the diagnostic model and asks a team of teachers to discover it. They are to make up problems and submit them for the buggy model to solve, until they can single out the bug. As in SOPHIE-II's game, the team format is helpful in getting participants to articulate their strategies and hypotheses. Indeed, experimentation with the game revealed that people are not very good at giving precise procedural descriptions verbally, as can be seen in Figure 8.3, which shows a session with BUGGY in the domain of addition. Apparently, student teachers found the game illuminating in that what looked at first like a completely random behavior could eventually be

BUGGY: Welcome to Buggy. I have chosen a bug.
Here is an example of the bug.

```
 17
+ 5
 13
```

Now you can give me problems to determine the bug.

Team:
```
 18   43
+ 6  +79
```

BUGGY:
```
 15   23
```

Team: Got the bug! Student adds the number to be carried in the last column worked in.

*This cryptic description is typical of those observed. The team is trying to say that if a column adds to more than 9, the unit digit and the carry digit are added together and written as the answer for that column. BUGGY does not attempt to understand the English descriptions, but heuristically generates five problems on which the players may test their hypothesis. The generation of these problems involves some knowledge about the manifestations of individual bugs.*

BUGGY: Here are some problems to test your theory about the bug. What is:

```
 21
+39
```

Team: 51

BUGGY: That's not the bug I have. Using my bug:

```
 21
+39
 15
```

Try giving me more examples ...

*The first hypothesis did account for the observed behavior. Unfortunately, it is not the student's bug.*

Team:
```
  51    99    68
+1707  +99  + 9
```

BUGGY:
```
  21    36    23
```

*The team needs to try problems of a greater diversity. This highlights the importance of considering a variety of hypotheses before positing a diagnosis. Notice the possibly humorous, probably disastrous results that would have transpired if the team had stopped earlier and tried to tutor the student on the basis of their initial belief.*

Team: Got the bug! The student is adding all the digits together.

BUGGY: Here are some problems to test your theory about the bug. What are:

```
  33   1091    8    28   90
+99   + 60  +34  +70  + 6
```

Team:
```
  24     17   15   17   15
```

BUGGY: Very good. My description of the bug is: The student always sums up all the digits, with no regard to columns.

**Figure 8.3 A BUGGY game with teacher students (after Brown and Burton, 1978a).**



traced to a consistent buggy procedure. Moreover, exposure to the game significantly improved their ability to detect error patterns (Brown et al., 1977a).

A similar result was achieved when the game was tried in a classroom with seventh and eighth graders. This "role reversal," in which students did the diagnosing, had a rather interesting effect. At first, students would find the program's mistakes really "dumb," but after some exposure, they would realize that there was always a systematic explanation for its behavior; they became less judgmental once they saw that surface errors revealed deeper bugs. Brown and Burton find this result "particularly exciting, since it paves the way for students to see their own faulty behavior not as being a sign of their own stupidity, but as a source of data about their own errors" (Brown and Burton, 1978a, p. 172).

## 8.2 DEBUGGY: a diagnostic system

Burton (1982) describes DEBUGGY, a sophisticated diagnostic system extending the BUGGY model. He discusses the issues of search strategies and inconsistent data involved in inferring the deep-structure model from problems solved by students. Two versions of DEBUGGY are presented: an off-line version, which analyzes tests taken by students, and an on-line version, IDEBUGGY, which diagnoses the student's procedure incrementally while giving him new problems to solve.

### 8.2.1 Off-line diagnosis: heuristic search strategies

To construct the student model as an individualized procedural network, the simplest method that comes to mind is of course generate-and-test: each bug in turn replaces a subprocedure in the correct model, until a bug is found with which the model reproduces all the student's answers on the problem set. In reality, however, the situation is complicated by the need to deal with compound bugs, whereby primitive bugs combine to manifest in ways that can be quite nontrivial. A study of subtraction problems (VanLehn, 1982) shows that 37% of the students manifest multiple bugs, sometimes as many as four together. Even though the number of primitive bugs is often limited (about 110 observed bugs for subtraction), trying all combinations, even only up to four, will generate too large a search space for exhaustive testing.

To make the search computationally tractable, DEBUGGY assumes that the simple bugs that interact to form multiple ones can also be detected individually. This means that there exists at least one problem in which either the simple bugs appear in isolation or the complex bug is