COGNITIVE SCIENCE **13**, 415–465 (1989)

Non-LIFO Execution of Cognitive Procedures

KURT VANLEHN, WILLIAM BALL, BERNADETTE KOWALSKI
Carnegie-Mellon University

Many current theories of human problem solving and skill acquisition assume that people work only on the unsatisfied goal that was created most recently. That is, the architecture obeys a last-in-first-out (LIFO) constraint on the selection of goals. This restriction seems necessary for the proper functioning of automatic learning mechanisms, such as production compilation and chunking. It is argued that this restriction is violated by some subjects on some tasks. In particular, 8 subjects (from a sample of 26) execute subtraction procedures in a way that violates the LIFO constraint. Although there is a great deal of between- and within-subject strategy variation in the 8 subjects' behavior, it can be simply explained by hypothesizing that (1) the goal selection is not necessarily LIFO, (2) goal selection knowledge is represented by explicit preferences, and (3) the 8 subjects have just a few preferences that are overgeneralized, overspecialized, or missing. The rest of their preferences are correct. On the other hand, LIFO-based models seem unable to explain the strategy variations in any simple way. Thus, it seems that part of the flexibility in human problem solving comes from having a choice of which goal to work on next. Fortunately, it is simple to ammend automatic learning mechanisms so that they will function correctly in a non-LIFO architecture.

1. INTRODUCTION

It is possible that people can select any pending goal they can recall as their next focus of attention. The model of problem solving presented in this article has this property, but most models—including GPS (Ernst & Newell, 1969), HPS (Anzai, 1978; Anzai & Simon, 1979), ACT* (Anderson, 1983, 1987; Anderson, Farrell, & Saurers, 1984; Anderson & Thompson, 1986; Soar (Laird, Newell, & Rosenbloom, 1987), and Sierra (VanLehn, in press)—

We would like to thank Jamesine Friend for conducting the experiment, and Austin Henderson for writing the data collection program. Micki Chi, Paul Rosenbloom, Clayton Lewis and Stellan Ohlsson provided valuable comments on the manuscript. Micki Chi's thoughtful advice was particularly important to the development of the research. This research was supported by the Cognitive Science Program, Psychological Sciences Division, Office of the Naval Research, under Contract No. N00014-88-K-0688. Reproduction in whole or in part is permitted for any purpose of the United States Government. Approved for public release; distribution unlimited.

Correspondence and requests for reprints should be sent to Kurt VanLehn, Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA 15213.

allow students to select only the most recently created pending goal. That is, they impose a LIFO (last in first out) restriction on goal selection.[1] For instance, suppose that processing goal A creates subgoals B and C, then subgoal B is selected and its processing creates sub-subgoal D. At this point, there are at least two pending goals, D and C, in working memory. A LIFO architecture has no choice; it must select goal D because D was created after C. A non-LIFO architecture can select either goal. On the face of it, the LIFO restriction seems a little strange. If working memory is equated with information that can be easily recalled, then the LIFO restriction asserts that people find it impossible to work on certain goals that they can easily recall.

There are no compelling computational reasons for placing a LIFO restriction on problem solving. Indeed, many contemporary AI problem-solving systems have non-LIFO architectures. Typically, whenever the currently executing goal (often called a task) generates subgoals, they are placed in a set of pending tasks. When the currently executing task completes, the architecture selects the next task from the set of pending tasks.[2] Although the task selected may be one of the ones just created, it does not have to be, and this makes the architecture non-LIFO. Despite the widespread use of non-LIFO problem-solving architectures in AI, no one has seriously considered whether *human* problem solving might be non-LIFO. Most problem-solving systems, that have been claimed as models of human problem solving have been LIFO, but that may have been more a matter of convenience than theoretical conviction.

There are many difficulties in determining whether people obey a LIFO restriction. Here are four:

1. Usually it is not known exactly what the subjects' procedures are. If their behavior is consistent with a LIFO execution of one procedure, and a non-LIFO execution of a different procedure, but it is not known which procedure they have, then how can one tell whether the LIFO constraint is being obeyed?
2. In many task domains, the LIFO restriction is imposed by the task itself. For instance, if one is using a goal-recursive strategy for solving the Tower of Hanoi puzzle, then the rules of the puzzle force one to attend to the Move-disk goals in LIFO order (Simon, 1975). Only task domains that give the subject the freedom to use a non-LIFO order are useful for testing the LIFO hypothesis.

---

[1] If several goals are created at the same time, then some models allow selection among them. But in no case is the model allowed to choose a goal that was created before some other pending goal.

[2] Architecturess vary in selection schemes they use. Some use simple numerical priorities attached to tasks. Some use heuristic rules. Others can use the full power of the problem-solving system by "going meta" and taking on goal selection as a problem in itself.

3. When a person follows a written procedure (e.g., while cooking), the goals of that procedure probably are not subject to the LIFO restriction. Suppose the procedure is memorized then followed from (declarative) memory. Presumably the goals are still not subject to the LIFO constraint. Suppose the declarative representation of the procedure is compiled into a procedural representation. Now the LIFO constraint should apply. But how does one tell whether a given subject's procedure is represented declaratively or procedurally?
4. Although the LIFO restriction applies to goals retrieved from working memory, that is not how some goals are recalled. Instead, they are constructed from the externally visible problem state (VanLehn & Ball, in press). It is not clear whether the LIFO restriction applies to reconstructed goals.

In short, the issue of LIFO versus non-LIFO architectures is quite complicated. This article certainly does not settle the issue, but merely introduces some new data from a simple task domain that avoids most of the complexities mentioned above.

The task domain is ordinary, multicolumn subtraction. This task domain has been extensively investigated, and detailed models exist for the knowledge representations and learning processes that students seem to employ (Brown & VanLehn, 1980; VanLehn, 1983a; VanLehn, in press; Young & O'Shea, 1981). Thus, Difficulty 1 is avoided. The subtraction procedure is such that non-LIFO executions are possible, so Difficulty 2 is avoided. The subjects have practiced subtraction for many hours. As shall be argued later, this suggests that the subtraction procedures are encoded as procedural knowledge and therefore should be subject to the LIFO restriction. Thus, Difficulty 3 is avoided. Although the task domain does allow goals to be stored externally and reconstructed, it will be argued that mechanisms for storing goals are "implementation" details, which should be considered separately from architectural principles, such as the LIFO constraint. Thus, Difficulty 4 is shown to have no force. In short, the task domain of subtraction is an ideal vehicle for testing the claim that people's use of goals is not subject to the LIFO constraint.

There is another advantage of subtraction that is a bit less obvious, but quite important nonetheless. If subjects reliably mention their goals as they selected them, then the goal structure mentioned earlier could be used,

```
Goal A
  Subgoal B
    Sub-subgoal D
  Subgoal C
```

If they selected goals in the order A, B, C, and D, and said so, then they have demonstrably violated the LIFO restriction. But verbal protocol data are notoriously incomplete, and seldom do subjects mention all the goals

that have played a role in their problem solving. Fortunately, in subtraction, the primitive (lowest) goals in the hierarchy correspond to reliably visible actions, such as writing a digit. This gives the experimenter partial evidence about the sequence of goals selections. This is not always good enough. For instance, suppose goals C and D correspond to reliably visible actions and the experimenter sees the subject execute C then D. This behavior is consistent with both a non-LIFO execution of the goal structure (A, B, C, D) as well as a LIFO execution (A, C, B, D). Thus, this particular goal structure is too simple to allow an experimenter to test whether or not the subject uses LIFO or non-LIFO goal selections. Testing requires a more complicated goal structure, such as:

Goal A
  Subgoal B
    Sub-subgoal B1
    Sub-subgoal B2
  Subgoal C
    Sub-subgoal C1
    Sub-subgoal C2

where goals B1, B2, C1, and C2 correspond to reliably visible actions. Now, if the subject executes the actions for goals B1, C1, B2, and C2, in that order, the experimenter is fully justified in concluding that the subject has used non-LIFO goal sections, as there are only two orders for goal section that will generate this sequence (one is A, B, B1, C, C1, B2, C2, and the other is A, C, B, B1, C1, B2, C2), and neither obey the LIFO restriction on goal selection.[3] Fortunately, subtraction procedures generate goal structures of the necessary kind, which is yet another reason for using subtraction as the task domain.

Arguing for the generality of these findings will not be attempted since it is both too hard and too easy. If one ignores the four difficulties listed earlier, then cases of non-LIFO execution are easily found: When you fix dinner, you have no problem interweaving subgoals from the Cook-dinner goal with subgoals from the Clean-up-breakfast-dishes goal, even though the LIFO restriction says that you must complete all the subgoals of Clean-up-breakfast-dishes before starting to work on Cook-dinner (or vice versa). But such easy counterexamples to the LIFO constraint are subject to all the difficulties mentioned above. For instance, the goal structure of the knowledge involved in cooking and cleaning up the kitchen is not really known, nor whether it is represented procedurally or declaratively. Thus, although

[3] Computer scientists may recognize this pattern of execution as a kind of co-routinizing or pseudo-parallel processing that can only be accomplished if the programming language has a spaghetti stack. An ordinary stack will not support suspending the execution of procedure B in order to initiate execution of C, then later resuming the execution of procedure B.

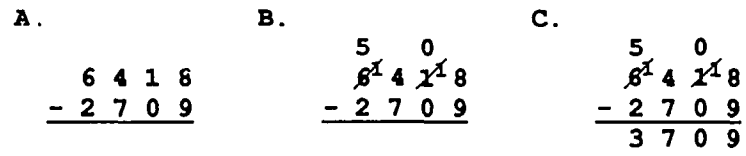

Figure 1. Initial, intermediate and final states of a nonstandard solution

it is easy to make plausible claims about generality, it is difficult to support them properly, so this article restricts its claims to subtraction.

The argument to be presented is a classic case of finding a model that captures the variance in a set of data. In this case, the data are protocols of 26 subjects solving subtraction problems. It will be shown that 8 of these subjects alternate between a standard execution sequence, wherein all the subgoals of processing a column are completed before moving on to work on the next column, and a variety of nonstandard execution sequences. For instance, some subjects do all the borrowing that a problem requires before answering any column, then they answer the columns, starting with the far left one and proceeding toward the units column (see Figure 1). The variance in the data is exactly the strategy shifts of the subjects. For instance, one subject did 4 problems with the standard execution strategy, 4 with the nonstandard strategy just mentioned, and 2 with a second nonstandard strategy. Two further problems were solved with blends of the three strategies.

After presenting the data, a problem-solving model is presented that is based upon the assumptions that (1) goals are not subject to a LIFO restriction and (2) people have explicit goal selection preferences as part of their knowledge of the subtraction procedure. A typical goal selection preference is "If there is a pending goal of type Process-column and a pending goal of type Borrow, then prefer the goal of type Borrow."

Next, it is shown that almost all the variance in the data can be captured by introducing small perturbations into the set of goal selection preferences that defines the standard execution strategy. Most perturbations consist of simply deleting a goal selection preference, wrapping a condition around it, reversing its direction, generalizing it, or specializing it. In short, the explanation for the variance in the data is that most of the 26 subjects learned a complete, standard procedure, but 8 either failed to encode a few of the standard goal preferences completely or they did encode them but later decided to revise them slightly. Either way, the 8 subjects end up with a procedure that is almost identical to the standard procedure, with only a few preferences modified.

Although this explanation for the variance is simple and intuitively compelling, it does depend upon having a non-LIFO problem-solving architecture. Because the LIFO constraint has been a fixture in successful problem-solving models for almost a decade, cautions should be taken before abandoning

such a traditional and useful assumption. So the next step in the argument is to consider what it would take to capture the variance in these data using a LIFO architecture. Several ways to model the data with LIFO architectures are tried. Although all of the alternatives worked, they all required some rather implausible extra assumptions.

Lastly, the reason for wanting a LIFO architecture in the first place is examined. The chief advantage of the LIFO restriction seems to be that it constrains automatic learning mechanisms, such as knowledge compilation (Anderson, 1987) and chunking (Laird, Rosenbloom, & Newell, 1986). A simple technique for constraining automatic learning without putting a LIFO restriction on goal selection is presented. This leads to the conclusion that there is no reason for retaining the LIFO constraint, and several reasons for dropping it.

2. THE EXPERIMENT AND ITS RESULTS

The experiment involved collecting protocols from students learning multicolumn subtraction. One advantage of this task domain is that subtraction is a fairly pure example of procedural knowledge. Students, much to the regret of their teachers, do not seem to acquire a deep understanding of the algorithm (Resnick, 1982; Resnick & Omanson, 1987; VanLehn, 1986). As far as most of them are concerned, the procedure is just so much symbol manipulation. This means that a representation can be simple and yet still suffice for representing their operative knowledge.[4] Moreover, a fair amount is already known about subtraction procedures and how they are acquired (Brown & VanLehn, 1980; VanLehn, 1983a; VanLehn, 1983b; VanLehn, in press). Before discussing the experiment itself, some additional background on this task domain is presented.

2.1. Background

Prior to the experiment described here, three phenomena were already known to occur in the behavior of subtraction students. Because these phenomena also occur in the present data, the following paragraphs briefly describe them and the best current explanations of them, as well as introducing several technical terms (in italics) that are needed later.

1. Both students and adults are known to make *slips,* which are unintentional mistakes such as omitting a borrow or misremembering a number

[4] Some students know much about the deep structure of the algorithm, but they do not seem to use it when they solve problems. Moreover, they do not seem to care if their solution violates the subtraction principles that they can state (Resnick & Omanson, 1987). The procedural representations here are meant to cover only the parts of their knowledge that affect problem solving.

fact (e.g., thinking 13 − 7 = 5). Norman (1981) has proposed a preliminary model of slips.

2. Subtraction is taught in the second, third and fourth grades in the United States. About 40% of the students in those grades seem to have *buggy* subtraction procedures (Brown & Burton, 1978; VanLehn, 1982). A buggy procedure is a correct procedure that has one or more small changes to its structure (*bugs*) that cause it to generate incorrect answers on some problems. For instance, one common bug is called Diff-0-N = 0 because students with this bug answer columns with a zero on top by simply putting zero in the answer instead of borrowing. One can model this by simply adding a rule to a set of rules representing a correct procedure (Young & O'Shea, 1981). Other bugs can be modelled by deleting rules or substituting rules.
3. In the grades where subtraction is taught, about 18% of the students exhibit *bug migrations,* wherein they exhibit one bug on some problems and different bugs on other problems (VanLehn, 1982). For instance, a common bug migration is to do Diff-0-N = 0 on some problems and Diff-0-N = N on others.

Brown and VanLehn (1980) proposed that most bugs and bug migrations are caused by *repairs* to *impasses.* An *impasse* occurs when the student's procedure requires them to do something that they believe cannot or should not be done. A *repair* is problem solving conducted with the goal of getting past the impasse. Typical repairs are to skip the action that can/should not be done or to substitute a similar action. In the case of a bug migration between Diff-0-N = 0 and Diff-0-N = N, the impasse occurs when the student starts to borrow and notices the top digit in the column is a zero. Perhaps he or she has heard "you can't borrow from zero," and thinks the rule applies to this situation. In any case, for some reason the student reaches an impasse on borrow columns with zeros on top. The two bugs are generated by two different repairs, both of which substitute actions for the borrow. The bug Diff-0-N = N comes from substituting the normal take-difference action for the borrow. The bug Diff-0-N = 0 comes from substituting for the borrow the action that is normally used on columns whose bottom digit is missing (e.g., the tens column of 34 − 8). These two bugs illustrate how bug migrations, and for that matter, bugs themselves, are generated by repairs to impasses.

VanLehn (1987, in press) developed a computer program, called *Sierra,* which simulates the acquisition of bugs. Sierra combines example-driven learning with impasses and repairs. In one particularly rigidly controlled demonstration, it was shown that 33% of the 75 observed subtraction bugs, including almost all of the most common bugs, could be acquired by Sierra (VanLehn, in press). Hand simulation indicates that the theory could be extended to capture 85% of the bugs (VanLehn, 1986).

The data for testing Sierra came from testing thousands of students from around the world (849 from California, 288 from Massachusetts, 1325 from Nicaragua, and several smaller studies in Pennsylvania, New York, Utah, and the Philippines). In all cases, the students took diagnostic tests of about 20 problems in length,[5] with each problem presented in vertical format, for example,

```
 5067
-  92
```

Test data were not scored or aggregated prior to analysis. Instead, each student's answers were analyzed as digit strings. A student was counted as being successfully modelled if the student's answers to each problem were matched by the model's answers, digit for digit (with some allowances made for slips).

Although this method of testing and analysis is more detailed than earlier ones (see Brown & Burton, 1978, for a complete discussion), it omits one crucial piece of information. One cannot determine the chronological sequence in which the writing actions were made. One would need protocol data, such as one could collect by video taping, in order to obtain the exact sequence of writing actions. The experiment described here was designed to collect such data and thereby (it is hoped) provide additional support for Sierra. As will be seen, this purpose was not achieved, but, serendipitously, a new phenomenon was discovered instead.

2.2. Subjects and Methods

By hypothesis, impasses occur because a student's knowledge of a procedure is not quite complete. Since it is difficult to induce such a state of knowledge in the laboratory with any reliability, a screening technique was used in order to find subjects who might naturally exhibit impasses. This strongly biased subject selection means that the resulting group of subjects is not a random sample of the population. Thus, frequency of occurrence of the phenomena described later could be inflated when compared to the frequency of occurrence in the general population.

Screening was done in the context of the California study mentioned earlier. The tests used, the methods of administration, and the results are reported elsewhere (VanLehn, 1982). Since bringing protocol-taking equipment out to the schools was anticipated, only three classrooms were selected for participation in this experiment. The students in these classrooms had been tested twice, approximately two months apart with the diagnostic tests mentioned earlier. On the basis of these screening tests, 33 third graders

[5] Problems were chosen to maximize the power of the test to differentiate buggy procedures. See Burton (1982) for a description of Debuggy, the program used to generate test items and analyze the students' solutions.

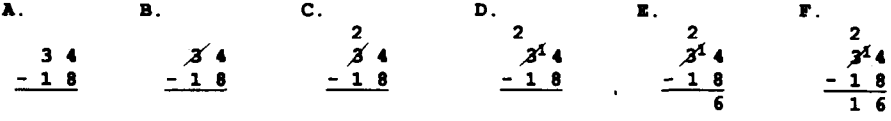

Figure 2. A correct solution, shown as a sequence of problem states

were selected. In order to have stringent data to test this theory, students who had uncommon bugs or whose behavior was not systematic enough to be modelled by bugs were selected. The belief was that these students would be more likely to exhibit impasses.

These 33 subjects were tested individually in a small room adjacent to their classrooms. Each student solved an individualized paper-and-pencil test whose items were designed to elicit the errors noted on that student's pretests. Each test consisted of about 13 subtraction problems, presented in vertical format. In order to collect the exact chronology of the students' writing actions, the test page was taped to an electronic tablet, and students filled out the test with a special pen. Equipment malfunctions caused the data from 7 students to be lost. Tablet data from each of the remaining 26 students were coded as a sequence of writing actions. For instance, a correct solution to the problem shown in Figure 2, frame A, would be coded as the 5-action sequence

[Slash(2), Decrement(2), Add-ten(1), Take-difference(1), Take-difference(2)]

where "1" means the units column, "2" means the tens column, and so forth. The states after each action are shown in Frames B through F, of the figure. Although the students' comments were also audio taped, they said so little that their verbal protocols were not transcribed or analyzed. Consequently, the raw data from this experiment consist of 26 nonverbal protocols, one for each subject.

2.3. Results

Without verbal reports, evidence for impasses would have to come from the actions and the pauses between them. It had been hoped to develop a criterion for the existence of impasses by measuring the pauses between actions, and assuming that the longer pauses were caused by impasse-repair episodes. However, fragmentary verbal data and the notes of the experimenter indicated that long pauses seemed to be caused mostly by counting in order to determine a number fact, such as the difference between 15 and 7. Against this high background variation in pause length, it would be difficult to set a reliable criterion for the duration of impasse-repair episodes relative to the duration of nonrepair actions. Consequently, one purpose for the experiment, which was to collect protocol data on what happens at impasses could not be achieved, because no reliable way to detect impasses directly was available.

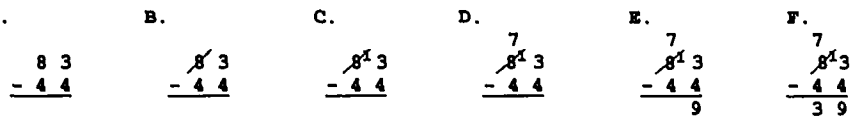

Figure 3. Hilda's solution, shown as consecutive states

Another reason for the experiment was simply to check if the models developed for data in the form of numerical answers would also fit protocol data. The earlier research had produced a large set of correct and buggy procedures, which will henceforth be called the *standard* procedures. Since this set had been developed with a very large sample of students, and this experiment had a rather small sample, it was expected that for each student in this experiment, there should be some standard procedure that would accurately simulate the student's protocol. Of the 26 students, 15 met our expectation, 8 did not, and 3 students could not be analyzed because they made no scratch marks (i.e., the marks made in the top row of the problem that indicate the actions of borrowing). The rest of this article focuses on the behavior of the 8 students whose protocols could not be modelled by a standard procedure. They will be called the *nonstandard* students for ease of reference. Appendix 1 presents their protocols and the analysis of them. The following comments summarize those analyses.

For expositional purposes, it is convenient to consider two types of behavior. The first type of behavior involves nonstandard ways of doing borrowing. For instance, on the problem shown in A of Figure 3, Hilda first puts a scratch mark through the 8, then adds 10 to the 3, then decrements the 8, and finally answers the units column, as shown in B through E of the figure. Her actions will be abbreviated, so that "S" stands for slash, "A" for adding 10, "D" for decrement, and "-" for column difference. Then Hilda exhibits the permutation SAD-. The most commonly taught borrow order is SDA-, although ASD- is also taught, so it was expected that students would exhibit either one or the other of these two standard orders, but not both. However, Hilda's SAD- order violates this expectation. She was not alone in this respect. Of the 8 nonstandard students, 7 exhibit nonstandard borrowing permutations. (The 15 standard students used either the SDA- order on the whole test, or the ASD- order on the whole test. They did not alternate among the two orders during the test, nor did they use nonstandard orders.) Table 1 gives a rough indication of the borrowing permutations exhibited by the nonstandard students by counting the number of times each permutation occurred, excluding unclear cases (e.g., borrowing across zeros). Possible permutations that never occurred are excluded from the table.

The second type of behavior involves nonstandard orders for processing columns. Although two students, Hilda and Paul, processed columns in the

TABLE 1
Borrowing Permutations Frequencies

Student	SDA-	SAD-	SA-D	ASD-	A-SD
Angela	19				
Hilda	3	6	2		
Janine				10	
Paul		2	11		
Pete		2	2		
Robby	1			3	
Tanya	2		4		
Trina	2				7
Total	27	10	19	13	7

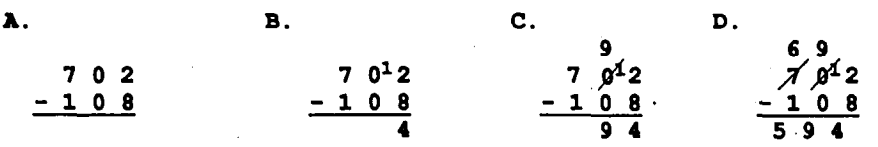

Figure 4. Janine's solution, shown as a sequence of states

standard, left-to-right order, the other nonstandard students used a variety of unusual orders. For instance, Janine used a "horizontal" ordering on 4 problems. She first did all the scratch marks required for borrowing in the whole problem, moving right to left across the top row of the problem. Then she filled in all the column's answers. Usually she did this second pass in right-to-left order, but on 1 problem, she filled in the thousands place first then moved to the right, filling in the other answer places as she went. Her way of solving the problem is called "horizontal" because it first does the top row then does the answer row. On 3 problems, Janine used a "vertical" ordering. She did all the marks in one column before moving on to the next. Thus, on the problem shown in A, Figure 4, she added 10 to the 2, and answered the units column (see Frame B). Then she moved to the tens column, where she wrote "9" above the upper zero and in the answer (Frame C). Then she moved to the hundreds column, where she decremented the 7 by one, and put "5" in the answer (Frame D). This ordering we called "vertical" because all actions that are aligned vertically are done together. On 2 other problems, Janine used a strategy that seems to be a mixture of the horizontal and vertical strategies.

Table 2 gives a rough indication of the usage of various ordering conventions for column processing. The cells in the table indicate the number of problems answered with the indicated convention.[6] The column in the table

[6] When a problem has no borrows, the column-processing convention is sometimes ambiguous, so these figures have some guesswork built into them.

TABLE 2
Frequencies of Column Processing Conventions

Student	Standard	Horizontal L. to R.	Horizontal R. to L.	Vertical	Unique
Angela	8				4
Hilda	12				
Janine	4	1	3	3	2
Paul	12				
Pete	9		3		1
Robby	11				2
Tanya			14		
Trina	9				3

labeled "Unique" lumps together various strategies that are unique to a single student. For instance, Robby has a particular strategy that he uses on 2 problems; Angela has a different one that she uses on 4 problems; both counts are shown in the "Unique" column. See Appendix 1 for descriptions of the actual ordering conventions used.

These two dimensions of variations—borrowing permutations and column-processing orders—represent the unexpected aspects in the students' behavior. The other aspects of their behavior have been observed before. All of the 8 nonstandard students exhibited slips. Four of the 8 students (Hilda, Robby, Tanya and Trina) had bugs. One of the 8 students (Robby) exhibited bug migration.

There are some rather obvious observations to make about these data. First, all 8 nonstandard students seem to be executing a standard procedure, but they often permute the order in which the standard procedure's actions are executed. On any given problem, the horizontal ordering convention, for instance, generates exactly the same *set* of actions as the standard procedure with standard execution. However, the *sequence* of actions generated by the horizontal convention is a permutation of the sequence generated by the standard execution of the standard procedure.

A second observation is that all the students (except Tanya) exhibited more than one execution strategy. Janine, for instance, used 3: a standard execution strategy, the horizontal execution strategy and the vertical execution strategy. Paul used 2: standard and horizontal. Sometimes these execution strategies were used in sequential runs. Paul, for instance, used the standard execution strategy for the first 9 problems, then a unique execution strategy, then the horizontal execution strategy for the last 3 problems. In other cases, there are correlations between features of the problem and the student's choice of execution strategy. For instance, Tanya used the ASD permutation when a borrow originated in the units column and the SDA permutation when the borrow originated in the tens column.

A third observation, which is perhaps not so obvious, is that the nonstandard execution strategies exhibited by most of the students (all except Angela, whose protocol is discussed later) happen to give the same answers as the standard execution strategy would. In particular, if the student's procedure were bug-free, then the student's answers would be correct (ignoring slips). This explains why the phenomena of nonstandard ordering had escaped notice in the earlier subtraction experiments—only the students' answers were known, and not the exact sequence of their writing actions. Thus, the experiment showed that the original Sierra models of subtraction procedures fit only 15 of the 26 students' *protocols* (i.e., only the standard students) even though they fit 25 of the 26 students' *answers* (i.e., all except Angela).

Given these observations, a simple hypothesis leaps to mind: all the students learned standard procedures, but 8 of the 26 are missing a few unimportant constraints on the ordering of actions. Perhaps they never encoded those unimportant ordering constraints, or perhaps they learned them but discovered later that it made no difference to the answer if they dropped them. In order to investigate this hypothesis further, a formal representation of knowledge was developed such that modifying "a few unimportant constraints" would yield exactly the behavior exhibited by the students. The next section presents the representation.

3. A CONCISE REPRESENTATION OF THE OBSERVED NONSTANDARDNESS

The knowledge representation selected here is based on the standard cognitive science concepts of goals, operators and control knowledge. Control knowledge is divided into two kinds: (1) goal selection information is used to choose a goal to attend to, called the current goal; (2) operator selection information is used to choose an operator appropriate for the current goal. Execution of the selected operator causes either a change in the state of the situation, such as writing a digit, or the creation of subgoals.

The formalisms used for these concepts are also standard ones. Operators and opeator selection information are represented as a goal-hierarchical production system, similar to the ones used by HPS (Anzai, 1978), Grapes (Anderson et al., 1984), and many other systems. Goal selection information is represented as a set of conditional preferences similar to the ones used by Soar (Laird et al., 1987), Prodigy (Carbonell, Knoblock, & Minton, in press; Minton, Carbonell, Etzioni, Knoblock, & Kuokka, 1987) and other systems. Except for the preferences, this representation of knowledge is very similar to the one used by Sierra.

The part of working memory that contains goals is called the goal store. One of the goals is marked as the current goal. Each production has exactly

one goal test as part of its condition; productions are considered for execution only if their goal tests match the current goal. A production may have one or more goal specifications on its action side; the goals that they create are added to the goal store. For instance, consider the following production:

If the goal is (ProcessColumn i), and
  the top digit of i is less than the bottom digit of i,
then
  set the goals (BorrowFrom i + 1), (Add10 i), (ColumnDifference i).

If the current goal is (*ProcessColumn 2*), which means to process the tens column (columns are numbered from right to left), then this production is considered for execution. Suppose that the problem is 234 − 190. Then the test that is the second element of the left side will succeed, because $3 < 9$, so the production is executed.[7] Three goals are created by the right side and added to the goal store.

The goal store contains only pending goals. Whenever a production, which matches a goal is executed, the goal is removed from the goal store. Thus, if the store contains

(ProcessColumn 2)
(ProcessColumn 3)

just before the production above fires, then it will contain

(BorrowFrom 3)
(Add10 2)
(ColumnDifference 2)
(ProcessColumn 3)

just after the production fires.[8]

Some goals are primitive, in that whenever they are selected as the current goal, the next cycle of the production system results in a change to the state of the problem rather than firing a production rule. For instance, (*Add10 2*) causes 10 to be added to the top digit in the tens column, and (*ColumnDifference 2*) causes the difference between the top and bottom digits in the tens column to be written in the answer row as the answer for that column. This completes the description of the notation used for proce-

[7] A technical detail: the conflict-resolution strategy is specificity—if the set of working-memory items matched by production A is a subset of those matched by production B, then A is discarded.

[8] The computer implementation does not physically remove goals from the goal store, but only marks them as "not pending." This makes it possible to back up to the goal if a failure occurs and choose a different production for achieving it. Since such backing up did not seem to occur in the data, this detail has been suppressed in order to simplify the exposition, and the contents of the goal store are equated with the set of pending goals.

TABLE 3
A Correct Subtraction Procedure

1. If the goal is (Subtract)
   then for each column i, set the goal (ProcessColumn i).
2. If the goal is (ProcessColumn i), and (Top i)<(Bottom i),
   then set the goals (BorrowFrom i+1), (Add10 i), (ColumnDifference i).
3. If the goal is (ProcessColumn i),
   then set the goal (ColumnDifference i).
4. If the goal is (BorrowFrom i), and (Top i)=0,
   then set the goals (BorrowFrom i+1), (Add10 i), (BorrowFrom i).
5. If the goal is (BorrowFrom i),
   then set the goals (Slash i), (Decrement i).

dures. As an illustration, Table 3 presents a correct subtraction procedure expressed in this production system notation.

Sierra did not need goal preferences, because it used its goal store as a last-in-first-out stack. The most recently added goal was considered to be the current goal. In the new model, the goal store is considered to be an unordered set, and goal preferences are used to select a goal from it.

Four notations for preferences were tried before finding one that yields simple, elegant student models.[9] The one settled upon represents a goal selection strategy as a set of preferences of the form,

If <condition>
then prefer <goal> over <goal>
else prefer <goal> over <goal>

[9] The following is a brief description of those notations and their inadequacies. (1) A goal selection strategy is represented as a set of preferences of the form "<goal type A>is better than<goal type B>." When the 8 students are fit by a strategy expressed this way, 33% of the interpreter cycles results in a multiple goal impasse (VanLehn & Ball, 1987). Essentially, this representation could not represent the fact that most students change ordering conventions during the course of the testing session. (2) A goal selection strategy is represented as a "big switch" among several sets of preferences of the form described in (1), above. Although this could represent the observed strategies, the sets used for any given student tended to overlap considerably, indicating that most of their preferences were constant, and only a few varied (VanLehn & Ball, 1987). (3) A goal selection strategy is represented as a discrimination net, whose leaves are goal types (VanLehn & Garlick, 1987). This functions identically to a set of rules of the form "If<condition>then<goal type>". This representation could not express some of the observed strategies (Kowalski & VanLehn, 1988). (4) A goal selection strategy is represented by a set of preferences of the form, "If<condition>then prefer<goal A>over <goal B>," where the goal patterns must have an explicit goal name in them (Kowalski & VanLehn, 1988). This representation was fine in every respect, but not as concise as the chosen representation, which allows variables for goal names and has "else" clauses as well as "then" clauses.

TABLE 4
Preferences for a Correct Subtraction Procedure

1.	For X in the set BorrowingGoals, prefer (X i) over (ProcessColumn i).
2.	For X in the set BorrowingGoals, prefer (X i) over (ColumnDifference i).
3.	For X in {BorrowFrom, Slash, Decrement}, prefer (Add10 i) over (X i).
4.	Prefer (Slash i) over (Decrement i).
5.	Prefer (ProcessColumn i) over (ProcessColumn i+j).
6.	For X in the set BorrowingGoals, prefer (X i) over (ProcessColumn j).
7.	For X in the set BorrowingGoals, prefer (X i) over (ColumnDifference j).
8.	Prefer (ColumnDifference i) over (ProcessColumn j).
9.	For X, Y in the set BorrowingGoals, prefer (X i+j) over (Y i).

A <condition> is like a condition in a production rule in that it can test working memory and/or the state of the external world. A <goal> is a pattern that matches items in the goal store. The goal patterns can mention constants, variables or constrained variables. For instance, the preference

If (Top i)>9
then prefer (ProcessColumn j) over (ColumnDifference i)
else prefer (ColumnDiffernce i) over (ProcessColumn j)

means: if the column difference in column *i* will be a "hard" one to calculate/recall, because the top digit is 10 or more, then prefer starting to process another column rather than taking the column difference in *i*. On the other hand, if the column difference will be a normal one, then it is preferable to take care of it before moving on to another column. That is, this preference prefers to procrastinate taking the difference in columns whose top digits are 10 or more.

For convenience in exposition, parts of a preference are sometimes dropped. For instance, if a goal preference always holds, then *Prefer <goal> over <goal>* is written.

The execution cycle is to (1) select a goal, then (2) select a production, then (3) execute the production. Preferences are used during Step 1. Goal selection begins by gathering the set of preferences whose conditions are true at this time. Next, it finds a subset of the goal store that is maximal, according to the true preferences. A goal is maximal if there is no goal that is preferred over it. Often, there is just one maximal goal, so it is chosen as the next current goal. If there are no maximal goals (e.g., because the true preferences have a cycle in them), then a "no goal" impasse occurs. If there are two or more maximal goals, then a "multiple goal" impasse occurs. This use of preferences is quite similar to the way preferences are used in Soar (Laird et al. 1987).

Table 4 shows the "standard" preference set, which will generate a standard, depth-first, left-to-right execution of the procedure of Table 3. Although there are many different sets of preferences that will yield standard

executions, this one is thought to correspond most closely to the preference sets acquired by the students. As will be seen later, all the students' strategies can be formed by making small deletions or additions to this set of preferences. Hence, it is worth a moment to examine it carefully.

The first three preferences concern satisfaction of a precondition of the *ProcessColumn* goal. The first step in processing a column is to test whether it needs a borrow. In order for the test to deliver a correct result, all pending actions, which could modify the column, must be executed before the test. The set *BorrowingGoals* is {*BorrowFrom, Add10, Slash, Decrement*}, so Preference 1 guarantees that such a goal will be selected before *ProcessColumn* when both refer to the same column. Similarly, Preference 2 concerns satisfaction of a precondition of *ColumnDifference,* and Preference 3 concerns satisfaction of a precondition of decrementing zero.

Preference 4 is a universal convention in handwriting: cross out the old stuff before writing the new stuff in.

Preference 5 causes the columns to be processed from right to left. This preference is mandatory because it concerns satisfaction of the preconditions of the test embedded in the *ProcessColumn* goal. In fact, all of the first 5 constraints are mandatory (although one could quibble about Preference 4) in that violating any one of them will result in incorrect answers being generated from a correct production system.

The remaining 4 preferences are mere conventions. Deleting or modifying them will not harm the correctness of the answers. Preferences 6, 7, and 8 rank the goals by type:

BorrowingGoals > ColumnDifference > ProcessColumn

Notice that the relative location of the goals does not matter to these preferences, whereas it is crucial in the mandatory preferences. Preference 9 causes the borrowing goals to be executed in left-to-right order. This is the most commonly taught ordering of the borrowing actions.

It should be noted that the distinction between mandatory and conventional preferences is important, particularly in the discussion of how preference sets are learned.

3.1. The 8 Student Models

For each of the 8 nonstandard students, Appendix 1 presents four items: the student's protocol, an idealized version of the protocol, a production system, and a preference set. The latter two constitute the model of the student here. When the student model is executed, it generates the idealized protocol exactly. So the discrepancies between the model and the data are captured in the differences between the idealized protocols and the real ones. Each such difference is highlighted and discussed in Appendix 1. In order to give an overview, Table 5 categorizes the discrepancies and discusses each briefly.

TABLE 5
Discrepancies between Idealized and Actual Procotols

Number of Occurrences	Description
22	Facts errors. Column differences or decrements are misremembered, e.g., 12−9=4.
10	Missing pieces of borrows. Sometimes there is a slash without a decrement, a decrement without a slash, a missing addition of ten that is detected later, etc.
9	Redoing a column difference. The students rewrite the answer to a column, possibly because they consider it illegible or as a result of checking the column subtraction.
9	Leading zero suppression. The student models will write leading zeros for problems like 303-279, whereas some students do not.
5	Extra pieces of borrows. Extra scratch marks are made for no apparent reason. In some cases, they seem to be ignored later.
4	Missing borrows. The student inexplicably fails to borrow for a column where she or he ordinarily would.
2	Extra borrows. The student inexplicably borrows for a column that she or he would ordinarily not borrow for.
2	Blanks treated as ones. In a column with a blank subtrahend, the student's answer was one less than it should be.
2	Missing answer. One column is missing an answer.
1	Impasses and repairs. The interpreter used with the student models did not have a complete set of repairs in it, so it was not able to generate exactly the repairs that some students made to their impasses.
1	Quit early. Hilda forgot to answer the last two columns of her last problem, even though she had borrowed from them.
1	Floundering. Pete made three slips during his solution of problem 8, and got so frustrated that he quit, copied the problem over (as problem 9) and tried again. Problem 8's solution was omitted from the idealized protocol.
1	Slash after decrement. One student reversed the usual order of slashing and decrementing.
1	Use Slash for Add10. One student used a slash where she would normally use an Add10, and later changed it.

All of them are either well explained by current theory (e.g., the cases of repairs to impasses) or clearly in the province of some other theory, such as Norman's (1981) theory of action slips or Siegler's (1982) theory of arithmetic facts.

The student models in Appendix 1 exhibit several important features. First, all the production systems represent standard procedures that have occurred many times before in analyses of the bug data. The second feature exhibited by the student models is that all of the preference sets are quite

similar to the standard preference set of Table 4. For instance, 3 students (Janine, Pete and Tanya) exhibit a horizontal execution strategy, wherein they perform all the borrowing actions required by the problem before filling in any of the problem's answers. The horizontal execution strategy can be formed by reversing the standard preference,

8. Prefer (ColumnDifference i) over (ProcessColumn j).

to become

8. Prefer (ProcessColumn j) over (ColumnDifference i).

and leaving the remainder of the standard preferences intact.

Sometimes, deleting or weakening a standard preference creates an execution strategy that exhibits just the right kind of nondeterminism. For instance, in place of the standard preference,

9. Prefer (X i+1) over (Y i), for X,Y in the set BorrowingGoals.

which says to perform the actions of borrowing from left to right, Hilda has the following preferences:

9a. Prefer (Slash i+1) over (Add10 j).
9b. Prefer (BorrowFrom i+1) over (Add10 j)

These preferences leave three possible permutations of the borrowing actions, and Hilda exhibits them all. The rest of Hilda's preference set is identical to the standard preference set.

Sometimes a pattern in the student's alternation among strategies was seen. Although the preferences could be written, like Hilda's, so that the choice was undetermined, preferences were written that captured as much of the variation as could be seen. For instance, Robby's column-processing strategy is represented by replacing

8. Prefer (ColumnDifference i) over (ProcessColumn j).

with

8. If the top digit of column i is greater than 10 and the current problem is either problem 6 or 7, then prefer (ProcessColumn j) over (ColumnDifference i) else prefer (ColumnDifference i) over (ProcessColumn j).

It is not known why Robby uses a nonstandard ordering on Problems 6 and 7 only, so the problem numbers in the preference's condition were deliberately mentioned. It is clear, however, that the columns whose answers are delayed are the ones where the column difference would be "hard" because the top digit is 10 or more. Because the condition is ad hoc, the preference causes an exact match to Robby's protocol, and thus serves as an accurate reduction of the data, albeit only a partial explanation of it. An alternative formulation would be to delete Preference 8 entirely. This would not cap-

ture what appears to be a valid (although partial) explanation of why Robby delays the answering of some columns. In general, whenever such explanations could be found, they were written into the preferences. When they could not be written in, the preference set was left undetermined. Usually patterns (partial explanations) were seen in the choice of column-processing strategies, but often patterns could not be seen in the choice of borrowing permutations.

3.2. Discussion

Overall, the most important observation to make is that all the students had preference sets that were nearly identical to the standard set. Also, the instability in any given student's apparent goal selection strategy can be simply and succinctly represented by conditionalizing or dropping a very few preferences. In short, a representation for the data has been found that reduces a large amount of protocol variance to a small amount of preference set variance.[10]

A second finding is that the differences between the students' preferences and the standard preferences tended to involve only conventional preferences. The preference sets of all the students except Angela include all the mandatory standard preferences. This means that all the students except Angela have "correct" preference sets, in that they will produce correct answers when used with a correct production set. (Angela has a conditionalized version of standard Preference 5, a mandatory preference, which causes her to give incorrect answers on 4 of the 12 problems she solves.)

As the introduction mentioned, there is an informal explanation that leaps to mind when one considers the nonstandard students' behavior. The explanation is that these 8 students lack a few "unimportant" standard constraints because either they overlooked them as they learned the procedure, or they learned them but later discovered that they were unimportant, so they started ignoring or modifying them. So far, this informal explanation has been borne out by formal modelling. A computationally sufficient representation exists such that the behavior of the nonstandard students is accurately reproduced by deleting or modifying a few "unimportant" parts of it. The "unimportant" parts turn out to be conventional standard preferences.

4. EXPLANATIONS BASED ON LIFO ARCHITECTURES

The preceding section presents successful models that violate the LIFO restriction. Before abandoning such a venerable restriction, it should be

[10] However, some of the models are nondeterministic, because their preference sets underconstrain some goal selection events. Thus, the models do not explain all the variance in the data. For discussion of the tradeoffs involved in this type of data analysis, see VanLehn and Ball, 1987.

seen how well the data can be modelled if the LIFO restriction is maintained. There are two basic approaches to modelling these data within the confines of a LIFO architecture. Both add complexity to the account based upon a non-LIFO interpretation. The first approach adds complexity to the representations used for the student's procedure. The second approach retains the simple procedural representations used with the non-LIFO model, and adds complexity to the execution of these procedures.

4.1. Big Switch Representations of Subtraction Procedures

Any collection of observed strategies can be represented by a collection of procedures, one per strategy, with a "big switch" that selects among them. For instance, to model Janine's behavior, one can write LIFO procedures for each of three strategies—horizontal, vertical, and standard—then write productions that act as a "big switch" to select among them. Appendix 2 presents hierarchical production systems for each of the 8 nonstandard subjects that suffice to model their behavior within the confines of a LIFO architecture. The LIFO models for 4 of the 8 subjects (Angela, Hilda, Paul, and Tanya) turn out to be quite simple and not much different from the production systems given for them in Appendix 1. This simplicity is because the nonstandardness in their behavior is due mostly to local permutations in the order of their borrowing actions (e.g., from the standard SDA- to SA-D), and that kind of variance doesn't require a big switch to represent it, even in a LIFO architecture. However, the other 4 students (Janine, Pete, Robby, and Trina) do require big switch representations.

If parsimony of knowledge representations were the only criteria, the LIFO hypothesis would be rejected. But parsimony is an argument of last resort, to be used only when empirical evidence fails to discriminate the hypotheses. The rest of this section will develop some empirical tests for differentiating the LIFO hypothesis from the non-LIFO hypothesis. Although the tests are not conclusive, what little evidence there is favors the non-LIFO hypothesis.

4.1.1. Blends of Strategies. When an iterative or recursive procedure is in the middle of its execution, its design causes certain properties to be true of the current state. For instance, when the vertical strategy is running, all the columns to the right of the current column will be completely finished, while all the columns to the left will be totally free of scratch marks. Different properties are true for the standard execution strategy and the horizontal one. When the intermediate state properties of two procedures are different, it is difficult to abort one procedure in the middle of a problem and start the other. It may take rather sophisticated problem solving to convert the state left by the first procedure into one that is suitable for the execution of the second procedure. Thus, if the subjects did indeed have a big switch procedure for subtraction, one would not expect to see them "throw the switch"

in the middle of problems. That would force them to engage in difficult problem solving for no particular reason.

On the other hand, if subjects have a non-LIFO procedure, then switching from, say, the standard to the horizontal strategy, amounts to reversing the direction of a single preference. This is done in the middle of solving a problem just as easily as it is done in between problems. The resulting behavior would appear to the observer as a "blend" of the two strategies. Thus, if blends of strategies occur, it is more plausible that subjects have a non-LIFO representation than a big switch representation. If blends do not occur, then they could equally well have either representation.

Of the 4 subjects whose LIFO representations require big switches, Janine displays clear instances of strategy blends in her solutions to Problems 10 and 13. (The LIFO procedure given in Appendix 2 does not successfully model her solutions to 10 and 13.) Pete's solution to Problem 10 may also be a blend of strategies, but that problem is difficult to analyze because Pete seems to be in the middle of acquiring a new strategy at that point. (The LIFO procedure in Appendix 2 does not model Pete's solution to Problem 10.) The other two subjects (Robby and Trina) do not switch strategies in the middle of problems. Thus, there is a little evidence that Janine has a non-LIFO representation, but the data are silent in the case of the other 3 subjects.

4.1.2. Bugs. Under both the LIFO and non-LIFO hypotheses, bugs are represented by perturbed production systems; the preferences, if any, are left alone. Under the non-LIFO hypothesis, the subjects have just one production system. Their preferences cause them to display different column-processing strategies on different problems. Thus, if the subject has a bug, it must appear regardless of the column-processing strategy used by the subject, since the bug is represented in the subject's production system, and that does not shift over the course of the testing session. On the other hand, under the LIFO hypotheses, the "big switch" subjects have different productions for each of their strategies. They could have acquired the bug in the context of only one of their strategies. Thus, if subjects display different bugs depending upon the strategy they are using, or display bugs only on some of their stategies and correct performances on others, then evidence for the LIFO representation exists. If the subjects show the same bugs under all strategies, then both hypotheses would be consistent with the data.

Unfortunately, the data are silent. Of the 4 students who switch column-processing conventions, 2 (Janine and Pete) are totally bug-free. Robby has a bug that shows up when he is doing the standard column-processing convention, but he does his nonstandard convention only on 2 problems where the conditions for the bug's occurence are absent. Trina has a bug, and it shows up under both her standard and nonstandard execution strategies.

However, the bug occurs in a rule that is shared by those two conventions regardless of which hypothesis is used to represent her procedural knowledge. In short, both the LIFO and non-LIFO hypotheses are consistent with the bug-versus-strategy data.

4.1.3. Summary of "Big Switch" Arguments. An attempt was made to differentiate between the two hypotheses by looking for blends of strategies, and for bugs that occurred only in some of the subjects' strategies. The strategic blend data had the ability to support the non-LIFO hypothesis, and they did, but only weakly. The bug-versus-strategy data had the ability to refute the non-LIFO hypothesis, but they did not. So these arguments provide a tiny bit of support for the non-LIFO hypothesis. However, the main significance of the arguments is to show that the two hypotheses actually are discriminable, even though these data do not do a good job of it.

4.2. Are Subtraction Goals Special?

The preceding section made the assumption that the goals generated in the course of solving subtraction problems have the same status as the goals one generates while solving puzzles, college physics problems, and all the other types of problems used in testing cognitive architectures. It was shown that this assumption, plus the hypothesis of a LIFO architecture, implied that some subjects had unparsimonious "big switch" procedures. This section examines the contrary position. It assumes that subjects have parsimonious representations for their procedures but the goals generated during the course of solving subtraction problems are atypical in that they are not subject to the LIFO restriction even though the architecture is indeed a LIFO architecture. Two different versions of this assumption will be examined.

4.2.1. Is Subtraction Declarative Knowledge? One way to release subtraction goals from the LIFO constraint is to assume that the subjects' procedures are declarative knowledge. As such, they do not run directly on the architecture, but instead are interpreted by a procedure that is running directly on the architecture. This interpreter procedure need not enforce a LIFO restriction even if the architecture does.

There are several pieces of evidence against the hypothesis that these students are interpreting their procedures (as opposed to executing them directly). First, the subjects in this experiment were nearing the end of the third grade in a school where subtraction instruction begins in second grade. They had between one and a half and two years of intermittent instruction on the subtraction procedure. Anderson, based upon experiments where students learn geometry, estimates that compiling from declarative to procedural knowledge takes only a short period of time, not years (Anderson, 1982, 1987). Surely, the students in this experiment had compiled their

knowledge before they reached this experiment. Second, verbal rehearsal of the procedure is usually common when a student is interpreting a declaratively encoded procedure (Anderson, 1983), but such rehearsal was absent in this experiment. Third, all the subjects in this experiment were relatively rapid, smooth solvers who seemed to pause only when they could not recall a number fact or when they detected a mistake that needed correction. Speed and lack of apparent effort at recall are hallmarks of a compiled skill. On these grounds, it appears that the students in this experiment were executing compiled procedures, rather than interpreting declarative knowledge.

These pieces of evidence bear only on Anderson's theory of the procedural-declarative distinction. However, regardless of what theory of the procedural-declarative distinction turns out to be correct, one will have to assume that the category one assigns to subtraction should also be assigned to the knowledge for Tower of Hanoi, geometry theorem proving, physics, and other tasks of the problem-solving literature, because the performances of the subjects here are not qualitatively different from the performances of subjects in these classic task domains.[11] Taking this position seriously would mean that protocol data from the classic experiments, as well as this experiment, are simply irrelevant to determining the underlying cognitive architecture. Such data bear only on the structure of the interpreters, and there might be arbitrarily many of them. Clearly, there are significant methodological advantages to the traditional assumption that there is just one cognitive architecture and that the procedures for solving the classic tasks, and subtraction as well, are executed directly by this architecture.

Although the specific version of the interpretation hypothesis that derives from Anderson's theory can be rejected empirically, the general version can be rejected only on methodological grounds, if at all. Nonetheless, this explanation of the data from this experiment clearly seems to have more problems than the others.

4.2.2. Are Subtraction Goals Stored Externally? Goals sometimes are forgotten and have to be reconstructed. For instance, Anderson (1983) says that he often fails to retrieve goals when solving the Tower of Hanoi and must reconstruct them from the current puzzle state by using Simon's (1975) perceptual strategy. Larkin (1989) and VanLehn and Ball (in press) have proposed that the problem state is routinely used as a sort of external storage

[11] In particular, suppose we assume (as programmers often do), that a program that is running on an interpreter is ten times slower than it would be if it could run directly on the architecture that the interpreter runs on. If this 10:1 ratio holds for the general cognitive system, and our subjects are running an interpreter while the subjects in the classic experiments are not, then our subjects should be ten times slower than the subjects in the classic experiments. But both sets of subjects seem to be working about as fast, so this conjunction of assumptions is untenable.

for goals. Pylyshyn (in press) comments that it would be just as revealing and more traditional to say that such goals are stored in long-term memory but the external problem state serves as a visual cue for retrieving them. Regardless of the mechanisms involved, it is clear to all involved that goals are not necessarily stored in working memory—sometimes one can get them from the external world.

If one views the LIFO restriction as deriving from some property of the part of working memory which stores goals, then it follows that goals that are "stored" externally are not subject to the LIFO constraint. So it seems to follow that parsimonious, standard subtraction procedures can be used even in a LIFO architecture. But if this line of reasoning is examined closely, it falls apart.

Consider the subtraction procedure shown in Table 3. If the goals it creates are stored in a LIFO memory, then it generates a standard execution strategy when executed. However, suppose the *ColumnDifference* goals are forgotten but later reconstructed from the external problem state just after the goal *Subtract* has completed. The resulting behavior is exactly the horizontal strategy. So it seems at first that a non-LIFO architecture can explain the data. But the price is assuming that all the *ColumnDifference* goals are "forgotten." This occurs routinely, so it must be a part of the person's knowledge—not really a case of working-memory failure. Obviously, the person has decided to delay the processing of *ColumnDifference* goals until after the problem's borrowing is done. That is, the person has adopted the preference "For X in the set BorrowingGoals, prefer (X i) over (ColumnDifference j)." In a LIFO architecture, this preference means that *ColumnDifference* goals should be dropped from the LIFO goal memory (or ignored) and reconstructed later. But if the architecture can do this on a routine basis, then in what sense is it still a LIFO architecture?

Almost all cognitive modelling has been conducted under the useful idealization that the goal store does not forget goals. The LIFO restriction has always been interpreted as a characterization of this idealized goal store. It is still thought to be useful to have an idealized goal store whose properties are spelled out carefully. The business about reconstructing goals from external and/or long-term memory is an "implementation" issue, albeit an important one. If one goes this route, then putting a LIFO restriction on the idealized goal store implies that some students have unparsimonious, big switch procedures, which leads to the problems discussed earlier.

On the other hand, if one chooses to dispense with the idealization and model goal storage at the level of retrieval/reconstruction from external/long-term memory, then it is patently clear that the goals are not subject to a LIFO restriction, for no one (it is assumed) would want to argue that the external world and/or long-term memory impose some kind of LIFO restriction on recall.

So, the availability of a non-LIFO place to store goals does not really help the LIFO theory of problem solving. To be consistent, either the theory has to recognize this non-LIFO storage facility as a first class goal store, in which case there is no sensible LIFO restriction, or the theory has to stick with an idealized LIFO goal store, in which case it is stuck with big switch procedures. The alternative favored here, of course, is to work with an idealized goal store that is non-LIFO. This keeps the theory of problem solving from becoming enmeshed in the "implementation" details of the goal memory, and it also avoids burdening the learning component of the theory with having to acquire complex procedural structures.

5. CONCLUSIONS

This article makes two main points. The first is that there is a greal deal of variance in the execution strategies of some subtraction students, both between subjects and within the individual subject's performance. The second point is that this variance can be simply represented as minor perturbations of standard procedures, provided that goal selection information is represented explicitly and the architecture does not enforce a LIFO restriction on goals. To these rather substantially supported points, a rather extended examination of how a LIFO architecture could model these data was added. The primary conclusion is that it could, but only at the price of assuming that some of the students had somehow acquired complex "big switch" procedures. Also devised were some empirical tests to determine whether their procedures were big switches or not; unfortunately, the data were silent.

Early accounts of problem solving did not impose a LIFO restriction. Miller, Galanter, and Pribram (1960) explicitly differentiate inflexible plans from flexible plans whose goals can be rearranged to suit the occasion. Although their examples indicate a non-LIFO sort of processing for flexible plans, they are not explicit on the point. Newell and Simon (1972) are explicit (to put it mildly!), and they sometimes use LIFO models and sometimes use non-LIFO models. In their famous chapter 14, wherein they present their theory of human problem solving, no LIFO restriction is placed on the goal store.

LIFO restrictions became common at about the same time that cognitive architectures began to include automatic learning mechanisms, such as production compounding (Anderson, 1982) and chunking (Rosenbloom, 1983). In order to yield plausible learning, one wants to compound two productions only when those productions serve the same goal. For instance, if a subtraction problem were immediately followed by an addition problem, then one would want to avoid forming a compound from the last production executed during a substraction problem's solution and the first production executed for the addition problem, even though those productions may have been executed consecutively. Since automatic learning mechanisms are part of

the architecture, it makes sense to make the LIFO restriction be a part of the architecture. This makes it simpler to place the same-goal constraint on automatic learning.

Once the question is posed, it is clear that a non-LIFO architecture does not really make automatic learning any more complicated. In order to impose the same-goal constraint that seems critical for plausible automatic learning, the architecture need only maintain explicit information about the goal–subgoal relationship of the goals it creates. Because this information is often needed anyway, relaxing the LIFO restriction adds no new burdens to the architecture.

As far as can be seen, there is no good reason for believing the LIFO constraint, and several weak reasons for disbelieving it. So far, there is nothing in the data to contradict the intuition that if a person can recall a goal, then he or she can act upon it, even if that goal is not the most recently created pending goal.

■ Original Submission Date: November 22, 1988.

REFERENCES

Anderson, J.R. (1982). Acquisition of cognitive skill. *Psychological Review, 89,* 369–406.

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

Anderson, J.R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review, 94,* 192–210.

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

Anderson, J.R., & Thompson, R. (1986, June). Use of analogy in a production system architecture. Paper presented at the Illinois Workshop on Similarity and Analogy, Urbana-Champaign.

Anzai, Y. (1978). Learning strategies by computer. In *Proceedings of the Second Conference on Computational Studies of Intelligence.* Toronto: Canadian Society for Computational Studies of Intelligence.

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

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.

Burton, R.B. (1982). Diagnosing bugs in a simple procedural skill. In D.H. Sleeman & J.S. Brown (Eds.), *Intelligent tutoring systems* (pp. 157–183). New York: Academic.

Carbonell, J.G., Knoblock, C., & Minton, S. (in press). PRODIGY: An integrated architecture for planning and learning. In K. VanLehn (Ed.), *Architectures for intelligence.* Hillsdale, NJ: Erlbaum.

Ernst, G.W., & Newell, A. (1969). *GPS: A case study in generality and problem solving.* New York: Academic.

Kowalski, B., & VanLehn, K. (1988). *Inducing subject models from protocol data.* In V. Patel (Ed.) *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum.

Laird, J.E., Newell, A., & Rosenbloom, P.S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33,* 1–64.

Laird, J.E., Rosenbloom, P.S., & Newell, A. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning, 1,* 11–46.

Larkin, J. (1989). Display-based problem solving. In D. Klahr & K. Kotovsky (Eds.), *Complex information processing: The impact of Herbert A. Simon.* Hillsdale, NJ: Erlbaum.

Miller, G.A., Galanter, E., & Pribram, K.H. (1960). *Plans and the structure of behavior.* New York: Holt, Rinehart & Winston.

Minton, S., Carbonell, J.G., Etzioni, O., Knoblock, C., & Kuokka, D.R. (1987). Acquiring effective search control rules: Explanation-based learning in the Prodigy system. In P. Langley (Ed.), *Proceeedings of the Fourth International Workshop on Machine Learning.* Los Altos, CA: Morgan Kaufmann.

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.

Pylyshyn, Z. (in press). Architectures and strong equivalence: Commentary. In K. VanLehn (Ed.), *Architectures for intelligence.* Hillsdale, NJ: Erlbaum.

Resnick, L. (1982). Syntax and semantics in learning to subtract. In T. Carpeter, J. Moser, & T. Romberg (Eds.), *Addition and subtraction: A developmental perspective.* Hillsdale, NJ: Erlbaum.

Resnick, L.B., & Omanson, S.F. (1987). Learning to understand arithmetic. In R. Glaser (Ed.), *Advances in Instructional Psychology.* Hillsdale, NJ: Erlbaum.

Rosenbloom, P.S. (1983). The chunking of goal hierarchies: A model of practice and stimulus-response compatibility. Doctoral dissertation, Carnegie-Mellon University, Pittsburgh, PA. (CMU Computer Science Technical Report #83-148)

Siegler, R.S. (1987). Strategy choices in subtraction. In J. Sloboda & D. Rogers (Eds.), *Cognitive Processes in Mathematics.* Oxford, England: Oxford University Press.

Simon, H.A. (1975). The functional equivalence of problem solving skills. *Cognitive Psychology, 7,* 268–288.

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). Human skill acquisition: Theory, model and psychological validation. In M.R. Genesereth (Ed.), *Proceedings of AAAI–83* (pp. 420–423). Los Altos, CA: Morgan Kaufmann.

VanLehn, K. (1983b). The representation of procedures in repair theory. In H.P. Ginsberg (Ed.), *The development of mathematical thinking.* Hillsdale, NJ: Erlbaum.

VanLehn, K. (1986). Arithmetic procedures are induced from examples. In J. Hiebert (Ed.), *Conceptual and procedural knowledge: The case of mathematics.* Hillsdale, NJ: Erlbaum.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence, 31,* 1–40.

VanLehn, K. (in press). *Mind bugs: The origins of procedural misconceptions.* Cambridge, MA: MIT Press.

VanLehn, K., & Ball, W. (1987). *Flexible execution of cognitive procedures* (Tech. Rep. No. PCG-5). Pittsburgh, PA: Department of Psychology, Carnegie-Mellon University.

VanLehn, K., & Ball, W. (in press). Teton: A large-grained architecture for studying learning. In K. VanLehn (Ed.), *Architectures for intelligence.* Hillsdale, NJ: Erlbaum.

VanLehn, K., & Garlick, S. (1987). Cirrus: An automated protocol analysis tool. In P. Langley (Ed.), *Proceedings of the Fourth Machine Learning Workshop* (pp. 205–217). Los Altos, CA: Morgan-Kaufmann.

Young, R.M., & O'Shea, T. (1981). Errors in children's subtraction. *Cognitive Science, 5,* 153–177.

APPENDIX 1: PROTOCOLS AND NON-LIFO MODELS

This appendix presents the protocols of each of the eight nonstandard students and our models of them. The syntax of the models has been explained in the text. The protocols use an abbreviated notation. Each action is indicated by a letter and a number. The letter stands for the type of the primitive goal:

A Add10
D Decrement
- ColumnDifference
S Slash
N WriteNine

The number stands for the column that the action was executed in. Thus, A2 means (Add10 2), the addition of ten to the top digit in the tens column.

In order to capture some of the variability in the protocols, ad hoc predicates on the problem states are used. The most common one is (*Problem x*), which is true whenever the current problem's number is contained in the set that is its argument. Thus, (*Problem* {*3, 7*}) is true during problems 3 and 7.

Angela

Angela always uses a standard, correct procedure. On eight problems, she uses the standard execution strategy. In four problems (5, 9, 10, and 11), she permutes the order of the ProcessColumn goals. She seems to delay processing a column if that column seems hard, for instance, because it requires borrowing from zero. We represent this by making the standard preference 5 conditional on the perceived difficulty of the problem. The predicate (*ColumnSeemsHard i*) has an ad hoc definition, since we do not have enough data to fully understand Angela's concept of column difficult.

Protocols

1. 562 − 3 — S2 D2 A1 -1 -2 -3

2. 742 − 136 — S2 D2 A1 -1 -2 -3

3. 50 − 23 — S2 D2 A1 -1 -2

4. 8305 − 3 — S3 D3 A2 S2 D2 A1 -1 Carry2 -2 -3
   Ideal: -1 -2 -3 -4

   Angela mistakenly borrows in the units column, which causes the units answer to be 12, so she carries tne ten to the ten's column. When she

comes to taking the column difference in the tens column, she interprets the 9 with a one over it (from the carry) as 9-1, so she writes 8 in the answer. The idealized protocol omits this extra-borrow slip. Also, Angela omits answering the last column, which the idealized protocol does not.

5. 106 S3 D3 A2 -2 -1 -3
   − 70

6. 716 S2 D2 A1 -1 -2 -3
   − 598 Ideal: S2 D2 A1 -1 S3 D3 A2 -2 -3

Angela mistakenly omits the borrow for the tens column, doing $0-9=9$ instead. The idealized protocol rectifies this missing-borrow slip. Her answer in column one is off by one: $16-8=7$.

7. 1564 S2 D2 A1 -1 S3 D3 A2 -2 S4 D4 A3 -3 -4
   − 887

8. 6591 S2 D2 A1 -1 -2 S4 D4 A3 -3 -4
   −2697 Ideal: S2 D2 A1 -1 S3 D3 A2 -2 S4 D4 A3 -3 -4

Angela mistakenly omits the borrow in the tens column, doing $8-9=1$ instead.

9. 311 S3 D3 A2 -2 Carry3 -3 <cross out answer 3> S2 D2 A1 -1
   − 214 Ideal: -2 -3 S2 D2 A1 -1

Angela processes the tens column first, as she does in several other problems. For some reason, she borrows. Perhaps she sees that the units column will cause a decrement in the tens. If so, she forgets her discovery by the time she gets to the -2, and takes 1 from 11, gets ten, and carries into the hundreds. After processing the hundreds, she apparently decides this carry was wrong, and thus the answer to the hundreds should be zero instead, so she crosses out her answer in the hundreds column. The idealized protocol retains the column processing order, but omits her extra borrow and the trouble that it causes.

10. 102 S3 D3 A2 -2 S2 D2 A1 -1 -3
    − 39

11. 9007 -1 S4 D4 A3 -3 S3 D3 A2 -2 -4
    −6880

12. 702 S3 D3 A2 S2 D2 A1 -1 -2 -3
    − 108

Arithmetic slip in the units column: $12-8=5$.

Production System

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i)<(Bottom i),
then set the goals (BorrowFrom i+1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i)=0,
then set the goals (BorrowFrom i+1), (Add10 i), (BorrowFrom i).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

Goal Selection Strategy

1. Prefer (X i) over (ProcessColumn i), for X in the set BorrowingGoals.
2. Prefer (X i) over (ColumnDifference i), for X in BorrowingGoals.
3. Prefer (Add10 i) over (X i), for X in {BorrowFrom, Slash, Decrement}.
4. Prefer (Slash i) over (Decrement i).
5. If (ColumnSeemsHard i)
   then prefer (ProcessColumn i+j) over (ProcessColumn i)
   else prefer (ProcessColumn i) over (ProcessColumn i+j).
6. Prefer (X i) over (ProcessColumn j), for X in the set BorrowingGoals.
7. Prefer (X i) over (ColumnDifference j), for X in BorrowingGoals.
8. Prefer (ColumnDifference i) over (ProcessColumn j).
9. Prefer (X i+j) over (Y i) for X, Y in the set BorrowingGoals.

Hilda

Hilda has a standard bug, called Borrow-Across-Zero, that skips over zeros during borrowing. Hilda's execution strategy is standard, except that she exhibits three different permutations of the borrowing actions. Since we do not understand what determines her choice among them, we represent her execution strategy by weakening standard preference number 9 in such a way that the choice among the three permutations is undetermined.

Protocols

1. 647 -1 -2 -3
   − 45

2. 885 -1 -2 -3
   − 205

3. 83 S2 A1 D2 -1 -2
   − 44

4. 8305 -1 -2 -3 -4
   − 3

5. 50 S2 D2 A1 -1 -2
   − 23

6. 562 S2 A1 D2 -1 -2 -3
   − 3

7. 742 S2 A1 -1 D2 -2 -3
   − 136

8. 106 -1 S3 D3 A2 -2 -3
   − 70

9. 9007 -1 S4 A2 -2 D4
   −6880 Ideal: -1 S4 A2 -2 D4 S4 A3 -3 D4 -4

   Hilda reaches an impasse when she attempts the slash-decrement for the borrow in the hundreds column because the 9 has already been decremented. Her repair is to quit. The idealized protocol pretends that she had no impasse.

10. 4015 S2 D2 A1 -1 -2 S4 A3 D4 -3 -4
    − 607

11. 702 S2 S3 A1 D3 -1 -2 -3
    − 108 Ideal: S3 A1 D3 -1 -2 -3

    Hilda performs the slash to the top of the tens column before she notices that it is zero, and thus should be skipped over. The idealized protocol omits the S2. Also, Hilda makes a facts error in the units column, 12−8=5.

12. 2006 -1 <redo -1> S3 S4 A2 D4 -2
    − 42 Ideal: -1 S4 A2 D4 -2 -3 -4

    Hilda makes several slips, which the idealized protocol rectifies. She redoes her units column difference. She makes a slash in the hundreds column before noticing that its top digit is zero. She quits before answering the hundreds and thousands column.

Production System

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i)<(Bottom i),
then set the goals (BorrowFrom j), (Add10 i), (ColumnDifference i),
where j is the first column to the left of i with a non-zero top digit.

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

Goal Selection Strategy

1. Prefer (X i) over (ProcessColumn i), for X in the set BorrowingGoals.
2. Prefer (X i) over (ColumnDifference i), for X in BorrowingGoals.
3. Prefer (Add10 i) over (X i), for X in {BorrowFrom, Slash, Decrement}.
4. Prefer (Slash i) over (Decrement i).
5. Prefer (ProcessColumn i) over (ProcessColumn i+j).
6. Prefer (X i) over (ProcessColumn j), for X in the set BorrowingGoals.
7. Prefer (X i) over (ColumnDifference j), for X in the set {BorrowFrom, Slash, Add10}.
8. Prefer (ColumnDifference i) over (ProcessColumn j).

9a. Prefer (Slash i) over (Add10 j).

9b. Prefer (BorrowFrom i) over (Add10 j)

Janine

Janine uses a standard, correct production system. It has a borrow from zero routine that substitutes a *WriteNine* operation (abbreviated as "N' in the protocols) for the more common combination of adding ten and decrementing. This change necessitates a slight change to standard preference number 4. The major differences from the standard execution strategy occur because Janine uses three different conventions for processing columns. On four problems (1, 2, 4, and 8), she uses the standard execution strategy. On four problems (5, 6, 11 and 12), she uses a horizontal convention. On three problems (3, 7 and 9), she uses a vertical convention. On two problems (10 and 13), she uses a mixture of all three strategies. We currently do not understand the problem characteristics, if any, that cause her to choose one execution strategy over another, so a simple way to model Janine's behavior would be to delete standard preferences 6, 7 and 8, which establish the conventional ordering for ProcessColumn and Column difference with respect to the borrowing goals and each other. However, we believe that Janine actually has knowledge of the standard, horizontal and vertical conventions, so we prefer a more complicated model that uses (Problem x) to turn off preferences 6, 7 and 8 on some problems. Two new preferences (10 and 11) are added, and they are also conditional on the problem being processed. The conditions are written so as to put no constraints on the column processing order in problems 10 and 13, because we see no pattern to her behavior on those problems.

Protocols

1. 83 A1 S2 D2 -1 -2
   − 44

2. 50 A1 S2 D2 -1 -2
   − 23

3. 742 A1 -1 S2 D2 -2 -3
   − 136

4. 106 -1 A2 S3 D3 -2
   − 70 Ideal: -1 A2 S3 D3 -2 -3

   Janine suppresses the answer's leading zero; the idealized protocol does not.

5. 716 A1 S2 D2 A2 S3 D3 -1 -2 -3
   − 598

6. 1564 A1 S2 D2 A2 S3 D3 A3 S4 D4 -1 -2 -3
   − 887 Ideal: A1 S2 D2 A2 S3 D3 A3 S4 D4 -1 -2 -3 -4

   Janine suppresses the answer's leading zero.

7. 102 A1 -1 <redo -1> S2 N2 -2 S3 D3
   − 39 Ideal: A1 -1 S2 N2 -2 S3 D3 -3

   Janine redoes the units column's answers to correct the facts error 12−9=4, and she suppresses the answer's leading zero. The idealized protocol does neither of these.

8. 9007 -1 A2 S3 N3 S4 D4 -2 -3 -4
   −6880

9. 702 A1 -1 N2 -2 S3 D3 -3 <rewrite result of D3>
   − 108 Ideal: A1 -1 S2 N2 -2 S3 D3 -3

   Janine omits the slash in the tens column; she rewrites the 6 over the 7 after she has finished the problem.

10. 2006 -1 A2 -2 N3 S3 S4 D4 -3 -4
    − 42 Ideal: -1 A2 -2 S3 N3 S4 D4 -3 -4

    Janine inexplicably reverses the order of the Slash and the WriteNine in the hundreds column. The idealized protocol has them in their usual order.

11. 10012 A1 S2 D2 A2 S3 N3 S4 N4 S5 D5 -1 -2 -3 -4
    − 214 Ideal: A1 S2 D2 A2 S3 N3 S4 N4 S5 D5 -1 -2 -3 -4 -5

    Janine suppresses the answer's leading zero.

12. 8001 A1 S2 N2 S3 N3 S4 D4 -4 -3 -2 -1
    − 43

13. 401 A1 S2 N2 -1 -2 S3 D3 -3
    − 206

Production System

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i)<(Bottom i),
then set the goals (BorrowFrom i+1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i)=0,
then set the goals (BorrowFrom i+1), (Slash i), (WriteNine i).

If the goal is (BorrowFrom i),
the set the goals (Slash i), (Decrement i).

Goal Selection Strategy

1. Prefer (X i) over (ProcessColumn i), for X in the set BorrowingGoals.
2. Prefer (X i) over (ColumnDifference i), for X in BorrowingGoals.
3. Prefer (Add10 i) over (X i) for X in {BorrowFrom, Slash, Decrement}.
4a. Prefer (Slash i) over (Decrement i).
4b. Prefer (Slash i) over (WriteNine i).
5. Prefer (ProcessColumn i) over (ProcessColumn i+j).
6. If (Problem {1, 2, 4, 5, 6, 8, 11, 12}) then
   prefer (X i) over (ProcessColumn j), for X in the set BorrowingGoals.
7. If (Problem {1, 2, 4, 5, 6, 8, 11, 12}) then
   pefer (X i) over (ColumnDifference j), for X in BorrowingGoals.
8. If (Problem {5, 6, 11, 12})
   then prefer (ProcessColumn i) over (ColumnDifference j)
   else prefer (ColumnDifference j) over (ProcessColumn i).
9. Prefer (X i) over (Y i+1) for X, Y in the set BorrowingGoals.
10. If (Problem {12})
    then prefer (ColumnDifference i+j) over (ColumnDifference i)
    else prefer (ColumnDifference i) over (ColumnDifference i+j).
11. If (Problem {3, 7, 9})
    then prefer (X i) over (Y i+1), for any X, Y.

Paul

Paul has a standard, correct subtraction procedure, and uses the standard column processing order. He exhibits two different nonstandard permutations of the borrowing actions when he does a borrows from non-zero digits. He uses the standard permutation when he does borrowing from zero. In order to represent this, we use an ad hoc predicate, (*BFZinProgress*), which is true whenever a borrowing from zero is being performed.

Protocols

1.
```
    647   -1 -2 <redo -1> <redo -2> -3
  -  45   Ideal: -1 -2 -3
```

Paul rewrites his first two answers, apparently because they were illegible. The idealized protocol omits these rewrites.

2. 8305 -1 -2 -3 -4
   − 3

3. 885 -1 -2 -3
   − 205

4. 83 S2 A1 -1 D2 -2
   − 44

5. 50 S2 A1 -1 D2 -2
   − 23

6. 562 S2 A1 -1 D2 -2 -3
   − 3

7. 6591 S2 A1 -1 D2 S3 A2 D3 -2 S4 A3 -3 D4 -4
   −2697

8. 311 S2 A1 -1 D2 S3 A2 -2 D3 -3
   − 214

9. 1813 S2 A1 -1 D2 S3 A2 -2 D3 -3 -4
   − 215

10. 4015 S2 A1 -1 D2 -2 S4 A3 D4 -3 -4
    − 607

11. 10012 S2 A1 -1 D2 S5 D5 A4 S4 D4 A3 S3 D3 A2 -2 -3 -4
    − 214 Ideal: S2 A1 -1 D2 S5 D5 A4 S4 D4 A3 S3 D3 A2 -2 -3 -4 -5

Paul suppresses the answer's leading zero; the idealized protocol does not.

12. 8001 S4 D4 A3 S3 D3 A2 S2 D2 A1 -1 -2 -3 -4
    − 43

Production System

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i)<(Bottom i),
then set the goals (BorrowFrom i+1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i)=0,
then set the goals (BorrowFrom i+1), (Add10 i), (BorrowFrom i).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

Goal Selection Strategy

1. Prefer (X i) over (ProcessColumn i), for X in the set BorrowingGoals.
2. Prefer (X i) over (ColumnDifference i), for X in BorrowingGoals.
3. Prefer (Add10 i) over (X i), for X in{BorrowFrom, Slash, Decrement}.
4. Prefer (Slash i) over (Decrement i).
5. Prefer (ProcessColumn i) over (ProcessColumn i+j).
6. Prefer (X i) over (ProcessColumn j), for X in the set BorrowingGoals.
7. If (BFZinProgress)
   then prefer (X i) over (ColumnDifference j) for X in BorrowingGoals
   else prefer (X i) over (ColumnDifference j)
   for X in the set {BorrowFrom, Add10, Slash}.
8. Prefer (ColumnDifference i) over (ProcessColumn j).
9. If (BFZinProgress)
   then prefer (X i+j) over (Y i) for X, Y in the set BorrowingGoals
   else
   prefer (BorrowFrom i+1) over (Add10 i), and
   prefer (Slash i+1) over (Add10 i), and
   prefer (Add10 i) over (Decrement i+1).

Pete

Pete has a standard, correct production system. Pete seems to learn a new execution strategy somewhere in the vicinity of problem 10. On problems 1 through 9, he exhibits the same pattern of borrow permutations that Paul does; see the section on Paul for discussion. On problem 10, Pete exhibits a transitional strategy that is like the pattern for borrowing from zero, but it is applied to a problem that has no borrowing from zero. We do not model Pete's solution to this problem. On problems 11 through 13, Pete exhibits a horizontal execution strategy. Also, Pete exhibits two permutations of his borrowing actions during problems 11 through 13. For borrows into the leftmost column, he uses the order Slash-Decrement-Add10. For other columns, he uses Slash-Add10-Decrement. To represent this, we use the ad hoc predicate (*Penultimate i*), which is true if i is the next to last column in the problem.

Protocols

1. 647 -1 -2 -3 <redo -2>
   − 45 Ideal: -1 -2 -3

   Pete rewrites his answer to the tens column. The idealized protocol omits this.

2. 885 -1 -2 -3
   − 205

3. 83 S2 A1 D2 -1 -2
   − 44

4. 8305 -1 -2 -3 -4
   − 3

5. 50 S2 A1 -1 D2 -2
   − 23

   Facts error in column 1: $10-3=6$.

6. 562 S2 A1 D2 -1 -2 -3
   − 3

7. 742 -1 -2 -3
   − 136

   Pete fails to notice a borrow is necessary in the units column. Since we do not know what his scratch marks would be in this case, the idealized protocol pretends the problem is $742-131$, which requires no borrowing.

8. 106 S3 A2 D2 S2 A1
   − 70

   Pete makes several slips on this problem, and ultimately gives up, copies the problem (which appears as 9 below) and tries again. His slips are borrowing for the units column, leaving out the decrement in the hundreds column, and switching the slash and the decrement in the tens column. This problem's solution is excluded from the idealized protocol.

9. 106 S3 A2 D2 A1 -1 -2
   − 70 Ideal: S3 D3 A2 S2 D2 A1 -1 -2 -3

   Pete again makes several slips on this problem: borrowing for the units column, leaving out the decrement in the hundreds column, and leaving out the slash in the tens column. Also, he suppresses the answer's leading zero. This problem is the best evidence in the protocol that Pete knows about borrowing from zero, so it is necessary to leave it in. Thus, the idealized protocol pretends the problem is $106-79$, which requires a borrow in the units column. The idealized protocol rectifies Pete's other slips and does not suppress the leading zero.

10. 311 S3 D3 A2 S2 D2 A1 -1 -2 -3
    − 214

11. 6591 S2 A1 D2 S3 A2 D3 S4 D4 A3 -1 -2 -3 -4
    −2697

    Facts error in the units column: $11-7=5$.

12. 1564 S2 A1 D2 S3 A2 D3 S4 D4 A3 -1 -2 -3
    − 887 Ideal: S2 A1 D2 S3 A2 D3 S4 D4 A3 -1 -2 -3 -4

Pete suppresses the answer's leading zero; the idealized protocol does not. Facts error in the hundreds column: $14-8=7$.

13. 716 S2 A1 S3 D3 D2 A2 -1 -2 -3
− 598 Ideal: S2 D2 A1 S3 D3 A2 -1 -2 -3

Pete seems to forget to do the decrement in the tens column, but he catches his mistake just before adding ten in the tens column. The idealized protocol does not make this slip.

Production System

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i) < (Bottom i),
then set the goals (BorrowFrom i + 1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i) = 0,
then set the goals (BorrowFrom i + 1), (Add10 i), (BorrowFrom i).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

Goal Selection Strategy

1. Prefer (X i) over (ProcessColumn i), for X in the set BorrowingGoals.
2. Prefer (X i) over (ColumnDifference i), for X in BorrowingGoals.
3. Prefer (Add10 i) over (X i), for X in {BorrowFrom, Slash, Decrement}.
4. Prefer (Slash i) over (Decrement i).
5. Prefer (ProcessColumn i) over (ProcessColumn i + j).
6. Prefer (X i) over (ProcessColumn j), for X in the set BorrowingGoals.

7a. If (BFZinProgress)
then prefer (X i) over (ColumnDifference j) for X in BorrowingGoals
else prefer (X i) over (ColumnDifference j)
for X in the set {BorrowFrom, Add10, Slash}.

7b. If (Problem {5})
then prefer (ColumnDifference i) over (Decrement j)
else prefer (Decrement j) over (ColumnDifference i).

8. If (Problem {11, 12, 13})
then prefer (ProcessColumn i) over (ColumnDifference j),
else prefer (ColumnDifference j) over (ProcessColumn i).

9. If (BFZinProgress) or
((Problem {11, 12, 13}) and (Penultimate i))
then prefer (X i + j) over (Y i) for X, Y in the set BorrowingGoals.
else

prefer (BorrowFrom i + 1) over (Add10 i), and
prefer (Slash i + 1) over (Add10 i), and
prefer (Add10 i) over (Decrement i + 1)..

10. Prefer (ColumnDifference i) over (ColumnDifference i + j).

Robby

Robby seems to have a common bug, called Stops-Borrow-At-Zero. (There is a second analysis, which attributes to him a procedure that does not know how to borrow across zero. When given a borrow-from-zero problem, he reaches an impasse when he tries to decrement a zero. On two problems (10 and 11), he repairs by skipping the BorrowFrom goal. On another problem (13), he repairs by relocating the BorrowFrom leftwards. However, the evidence for a relocation repair in problem 13 is weak, because Robby only does the slash and not the decrement of the supposedly relocated BorrowFrom.) Robby's execution strategy is usually the standard one, but on two problems (6 and 7), he delays doing "hard" column differences until the end of the problem, where "hard" appears to mean that the top digit in the column is ten or more. Also, Robby exhibits several permutations of the borrowing operations, which we model by deleting standard preferences 7 and 9, and adding a new preference, 10. The new preference causes slash-decrement pairs to be executed continguously, with no intervening actions. Robby is by far the sloppiest student in the experiment. He makes many facts errors, and he frequently omits parts of the borrowing subprocedure.

Protocols

1. 885 − 205 -1 -2 -3

2. 8305 − 3 -1 -2 -3 -4

3. 83 − 44 S2 D2 A1 -1 -2

4. 967 − 607 -1 -2 -3

   Facts slip in hundreds column: 9 − 6 = 4.

5. 106 − 70 -1 A2 S3 D3 -2 <redo -2> -3
   Ideal: -1 A2 S3 D3 -2 -3

   Robby correctly answers the tens column, then "corrects" it to 10 − 7 = 4.

6. 6591 − 2697 A1 S2 D2 -2 A3 S4 D4 -4 -3 -1

Robby mistakenly omits the borrow in the tens column. Since it is not clear what he would have done if he had noticed the borrow there, the idealized protocol keeps his action sequence and pretends that the problem is 6591 − 2677, which requires no borrow in the tens column.

7. 108 -1 A2 S3 D3 -3 -2 <redo -2>
− 60 Ideal: -1 A2 S3 D3 -3 -2

Robby makes a facts error in the tens column, detects it, and redoes the column difference. The idealized protocol gets it right the first time.

8. 1236 A1 S2 -1 A2 S3 -2 A3 -3 -4
− 497 Ideal: A1 S2 D2 -1 A2 S3 D3 -2 A3 S4 D4 -3 -4

Robby makes several slips, which are rectified in the idealized protocol. He omits D2, D3 and the whole of (BorrowFrom 4). Also, he makes a facts error in column 2 (12 − 9 = 4) and in column 3 (11 − 4 = 4).

9. 1813 A1 -1 -2 S3 D3 -3 -4
− 215 Ideal: A1 -1 S2 D2 A2 -2 S3 D3 -3 -4

Facts error in the units column: 13 − 5 = 7. Robby does not write the (BorrowFrom 2) and A2, but he acts as if he did, and give 10 − 1 = 9 as the answer in column 2.

10. 102 A1 -1 A2 S3 D3 -2 -3
− 39

Facts error in the units column: 12 − 9 = 4.

11. 9007 -1 A2 -2 A3 -3 S4 D4 -4
−6880

Facts error in the thousands column's decrement: 9 − 1 = 7.

12. 4015 A1 -1 D2 -2 A3 S4 D4 -3 -4
− 607 Ideal: A1 -1 S2 D2 -2 A3 S4 D4 -3 -4

Robby omits the S2.

13. 104 S3 A1 -1 A2 -2 -3
− 27 Ideal: A1 -1 A2 -2 -3

The initial Slash may be due to a repair to the decrement-zero impasse, or it may be a slip of some kind. The idealized protocol omits it.

Production System

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i) < (Bottom i),
then set the goals (BorrowFrom i + 1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i) = 0,
then do nothing.

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

Goal Selection Strategy

1. Prefer (X i) over (ProcessColumn i), for X in the set BorrowingGoals.
2. Prefer (X i) over (ColumnDifference i), for X in BorrowingGoals.
3. Prefer (Add10 i) over (X i), for X in {BorrowFrom, Slash, Decrement}.
4. Prefer (Slash i) over (Decrement i).
5. Prefer (ProcessColumn i) over (ProcessColumn i + j).
6. Prefer (X i) over (ProcessColumn j), for X in the set BorrowingGoals.
7.
8. If (Top i) > 9 and (Problem {6, 7})
   then prefer (ProcessColumn j) over (ColumnDifference i)
   else prefer (ColumnDifference i) over (ProcessColumn j).
9.
10. Prefer (Decrement i) over (X j),
    for X in {Add10, ColumnDifference, BorrowFrom}.

Tanya

Tanya has a moderately common bug, called Diff-0-N = 0-Except-After-Borrow, which merely writes zero in the answer instead of borrowing whenever a column has zero as its top digit, and the zero is from the original top row rather than being created by an earlier borrow that decremented a one. This bug is represented by adding an extra condition—that the original value of the top digit in the column be non-zero—to the production that initiates borrowing, and assuming that ColumnDifference always takes the absolute difference of the digits in the column. Also, Tanya has the same version of borrowing from zero as Janine—see the comments in Janine's section for discussion. Tanya's execution strategy is almost perfectly stable. She always uses the horizontal convention, which completes all borrowing before doing any column-answering. This is represented by reversing standard preference number 8. Tanya systematically employs two permutations of the borrowing goals. If the borrow originates in the units column, Tanya does the Add10 first; if the borrow originates in the tens column, she does the BorrowFrom first. Only problem 11 is an exception, and the exception may be due to a slip. Tanya's policy on borrowing permutations is represented by putting a condition around standard preference number 9.

Protocols

1.
```
   647  -1 -2 -3
 -  45
```

2. 885 -1 -2 -3
   − 205

3. 83 A1 S2 D2 -1 -2
   − 44

4. 8305 -1 -2 -3 -4
   − 3

5. 50 -1 -2
   − 23

6. 106 -1 -2 -3 <redo -3>
   − 70 -1 -2 -3

   Tanya first writes 1 in the units column answer, then "corrects" it to 0. Ideal protocol leaves it as 1.

7. 716 A1 S2 D2 S3 D3 A2 -1 -2 <redo -2> -3
   − 598 Ideal: A1 S2 D2 S3 D3 A2 -1 -2 -3

   Tanya rewrites her answer to column 2; the idealized protocol does not.

8. 311 A1 S2 D2 S3 D3 A2 -1 -2 -3
   − 214

   Facts error in column 1: $11-4=6$.

9. 102 A1 N2 S3 D3 -1 -2 -3
   − 39

   Facts error in column 1: $12-9=9$.

10. 9007 -1 -2 -3 -4
    −6880

11. 4015 A1 S2 D2 A2 N3 -1 -2 -3 -4
    − 607 Ideal: A1 S2 D2 -1 -2 -3 -4

    Tanya slips, starts a borrow in the tens column, but stops before completing it. The idealized protocol omits the borrow.

12. 702 A1 N2 S3 D3 -1 -2 -3
    − 108

13. 205 -1 -2 -3
    − 30

    Slip in column 3: writes 1 in answer.

14. 100 -1 -2 -3
    − 60

    Slip in column 3: writes 0 in the answer.

Production System

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i)<(Bottom i),
and (OriginalTop i) is not equal to zero,
then set the goals (BorrowFrom i+1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i)=0,
then set the goals (BorrowFrom i+1), (WriteNine i).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

Goal Selection Strategy

1. Prefer (X i) over (ProcessColumn i), for X in the set BorrowingGoals.
2. Prefer (X i) over (ColumnDifference i), for X in BorrowingGoals.
3. Prefer (Add10 i) over (X i), for X in {BorrowFrom, Slash, Decrement}.
4a. Prefer (Slash i) over (Decrement i).
4b. Prefer (Slash i) over (Write9 i).
5. Prefer (ProcessColumn i) over (ProcessColumn i+j).
6. Prefer (X i) over (ProcessColumn j), for X in the set BorrowingGoals.
7. Prefer (X i) over (ColumnDifference j), for X in BorrowingGoals.
8. Prefer (ProcessColumn i) over (ColumnDifference j).
9. If i=1 and (BFZinProgress)
   then prefer (X i) over (Y i+1) for X, Y in the set BorrowingGoals,
   else prefer (Y i+1) over (X i) for X, Y in the set BorrowingGoals.
10. Prefer (ColumnDifference i) over (ColumnDifference i+j).

Trina

Trina has a common bug, called Don't-Decrement-Zero, which makes her borrow-from-zero routine incorrect. The correct procedure is to add ten to the zero and then later to decrement; Trina's bug omits the decrementing. Trina's strategy for processing columns is generally standard, except on problems 9, 10 and 12. On those problems, as soon as the actions of the last possible borrow are completed, she answers the remaining columns in left to right order. This aspect of her strategy is represented by wrapping a conditional around standard preference number 5. In problem 10, she delays the column difference in the tens column, so standard preference 8 also has a condition wrapped around it. Trina also shifts unsystematically among three permutations of the borrowing actions, which is represented by deleting standard preferences numbers 7 and 9, and adding a new preference, 10.

The effect of the new preference is to cause slash-decrements to be executed continguously, with no intervening actions.

Protocols

1. 50 S2 D2 A1 -1 -2
   − 23

2. 562 A1 -1 S2 D2 -2 -3
   − 3

3. 742 A1 -1 S2 D2 -2 -3
   − 136

4. 106 -1 <redo -1> A1 -2 S3 D3
   − 70 Ideal: -1 A1 -2 S3 D3 -3

   Trina rewrites her answer in column 1, and she suppresses the answer's leading zero. The idealized protocol does neither.

5. 716 A1 -1 S2 D2 S3 D3 A2 -2 -3
   − 598

6. 102 A1 -1 A2 S3 D3 -2
   − 39 Ideal: A1 -1 A2 S3 D3 -2 -3

   Trina suppresses the leading zero. The ideal protocol does not.

7. 9007 -1 A2 A3 S4 D4 -2 -3 -4
   −6880

8. 4015 A1 -1 S2 D2 -2 A3 -3 S4 D4 -4
   − 607

9. 702 S2 D2 A1 -1 A2 S3 D3 -3 -2
   − 108 Ideal: A1 -1 A2 S3 D3 -3 -2

   Trina inexplicably begins by slashing the top zero in the tens column and "decrementing" it to zero. The idealized protocol omits these actions. Both Trina and the ideal protocol do $10-0=0$ in the tens column. That is, they both omit carrying.

10. 2006 -1 A2 A3 S4 D4 -4 -3 -2
    − 42

    Both Trina and the ideal protocol omit carrying from the hundreds column.

11. 10012 A1 -1 S2 D2 A3 A4 S5 D5 A2 -2 -3 -4 -5
    − 214

12. 8001 A1 -1 S2 A3 S4 D4 -4 -2 <Write 10 above column 2>
    − 43 Ideal: A1 -1 A2 A3 S4 D4 -4 -3 -2

Trina makes two slips, which the idealized protocol rectifies. First, she fails to answer the hundreds column. Second, she does an S2 instead of an A2. Apparently, she catches this error later, because her answer to the second column is 6, indicating that she interpreted the slash mark as an Add10. Nonetheless, she changes her slash to a "10" after she has already completed the problem.

Production System

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i) < (Bottom i),
then set the goals (BorrowFrom i + 1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i) = 0,
then set the goals (BorrowFrom i + 1), (Add10 i).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

Goal Selection Strategy

1. Prefer (X i) over (ProcessColumn i), for X in the set BorrowingGoals.
2. Prefer (X i) over (ColumnDifference i), for X in BorrowingGoals.
3. Prefer (Add10 i) over (X i), for X in {BorrowFrom, Slash, Decrement}.
4. Prefer (Slash i) over (Decrement i).
5. If (Problem {9, 10, 12})
   and there is a slash-decrement in the leftmost column,
   then prefer (ProcessColumn i + j) over (ProcessColumn i)
   else prefer (ProcessColumn i) over (ProcessColumn i + j).
6. Prefer (X i) over (ProcessColumn j), for X in the set BorrowingGoals.
7.
8. If (Not (Problem {10}))
   then prefer (ColumnDifference i) over (ProcessColumn j).
9.
10. Prefer (Decrement i) over (X j),
    for X in {Add10, ColumnDifference, BorrowFrom}.

APPENDIX 2: LIFO MODELS

This section presents LIFO models for the 8 non-standard students. The models are expressed as hierarchical production systems. The notational conventions used in the non-LIFO models are used here, with one modification. If a production creates several goals, the default interpretation is that

the goals are to be completed in the order of their occurrence in the rule. Thus, the production *If C then set the goals A, B* means to complete goal A before starting to work on B. This default interpretation can be overridden by preferences attached to the production. The production *If C then set the goals A, B. If (Problem {12,13}) then prefer B over A* means to finish goal A before starting B for all problems except problems 12 and 13; on those problems, goal B is to be finished before work begins on goal A. Similarly, the default interpretation of an action side of the form "for each column i, set the goal...." is to select the goals in right-to-left order by their arguments; but preferences can modify this order. The preferences attached to a production can only mention goals that appear in the action side of the production. Thus, this notation is strictly less powerful than the non-LIFO notation.

Angela

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i);
If (ColumnSeemsHard i)
then prefer (ProcessColumn i+j) over (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i)<(Bottom i)
then set the goals (BorrowFrom i+1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i) and (Top i)=0,
then set the goals (BorrowFrom i+1), (Add10 i), (BorrowFrom i).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

Hilda

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i);

If the goal is (ProcessColumn i), and (Top i)<(Bottom i)
then set the goals (Slash j), (Add10 i), (Decrement j), (ColumnDifference i),
where j is the first column to the left of i that has a non-zero top digit.
If (Problem {5, 8}) then prefer (Decrement j) over (Add10 i),
else if Problem {7, 9}) then prefer (ColumnDifference i) over (Decrement j).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

Janine

If the goal is (Subtract) and (Problem {1,2,4,8}),
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i)<(Bottom i),
then set the goals (Add10 i), (BorrowFrom i+1), (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i)=0,
then set the goals (Slash i), (WriteNine i), (BorrowFrom i+1).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

If the goal is (Subtract) and (Problem {5,6,11,12}),
then set the goals (DoAllBorrows), (DoAllAnswers).

If the goal is (DoAllBorrows),
then for each column i, set the goal (Borrow i).

If the goal is (Borrow i) and (Top i)<(Bottom i),
then set the goals (Add10 i), (BorrowFrom i+1).

If the goal is (Borrow i),
then do nothing.

If the goal is (DoAllAnswers),
then for each column i, set the goal (ColumnDifference i).
If Problem {12})
then prefer (ColumnDifference i+j) over (ColumnDifference i).

If the goal is (Subtract) and (Problem {3,7,9}),
then for each column i, set the goal (Vertical i).

If the goal is (Vertical i) and (Top i)=0 and (IncomingBorrow i),*
then set the goals (Slash i), (WriteNine i), and (ColumnDifference i).

If the goal is (Vertical i) and (Top i)>(Bottom i) and (IncomingBorrow i),
then set the goals (BorrowFrom i), (ColumnDifference i).

If the goal is (Vertical i) and (IncomingBorrow i),
then set the goals (BorrowFrom i), (Add10 i), (ColumnDifference i).

If the goal is (Vertical i) and (Top i)<(Bottom i),
then set the goals (Add10 i), (ColumnDifference i).

If the goal is (Vertical i),
then set the goal (ColumnDifference i).

Paul

If the goal is (Subtract)
then for each column i, set the goal (ProcessColumn i).

---

* The predicate (*IncomingBorrow i*) is true if there is a column i-1 and either the action Add10 or the action WriteNine has been performed on it.

If the goal is (ProcessColumn i), and (Top i)<(Botom i) and (Top i+1)=0,
then set the goals (BorrowFrom i+1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i) and(Top i)<(Bottom i),
then set the goals (Slash i+1), (Add10 i), (ColumnDifference i), (Decrement i+1).
If either (Problem {7}) and i=2 or (Problem {10}) and i=3,
then prefer (Decrement i+1) over (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i)=0,
then set the goals (BorrowFrom i+1), (Add10 i), (BorrowFrom i).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

Pete

If the goal is (Subtract) and (Problem {1,2,3,4,5,6,7,9},
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColulmn i) and (Top i)<(Bottom i) and (Top i+1)=0,
then set the goals (BorrowFrom i+1), (Add10 i), (ColumnDifference i).

If the goal is (ProcessColumn i) and (Top i)<(Bottom i),
then set the goals (Slash i+1), (Add10 i), (ColumnDifference i), (Decrement i+1).
If (Problem {3, 6}) then prefer (Decrement i+1) over (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i)=0,
then set the goals (BorrowFrom i+1), (Add10 i), (BorrowFrom i).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

If the goal is (Subtract) and (Problem {11, 12, 13}),
then (DoAllBorrows), (DoAllColumns).

If the goal is (DoAllBorrows),
then for each column i, set the goal (Borrow i).

If the goal is (Borrow i) and (Top i)<(Bottom i),
then set the goals (Slash i+1), (Add10 i), (Decrement i+1).
If (Penultimate i), then prefer (Decrement i+1) over (Add10 i).

If the goal is (Borrow i),
then do nothing.

If the goal is (DoAllAnswers),
then for each column i, set the goal (ColumnDifference i).

Robby

If the goal is (Subtract) and (Problem {1,2,3,4,5,8,9,10,11,12,13}),
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i) < (Bottom i),
then set the goals (Add10 i), (BorrowFrom i + 1), (ColumnDifference i).
If either (Problem {9,11}) or both i = 2 and (Problem {12}),
then prefer (ColumnDifference i) over (BorrowFrom i + 1),
else if (Problem {3}),
then prefer (BorrowFrom i + 1) over (Add10 i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i) = 0,
then do nothing.

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

If the goal is (Subtract) and (Problem {6,7}),
then (DoColumns), (FinishUp).

If the goal is (DoColumns),
then for each column i, set the goal (DoProcessColumn i).

If the goal is (DoProcessColumn i) and (Top i) < (Bottom i),
then set the goals (Add10 i), (BorrowFrom i + 1).

If the goal is (DoProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (FinishUp),
then for each column i, set the goal (Check&Answer i),
where i goes rightward from the leftmost column.

If the goal is (Check&Answer i) and the answer place of i is blank,
then set the goal (ColumnDifference i).

If the goal is (Check&Answer i),
then do nothing.

Tanya

If the goal is (Subtract)
then set the goals (DoAllBorrows), (DoAllAnswers).

If the goal is (DoAllBorrows),
then for each column i, set the goal (Borrow i).

If the goal is (Borrow i), and (Top i) < (Bottom i),
and (OriginalTop i) is not equal to zero,
then set the goals (BorrowFrom i + 1), (Add10 i).
If i = 1, then prefer (Add10 i) over (BorrowFrom i + 1).

If the goal is (Borrow i),
then do nothing.

If the goal is (BorrowFrom i), and (Top i) = 0,
then set the goals (WriteNine i), (BorrowFrom i + 1).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

If the goal is (DoAllAnswers)
then for each column i, set the goal (ColumnDifference i).

Trina

If the goal is (Subtract) and (Problem {1,2,3,4,5,6,7,8,11}).
then for each column i, set the goal (ProcessColumn i).

If the goal is (ProcessColumn i), and (Top i) < (Bottom i),
then set the goals (Add10 i), (ColumnDifference i), (BorrowFrom i + 1).
If either (Problem {1}) or both i = 2 and (Problem {5,11}),
then prefer (BorrowFrom i + 1) over (Add10 i),
else if (Problem {7}),
then prefer (BorrowFrom i + 1) over (ColumnDifference i).

If the goal is (ProcessColumn i),
then set the goal (ColumnDifference i).

If the goal is (BorrowFrom i), and (Top i) = 0,
then set the goals (Add10 i), (BorrowFrom i + 1).

If the goal is (BorrowFrom i),
then set the goals (Slash i), (Decrement i).

If the goal is to (Subtract) and (Problem {9,10,12}),
then set the goal (RecursiveSub i).

If the goal is (RecursiveSub i) and the leftmost column has a slash in it,
then for each column i, set the goal (ColumnDifference i), where
i goes from the leftmost column to the rightmost unanswered column.

If the goal is (RecursiveSub i) and Problem {10}) and i = 2,
then (Add10 i), (BorrowFrom i + 1), (RecursiveSub i + 1).

If the goal is (RecursiveSub i),
then set the goals (ProcessColumn i), (RecursiveSub i + 1).