In H.P. Ginsberg (ed) The development of mathematical thinking. New York: Academic. 1983.

CHAPTER 5

KURT VAN LEHN

On the Representation of Procedures in Repair Theory

Over the months or years that it takes students to master a procedure such as ordinary place-value subtraction, their performance is characterized by many systematic errors or *bugs* that indicate a flaw or incomplete understanding of the procedure. Not only are there a very large variety of bugs across the population, but the bugs a student exhibits often shift radically over short periods of time. Nonetheless, there are developmental trends indicating how formal instruction influences the student's (mis-)conceptions of the skill. Repair Theory (Brown & VanLehn, 1980) aims to account for all these phenomena—the large variety of bugs, their short-term instability, and their long-term relationship to instruction.

The theory draws its name from the belief that when a student has unsuccessfully applied a procedure to a given problem he or she will attempt a *repair*. Suppose that the student is missing a fragment of a procedural skill, either because the fragment was never learned or maybe it was forgotten. Attempting to rigorously follow the impoverished procedure will often lead to an *impasse*. That is a situation in which some current step of the procedure dictates a primitive action that the student believes cannot be carried out. In ordinary subtraction, an impasse would follow from an attempt to decrement a zero, provided the student knows (or discovers) that the decrement primitive has a precondition that is input argument can't be zero. When a constraint or precondition gets violated, the student, unlike a typical computer program, is not just apt to quit. Instead the student will often be inventive, invoking problem-solving skills in an attempt to

ISBN 0-12-284780-6

repair the impasse so that he or she can continue to execute the procedure, albeit in a potentially erroneous way. Many bugs can be explained as "patches" derived from repairing a procedure that has encountered an impasse while solving a particular problem.

The repair concept has been incorporated into a formal computational theory. It postulates that each student has a *core procedure* that represents the student's current knowledge of the skill. The core procedure is what the student applies to solve test problems, do exercise problems, and interpret further instruction. Because applying the core procedure to solve a problem often involves reaching an impasse and repairing, it cannot be observed directly but only inferred. Although there are many bugs in the population, there are apparently few core procedures. The large variety of "surface procedures" is accounted for by "multiplying" a small set of core procedures by a small set of repairs. Not only are core procedures less variegated, but they are more stable than bugs. Much short-term shifting among bugs is due to applying various repairs to the underlying impasses of a stable core procedure.

A database of hundreds of subtraction bugs has been collected by testing thousands of students in all stages of subtraction instruction (Brown & Burton, 1978; VanLehn, 1981). In addition to providing evidence for the repair process, these data allowed inference of a set of several dozen core procedures, including details about the form or structure of each procedure. This enables much deeper questions to be investigated: What are the causes of core procedures? How are they related to instruction? Why were they acquired and not others?

The layers of Repair Theory can be graphically stated, using "→" to mean *explains*.

$$\text{instruction} \rightarrow \text{core procedures} \rightarrow \text{bugs} \rightarrow \text{errors}$$

Connecting the layers, the theory has three parts: (*a*) an acquisitional study that links instruction with core procedures, (*b*) an applicational study that explains idealized systematic errors (bugs) from repair of core procedures, and (*c*) an empirical study that abstracts bugs from actual student performances by filtering out their unintentional, "careless" mistakes (e.g., errors in recalling the basic subtraction facts). All three parts of the theory are under active development, so this chapter will not dwell on the details of any one beyond that which is necessary as background to its main purpose, which is to present part of the infrastructure of the theory.

Like many recent theories of complex human behaviors, such as problem solving or skill acquisition, that have been expressed computationally, Repair Theory uses a *knowledge representation language* to express knowledge held by the subject. This chapter shows that certain aspects of the knowledge representation language are crucial to the theory's success. These arguments are offered as

one approach to understanding *mental representations* and their interface with cognitive development.

Knowledge representation languages have been the subject of great debate in the Artificial Intelligence community. The debate centers on how easily the language allows one to encode knowledge. This emphasis on representational ease is well placed. Several decades of experience in trying to construct computer programs that behave intelligently have convinced everyone that a good representation allows intelligent behavior to be captured quite simply, while a bad one makes this task complex or even impossible. By extension, the representational language used by a computational theory of cognition, such as repair theory, must have a profound impact on the simplicity, if not the overall conception, of the theory. It is an important issue to which psychologists should attend.

But psychologists have, for the most part, stayed out of the representational fray. The representation languages they choose are usually varieties of production systems or semantic nets. These are favored, I think, because they employ constructions, such as short-term memory and associations, that have enjoyed the attention of a great deal of psychological research. But just because constructions bearing names like "short-term memory" are a part of language does not mean that a theory employing the language depends in any strict way on the nature of those constructions. The theory could be a smashing success and yet its validation imputes no credit to the construction if it happens to be the case that the construction is not used *crucially* in the theory—that is, when other constructions could have been used in place of one bearing the fancy name, and the theory would succeed just as well. Worse yet is the case where alternatives to the advocated construction actually improve the theory. These are the kinds of trouble that a theory can get into if the relationships between representation language and the theory's predictions are not well understood.

Despite its importance, it is difficult to support a claim that a certain representation language is theoretically crucial. There are two reasons for this. One is that knowledge representations are quite indirectly related to observable behavior. Whereas one can make a fairly direct mapping between, say, a process model's actions and the subject's actions, such a direct mapping is unreliable in the case of knowledge. What subjects say or do is more plausibly the product of some interpretation or use of their knowledge. Consequently, the only way one can "see" the format of that knowledge is by mapping the observations backward through the interpretation/use processes. To support a claim about the knowledge representation language, one must use very complex arguments that bring together many backward-mapped observations. Moreover, in a theory as complex as Repair Theory, it is not sufficient to simply state the principles, show that the theory conforms to them, and assess the empirical adequacy. This would be treating the theory as a black box, making it impossible for anyone but its

creators to change it or even fully understand it. To make a complex theory a useful and viable contribution, one must reveal the system of inferences that motivate its constructions. One should show which principles depend upon which components and which empirical observations. This is particularly important in the case of the representation language since there is a traditional tendency in psychology, even in information-processing psychology, to misunderstand (if not totally ignore) the effects of knowledge representations on theories.

The second reason for the difficulty in sustaining claims about knowledge representation languages lies in the logic required by such claims. To show that some feature of the language is crucial is to show that it is *necessary* in order for the theory to meet some criteria of adequacy. To show that it is *sufficient* is not enough. Indeed, any successful theory that uses a knowledge representation language is a sufficiency argument for that language. But when there are two theories, one claiming that x is sufficient and another claiming that y is sufficient, sufficiency itself is no longer persuasive. One must somehow show that x is better than y. Indeed, this sort of *competitive argumentation* is the only realistic alternative to necessity arguments. Such arguments form a sort of successive approximation to necessity. But to form competitive arguments requires knowing the alternatives. The more exhaustive the set of alternatives, the more closely the argument approximates a necessity argument. Although the use of formal knowledge representation languages in the cognitive sciences has been popular recently, the space of representational alternatives is not yet well understood. Instead, there is only a vague fog of jargon in which specific knowledge representation languages are embedded. There is as yet no understanding of what the important issues are that differentiate languages from each other and what range of alternatives exist for each issue. In short, this chapter is about to venture into an unknown space in search not only of the winning alternatives, but of the issues themselves.

This journey requires extensive preparation. The preparation involves motivating the major components of the theory at a certain medium level of detail. The level of detail is set high enough that the form of the mental representations plays no role. This provides a rich structure of foundational assertions for the argumentation about mental representations to work with. The arguments themselves have the form "if Repair Theory is to conform to the principles motivated at the medium level of detail and yet produce accurate empirical predictions (i.e., have the right low-level detail), then the representation language *must* have such-and-such an attribute." The first two sections of this chapter describe and motivate the theory at the medium level of detail, and the next two sections present the arguments.

More specifically, the first section discusses the empirical and applicational parts of Repair Theory, that is, how bugs are related to student performance, and how repair of core procedures generates bugs. The theory's formal expression is

a *process model of application*—the interpretation and repair of core procedures. The second section presents the current thinking on how to account for the acquisition of core procedures, which differs from that presented in Brown and VanLehn (1980). It is part of a research paradigm that has emerged rather recently in developmental psychology. (For a review, see Keil, 1981.) Rather than proposing a process model for acquisition, it aims to discover *constraints on sequences of knowledge structures* that not only sharply limit the class of naturally learnable structures, but also the structural relationships that an intermediate state of knowledge can have with its predecessors and successors. The second section introduces this paradigm and applies it to a domain that has not felt its touch before: skill acquisition in formal instructional settings. The third and fourth sections present the target arguments of the chapter, concerning respectively the mental representations for the execution (or "run-time") state of core procedures and their long-term (or "schematic") form. The fifth section comments on the methodology and technology used to discover the arguments presented in this chapter.

THE REPAIR PROCESS

The initial task chosen for investigation is ordinary multidigit subtraction. Its main advantage, from a psychological point of view, is that it is a virtually meaningless procedure. Most elementary school students have only a dim conception of the underlying semantics of subtraction, which are rooted in the base ten representation of numbers. When compared to the procedures they use to operate vending machines or play games, subtraction is as dry, formal, and disconnected from everyday interests as the nonsense syllables used in early psychological investigations were different from real words. This isolation is the bane of teachers but a boon to the psychologist. It allows one to study a skill formally without bringing in a whole world's worth of associations.

In the last several years, a detailed study of thousands of student's subtraction performances has provided an extensive, precise catalog of subtraction misconceptions (Brown & Burton, 1978; Burton, 1981; VanLehn, 1981). This catalog is the major database used to develop and validate Repair Theory. It is worth a moment to discuss its nomenclature and the method of its collection before moving on to describe the repair process.

Bugs Are Precise Descriptions of Systematic Errors

It has long been known that many of the errors that students make while learning a procedural skill, such as ordinary place-value subtraction, are *system-*

atic in that the errors appear to stem from consistent application of a faulty method, algorithm, or rule (Ashlock, 1976; Brownell, 1941; Brueckner, 1930; Buswell, 1926; Cox, 1975; Lankford, 1972; Roberts, 1968). These errors occur along with the familiar unsystematic or "careless" errors that occasionally occur in expert performance as well as the learner's behavior. The common opinion is that careless errors or *slips*, as current research prefers to call them (e.g., Norman, 1981), are performance phenomena, an inherent part of the "noise" of the human information processor. Systematic errors on the other hand are taken as stemming from mistaken or missing knowledge about the skill, the product of incomplete or misguided learning. They are the results of *misconceptions*. By studying where conceptions of procedures break down, insight can be gained into the structure of knowledge about procedures.

The basic idea is that misconceptions could be formally represented and precisely described as *bugs* in a correct procedure for the skill. In brief, a bug is a slight modification or perturbation of a correct procedure. The bug-based notation is complete in the sense that it not only describes which problems the students gets wrong, but the content of each wrong answer and the steps followed by the student in producing it. The bug-based notation is the basis of the DEBUGGY diagnostic system (Burton, 1981) and its predecessor, BUGGY (Brown & Burton, 1978). To illustrate the notion of bug, consider the following subtraction problems, which display systematic errors:

1				1	4 5		
2	7	017	6	2	6		4
$3^{1}0^{1}6$	$8^{1}0$	$1\,8^{1}3$	$7^{1}0\,2$	$3\,0^{1}0^{1}5$	$7^{1}0^{1}0^{1}2$	3 4	$2\,5^{1}1$
−1 3 8	− 4	− 8 5	− 1 1	− 2 8	− 2 3 9	−1 4	− 4 7
7 8	7 6	8 8	6 9 1	1 0 8 7	4 8 7 3	2 4	2 4 4

(The small numbers stand for the student's scratch marks.) One could vaguely describe these problems as coming from a student having trouble with borrowing, especially in the presence of zeros. More precisely, the student misses all the problems that require borrowing from zero. One could say that he or she has not mastered the subskill of borrowing across zero. This description of the systematic error is fine at one level: It is a testable prediction about what new problems the student will get wrong. It predicts for example that the student will miss 305 − 117 and will get 315 − 117 correct. Systematic errors described at this level are the data upon which several psychological and pedagogical theories have been built (e.g., Durnin & Scandura, 1977). It has become common to use testing programs based on this notion for placement, advancement, and remediation in structured curricula, such as mathematics. Such testing programs are often labeled *domain referenced* or *criterion referenced*.

Once one looks beyond what *kinds* of exercises the student misses and looks at the actual steps taken in answering it, one finds in many cases that these step sequences can be precisely *predicted* by using a procedure that is a small perturbation in the fine structure of the correct procedure. Such perturbations—called bugs—serve as a precise description of the errors.

The "student" whose work we just considered has a bug called Borrow-Across-Zero. This bug modifies the correct subtraction procedure by deleting the step wherein the zero is changed to a nine during borrowing across zero. (This bug and others like it are described thoroughly in the appendices of VanLehn, 1981 and Brown and VanLehn, 1980.) This deletion creates a procedure for answering subtraction problems. As a hypothesis, it predicts not only which new problems the students will miss, but also what each answer will be and the sequence of steps that will be used in obtaining it. Since bug-based description of systematic errors predict behavior at a finer level of detail than missing-subskill/domain referenced testing, it has the potential to form a better basis for cognitive theories of learning and errors.

The bug description is an idealization of behavior in that it excludes slips. The tens column of the third problem shows a typical slip. The student has answered $17 - 8 = 8$. Slips are filtered from the raw data not because they are uninteresting, but because Repair Theory is probably not fine-grained enough to model them. One interesting trait of slips is "echoing." When a student makes a facts error slip, such as $17 - 8$, the digit written is often the same as one that has recently been the focus of attention, in this case an 8 (also, 8 was the answer in the preceding column). Such patterns as echoing fuel the ongoing development of theories of slips (Norman, 1981). Repair theory concerns itself more with what the student intended to do rather than what the student actually did. In a sense Repair Theory is a *competence* theory: It studies what people *can* do rather than what they *do* do. Greeno and Brown (1981) defend the importance of this kind of study. Burton (1981) and VanLehn (1981) discuss the difficulties of implementing this methodology objectively, and in particular, the algorithms used to filter out slips from the subtraction data.

It is often the case that a student has more than one bug at the same time. Indeed, the example given earlier illustrates co-occurrence of bugs. The last two problems are answered incorrectly, but the bug Borrow-Across-Zero does not predict their answers. (It predicts the two problems would be answered correctly.) A second bug called Diff-N $-$ N $=$ N is present. When the student comes to subtract a column where the top and bottom digits are equal, instead of writing zero in the answer, he or she writes the digit that appears in the column.

Using DEBUGGY, thousands of students have been analyzed. The chart that follows summarizes the widespread extent of the phenomena by showing that about 40% of the students making errors in the two largest samples, one domestic

and one foreign, were analyzed as having bugs. (These and other data following are taken from VanLehn, 1981; the 40% figure has been confirmed independently for British 10-year-olds by Young and O'Shea, 1981.):

Nicaraguan grades 5 & 6		American grades 3 & 4		Category
37		112		No errors
116	(9%)	223	(22%)	All errors due to slips
505	(39%)	417	(40%)	Most errors due to bugs
667	(52%)	386	(37%)	Cause of most errors unknown
1325		1138		Totals

It was common for a student to have more than one bug. Of the 417 American students that DEBUGGY analyzed as having bugs, 150 (36%) received a multibug diagnosis. Most of these diagnoses consisted of two or three bugs, but there were several cases of four bugs co-occurring. Overall, 77 distinct bugs occurred. (In this chapter *occurred* means that some student had the bug as his or her diagnosis, or if he or she was diagnosed as having a set of bugs, as part of this diagnosis.) The large variety of bugs and the complex patterns of relationships among them are a rich, precise database for developing and verifying a theory of how people understand and misunderstand procedures.

An Informal Introduction to Repair

When a student gets stuck while executing a flawed procedure (the product of mislearning or forgetting), he or she is unlikely to just quit as a computer does when it can't execute the next step in a procedure. Instead, the student will do a small amount of problem solving, just enough to get "unstuck" and complete the subtraction problem. These local problem-solving strategies are called *repairs* despite the fact that they rarely succeed in rectifying the broken procedure, although they do succeed in getting it "unstuck." Repairs are quite simple tactics, such as skipping the operation that can't be performed or backing up to the last branch point in the procedure and taking a different path. They do not in general result in a correct solution to the subtraction problem, but instead result in a buggy solution. For example, suppose the student has never borrowed from zero. The first time the student is asked to solve a borrow-from-zero problem, such as (a),

```
                    2              1
                                   2
(a)  3 0 5   (b)  3 10 15   (c)  3 10 15
   - 4 8        - 4 8          - 4 8
                  2 6 7          1 6 7
```

the student begins processing the units column by attempting to borrow from the tens column and immediately reaches an impasse because zero cannot be decremented. The student is stuck and so does a repair. One repair is simply to skip the decrement operation. This leads ultimately to the solution shown in (b). If the student uses this repair to the borrow-from-zero impasse throughout a whole subtraction test, he or she will be diagnosed as having a bug called Stops-Borrow-At-Zero. Suppose the student chooses a different repair, namely to relocate the decrement operation and do it instead on a nearby digit that is not zero, such as the nearest digit to the left in the top row, namely the three. This repair results in the solution shown in (c); the three has been decremented twice, once for the (repaired) borrow originating in the units column and once for the borrow originating in the (unchanged) tens column. If the student always chooses this repair to the impasse, he or she will be diagnosed as having the bug Borrow-Across-Zero.

The preceding story illustrates the repair process. It seems like plausible human behavior. What is needed next is empirical evidence, both to verify its existence and sharpen the understanding of how it works. Then we can move on to a formal model.

Empirical Motivation for Repair

A certain subset of the bugs shows a clear *cross product* or matrix-like pattern. In this pattern, concrete empirical evidence for the repair process is found. For simplicity, the pattern will be illustrated in terms of just four members of the subset. The four bugs are

1a. Smaller-From-Larger
1b. Zero-Instead-of-Borrow
2a. Smaller-From-Larger-Instead-of-Borrow-From-Zero
2b. Zero-Instead-of-Borrow-From-Zero

The first two bugs are related in that they miss problems that require borrowing. Smaller-From-Larger answers columns that require borrowing with a number that is the absolute difference of the two numbers. Zero-Instead-of-Borrow writes zero in such columns. The only problems that these bugs will answer correctly are those that do not require borrowing. The following problems illustrate these two bugs:

Smaller-From-Larger:	345	345	207
	−102	−129	−169
	243 ✓	224 ×	162 ×

Zero-Instead of-Borrow:	345	345	207
	−102	−129	−169
	243 ✓	220 ×	100 ×

Both bugs get the first problem correct, and miss the other two because they involve borrowing. (Correctly answered problems are marked with ✓, and incorrectly answered problems with ×.) This pattern of correct and incorrect answers unifies these bugs.

The other two bugs will miss only problems that require borrowing across a zero. Their answers to the the same problems would be:

Smaller-From-Larger-Instead-of-Borrow-From-Zero:	3 4 5	3 4¹5	2¹0 7
	−1 0 2	−1 2 9	−1 6 9
	2 4 3 ✓	2 1 6 ✓	4 2 ×
Zero-Instead-of-Borrow-From-Zero:	3 4 5	3 4¹5	2¹0 7
	−1 0 2	−1 2 9	−1 6 9
	2 4 3 ✓	2 1 6 ✓	4 0 ×

Both bugs get the second problem correct because it involves only simple borrowing. They miss the third problem because it involves borrowing from a zero. So, one dimension of the cross product pattern has been established: Bugs are grouped by the pattern of correct and incorrect answers.

The crucial second dimension of the cross product pattern is seen in the fact that the ways that the bugs in the second group miss problems *parallel* the ways that the bugs in the first group miss problems. Smaller-From-Larger-Instead-of-Borrow-From-Zero answers the units column with absolute difference, just as Smaller-From-Larger used absolute differences to answer the problems it missed. Both Zero-Instead-of-Borrow and Zero-Instead-of-Borrow-From-Zero answer the columns they miss with zero. This pattern in the answers from borrow-from-zero columns relates to the answers of the first group on borrow columns. In short, what we have here is a cross product pattern among four bugs. It could be represented informally by:

Let C = {borrow, borrow-from-zero}
Let R = {absolute-difference, max-of-zero-and-difference}
Claim: any pair in the set C × R represents a bug. Roughly speaking, the R operation is substituted for the C subprocedure.

For example, the pair ⟨borrow, absolute-difference⟩ represents Smaller-From-Larger since its hallmark is absolute differences instead of borrows.

From the standpoint of Repair Theory, the cross product pattern is just the repairs revealing themselves in the data. In Repair Theory, the set C reflects the set of *core procedures*. Core procedures are the product of mislearning or forgetting. They are named this way because, like an apple's core, they are not directly observable. Their structures are obscured by the effects of repair. The set R reflects the set of repairs. When a core procedure is executed on a problem that causes it to reach an impasse, the impasse is patched with one of the repairs, resulting in a bug.

A crucial fact about the repair process comes out clearly in the cross product pattern. It is the *independence* of repairs and impasses. Every repair is applicable to every impasse (core procedure). In principle, a bug will be found for each pairing of an applicable repair with a core procedure.

Of course, some pairs are much more popular than others, and some core procedures are more common than others. Combining an unpopular repair with an uncommon core procedure sometimes predicts a bug that has not yet been observed. This is indeed the case with the aforementioned four bugs, whose frequencies of occurrence in the 1138-student American sample are as follows:

	Borrow	Borrow-from-zero
Absolute-difference	124 (SFL)	5 (SFLIBFZ)
Max-of-zero	10 (ZIB)	0 (ZIBFZ)

The core procedure that doesn't know borrowing is more common than the one that doesn't know borrowing across zero. The repair that results in an absolute difference is more popular than the one resulting in taking the maximum of zero and the column difference. Impasse-repair independence explains why there have as yet been no occurrences of the bug resulting from pairing the unpopular repair with the unpopular core procedure. Note that this frequency distribution could not be explained in this way without reference to the repair process, which adds more evidence for the existence of that process.

Independence of repairs and impasses is one central feature of the repair process. Another relates to *when* the repair process is carried out by the subject. It could be that repair is something like forgetting or mislearning. It could happen while the student is sleeping, or watching the teacher, or explaining the procedure to a friend. All one can see in the cross product pattern is the result of repair, and not *when* it happened. However, there are data indicating that repair actually occurs during solution of problems, as the story earlier implied. These data involve a behavior labeled *bug migration*.

Bug migration is the phenomenon of a student switching among two or more bugs during a short period of time with no intervening instruction. The bugs the student is switching among are related in that they result from applying different repairs to the same impasse. That is, the student appears to have the

same core procedure throughout the period of observation, but chooses to repair its impasses differently at different times. Switching repairs establishes that the repair process takes place during that period. Two periods of bug migration have been observed: intertest and intratest.

Intratest bug migration (which was called *tinkering* in earlier reports) occurs when a student answers part of the test as if he or she had one bug, and the other part as if he or she had a different bug. That is, the student appears to have the same core procedure throughout the test, but chooses to repair its impasse differently on different test problems. This establishes that repair happens during a test. Many instances of intratest bug migration have been observed (VanLehn, 1981). Figure 5.1 presents one. (Figure 5.1 is an exact reproduction of a test taken by student 22 of class 34. She misses only six problems, namely the ones that require borrowing from zero. The first two problems she misses [the second and third problems on the fourth row] are answered as if she had the bug Stops-Borrow-At-Zero. That is, she gets stuck when she attempts to decrement a zero and repairs by skipping the decrement operation. The next two problems she misses [the first two problems on the last row] are answered as if she had the bug

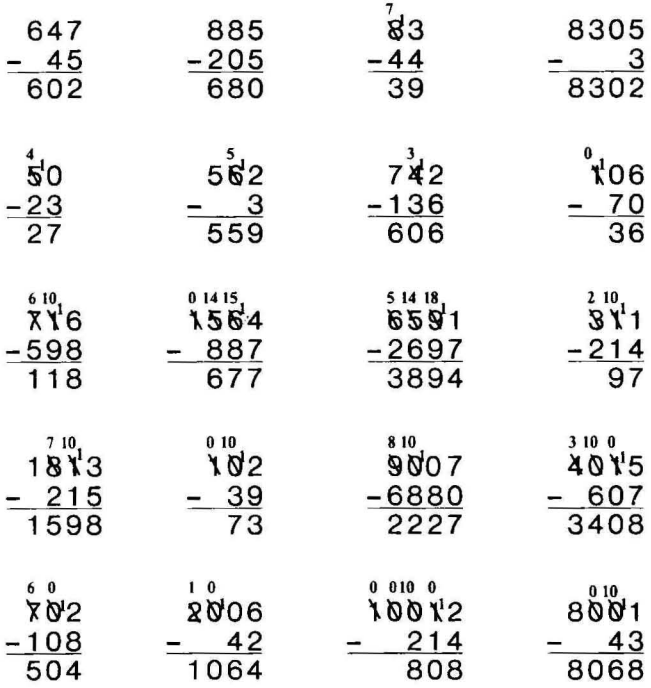

Figure 5.1. An example of intratest bug migration.

Borrow-Across-Zero. She hits the same impasse, but repairs by relocating the decrement leftward. On the third problem of the last row, she uses two repairs within the same problem. For the borrow originating in the tens column, she backs up from the decrement-zero impasse and writes a zero as the answer in the tens column [as if she had the bug Zero-Instead-of-Borrow-From-Zero]. In the hundreds column, she takes the same left-relocate repair that she used on the preceding two problems. On the last problem, she reverts to the original repair of skipping the stuck decrement of both borrows.)

The student of Figure 5.1 is typical in that her repairs occur in "runs." The first two repairs are one kind, the next two are another, and so on. This observation suggests that there can be a temporary association of an impasse with a repair. In Repair Theory, these pairs are called *patches*. Apparently, the first time the student of Figure 5.1 hit the impasse, she searched for an applicable repair and not only used it, but created a patch to remember that she used it. On the next problem, she again encounters the impasse, but instead of searching for a new repair, she just retrieves the patch and uses its repair. She completes the next problem without encountering the impasse, which is apparently enough to cause her to forget her patch, since the next time she hits the impasse, she repairs it a new way. Either the patch was forgotten during the nonimpasse problem or she chose to ignore it and try a different repair. The latter possibility is supported by her behavior at the end of the test, where she is applying different repairs for each impasse even when the impasses occur in the same problem. In short, there seems to be some flexibility in whether patches are ignored and perhaps also in how long they are retained.

Intertest bug migration is detected by testing students twice a short time apart (say, 2 days) with no intervening instruction. It is the phenomena of a student having a consistent bug on each test, but not the same bug. The bugs are related in that they can be generated by different repairs to the same impasse. It appears that the student has retained the same core procedure between the two tests, but the patch that was used on the first test was not retained. Instead, a new repair was selected, stored in a patch, and used consistently throughout the second test. Although intertest bug instability is the norm rather than the exception (only 12% of the bugs remained stable in one study (VanLehn, 1981); Bunderson (1981) reports no stable bugs at all), bug migration accounts for well over half of it (VanLehn, 1981).

Both kinds of bug migration were predicted in advance of their observation (Brown & VanLehn, 1980). They fall out as a natural consequence of viewing the repair process as modifying the execution (short-term) state of the processor that interprets the stored core procedure. An alternative is to view repair as modifying the core procedure. This "core procedure modification" view accounts for stable bugs, but not for bug migration. (Bugs are sometimes held for months or years [Brown & Burton, 1978; VanLehn, 1981].) Repairing the pro-

cessor state leaves open the question of whether patches are ever abstracted and stored in long-term memory as modifications to the core procedure—it is not necessary to assume such an abstraction process, even though it is highly plausible since it suffices to explain long-standing bugs as a long-term patch retention (in fact, that is the approach used by the formal model that will be introduced in the next section). Whether patches are incorporated into the core procedure has been left unresolved for the time being.

So, from this overview of the data two important aspects of the repair process have been abstracted, as well as a variety of evidence for the existence of the process itself. First, in principle any repair can be applied to any impasse. Second, repairs are modifications to the execution state of the core procedure's interpreter. The basic patterns in the bug data were the cross-product pattern and the bug migration classes. (NB: A bug migration class is the set of bugs that switch with each other.)

A Process Model for Interpretation and Repair of Core Procedures

The previous section introduced some of the insights that Repair Theory is based on. This section presents the theory more formally.

Repair Theory has two parts: a constraint-based account of how core procedures arise, and a process model that interprets and repairs a core procedure to generate or simulate a buggy behavior. This section discusses the second part, which is referred to as the application model since it applies the core procedure to given problems.

Figure 5.2 is a block diagram of the model's components. The model is given a core procedure and a sequence of stimuli. For subtraction, the stimuli represent subtraction problems as they appear on the test page. The model solves each problem, producing a sequence of actions that are asserted to match a subject's actions while solving the same problems. In the sense that its action sequences are expected to map onto subjects' action sequences, idealized as bugs, it is a process model. No other claims are made about the mapping. In particular, the speeds at which the actions are generated is considered irrelevant. Also, the theory is neutral about the ontological status of the model's components.

The application model is composed of the *interpreter* and the *local problem solver*. The interpreter executes the core procedure on each input, one after the other. Executing the core procedure involves constant changes to the execution state. At any given time execution state reflects not only where in the core procedure the interpreter is, but what aspects of the external state, namely the test problem, the interpreter is attending to. Reaching an impasse occurs if an attempt

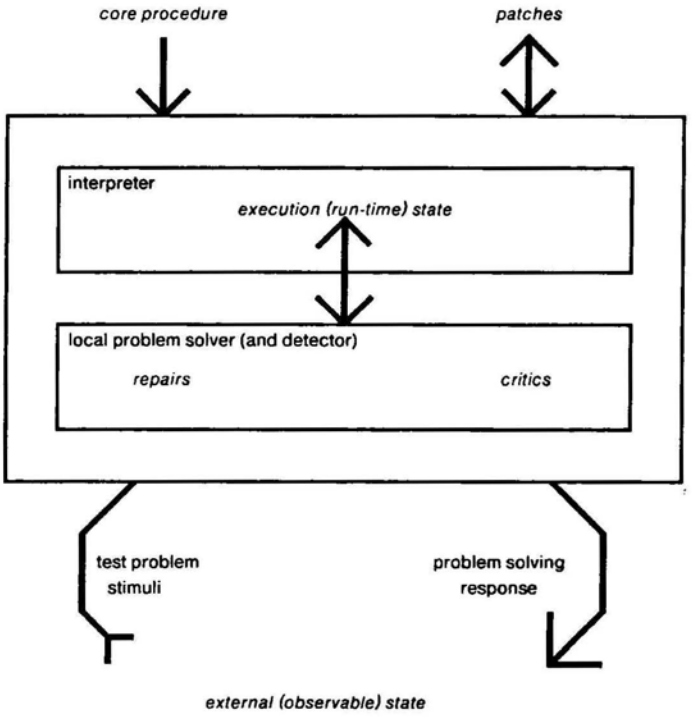

Figure 5.2. Components of the application model.

is made to execute a primitive action in a situation that violates the action's preconditions, such as attempting to decrement a zero or a blank. When this happens, control shifts from the interpreter to the local problem solver. If there is a patch for this impasse in the patch set, the patch's repair is applied. (The theorist can override this in order to match bug migrations—the default actions of the model make it produce stable bugs.) Otherwise, the local problem solver calculates which repairs are applicable, and one is selected. (The theorist can control which of the applicable repairs is selected, which in turn controls which bugs the model generates.) A patch pairing the selected repair with the impasse is created and stored in the patch set. (NB: The patch retention is not yet fully understood; although it will be shown that execution state must be a stack and not a buffer, no similar claim can be made about the architecture of patch storage.)

Regardless of whether the repair was retrieved from a patch or searched out, the local problem solver applies the repair, which modifies the interpreter's execution state. Now control switches back to the interpreter and the core procedure continues, but not of course from exactly where it left off. Some core

procedures never reach an impasse on any problem. Others hit several impasses, sometimes on the same problem.

The local problem solver also contains a set of *critics*. A critic watches the core procedure as it's interpreted or repaired and objects if any action would violate the critic's condition. During interpretation, critic violations are handled just like precondition violations—an impasse occurs and is repaired. Critics also figure in determining which repairs are applicable at a given impasse. Repairs that would cause an immediate violation of a critic are not considered applicable. A typical critic for subtraction is one that objects to writing a two-digit column answer. There are a number of theoretical problems with critics, some of which are mentioned in Brown and VanLehn (1980). Since development of the critic idea is currently in a state of flux, this chapter will have little to say about critics.

In the preceding motivational section, several facts about the repair process were uncovered. These and others can now be expressed as formal constraints that are obeyed by the model:

Impasse-repair independence: Any impasse can be repaired by any applicable repair.

Local manipulation: The only information that repairs can change is the interpreter's execution state. Most commonly, they substitute an action for the "current" action, that is, the action the interpreter was about to execute but could not. They cannot execute the action themselves nor change the core procedure. All they can do is reset the execution state so that the interpreter does something different when control returns to it.

Core procedure immutability: The application model cannot change the core procedure. It does, however, change the set of patches, and this is how stable bugs and various patterns of bug migrations are captured.

Local applicability: A repair is applicable if and only if its manipulation of the execution state allows the interpreter to run. That is, repairs must "unstick" the interpreter. It may only be unstuck for a moment—a new impasse might occur immediately after the patched action is executed by the interpreter. In fact, it is rather common for two impasses to occur in a row. Another view of this constraint is that repairs cannot "look ahead" to see if their manipulation will cause ill effects later on. The local problem solver can only check preconditions and critics in the execution state that the proposed repair's manipulation would create. If none is violated, then that repair is applicable.

Domain independence: Repairs are domain independent. This is, the same set of repairs is found in other tasks besides subtraction.

One purpose of these constraints is to limit the degrees of freedom or *tailorability* of the model. The existence of a given repair can only be inferred

from data, in this case by mapping backward from bug migration classes and cross-product patterns of bugs. There is no a priori exhaustive set of repairs. Since the set of bugs must be kept open, it is very important to constrain what the members of that set can do. If a repair can be arbitrarily powerful, then virtually any behavior can be "explained" by postulating a repair to fit it, which makes the theory irrefutable and vaccuous. Some of these principles directly constrain what repairs can be, as the *local manipulation* principle does. Later, more locality constraints will be proposed and play crucial roles in arguments.

Another way to limit tailorability is to make the theorist pay a heavy price for adding ad hoc repairs. That is, any change to increase the empirical coverage must make other predictions that may or may not be correct. In short, changes have *entailments*. For example, the impasse-repair independence principle and the local applicability principle work together to insure that adding a new repair in order to generate a certain observed bug will cause the theory to predict many new bugs (surface procedures), namely all those that are derived by applying the new repair to all other impasses. Some of these predicted bugs may exist, in which case the addition is good. But more often, the procedures are not bugs and in fact may be such absurd predictions that it is clear that the theory should not be making them. So, adding a repair may entail making many dubious if not incorrect predictions. By strict adherence to the principles of the theory, tailorability can be limited.

Observational, Descriptive, and Explanatory Adequacy

For the purposes of this chapter, it is sufficient to take a relatively well-known set of criteria for evaluating the theory (see Brown & VanLehn, 1980, for a different set). Pioneered by Chomsky (1965), observational, descriptive, and explanatory adequacy have become a standard in linguistics. In the context of Repair Theory, they have a particularly crisp characterization.

Observational adequacy is the ability of the theory to cover its observations. Repair theory currently uses two kinds of data. Hence, there are two instantiations of the criterion:

1. The model should be able to generate all the bugs.
2. The model should be able to generate all the bug migration classes.

Other kinds of observations could perhaps be enlisted later in testing observational adequacy. In particular, studying the co-occurrence frequencies of bugs (i.e., how often a pair of bugs occur together as part of a multiple bug diagnosis) and the overall frequencies of bugs seems a fruitful direction.

Descriptive adequacy measures the ability of the theory to cover "poten-

tial'' facts about behavior, which by their very nature cannot be observed. In Repair Theory, the most common ''potential'' facts are *star bugs*. Sometimes the model produces behavior that not only does not simulate any observed bug or bug migration, but is so absurd that it is highly doubtful that any subject will behave like that. Such absurd behaviors are called *star bugs* (after the linguistic convention of placing a star before sentences that are not a part of the language). A trivial example of a star bug is the model going into an infinite loop. A second, more subtle example is the behavior the model exhibits when it is able to borrow perfectly, even borrowing across several zeros correctly when required to, and yet despite its borrowing expertise, it is unable to complete the borrowing by subtracting the column that originated the borrow and leaves those columns' answers blank. This leads to problems answered like this:

```
  8 9
  9 0 15 7
- 1 3 8 2
-------
  7 6   5
```

A star bug can never be an observation in the sense that bugs and bug migration classes are observations. Its assertional content is that a certain behavior will *never* be observed. That modality cannot be directly tested.

Descriptive adequacy insists that the model generate no star bugs. Generating star bugs indicates that some constraint upon it is missing; the theory is at best incomplete and at worst just plain wrong.

I will sometimes lump descriptive adequacy and observational adequacy under the nonstandard term *empirical adequacy*.

Explanatory adequacy is the ability of the theory to allow constraints on acquisition to be formulated. This is perhaps the most important measure of the three, and also the most subtle. The idea behind it is that acquisition is such a complicated process that it is best to approach it gradually by first finding out what constraints it obeys. This paradigm will be explained in detail in the next section. Put briefly here its instantiation for Repair Theory is that the bug database is mapped backward, so to speak, through repair in order to *deduce* the set of core procedures that spans the developmental period. Not only can one deduce what each core procedure does, one can infer something about its *form* as well. These inferences are some of the most interesting and difficult in the theory. They will be presented in detail in the following sections. It suffices to say here that working with the bug data allows one to discover the content and even the form of core procedures. Knowing the set of ''observed'' core procedures puts one in an excellent position to determine constraints that are obeyed by *all* core procedures, which not only makes testable predictions about core procedures that have not yet been observed, but leads perhaps to an understanding of core procedure acquisition. Changes to the theory (e.g., a new syntax for

the core procedure representation language) that allow a new constraint on the "observed" core procedures to be formulated increases the explanatory adequacy of the theory.

The empirical adequacy of the original version of Repair Theory was reported by Brown and VanLehn (1980). Using a highly constrained set of nine core procedures, it generated 21 of the observed bugs (i.e., about one quarter of the observed bugs) and 1 star bug. It also predicted the eventual observation of 10 bugs and several bug migration classes. Empirical work since that first report on Repair Theory verified the existence of one of the 10 predicted bugs, as well as establishing the existence of several predicted bug migration classes (VanLehn, 1981). With a less constrained set of core procedures, the theory generated 43 of the observed bugs (i.e., about one half). Due to the way the theory was tested with this larger core procedure set, the sets of star bugs, predicted bugs, and bug migrations that it generated are not known, except that they include the corresponding sets generated by the more constrained set of core procedures.

The main difference between these versions of the theory and the one discussed here is that core procedures are being generated in a different way. This change was made mostly in order to increase the theory's explanatory adequacy, but it also improved the descriptive adequacy by eliminating the generation of the star bug.

However, the point of this chapter is not to revise the theory in order to improve its empirical adequacy. Rather, it is to present the *connection* between empirical data and existing theory. Even though the theory has been elaborated beyond even the version reported here and its predictions have improved, that development is proceeding so rapidly that it would be pointless to report it now in the middle of its flight. On the other hand, the arguments behind the theory reported here are completely stable. So far, the ongoing revisions set them in a new context but do not affect their validity. Establishing the empirical or explanatory necessity of a certain architecture or constraint seems to fix it as being a permanent feature of succeeding versions. It would perhaps not be too strong to say that adding new arguments to this theory, and perhaps any theory, seems much more important than adding more data points to its empirical coverage.

CONSTRAINTS ON ACQUISITION OF NONNATURAL KNOWLEDGE

Recent research has viewed the child's acquisition of mathematical skills as a highly structured phenomenon. Other chapters in this volume show that development of even very simple procedural skills, such as mental arthmetic (Resnick), are characterized by intermediate stages of well-formed, stable procedures. The emphasis of this research is on accurate description of the child's

knowledge and how it evolves in a school setting. In the past, developmental psychology tended to avoid studying learning that is accompanied by formal instruction. On the other hand, educational research tended to describe intermediate states of skill acquisition in prescriptive terms, such as *bad habits of work, systematic errors,* or *weakness in component skills.* A recent trend is to view the misconceptions that students pass through from a developmental rather than an educational standpoint.

It has been pointed out (Keil, 1981) that there are two traditional views of cognitive development: as the acquisition of knowledge through simple and general mechanisms (learning theory views) and as a sequence of radical restructurings of knowledge (stage theory views). A new viewpoint has emerged recently. It emphasizes what remains constant in the developing cognition. It looks for constraints that are obeyed by knowledge at all stages. In a sense, this view is a natural outgrowth of the earlier views. Learning theories studied acquisition over short periods of times, often a single laboratory session. Stage theories chopped a long-term development into medium-sized lengths of time, stages. The new view studies the whole of a long-term development in order to characterize what is constant about the knowledge throughout its acquisition.

This might seem a strange thing for a developmental psychologist to do. How could knowing what is the same about knowledge at all stages be important in understanding how it changes? The crucial assumption is that the properties of knowledge that are observed to be constant are in fact *constraints* on the acquisition process. They impose real limitations on the output of whatever mechanisms implement learning. Or perhaps they are a consequence of the structure of those mechanisms or the mental representations the mechanisms work with. How these constraints are imposed by processes, mechanisms, or representations is not yet known. Although the proponents of this view tend to speak in terms of a "generate and filter" mechanism, where simple induction generates new knowledge from experience while the constraints filter some of it out, this is not meant, I think, as a proposal for an actual learning mechanism. The mechanisms are unknown.

It can be objected that merely to say that there are constraints on the knowledge that can be acquired is vaccuous. The mind is not a *tabula rasa.* Since there is an infinite set of hypotheses that are consistent with any finite set of observations, to explain how people converge on knowledge states that are identical (or at least very similar) requires postulating some kind of prior constraints. So, to say there are constraints is nothing new.

The rejoinder to this objection is that the constraints that have been found so far are very strong and very specific to the domain of knowledge they apply to. For example, constraints on the syntax of languages mention clauses and noun phrases—which are certainly domain specific terms—in order to sharply limit syntactic transformations of each with respect to the other (Chomsky, 1980). So

the substance of the constraint theory view lies in the strength and domain specificity of constraints. One of its leading proponents, F. C. Keil (1981), puts it succinctly:

> Knowledge acquisition cannot proceed successfully unless the inductive devices that apply to various cognitive domains are constrained in their outputs. This much is a logical necessity. The important and more controversial claim to be made here is that, when one examines what sorts of constraints are present, one discovers that they are highly elaborated and domain-specific, and that such specificity is necessary to explain the ease and rapidity with which so much of human knowledge is acquired. The solution to the mystery of how children acquire so much so fast lies not in the formulation of ever more powerful and sophisticated induction procedures, but rather in the specification of how relatively simple induction procedures are limited in particular domains [pp. 199–200].

The work reported in this chapter grew out of application of the constraint theory view to the development of skill in mathematical procedures. It was originally thought that procedural skill might be a domain like language or vision in that there would be "highly elaborated and domain-specific" constraints on its knowledge structures. This led to a direct application of Chomskyan methods to its study. The results of these initial studies, which are presented next, prompted a major revision of the approach.

Two Approaches Fail: Chomsky's and Keil's

The purest Chomskyan approach is to establish such strong constraints on the output of acquisition that all procedures that meet these constraints are good core procedures. That this approach is Chomskyan can be seen by comparing it to the Chomskyan approach to grammar acquisition. Such strong constraints are placed on grammars that the learner has no choice but to learn a natural language. The language environment that the child grows up in (e.g., English, French, Walbiri) is needed only to select among the few alternatives that the constraints have left open, thereby determining which natural language is acquired. Mapping the Chomskyan approach to acquisition of procedural skills involves substituting "subtraction" or "addition" for "English" or "French." The child cannot help but learn a core procedure of a certain highly constrained form. The goal of the theory, on this approach, is to uncover what the constraints on core procedures are.

This approach gains plausibility when one considers that grammars and procedures are formally isomorphic. Mathematically, it has been shown that any context-free grammar can be converted to a push-down automation and vice versa. More realistically, transformational grammars and augmented transition

nets are generally held to be notational variants at one level, and it is highly likely that any procedure can be expressed as an augmented transition net. So, it would not be surprising if the constraints-on-grammars approach could be applied to acquisition of procedures.

Strong constraints on the form of core procedures were indeed discovered with the Chomskyan approach, but they were not strong enough to restrict the set of computations that could be expressed as core procedures. The problem is that whereas there are good arguments for expressing core procedures in a certain formal language as opposed to any other, that language is nearly isomorphic to lambda calculus. It is widely accepted that lambda calculus has sufficient expressive power to represent any procedure (i.e., Church's hypothesis). To say that a core procedure can be any procedure expressible in this lambda calculus-like language is equivalent to saying a core procedure can be any procedure, which is saying nothing at all. The language constrained the *form* of the core procedures, but did not narrow the set of core procedures that could be expressed. In other words, putting constraints on the language used to express core procedures is not enough. The other approaches retain the constraints discovered with the Chomskyan approach and add more.

A second problem with the pure Chomskyan approach is that it only explains why mature knowledge has the form it does. It does not account for patterns in the development of knowledge. Keil's work on ontological knowledge is a good example of this incompleteness. He found that a certain kind of ontological knowledge was subject to the constraint that it be dendralic, that is, a tree (Keil, 1981). However, he also found that children's trees were "collapsed" versions of adult trees. He states, "The collapses are highly systematic, involving more than simple conformity to the [tree] constraint, [p. 208]." That is, a child's ontological knowledge was not just any tree, it was a certain kind of homomorphic image of an adult tree. This observation cannot in principle be captured by unary constraints on knowledge states, but they can be expressed by binary constraints: for example, a structure preserving transformation (homomorphism) relating the child's knowledge to a certain ideal or adult knowledge. Similar observations in the domain of procedural skills led Brown and VanLehn (1980) independently to adopt the same homomorphism approach as Keil.

In Brown and VanLehn (1980), all core procedures were generated by deleting parts of a mature, adult core procedure. Each deletion generated an incomplete core procedure, which like Keil's collapsed ontological trees, characterized a child's partially developed knowledge. In general, this was a rather successful approach in that it lent itself well to an exhaustive formal treatment, the precision of which led to deeper insights into the nature of acquisition. However, the deletion approach itself was abandoned due to some problems that will now be discussed.

As a step toward understanding acquisition, one needs to know which of all the possible deletions to a core procedure in fact occur. Thus, deletion was

formalized as an operator that applied freely, deleting any rule or set of rules (the major syntactic unit in the language for expressing procedures is a rule). As expected, unconstrained deletion sometimes generated star bugs, that is, procedures that are so absurd that it is doubtful that they will ever be observed as bugs. The deletion operator must be constrained so that the theory will not predict that this behavior will occur as a bug. Brown and VanLehn (1980) used a set of "deletion blocking principles" to constrain deletion and avoid predicting star bugs. It is in these constraints on deletion that any understanding of acquisition lurks.

Although the details will not be presented here, this approach did not succeed empirically. The most important of the deletion blocking principles, the one dwelled upon in Brown and VanLehn (1980), had apparent implications for acquisition, but some of the others did not. Yet if the unmotivated deletion blocking principles were dropped, the theory generated star bugs.

A second empirical problem was that there are core procedures that cannot be generated in any easy way by rule deletion. As an example, there is a bug that always decrements the leftmost, top digit of a problem regardless of where the column that caused the borrow is. This bug can be explained informally by supposing that the subject who has this bug was tested at a point in his or her schooling where he or she had only practiced borrowing on two column problems. In such problems, the correct digit to decrement is indeed the leftmost, top digit. The subject has not yet had problems of the proper form to discriminate between the "leftmost" abstraction and the "left-adjacent" abstraction. It would be difficult if not impossible to generate a core procedure that used the "leftmost" abstraction in place of the "left-adjacent" abstraction using rule deletion.

Even if such a deletion operator could be formulated, it would probably not capture an important aspect of that core procedure, namely that it depends on the fact that the subtraction curriculum often involves introducing borrowing with two-column problems. Had subjects been taught borrowing some other way, the theory would still generate the "leftmost" core procedure, a prediction that would presumably be false in this hypothetical population. The deletion approach is not expressing acquisitionally relevant information in that it does not show dependence of core procedures on the teaching that subjects receive; it only shows dependence on the form of the mature skill. In cases where the same form can be acquired in the context of different teaching sequences, the deletion approach cannot make different predictions about the intermediate states of knowledge that students in the different teaching sequences can have. It is this reason more than any other that advises against applying Keil's homomorphism approach to taught knowledge. The following speculations amplify this point.

Chomsky, Keil, and others suggest that constraints are at least partially a product of human evolution (Chomsky, 1980; Keil, 1981). This implies that strong, domain-specific constraints could only be found for "natural" knowl-

edge or "faculties," that is, knowledge that is easily imagined to be the product of evolution. Such knowledge is acquired easily and rapidly, usually without formal tutelage. It has some survival value and has been a part of the culture long enough for that survival value to cause significant natural selection. Given this, one would expect Chomskyan approaches to subtraction to fail since subtraction is clearly nonnatural knowledge (yet it is knowledge that everyone can learn). It meets none of the criteria for a "faculty." Unlike counting and simple mental arithmetic, the written multicolumn subtraction algorithm has not been observed to develop without instruction. Also, it has been common in our culture only in the last two centuries, which makes it hard to imagine it as a product of our species' evolution. In short, as nonnatural knowledge, subtraction could not be expected to exhibit strong domain-specific constraints, either of the unary or the homomorphic type. But pursuing this speculative line of reasoning a little further leads to a proposal for some contraints that it could be expected to satisfy.

Consider all the knowledge an adult has and take away natural knowledge domains: vision, language, ontology, spatial reasoning, counting, and so on. There is still a great deal of information left; call it nonnatural knowledge. Some of this nonnatural knowledge has, I assume, enough survival value that an organism that supported its transmission across generations would have a clear selectional advantage over others. Yet, by assumption, there is no direct genetic support for this knowledge per se. Indeed, this is a potential feature since it frees the knowledge to evolve at speeds far in excess of those that characterize changes in the genotype. This question is, how does this transmission work, from evolution's point of view?

If an organism is to be thrust into an information-rich world, evolution would do well to encode a special, highly restricted receptivity for knowledge with a proven survival value: natural knowledge. But restricted receptivity for nonnatural knowledge cannot be based on the structure of that knowledge in isolation since evolution cannot commit the organism in advance. To put it metaphorically, since the genes cannot decide what's useful, they must leave the decision to someone else and provide a special route for the chosen information to be written into the organism's mind. Note that it would not work to let just *any* information be absorbed by the young organisms—there is such a buzzing, blooming confusion of information in the world that information with survival value would be buried in useless information. Evolution should provide a faculty that locks out most information, but provides a "key" or "carrier wave" that allows the chosen knowledge to bypass the filter and become a part of the young organism's knowledge.

My guess is that this coded conduit is active during "teaching," an activity that sometimes goes on between teachers and pupils, masters and apprentices, or neolithic hunters and their children. Something in the structure of the instructional interaction is guiding the target knowledge (e.g., subtraction or chipping out flint arrowheads) through the barriers, allowing the student to learn it. Per-

haps repetition is crucial—but nature is full of distracting, useless repeitions. More likely, it is the movement from simple cases to complex ones, regulated by the teacher's assessment of the student's comprehension, perhaps in combination with repetition. The essence of the approach described in the next section is to find formal constraints on the natural conduit that allows instruction in non-natural knowledge to succeed.

A Third Approach: Perfect Prefix Learning with Deletion

The third approach, the one taken in this chapter, is motivated by the belief that core procedures are due not only to incomplete or flawed learning, but also to incomplete traversal of the teaching sequence. In point of fact, many subjects in the subtraction database had not been taught all of the procedural skill at the time they were tested (e.g., they had not been shown how to borrow from zero, but only how to borrow from nonzero digits). In many case, subjects appear to have mastered what they had been taught, but they had only been taught the initial segments or "prefix" of the instructional sequence. The key idea is to split generation of core procedures into *perfect prefix learning* and *deletion*. That is, instead of attacking the core procedure problem with one formalism that captures all aspects, it is broken into two subproblems: flawless learning of partial instruction and the relationship between flawless core procedures and the other, flawed ones. This division is a rather natural one, given the kinds of core procedures one finds. The following two bugs illustrate why both kinds of generation are needed. The first bug can be generated by perfect prefix learning and repair:

Stops-Borrow-At-Zero:	3 4 5	$3\ 4^{\overset{3}{1}}5$	$3^{\overset{2}{1}}0^{\overset{1}{1}}7$
	$\underline{-1\ 0\ 2}$	$\underline{-1\ 2\ 9}$	$\underline{-1\ 6\ 9}$
	2 4 3 ✓	2 1 6 ✓	1 4 8 ×

The core procedure behind this bug does not know how to borrow across zeros. It borrows correctly from nonzero digits, as shown in the second problem. On the third problem, it attempts to decrement the zero, hits an impasse, and repairs by skipping the decrement operation entirely. The point is that this bug has a complete, flawless knowledge of borrowing from nonzero digits, but precisely at one of the known junctures in the subtraction curriculum, its understanding stops. Now compare this knowledge state with the one implicated by the following bug:

Borrow-From-Zero:	3 4 5	$3\ 4^{\overset{3}{1}}5$	$3\ 0^{\overset{9}{1}}7$
	$\underline{-1\ 0\ 2}$	$\underline{-1\ 2\ 9}$	$\underline{-1\ 6\ 9}$
	2 4 3 ✓	2 1 6 ✓	2 3 8 ×