and informal electronic learning. *Journal of Educational Computing Research, 1*, 179–201.
Bundy, A. (1983). *The computer modeling of mathematical reasoning*. Orlando, FL: Academic Press.
Collins, A. (1986). Teaching reading and writing with personal computers. In J. Orasanu (Ed.), *A decade of reading research: Implications for practice* (pp. 171–187). Hillsdale, NJ: Lawrence Erlbaum Assoc.
Collins, A., & Smith, E. E. (1982). Teaching the process of reading comprehension. In D. K. Detterman & R. J. Sternberg (Eds.), *How much and how can intelligence be increased?* (pp. 173–185). Norwood, NJ: Ablex.
Flower, L. S., & Hayes, J. R. (1980). The dynamics of composing: Making plans and juggling constraints. In L. W. Gregg & E. R. Steinberg (Eds.), *Cognitive processes in writing* (pp. 31–50). Hillsdale, NJ: Lawrence Erlbaum Assoc.
Palincsar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and monitoring activites. *Cognition & Instruction, 1*, 117–175.
Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

VanLehn, K. (1988). Toward a theory of impasse-driven learning. In H. Mandl & A. Lesgold (Eds.), Learning Issues for Intelligent Tutoring Systems (pp. 19-41). New York, NY: Springer.

2

Toward a Theory of Impasse-Driven Learning

KURT VANLEHN

Introduction

Learning is widely viewed as a knowledge communication process coupled with a knowledge compilation process (Anderson, 1985). The communication process interprets instruction, thereby incorporating new information from the environment into the mental structures of the student. Knowledge compilation occurs with practice. It transforms the initial mental structures into a form that makes performance faster and more accurate. Moreover, the transformed mental structures are less likely to be forgotten. At one time, psychology concerned itself exclusively with the compilation process by using such simple stimuli (e.g., nonsense syllables) that the effects of the communication process could be ignored. The work presented here uses more complicated stimuli, the calculational procedures of ordinary arithmetic. For such stimuli, the effects of the knowledge communication process cannot be ignored. Later in this chapter it is shown that certain types of miscommunication can cause students to have erroneous conceptions. The long-term objective of the research reported here is to develop a theory of the neglected half of learning, knowledge communication. The experimental methods employed are designed to show the effects of knowledge communication and hide the effects of knowledge compilation. Consequently, whenever the term *learning* appears, it is intended to mean knowledge communication.

Earlier work (Brown & VanLehn, 1980; VanLehn, 1982, 1983a, 1983b, in press) has shown that students often reach "impasses" while trying to use a procedural skill that they are acquiring. An impasse occurs when the step that they believe should be executed next cannot be performed. If they are in a test-taking situation, where they may not receive help, they perform a specific, simple kind of problem solving, called "repair." This chapter speculates about what happens when impasses occur during instructional situations, where help is available. The conjecture is that the help that the student receives—either from the teacher, from examining the textbook, or from other information sources—is reduced to the sequence

of actions that will get the student past the impasse. This action sequence is generalized to become a new subprocedure. The new subprocedure is inserted into the old procedure at the location where the impasse occurred. The proposed learning process is called impasse-driven learning.

The research presented here began with the "buggy" studies of Brown and Burton (1978). Those studies found that students of certain procedural skills, such as ordinary multicolumn subtraction, had a surprisingly large variety of bugs (i.e., small, local misconceptions that cause systematic errors). Early investigations into the origins of bugs yielded a theory of procedural problem solving, named Repair Theory (Brown & VanLehn, 1980). Among other accomplishments, Repair Theory predicted the occurrence of certain patterns of short-term instabilities in bugs. These instabilities were subsequently found (VanLehn, 1982). Recent research has investigated the relationship between the curriculum, the students' learning processes, and the acquisition of bugs. A learning theory has been added to Repair Theory, yielding an integrated explanation for the acquisition of correct and buggy procedures (VanLehn, 1983a; 1983b, in press). It describes learning at large "grain size." Given a lesson and a representation of what the student knows before the lesson, the theory predicts what the student's knowledge state will be after the lesson.

Recently, attention has turned toward describing learning at a finer grain size. The object of the current research is to describe the student's cognitive processing during a lesson. The research strategy is to augment Repair Theory, which already provides a fine-grained account of problem-solving processes during diagnostic testing sessions, so that it provides a fine-grained account of learning. If this strategy succeeds, the cognitive processes will account for both problem-solving behavior and lesson-learning behavior. In contrast to the old theory, the new theory will (1) provide a more integrated account of cognition and (2) describe learning behavior at a finer grain size. In order to make it easier to contrast old and new theories, the new theory is identified as RT2, and the old theory, which is a conglomerate of Repair Theory and the large-grained learning theory, is referred to as RT1.

This chapter introduces RT2. It is, for the most part, speculation. Unlike RT1, RT2 has not been implemented as computer simulation, nor has its internal coherence and empirical accuracy been scrutinized with competitive argumentation (VanLehn, Brown, & Greeno, 1984). Although the ideas behind RT2 are simple extensions of the principles of RT1, they seem capable of explaining much about human behavior. Moreover, they relate to current research in machine learning and language acquisition. A discussion of RT2, even in its current underdeveloped form, should be at least timely, and perhaps interesting as well.

The chapter begins with a discussion of the task domain and the kinds of behavior one finds students displaying. It then introduces the old theory, RT1. Readers who are familiar with RT1 from earlier publications may safely skip the first two sections. The remainder of the chapter presents RT2 and discusses its relationship to other work in cognitive science.

Learning Elementary Mathematical Skills

The goal of this research is to develop a rigorously supported theory of learning by taking advantage of the new modeling power of artificial intelligence (AI). The long-term research strategy is to begin by studying a particular kind of cognition, then if all goes well, to test the theory's generality on other kinds of cognition. The initial studies focused on how elementary school students learn ordinary, written arithmetic calculations. The main advantage of arithmetic procedures, from a methodological point of view, is that they are virtually meaningless to most students (Resnick, 1982). Most students treat arithmetic procedures as arbitrary formal manipulations (i.e., "symbol pushing"). Although this may frustrate teachers, it allows psychologists to study a complex skill without having to model a whole world's worth of background knowledge.

This section introduces the domain. First, it describes the instruction that students receive, and then it describes the behavior they produce. The theory's main job is to explain what kinds of mental structures are engendered by that instruction and how those structures guide the production of the observed behavior.

Learning From Lesson Sequences of Examples and Exercises

In a typical American school, mathematical procedures are taught incrementally via a lesson sequence that extends over several years. In the case of subtraction, there are about 10 lessons in the sequence that introduce new material. The lesson sequence introduces the procedure incrementally, one step per lesson, so to speak. For instance, the first lesson might show how to do subtraction of two-column problems. The second lesson demonstrates three-column problem solving. The third introduces borrowing, and so on. The 10 lessons are spread over about 3 years, starting in the late second grade (i.e., at about age 7). These lessons are interleaved with lessons on other topics, as well as many lessons for reviewing and practicing the material introduced by the 10 lessons. In the classroom a typical lesson lasts an hour. Usually, the teacher solves some problems on the board with the class, then the students solve problems on their own. If they need help they ask the teacher, or they refer to worked examples in the textbook. A textbook example consists of a sequence of captioned "snapshots" of a problem being solved (Figure 2.1). Textbooks have very little text explaining the procedure, perhaps because young children do not read well. Textbooks contain mostly examples and exercises.

Take a ten to make 10 ones.	Subtract the ones.	Subtract the tens.
2 15 ~~3~~ ~~5~~ − 1 9	2 15 ~~3~~ ~~5~~ − 1 9 6	2 15 ~~3~~ ~~5~~ − 1 9 1 6

FIGURE 2.1. A typical textbook example.

DESCRIBING SYSTEMATIC ERRORS WITH "BUGS"

The observable output of the students' learning process is their performance while solving exercise problems. Error data are a traditional measure of such performance. There have been many empirical studies of the errors that students make in arithmetic (Ashlock, 1976; Buswell, 1926; Brueckner, 1930; Brownell, 1935; Cox, 1975; Roberts, 1968; Lankford, 1972). A common analytic notion is to separate systematic errors from slips (Norman, 1981). Systematic errors appear to stem from consistent application of a faulty method, algorithm, or rule. Slips are unsystematic, "careless" errors (e.g., facts errors, such as $7 - 3 = 5$). Since slips occur in expert performance as well as student behavior, the common opinion is that they are due to inherent "noise" in the human information processor. Systematic errors, on the other hand, are taken as stemming from mistaken or missing knowledge, the product of incomplete or misguided learning. Only systematic errors are used in testing the present theory. See Siegler and Shrager (1984) for a developmental theory of addition slips.

Brown and Burton (1978) used the metaphor of bugs in computer programs in developing a precise, detailed formalism for describing systematic errors. A student's errors are accurately reproduced by taking a formal representation of a correct procedure and making one or more small perturbations to it, such as deleting a rule. The perturbations are called bugs. A systematic error is represented as a list of one or more bugs. Bugs describe systematic errors with unprecedented precision. If a student makes no slips, then his or her answers on a test exactly match the buggy algorithm's answers, digit for digit. Bug data are the main data for testing this theory.

Burton (1982) developed an automated data analysis program, called Debuggy. Using it, data from thousands of students learning subtraction were analyzed, and 76 different kinds of bugs were observed (VanLehn, 1982). Similar studies discovered 68 bugs in addition of fractions (Shaw, 1982). Similar studies discovered 68 bugs in addition of fractions (Shaw, Standiford, Klein, Tatsuoka, 1982), several dozen bugs in simple linear equation solving (Sleeman, 1985), and 57 bugs in addition and subtraction of signed numbers (Tatsuoka & Baillie, 1982).

It is important to stress that bugs are only a notation for systematic errors and not an explanation. The connotations of "bugs" in the computer programming sense do not necessarily apply. In particular, bugs in human procedures are not always stable. They may appear and disappear over short periods of time, often with no intervening instruction, and sometimes even in the middle of a testing session (VanLehn, 1982). Often, one bug is replaced by another, a phenomenon called bug migration.

Mysteries abound in the bug data. Why are there so many different bugs? What causes them? What causes them to migrate or disappear? Why do certain bugs migrate only into certain other bugs? Often a student has more than one bug at a time—why do certain bugs almost always occur together? Do co-occurring bugs have the same cause? Most importantly, how is the educational process involved in the development of bugs? One objective of the theory is to explain some of these bug mysteries.

Another objective is to explain how procedural skills are acquired from multiyear curricula. This objective seems to require longitudinal data, where each student in the study is tested several times during the multiyear period. Such data are notoriously difficult to acquire. Bug data are readily available and nearly as good. The bug data discussed here were obtained by testing students at all stages in the curriculum. Thus, the bug data are like between-subjects longitudinal data. Instead of testing the same student at several times at different stages of his or her learning, different students at different stages are tested just once. As shown later, such data can perform nearly as well as longitudinal data in testing a learning theory, and yet they are much easier to collect.

An Introduction to the Model: Explaining Always-Borrow-Left

Most of the mental structures and processes proposed by the theory can be introduced and illustrated by going through an explanation for a certain subtraction bug, called Always-Borrow-Left. Students with this bug always borrow from the leftmost column in the problem no matter which column originates the borrowing. Problem A shows the correct placement of borrow's decrement. Problem B shows the bug's placement.

(A)	(B)	(C)
5 3 ~~6~~ 15 − 1 0 9 2 5 6	2 ~~3~~ 6 15 − 1 0 9 1 6 6	5 ~~6~~ 15 − 1 9 4 6

Always-Borrow-Left is moderately common. In a sample of 375 students with bugs, six students had this bug (VanLehn, 1982). It has been observed

for years (Buswell, 1926, p. 173, bad habit number s27). However, this theory is the first to offer an explanation for it.

The explanation begins with the hypothesis that students use induction (generalization of examples) in learning where to place the borrow's decrement. All the textbooks used by students in our sample introduce borrowing using only two-column problems, such as in preceding problem C. Multicolumn problems, such as A, are not used. Consequently, the student has insufficient information to induce an unambiguous description of where to place the borrow's decrement. The correct placement is in the left-adjacent column, as in A. However, two-column examples are also consistent with decrementing the leftmost column, as in B.

The next hypothesis of the theory is that when a student is faced with such an ambiguity in how to describe a place, the student takes a conservative strategy and saves all the relevant descriptions. When inducing from two-column problems (e.g., C), the student describes the borrow-from column as "a column that is both left-adjacent to the current column and the left-most column in the problem."

Suppose that our student is given a diagnostic test at this point in the lesson sequence and that the test contains borrowing problems of all kinds. Suppose the student is faced with solving problem D.

(D)
$$\begin{array}{r} 3\,6\,5 \\ -\,1\,0\,9 \\ \hline \end{array}$$

(E)
$$\begin{array}{r} 3\,6\,{}^{1}5 \\ -\,1\,0\,9 \\ \hline \end{array}$$

The student starts to borrow, gets as far as E, and is suddenly stuck. The student's description of where to borrow from is ambiguous because there is no column that is *both* left-adjacent and the leftmost column. In the terminology of the theory, getting stuck while problem solving is called reaching an *impasse*.

It is hypothesized that whenever students reach an impasse on a test, they engage in *local problem solving*. Local problem solving is just like classical puzzle solving (Newell & Simon, 1972) in that there is an initial state, a desired final state, and state-change operators. Here, the initial state is being stuck, and the desired final state is being unstuck. Unlike traditional problem solving, in local problem solving the state-change operators do not change the state of the exercise problem. Instead, they change the *state of the interpreter* that is executing the procedure. The operators do things like pop the stack of goals or relax the criterion for matching a description to the exercise problem. They do not do things like writing digits on the test paper. Because the local problem solver modifies the state of the procedure's interpretation, it is a kind of *metalevel* problem solving. The sequences of metalevel operators that succeed in getting students unstuck are called *repairs*. Note that what is being repaired is, roughly speaking, the impasse. Repairs do not change the procedure. To

put it in terms of Newell's (1980) problem space hypothesis, the procedure works in one problem space, and local problem solving works in a second problem space that is "meta" to the base problem space. Returning to our stuck student, three common repairs to the impasse follow:

(F)
$$\begin{array}{r} {\scriptstyle 2} \\ \not{3}\,6\,{}^{1}5 \\ -\,1\,0\,9 \\ \hline \end{array}$$

(G)
$$\begin{array}{r} {\scriptstyle 5} \\ 3\,\not{6}\,{}^{1}5 \\ -\,1\,0\,9 \\ \hline \end{array}$$

(H)
$$\begin{array}{r} 3\,6\,{}^{1}5 \\ -\,1\,0\,9 \\ \hline 6 \end{array}$$

In F, the student has relaxed the description of which column to borrow from by ignoring the restriction that the column be left-adjacent to the current column. The remaining restriction, that the column be the leftmost column in the problem, has the student decrement the hundreds column, as shown in F. This is one repair. It generates the bug Always-Borrow-Left. Another repair is shown in G. Here, the student has relaxed the borrow-from description by ignoring the leftmost requirement. The decrement is placed in the left-adjacent column, yielding G. This repair generates a correct solution to the problem. In H, the student has chosen to skip the borrow-from entirely and go on to the next step in the procedure. This repair generates a bug that is named Borrow-No-Decrement-Except-Last, because it only executes a borrow-from when it is unambiguous about where to place the decrement; and that occurs only when the borrow originates in the last possible column for borrow. To sum up, three different repairs to the same impasse generate two different bugs and a correct version of subtraction.

As mentioned earlier, students' bugs are not like bugs in computer programs, because students' bugs are unstable. Students shift back and forth among bugs, a phenomenon called bug migration. The theory's explanation for bug migration is that the student has a stable underlying procedure but the procedure is incomplete in such a way that the student reaches impasses on some problems. The student can apply any repair she can think of. Sometimes she chooses one repair, and sometimes she chooses others. The different repairs manifest themselves as different bugs. So some bug migration is caused by varying the choice of repairs to a stable, underlying impasse. In particular, the theory predicts that the three repairs just discussed ought to show up as a bug migration. In fact they do. Figure 2.2 is a verbatim presentation of a diagnostic test showing the predicted bug migration.

This discussion of the bug Always-Borrow-Left illustrates many of the assumptions of the theory. First, procedures are the result of generalization of examples, rather than, say memorization of verbal or written recipes. The main evidence for this assumption is that there are accidental, visual characteristics of the examples, namely, the placement of the de-

A	B	C	D	E	F	G
7 12 ~~9~~ ~~2~~ − 4 3 3 9	4 10 ~~5~~ ~~9~~ − 2 3 2 7	1 0 9 − 7 0 3 9	~~5~~ 5 6 4 − 8 8 7 1 8 7	12 1 0 ~~2~~ − 3 9 7 3	1 17 ~~2~~ ~~7~~ − 8 1 9	9 0 0 − 6 8 8 2 2 2

H	I	J	K	L	M
7 1 6 − 5 9 8 1 1 8	3 1 1 − 2 1 4 9 7	8 8 5 − 2 0 5 6 8 0	4 ~~5~~ 15 11 ~~6~~ ~~5~~ 9 ~~7~~ − 2 6 9 7 2 9 0 4	8 3 5 5 − 3 8 3 5 2	6 10 11 ~~8~~ 0 ~~8~~ ~~7~~ − 4 3 6 0 6 8

N	O	P	Q	R	S
2 ~~3~~ 10 15 ~~4~~ ~~8~~ ~~7~~ ~~5~~ − 6 0 7 2 4 1 8	6 3 7 − 3 5 6 0 2	4 10 ~~5~~ 0 ~~8~~ − 4 4 0 6	6 12 ~~7~~ 0 ~~2~~ − 1 0 3 6 0 9	12 ~~4~~ 0 0 1 ~~2~~ − 2 1 4 2 0 8	6 12 ~~7~~ 4 ~~2~~ − 1 3 6 6 1 6

FIGURE 2.2. Verbatim presentation of a test by subject 8 of class 17 showing three repairs to the same impasse. On problems D, E, and G, one repair generates the bug Borrow-No-Decrement-Except-Last. (N.B., The subject does not always use scratch marks to indicate borrowing.) On problems H and I, another repair generates the correct borrow-from placement. On problems K, M, N, P, Q, R, and S, a third repair generates the bug Always-Borrow-Left. There are slips on problems D, P, Q, and S. There is a second impasse in processing the hundreds column of problem R.

crement, that a nonexample source of instruction, such as a verbal recipe, would not mention. The appearance of these visual characteristics in the acquired procedure is evidence that they were learned by induction (see VanLehn, (1986) for a full defense of this idealization).

A second assumption is that learning occurs in the context of a lesson sequence and that many bugs are caused by testing students who are in the middle of the lesson sequence on exercise types that they have not yet been taught how to solve. Perhaps such bugs should be welcomed as signs of a healthy learning process that may eventuate in a correct understanding of the procedure. Such a view of bugs is radically different from the traditional view, which considers bugs to be "bad habits" that need to be remediated. On the other hand the bad-habit view may be appropriate for older students, some of whom have bugs long after the lesson sequence has been completed (VanLehn, 1982).

Another set of assumptions involves the notions of interpretation, impasses, and repairs. A particularly important hypothesis is that repairs occur at the metalevel and change only the state of the interpretation. This hypothesis predicts the existence of bug migration. In fact, this prediction was made before any evidence of bug migration had been found (Brown & VanLehn, 1980). The surprising success of this forecast and the fact that it is an almost unavoidable consequence of the hypothesis provide strong support for the theory.

The Stable Bug Problem

Although some students' behaviors can be characterized as bug migrations, other students appear to have the same bug throughout a test. When such students are tested again 2 days later, they often have the same bug (VanLehn, 1982). Some students even show the same bug when tested twice 6 months apart (VanLehn, 1982). Such data encourage the interpretation that some students have learned their bugs. That is, their bugs have become a part of the knowledge structure they use to encode their procedure. Such relatively permanent bugs are called "stable" in order to differentiate them from bugs that may exist only for a short time, then migrate/change into other bugs[1].

Stable bugs present a problem for Repair Theory. Repairs do not modify the core procedure, but instead modify the state of the interpreter that is executing the core procedure. After a repair has been accomplished and the interpreter is running again, there is no trace of the effects of repair on the core procedure. Bug migrations are explained by assuming that the students apply different repairs at different occurrences of the impasse. In order to explain a stable bug, one must assume that the student chooses to apply the same repair every single time the impasse occurs. Intuitively, this seems quite unlikely.

One way to explain stable bugs within the RT1 framework is to assume that the set of possible repairs is different for different individuals. Some students may only know about one repair, so they always choose that repair at an impasse. They will appear to have a stable bug. However, this hypothesis has difficulties. There are stable bugs that can only be generated by assuming that the students have two different impasses and that the student repairs the first one with one repair and the second one with a different repair. Students with such bugs must know at least two repairs, yet they consistently choose the same one at each choice point. Assuming that different students have different repairs does not help explain such multi-impasse stable bugs.

[1] The proportion of students whose errors are due to stable bugs varies significantly with the grade level. In one study, 49% of the third graders had stable subtraction bugs versus 27% of the fourth graders and 13% of the fifth graders (VanLehn, 1982). The variation is due to the fact that more older children know the correct algorithm: 19% of the third graders were bug free versus 39% of the fourth graders and 60% of the fifth graders.

The Patch Hypothesis

As another potential explanation of stable bugs, one could augment RT1 by assuming that there is some memory of previous episodes of impasses and repairs. Stable bugs are generated by assuming that the student recognizes the impasse as one that has occurred before and recalls the repair that was selected before and employed successfully. To perform such recall, the student must have some memory of the impasse and the repair. That is, the student's knowledge of the skill must consist of a set of impasse-repair pairs in addition to the core procedure. Such pairs are called patches (Brown & VanLehn, 1980). Thus, if the students have a stable bug, then they have a patch for that impasse. If they do not have a patch, then the impasse will cause bug migrations.

There are problems with the hypothesis that the student's knowledge consists of patches as well as the core procedure. First, it seems inelegant and unparsimonious. Patches are essentially, condition–action rules. The condition is a description of particular interpreter states (i.e., a certain kind of impasse). A patch's action is some modification to be made to the interpreter's state. The core procedure is also made up of condition—action rules. The only differences between patches and the core procedure's rules are that the rules' conditions can test the external problem states (i.e., the state of a partially completed subtraction problem) and the rules' actions can modify the external problem state. That is, the patches operate exclusively on the interpreter's state, while core procedure's rules operate on the external problem state as well. Nonetheless, there are more similarities than differences between patches and rules. It would be parsimonious to combine them.

The second problem with patches is that they must be somewhat abstract in order to function properly. In order for the patch to apply to multiple occurrences of an impasse, it must be a *description* of the interpreter's state. Thus, if a patch is acquired from, say, the first occurrence of an impasse, then the condition half of the patch must be abstract. It must not mention details of the interpreter state that are idiosyncratic to this particular occurrence, such as the values of digits in the problem. Similarly, the repairs must also be abstract descriptions of the modifications that were performed to the interpreter's state. Consequently, acquiring a patch is not simply a matter of storing a state and a state change. Rather, patch acquisition requires nontrivial abstraction.

A third, more technical problem with patches is that they do not interface well with the pattern matching component of the interpreter. In order to represent descriptions of the external problem state, the procedure employs patterns. Such patterns are just like the usual ones found in, for instance the conditions of production rules. They consist of sets of relations whose arguments are variables or constants. To employ such patterns, the interpreter must have a pattern matcher. The matcher tries to fit the pattern to the representation of the external problem state. If the pattern matches, then the objects matched by the variables are often "read" and become a part of the interpreter's state. We saw an instance of this earlier, in the discussion of the bug Always-Borrow-Left. A pattern is used to represent the idea that the place to borrow from is (1) the leftmost column in the problem, (2) a column that is adjacent to the column that is the current focus of attention, and (3) a column that is left of the current focus of attention. Speaking very approximately, the pattern for this concept employs three relations, one for each of the constraints listed previously. It has two variables; one for the current focus of attention and one for the column to be borrowed from. The following is an informal presentation of the pattern:

(Is-leftmost-column New-focus) and
(Is-adjacent-to New-focus Current-focus) and
(Is-left-of New-focus Current-focus)

If the pattern matches, the object that is matched to the New-focus variable, namely a particular column in the problem, becomes the focus of attention for the borrow-from subgoal. The bug Always-Borrow-Left is generated when this pattern fails to match. Such mismatching occurs when borrowing originates in the units column of problems with more than two columns. In such problems, there is no column that meets all three constraints. The bug is generated when the second one is relaxed, allowing the pattern to match and picking out the leftmost column of the problem as the focus of attention for borrowing-from. This causes the student to borrow from the leftmost column, which is exactly what the bug Always-Borrow-Left does.

If the patch hypothesis is incorrect, then it should be possible to build a patch for Always-Borrow-Left. The impasse half of the patch can be quite simple. It can achieve the appropriate degree of abstraction by merely referring to the pattern. The description in the condition half of the patch would read, "The pattern that fetches the borrow-from column does not match." However, there are problems implementing the repair half of the patch. The following paragraphs present three possible implementations, all of which fail.

The repair could also be expressed in terms of the pattern. It needs to say something like "relax the second relation of the pattern." However, if this is taken literally, it means actually modifying the pattern by removing the second relation from it. Such modifications change the procedure itself. This makes it hard to explain bug migration—one would have to assume that the relaxation repair puts the deleted relations back, for instance.

A second possibility for the Always-Borrow-Left patch involves interrupting the pattern-matching process. In order to accomplish the requisite relaxation, the repair would have to interrupt the pattern matcher right when it was about to apply the second constraint of the pattern and some-

how cause the pattern matcher to skip over that relation. Expressing this repair as a patch is difficult. It would require a precise specification, at the theoretical level, of a pattern-matching algorithm, thus embroiling the theory in a layer of irrelevant detail.

A third option for the Always-Borrow-Left patch is to include a revised pattern that has all the relations except the second. The interpretation of this description is for the local problem solver to perform pattern matching using this pattern and substitute the results into the interpreter's state just as if the original pattern had been matched. This option works, usually. However, it has the flaw that on some occasions, the pattern stored in the patch does not match. This causes an impasse *inside* the local problem solver. That is, there can be an impasse while a person is trying to fix another impasse. The local problem solver is running "meta" to the interpreter, trying to repair the interpreter's impasse. We could assume that there is a meta-meta level, where another local problem solver runs, trying to repair the impasse that occurred inside the metalevel local problem solver. Such "towers" of metalevel interpreters have begun to appear in AI (Smith, 1982), but their properties are largely unexplored at this time. It is probably best to avoid postulating such multilevel architectures of students until they are better understood computationally.

In summary, there are three methods for representing the repair half of the patch: (1) modifying the core procedure by deleting a relation from the pattern, (2) having the repair cause the pattern matcher to ignore the relation, and (3) storing a substitute pattern in the patch and matching it from inside the local problem solver. Because all these methods have defects, it seems that patches cannot represent the stability of bugs that, like Always-Borrow-Left, depend on pattern relaxation in their repairs. This is just one problem with the patch hypothesis. The others, mentioned earlier, are its lack of parsimony, since patches are quite similar to rules, and the fact that nontrivial abstraction is required for patches to be acquired.

Representing Stable Bugs With Malrules

The new version of the theory, RT2, takes the position that there are no patches. The student's knowledge of the skill consists only of a procedure. In order to represent stable bugs, the core procedure has "malrules."[2] Malrules are identical in format and function to the core procedure's regular rules. The difference is only that they cause the student to answer incorrectly, rather than correctly. Furthermore, RT2 assumes that malrules

[2] Derek Sleeman coined the term "malrule" for his method of describing bugs in an objective, theoretically neutral fashion (Sleeman & Smith, 1981). Although the malrules of RT2 are interpreted as lying at a deeper, more psychologically plausible level, the use of the term seems just as descriptive of how the rules function.

are acquired by the same learning mechanism as regular rules. Regular rules are acquired by induction of the teacher's examples. Malrules are acquired by induction from the "malexamples" produced by local problem solving.

Malrules are a much more parsimonious solution to the stable bug problem than patches. They are identical to rules, and they are acquired by the same mechanism as rules. Thus, malrules escape the first objections raised against patches.

Another objection was that patches could not represent stable pattern-relaxation bugs. This objection is also resolved by the malrule hypothesis. To illustrate how, consider the bug Always-Borrow-Left again. In the normal course of events, students are first taught borrowing with two-column problems. Later they are taught how to solve three-column borrow problems. Recall that after the first lesson, the pattern is overconstrained:

(Is-leftmost-column New-focus) and
(Is-adjacent-to New-focus Current-focus) and
(Is-left-of New-focus Current-focus)

The second borrowing lesson shows that when borrowing originates in the units column of three-column problems, it is the tens column that one borrows from. The learning mechanism uses such examples to eliminate the first relation from the pattern. That is, the learning mechanism does pattern relaxation.

If, on the other hand, malexamples had been presented where the hundreds column was borrowed from, then pattern relaxation would delete the second relation. Such malexamples can be generated when the learner is tested between the first and second lessons on borrowing. The overconstrained pattern will cause impasses, and the repair of those impasses generates the malexamples. On this account, stable bugs like Always-Borrow-Left seem to be caused, ironically, by learning from one's mistakes.

Learning Occurs at Impasses

The introduction of this chapter promised a description of a fine-grained learning process. Although the responsibilities of the learning process have been increased, by including the generation of malrules as well as rules, the large-grained description of the learning process has not yet been refined. This section ventures a finer grained description.

If learning occurs as a result of local problem solving, then the learning process is likely to be interwoven with the local problem-solving process. The main hypothesis is that inductive learning occurs at impasses. The "at" is used here in two senses. Learning occurs when an impasse occurs.

If there is no impasse, there is no learning.[3] The second sense of learning "at" impasses is more subtle. When an impasse occurs, the student is "at" some place in the procedure. That is, the interpreter for the procedure is reading some part of the control structure of the procedure. The hypothesis is that the control location of the impasse is the place where the newly learned piece of procedure will be inserted. That is, if the control structure is visualized as laid out spatially, say as a tree, then the hypothesis that learning occurs "at" impasses takes on a spatial interpretation. The spatial location of the impasse is the place where the new subprocedure will be attached to the existing procedure. So, the hypothesis has two independent aspects: (1) learning occurs "at" impasses in the temporal sense and (2) learning occurs "at" impasses in the control structure sense.

First, let's examine the implications of the temporal aspect. According to the old theory, RT1, the only activity that occurs in response to impasses is repair. The goal of repair is merely to get the interpreter past the impasse in any way possible. In particular, repair is not concerned with answering the problem correctly. Consequently, repairs rarely modify the interpretation in such a way that the problems are solved correctly. However, the impasse-driven learning hypothesis is intended to explain the acquisition of regular rules as well as malrules. To do this, the theory must be amended to allow other activities in response to impasses. Once the history of this research is reviewed, it will be easy to see what those activities should be.

The bug data that initiated the theory were collected in testing situations. The students were asked to answer problems without help from their teacher, friends, or textbooks. If they got stuck, they would have to rely on their own knowledge to get unstuck. Thus, they repaired. However, students are not always in test-taking situations when they solve problems. Often, they solve practice exercises in class or at home. In such situations help is permitted. Indeed, students are encouraged to ask for help if they get stuck. So, the second kind of activity that may occur at impasses is receiving help.

Help seems to be the source of information that allows correct rules to be learned at impasses. For instance, suppose that a student gets stuck while doing seat work. He raises his hand. His teacher comes over. He asks, "I got to here and got stuck. What am I supposed to do next?" The teacher shows him what to do, saying, "You do this, and then this, then this." This short sequence of actions is just what the student needs. Not only does it get him around the impasse, but it is an example of a new subskill. The student may abstract the actions, leaving behind details that are specific to the particular problem that is being solved, such as the numerical values of the digits. The abstracted actions become a new subprocedure, which the student can attach to his existing procedure. Thus, correct

[3] Knowledge compilation may occur without impasses, but that is not the kind of learning that the theory describes.

rules can be acquired by the same mechanism that acquires malrules, provided that there is some way of obtaining help at impasses.

The actual method of delivering help is probably of secondary importance. The student can obtain help by comparing his solution to his friend's solution. The comparison isolates a subsequence of actions, illustrating the new subprocedure just as effectively as asking the teacher, but it may cost the student some effort to make the comparison. With slightly more effort, the student could generate a subsequence by drawing a "near" analogy to a worked problem in the textbook (Anderson, Farrell, & Saurers, 1984). The student may even be able to generate the subsequence from a "far" analogy, given a little coaching from the teacher (Resnick & Omanson, 1987). For instance, some teachers might have the student think of the problem's base-ten numerals as piles of pennies, dimes, and dollars. Under the teacher's prodding, the student maps the impasse over to the monetary representation, solves the impasse there without violating principles of fair exchange (i.e., always change a dime for 10 pennies), then maps those monetary manipulations back into paper-and-pencil actions in the written representation. Because the impasse was solved correctly in the monetary representation, the written analog of that solution should be a correct subsequence of actions. The point is that there are a variety of ways to obtain a written action subsequence. Demonstration, comparison, near analogy, and far analogy are only a few of the many possible ways, although they may be the most common.

As soon as one discovers that there are several kinds of inputs to a cognitive process, one wonders whether those differences make any difference. Does it matter whether the student receives help via individualized demonstrations, versus comparison, etc? The simplest hypothesis is that it is the subsequence of actions that determines the contents of the subprocedure and not the source for that subsequence of actions. If two methods of obtaining help yield the same subsequence of actions for the impasse, then the subprocedure that the student induces should be the same.

Like most simple hypotheses, this one is likely to be only half right, at best. The various methods of obtaining help may require different cognitive resources, and that may affect the inductive learning process. For instance, suppose far analogy takes longer and requires more problem solving of the student than attending to the teacher's demonstration. The heavier demands of far analogy could interfere with the retention of the interpreter's state at the time of the impasse. This may decrease or perturb inductive learning, because retaining (or reconstructing) the interpreter's state is necessary for determining where to attach the new subprocedure. This interference could be considered, however, a second-order or "performance" factor.

The second aspect of the hypothesis that learning occurs "at" impasses concerns the place where the new subprocedure will be attached to the existing procedure. The hypothesis is that the attachment point will be the

same as the location of the interpreter at the occurrence of the impasse. The notion of location is complicated by the fact that the student's procedures have a hierarchical control structure. That is, the procedure has goals, which call subgoals, which call sub-subgoals, and so on. There may even be recursion: a goal calling itself. Consequently, at the time of an impasse, there may be a whole stack of goals pending. However, it is the lowest goal, the one that is a subgoal of all the others, that is suffering from the impasse.[4] This has implications for where new subprocedures will be attached, given the hypothesis that subprocedures are attached at the place in the goal structure where the impasse occurred. Roughly speaking, new subprocedures will tend to be attached low in the goal tree. This prediction is a necessary implication of the hypothesis.

It is also a true prediction for the data available now. In order to predict the locations of subprocedure attachments, as inferred from the arithmetic bug data, RT1 had an ad hoc hypothesis, called the lowest parent hypothesis (VanLehn, 1983a). It simply stipulated that new subprocedures be attached as low as possible in the goal hierarchy. This hypothesis is no longer needed. The attachment points are predicted from an independently motivated hypothesis, namely, that learning occurs at impasses.

Implications for Remediation

One of the most obvious facts about arithmetic is that remediation of bugs tends to work. Many students have arithmetic bugs when tested in the early grades (e.g., 49% in the third grade). The proportion decreases with grade level. The proportion of adults with bugs is much smaller. Apparently, students' bugs are being remediated somewhere in the educational process. The question addressed in this section is how this remediation takes place. The section entitled Representing Stable Bugs With Malrules discussed how malrules are learned. The section entitled Learning Occurs at Impasses discussed how regular rules are learned. This section speculates about how regular, correct rules may be learned when they have to compete with malrules that were learned earlier.

Suppose a student enters a remedial session with a stable bug. The stability of the bug indicates that a malrule was learned during some prior

[4] Here is an informal proof that only the lowest goal can be stuck. Suppose that some goal other than the lowest goal is stuck. This means that there is some subgoal of it that is pending but not stuck. But if that lower goal is not stuck, then it can continue until it succeeds or fails. In either case it would be removed from the stack of pending goals. So the only goals left in the stack when interpretation is forced to stop are (1) the stuck goal and (2) all the goals that depend on its completion in order for them to complete, that is, the supergoals of the stuck goal. In short it is always the lowest goal on the stack that is the current locus of control when an impasse occurs.

episodes of local problem solving. Those impasses no longer occur because the malrule circumvents them. Consequently, the student can work all the relevant problems without reaching an impasse. As mentioned earlier, if there are no impasses, there is no learning. Yet, remediation works. How can the hypothesis be reconciled with the facts?

The simplest reconciliation is to assume that remediation occurs when the teacher stops the student just as the student makes a wrong move in the problem-solving process. For instance the teacher may be carefully watching the student do a borrow and may interrupt just as the student has placed a scratch mark where no scratch mark should occur. Suppose further that the student interprets the teacher's interruption in a similar fashion to an impasse. The student observes the subsequence of actions that the teacher suggests, abstracts them, and plugs them into the existing procedure as a new subprocedure. Treating interruptions in tutorial situations as impasses could extend the theory to cover remediation. Such interruptions are rare events, which explains why stable bugs are often found. However, if we postulate that such remediation is effective when it does occur, and that tutorial efforts persist for years, then the assumption also explains why stable bugs eventually disappear.

However, there is a small problem. Typically, the teacher's interruption occurs just after the first incorrect action instead of before it. Yet that action is the result of a malrule that is running in place of the impasse. We want the new subprocedure to be placed where the impasse used to occur, just as if the instruction had been delivered then instead of now. This would cause the new subprocedure to be attached correctly. However, the teacher's interruption is too late. The impasse place has been passed. In principle, the student could be asked to reason backward in order to locate the impasse place. However, a better remediation technique may be to have the student solve the same problem again (or a very similar one) and interrupt just before the incorrect action. This interruption would be at the impasse place, or at least much closer. This is, of course, a testable suggestion concerning the effectiveness of two remediation strategies.

There is a more significant problem with the kind of remediation proposed so far. Suppose the interruption is completed, and the student has installed the new subprocedure at the impasse place. There is already another subprocedure attached there, the malrule. It was generated in response to that impasse. So both the rule (subprocedure) and malrule are attached at the same impasse place. Subsequently, when the student comes to that place in solving a problem, how will the student know whether to execute the rule or the malrule? Since both were constructed in order to handle the same impasse, both will be applicable. All other things being equal, the student will pick the rule half the time and the malrule the rest of the time. This predicts a new kind of instability, where buggy problem solving alternates with correct problem solving.

There is anecdotal evidence that this prediction might be close to the

mark. The evidence concerns the spontaneous reappearance of bugs after their supposed remediation. Several teachers have told me that when students with bugs are shown the correct procedure in a remedial session, they pick it up easily. They can solve dozens of problems successfully in the session. Apparently, they have learned the correct procedure. However, when tested again several weeks later, these students are either back to using their old buggy procedure or alternating between their old buggy procedure and the correct procedure. Resnick and Omanson (1987) have carefully documented several cases of such bug regression in a study designed to investigate new remediation strategies. Bug regression occurred despite the fact that the remediation was particularly thorough. Nonetheless, Resnick and Omanson report that 60% of the students reverted to using their buggy procedures for answering written subtraction problems when tested about 4 weeks later.

Bug regression makes intuitive sense, given the cognitive process sketched earlier. Suppose the student has enough context during the remediation session to differentiate the newly learned rule from the malrule, which was learned some time previously. The similarity between the learning context and the application context allows the student to reliably differentiate the correct rule from the older rule and thereby apply the new rule throughout the remediation session. However, at a later testing session, the context during the session may not be similar enough to the remediation session that the student can recall which rule is the one to use. This would cause the student to be uncertain about which rule was correct. The student might alternate rules in order to maximize the test score. Another possibility is that the malrule was learned during a testing session. The present context, another testing session, may be more similar to the context in which the malrule was learned than the context in which the correct rule was learned. This may cause the student to use the malrule exclusively. Thus, depending on when the malrule was learned, the students may either apply the buggy procedure exclusively, or they may alternate between the correct procedure and their buggy one. The predictions of impasse-driven learning are in accord with the phenomenon of bug regression.

General Discussion

Many cognitive theories of learning have hypothesized that learning is some kind of automatic phenomenon. Mental activity leaves a trace that somehow makes it easier to perform that activity the next time. Automatic learning has been the dominant paradigm for the last few decades of psychology, if not longer. Particular examples of this kind of learning for skill acquisition are Anderson's (1983) ACT* theory, Laird, Rosenbloom and Newell's (1986–1987) SOAR theory, and Anzai and Simon's (1979) theory

of learning by doing. These theories feature automatic learning of new material (i.e., task-specific productions) by repeated usage of older material (i.e., weaker, more general productions).

Automatic learning theories have begun to draw fire from computer scientists who have noted that the lack of control over what is learned causes the system to acquire vast quantities of useless knowledge (Minton, 1985). For instance, Roger Schank (1982) has rejected automatic learning as a totally impossible way to acquire common sense knowledge about, for example, how to dine in a restaurant. He points out that most mundane thinking is so banal and disconnected that to remember it all would be pointless and a poor model of our introspective experience of learning. To put it in a phrase, automatic learning would generate mental clutter.

Impasse-directed learning does not generate mental clutter. Learning only occurs when the current knowledge base is insufficient. Moreover, it is not just any incompleteness that causes learning. The incompleteness must be relevant enough to the person's affairs that it actually generates an impasse. The person's problem solving must require a piece of knowledge that is not there. Consequently, one learns only when there is a need to learn. Mental clutter is avoided, and only pertinent knowledge is acquired.

Related Models of Skill Acquisition

Impasse-directed learning is a species of failure-driven learning. Failure-driven learning is a common theoretical idea in the learning literature. For instance, in Wexler and Culicover's (1980) theory of language acquisition, whenever the learning model cannot understand a sentence, it randomly deletes a rule from its grammar, or it makes a change in an existing rule by randomly choosing from a small class of legal perturbations. Their theory is typical of a class of learning theories where negative reinforcement of an internal kind causes a more-or-less random change in the learner's knowledge. Impasse-driven learning is more specific than these theories in that it postulates exactly what kinds of negative reinforcement cause learning (i.e., impasses) and exactly what kinds of changes the learner makes to its knowledge. The impasse-driven learning hypothesis is a new member of the class of failure-driven learning theories.

The idea of impasse-driven learning is central to the SOAR architecture (Laird, Rosenbloom, & Newell, 1986, 1987). SOAR is a production system. When SOAR reaches an impasse, it does some problem solving at the metalevel. As it returns from that problem solving, it automatically builds a new production rule, whose conditions are exactly the conditions pertaining at the impasse and whose actions are the results of the metalevel problem solving that was just completed. If ever those conditions occur again, the production will fire, thus saving SOAR the effort of reaching an impasse and resolving it at the metalevel. SOAR's authors call this kind of learning "chunking," and the productions built this way are called chunks.

SOAR's authors claim that chunking is the only kind of learning that people do. However, this claim is not very restrictive, because the SOAR architecture allows arbitrary metalevel problem solving at impasses. The chunking mechanism saves the results, but the programmer can generate those results any way she wants by writing the appropriate problem solving into SOAR's metalevel. RT2 will be more specific than that. The theory will describe in detail the metalevel problem solving qua learning that occurs at impasses.

The impasse-driven learning hypothesis has appeared in the literature on formal theories of natural language acquisition. Robert Berwick (1985) developed a theory of how English syntax is learned. His theory is strikingly similar to RT2, despite the fact that the two theories were developed independently. Berwick assumes that a person has a grammar and a parser. The grammar and parser are analogous, respectively, to the procedure and interpreter postulated by RT2. As internal state, Berwick's parser employs a stack and some other temporary structures. These have analogs in RT2 as well. In Berwick's theory the parser can get stuck because no grammar rules apply (the analog of reaching an impasse in RT2). One of four actions is taken. All four actions modify only the parser's internal state just as repairs would. Two of Berwick's four "repairs" have nearly exact analogs to the repairs found in RT2. So the architecture postulated by Berwick for understanding English is nearly isomorphic to the one described here for following procedures.

Berwick (1985) goes on to state his version of the impasse-driven learning hypothesis. Grammar rules are induced when the parser gets stuck. Which rules are induced depends on external information, namely, a perceptually given understanding of the sentence. To put it intuitively, if the child cannot understand a sentence, she figures out what it meant from context, then invents a rule that would both get her parser unstuck and be consistent with the sentence's meaning. This process is analogous to that postulated here, except that the typical learner may appeal to a blackboard, a Dienes Blocks algorithm (given an implementation of Resnick and Omanson's (1987) suggestion) or some other source of information about the skill, rather than inferring its meaning from context.

Summary

Impasse-driven learning has been put forward as a conjecture about how students learn procedural skills. It employs the metalevel architecture proposed by Repair Theory. It postulates additional processes that run at the metalevel. When an impasse occurs, the student can either repair or seek help; both processes run at the metalevel and fix the problem of being at the impasse. When the impasse is fixed, the student can choose either to abstract the actions taken to resolve it, or not. In general, inducing a new subprocedure from the actions taken at the impasse will result in either a correct subprocedure, if help was sought, or a buggy subprocedure, if repair was used.

Impasse-driven learning seems to make the right sort of predictions about bugs and their stability. It predicts that bugs can migrate as well as be stable over long periods. It predicts that remediation of bugs will appear effective at the end of the remediation session, but that bugs will tend to reappear over time.

Impasse-driven learning also seems to correctly predict the shape of cognitive structures that are built by learning. It predicts that new subprocedures will be attached as deeply as possible in the goal-subgoal hierarchy of the student's procedures.

Impasse-driven learning is a form of failure-driven learning. Failure-driven learning has traditionally been advanced as more cognitively economical than automatic learning, its traditional opponent hypothesis, in that it predicts that new knowledge is acquired only when there is a need for that knowledge. Automatic learning tends to generate mental clutter—cognitive structures of little or no relevance to subsequent thinking.

Impasse-driven learning seems to have great potential generality. It has been investigated in a powerful general learning system, SOAR (Laird et al., 1986, 1987). It has been shown capable of learning English grammar (Berwick, 1985). The future of this hypothesis seems quite bright indeed.

Acknowledgement. This research was supported by the Personnel and Training Research Programs, Psychological Sciences Division. Office of Naval Research, under Contract No. N00014-86K-0349, Project Number 442a558. Reproduction in whole or in part is permitted for any purpose of the United States Government. Approved for public release; distribution unlimited.

References

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

Anderson, J. R. (1985). *Cognitive psychology and its implications*. New York: Freedman.

Anderson, J. R., Farrell, R., & Saurers, R. (1984). Learning to program in LISP. *Cognitive Science, 8*, 87–129.

Anzai, Y., & Simon, H. A. (1979). The theory of learning by doing. *Psychological Review, 86*, 124–140.

Ashlock, R. B. (1976). *Error patterns in computation*. Columbus, OH: Bell & Howell.

Berwick, R. (1985). *The acquisition of syntactic knowledge*. Cambridge, MA: MIT Press.

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

Brown, J. S., & VanLehn, K, (1980). Repair Theory: A generative theory of bugs

in procedural skills. *Cognitive Science, 4*, 379–426.
Brownell, W. A. (1935). Psychological considerations in the learning and teaching of arithmetic. In W. D. Reeve (Ed.), *The teaching of arithmetic*. New York: Teachers College, Bureau of Publications.
Brueckner, L. J. (1930). *Diagnostic and remedial teaching in arithmetic*. Philadelphia: Winston.
Burton, R. B. (1982). Diagnosing bugs in a simple procedural skill. In D. H. Sleeman, and J. S. Brown (Eds.) *Intelligent Tutoring Systems*. New York: Academic Press.
Buswell, G. T. (1926). *Diagnostic studies in arithmetic*. Chicago: University of Chicago Press.
Cox, L. S. (1975). Diagnosing and remediating systematic errors in addition and subtraction computation. *The Arithmetic Teacher, 22*, 151–157.
Laird, J. E., Rosenbloom, P. S., & Newell, A. (1986). Chunking in SOAR: The anatomy of a general learning mechanism. *Machine Learning, 1*, 11–46.
Laird, J. E., Rosenbloom, P. S., & Newell, A. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence, 33*, 1–64.
Lankford, F. G. (1972). *Some computational strategies of seventh grade pupils*. Charlottesville, VA: University of Virginia.
Minton, S. (1985). Selectively generalizing plans for problem-solving. In *Proceedings of IJCAI 85* (pp. 596–599). Los Altos, CA: Morgan-Kaufman.
Newell, A. (1980). Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. Nickerson (Ed.), *Attention and Performance VIII*. Hillsdale, NJ: Erlbaum.
Newell, A., & Simon, H. A. (1972). *Human problem-solving*. Englewood Cliffs, NJ: Prentice-Hall.
Norman, D. A. (1981). Categorization of action slips. *Psychological Review, 88*, 1–15.
Resnick, L. (1982). Syntax and semantics in learning to subtract. In T. Carpenter, J. Moser & T. Romberg (Ed.). *Addition and subtraction: A cognitive perspective*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
Resnick, L. B., & Omanson, S. F. (1987). Learning to understand arithmetic. In R. Glaser (Ed.), *Advances in instructional psychology*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
Roberts, G. H. (1968). The failure strategies of third grade arithmetic pupils. *The Arithmetic Teacher, 15*, 442–446.
Schank, R. (1982). *Dynamic memory: A theory of learning in computers and people*. Cambridge, England: Cambridge University Press.
Shaw, D. J., Standiford, S. N., Klein, M. F., & Tatsuoka, K. K. (1982). *Error analysis of fraction arithmetic-selected case studies* (Tech. Report 82-2-NIE). Urbana, IL: University of Illinois, Computer-based Education Research Laboratory.
Siegler, R. S., & Shrager, J. (1984). Strategy choices in addition: How do children know what to do? In C. Sophian (Ed.). *Origins of Cognitive Skill*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
Sleeman, D. (1984). An attempt to understand students' understanding of basic algebra. *Cognitive Science, 8*, 387–412.
Sleeman, D. H. (1985). Basic algebra revisited: A study with 14-year olds. *International Journal of Man-Machine Studies, 22*, 127–149.

Sleeman, D. H., & Smith, M. J. (1981). Modeling student's problem solving. *Artificial Intelligence, 16*, 171–187.
Smith, B. C. (1982). *Reflection and semantics in a procedural language* (Technical Report MIT-TR-272). Cambridge, MA: M.I.T. Laboratory for Computer Science.
Tatsuoka, K. K., & Baillie, R. (1982). *Rule space, the product space of two score components in signed-number subtraction: An approach to dealing with inconsistent use of erroneous rules* (Tech. Report 82-3-ONR). Urbana, IL: University of Illinois, Computer-based Education Research Laboratory.
VanLehn, K. (1982). Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. *The Journal of Mathematical Behavior, 3*, 3–71.
VanLehn, K. (1983a). *Felicity conditions for human skill acquisition: Validating an AI-based theory* (Tech. Report CIS-21). Palo Alto, CA: Xerox Palo Alto Research Center.
VanLehn, K. (1983b). Human skill acquisition: Theory, model and psychological validation. In *Proceedings of AAAI-83* (pp. 420–423). Los Altos, CA: Kaufman.
VanLehn, K. (1986). Arithmetic procedures are induced from examples. In J. Hiebert (Ed.), *Conceptual and procedural knowledge: The case of mathematics*. Hillsdale. NJ: Lawrence Erlbaum Assoc.
VanLehn, K. (in press). *Cognitive procedures: The acquisition and mental representation of basic mathematical skills*. Cambridge, MA: MIT Press.
VanLehn, K., Brown, J. S., & Greeno, J. G. (1984). Competitive argumentation in computational theories of cognition. In W. Kintsch, J. Miller, & P. Polson (Ed.), *Methods and tactics in cognitive science*. Hillsdale, NJ: Lawrence Erlbaum, Assoc.
Wexler, K., & Culicover, P. (1980). *Formal principles of language acquisition*. Cambridge, MA: MIT Press.