DOCUMENT RESUME

ED 221 386 SE 039 194

AUTHOR Resnick, Lauren B.
TITLE Syntax and Semantics in Learning to Subtract.
INSTITUTION Pittsburgh Univ., Pa. Learning Research and Development Center.
REPORT NO LRDC-1982/8
PUB DATE 82
NOTE 23p.

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Addition; *Computation; Educational Research; Elementary Education; *Elementary School Mathematics; *Error Patterns; *Mathematics Instruction; *Subtraction
IDENTIFIERS *Mathematics Education Research

ABSTRACT

This paper is concerned with the role of meaning and understanding in the acquisition of computational skill in subtraction. Evidence on the syntactic nature of subtraction errors is reviewed, with common "bugs" described. Then, data collected from interviews with four children are summarized in terms of their semantics of concrete representations of the base system, procedures for adding and subtracting with concrete representations, semantics of the writing code, procedures for written subtraction and addition, and mapping. Finally, linking syntax and semantics and why mapping works are discussed. (MNS)

SE

LEARNING RESEARCH AND DEVELOPMENT CENTER

1982/8

SYNTAX AND SEMANTICS IN LEARNING TO SUBTRACT

LAUREN B. RESNICK

University of Pittsburgh

SYNTAX AND SEMANTICS IN LEARNING TO SUBTRACT

Lauren B. Resnick

Learning Research and Development Center

University of Pittsburgh

1982

Reprinted by permission from Addition and Subtraction: A Developmental Perspective, T. P. Carpenter, J. M. Moser, and T. A. Romberg (Eds.), Lawrence Erlbaum Associates, Inc. Hillsdale, New Jersey, 1982.

The research reported herein was supported by the Learning Research and Development Center, supported in part as a research and development center by funds from the National Institute of Education (NIE), Department of Education. The opinions expressed do not necessarily reflect the position or policy of NIE and no official endorsement should be inferred.

10 Syntax and Semantics in Learning to Subtract

Lauren B. Resnick
University of Pittsburgh

This chapter is concerned with the role of meaning and understanding in the acquisition of computational skill. Until quite recently, discussions of meaningfulness in arithmetic learning have been characterized by a confrontation between those who advocate learning algorithms and those who argue for learning basic concepts. This confrontation, however, is neither necessary nor fruitful. In fact, it serves to direct attention away from the ways in which understanding and procedural skill may mutually support and influence one another.

Proponents of conceptually oriented instruction have frequently argued that a major cause of difficulty in learning arithmetic is a failure to relate rules of computational procedure to their underlying mathematical concepts. Substantial efforts on the part of mathematics educators have resulted in instruction intended to display for children the structure of important concepts such as the base system and positional notation, on the assumption that procedural algorithms would be easily acquired and retained when the conceptual basis was obvious. The data presented here show that intuitions concerning the importance of conceptual understanding are correct in a crucial respect: difficulties in learning are often a result of failure to understand the concepts on which procedures are based. But the data also show that even when the basic concepts are quite well understood, they may remain unrelated to computational procedure. Thus the conceptual teaching methods of the past were inadequate to the extent that they taught concepts *instead of* procedures and left it entirely to students to discover how computational procedures could be derived from the basic structure of the number and numeration system. Our research thus poses an important new problem for mathematics instruction: devising methods that help students to explicitly link meaning and procedure.

DISTINGUISHING SYNTAX AND SEMANTICS

Written subtraction can be analyzed as an algorithm defined by a set of *syntactic* rules that prescribe how problems should be written, an order in which certain operations must be performed, and which kinds of symbols belong in which positions. Although the syntax may reflect an underlying *semantics*, or meaning, an algorithm need not include any explicit reference to the semantics in order to be successfully performed. Figure 10.1 shows an algorithm for subtraction that is

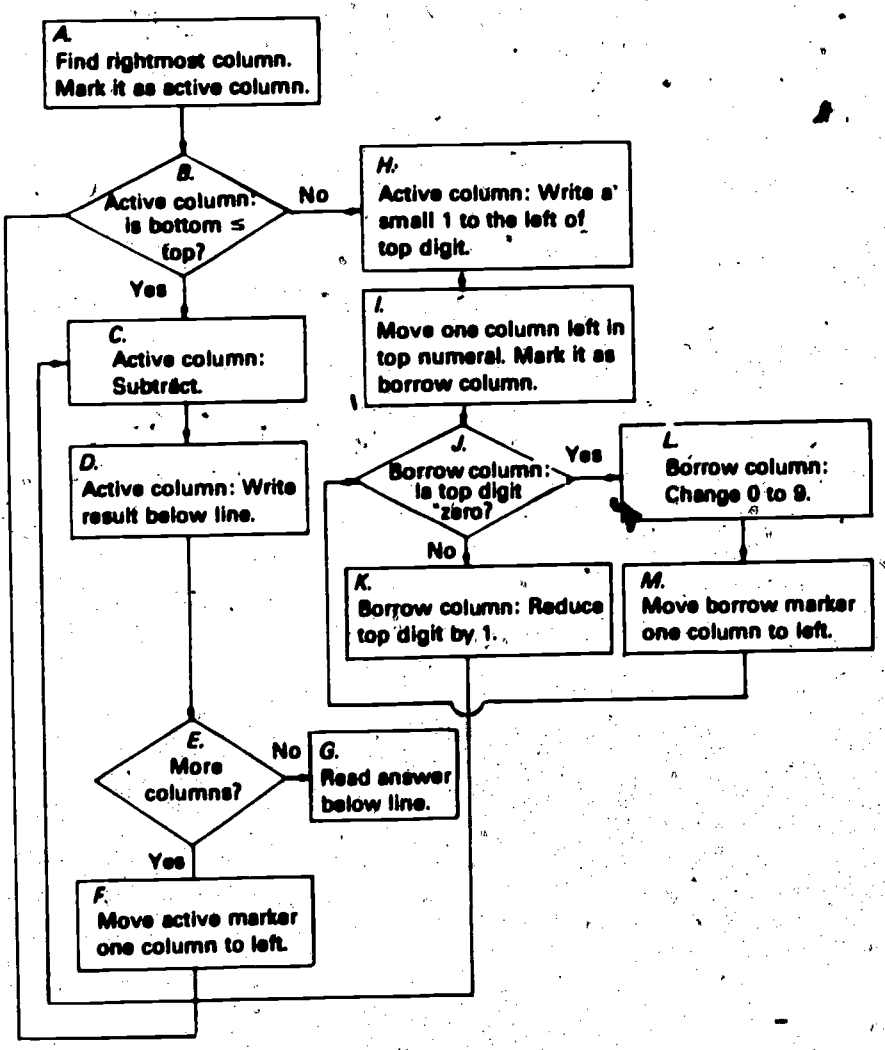


FIG. 10.1. Syntactic algorithm for subtraction.

entirely syntactic in nature. Followed strictly, it can solve any subtraction problem written out in aligned column format. Some of the syntactic rules are specified in the figure (e.g., writing 1 in a particular position at *H* and reducing by one at *K*). Other syntactic rules the algorithm obeys are not specified in the figure. For example, there can be only one digit per column, and each column must be acted upon at least once.

Because performing this algorithm requires no understanding of the base system, I claim that it includes no semantics. That is, the system performing this algorithm does not need to know such things as the fact that the small 1 inserted as part of borrowing really represents 10 (when it is in the rightmost, or units column), or that borrowing involves an exchange of quantities between columns that does not change the total quantity represented. Even the reason for borrowing is omitted. This kind of semantic knowledge would justify the syntax of the algorithm, but the algorithm can be run without reference to the semantics. The evidence I present here suggests quite strongly that many children may learn the syntactic constraints of written subtraction without connecting them to the semantic information that underlies the algorithm. This can lead to systematic errors in performance.

Evidence for the Syntactic Nature of Subtraction Errors

Initial evidence for the syntactic nature of written subtraction errors comes from the extensive work of Brown and his colleagues on the nature of children's errors in subtraction (Brown & Burton, 1978). They have shown that a substantial portion of children's errors result not from random mistakes, but from *systematically* following wrong procedures.

The wrong procedures are variants of correct ones; they are analogous to computer algorithms with bugs in them. A finite number of bugs, which in various combinations make up several hundred "buggy algorithms," have been identified for subtraction. Bugs presumably arise when a subtraction problem is encountered for which the child's algorithm is incomplete or inappropriate. The child tries to respond anyway and either applies the incomplete algorithm leaving out necessary steps, or tries to repair the algorithm to adjust to the new subtraction task. The resulting bug respects some of the constraints—syntactic and semantic—that are embedded in a full, correct algorithm, but violates others. (See Brown & VanLehn, this volume, for details of this "repair" theory.)

By examining some common subtraction bugs and considering the possible origins of each, it is possible to decide whether the bugs represent violations of primarily semantic or syntactic constraints. This analysis, part of which is presented in the following section, suggests that most buggy algorithms respect the syntactic requirements of written subtraction, whereas the constraints that are relaxed are the ones that express the semantics of the base-ten system and of

subtraction. This informal analysis appears consonant with both Brown and VanLehn's theory and with another theory by Young and O'Shea (1981). In the latter theory children construct buggy algorithms because they either have never learned the complete standard algorithm or have forgotten parts of it.

Figure 10.2 names a number of common bugs in order of their frequency, describes them, and gives examples of each. In what follows, I consider possible sources of these bugs and characterize each bug as semantic or syntactic in character.

1. Smaller-From-Larger. Repair theory suggests that this very common bug results from "switching arguments" to respond to a situation in which the system cannot make its normal move of subtracting the bottom from the top number in a column. In other words, the system makes the test at *B* in Fig. 10.1, but doesn't know how to borrow and decides that the subtraction should be done in the opposite direction. Young and O'Shea's analysis suggests that this bug derives from simply not making the test and is the normal or default way for the system to proceed *unless* the test is made and the various borrowing rules are thereby evoked. In both of these interpretations all the syntax of written subtraction without borrowing is respected. What is violated is the constraint that the bottom quantity *as a whole*, be subtracted from the top quantity *as a whole*. The semantics of multi-digit subtraction includes the constraint that the columns, although handled one at a time, cannot be treated as if they were a string of unrelated single-digit subtraction problems.

2. Borrow-From-Zero. Both repair theory and the Young and O'Shea analysis suggest that this bug derives from forgetting the part of the written procedure that is equivalent to steps *M-J-K* in Fig. 10.1 (moving the borrow marker left, and reducing the new column). The bug respects the syntactic requirement that, in a borrow, there must be a crossed-out and rewritten numeral to the left of the active column. It also respects the syntax of the special case of zero, where the rewritten number is always 9. However, it ignores the fact that the 9 really results from borrowing one column further left (the hundreds column) moving 100 as 10 tens into the tens column, and *then* borrowing from the 10 tens leaving 9 tens, or 90 (written as 9).

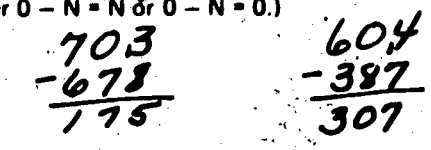
3. Borrow-Across-Zero. Repair theory offers two different derivations of this bug. The first is that this bug arises from the child's search for a place to do the decrementing operation with the condition that the column not have a zero in the top number. This would happen when the child doesn't know how to handle zeros or thinks they have "no value" and thus can be skipped. This solution respects the syntactic constraint that a small 1 must be written in the active column and that some other (nonzero) column must then be decremented. But the

1. **Smaller-From-Larger.** The student subtracts the smaller digit in a column from the larger digit regardless of which one is on top.

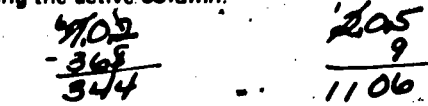
2. **Borrow-From-Zero.** When borrowing from a column whose top digit is 0, the student writes 9 but does not continue borrowing from the column to the left of the 0.

3. **Borrow-Across-Zero.** When the student needs to borrow from a column whose top digit is 0, he skips that column and borrows from the next one. (This bug requires a special "rule" for subtracting from 0: either 0 − N = N or 0 − N = 0.)

4. **Stop-Borrow-At-Zero.** The student fails to decrement 0, although he adds 10 correctly to the top digit of the active column. (This bug must be combined with either 0 − N = N or 0 − N = 0.)



5. **Don't-Decrement-Zero.** When borrowing from a column in which the top digit is 0, the student rewrites the 0 as 10 but does not change the 10 to 9 when incrementing the active column.



6. **Zero-Instead-Of-Borrow.** The student writes 0 as the answer in any column in which the bottom digit is larger than the top.

7. **Borrow-From-Bottom-Instead-Of-Zero.** If the top digit in the column being borrowed from is 0, the student borrows from the bottom digit instead. (This bug must be combined with either 0 − N = N or 0 − N = 0.)

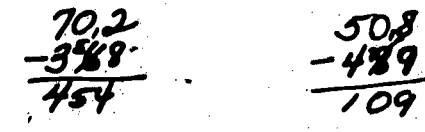


FIG. 10.2. Descriptions and examples of common subtraction bugs. (From "Diagnostic models for procedural bugs in basic mathematics skills," By J. S. Brown and R. R. Burton, *Cognitive Science*, 1978, 2, 155-192 and from personal communication with J. S. Brown, R. R. Burton, and K. VanLehn.)

semantic knowledge that the increment and decrement are actually addition and subtraction of 10 is ignored (or not known). Repair theory's second derivation, which agrees with Young and O'Shea's analysis, produces this bug by simply deleting the rule that changes 0 to 9 (*L* in Fig. 10.1). This too is a completely syntactic derivation, for it allows deletion of a rule without reference to the semantic information that justifies the operation.

4. Stop-Borrow-At-Zero. Both repair theory and Young and O'Shea's analysis interpret this bug as simply omitting a rule or an operation. Steps *I-J-K* of Fig. 10.1 are simply skipped. This bug fails to obey both syntactic and semantic constraints. Syntactically it produces only the increment part of the borrow operation—the 1 in the active column—but does not show a crossed-out number or the change of a 0 to a 9. Semantically it violates the justification for the borrow increment—that is, in order to add a quantity to the active column an equivalent quantity must be subtracted from another column.

5. Don't-Decrement-Zero. The change of 0 to 10 in this bug is the proper "semantic" move after borrowing from the hundreds column. But it produces an outcome that the child may not have encountered and thus does not respond to appropriately. Failure to change the 10 to 9 may result from a syntactic constraint that each column be operated on only once. This syntactic constraint is not "correct," but might be reasonably inferred from extensive experience with problems that contain no zeros. If so, the syntactic constraint is in direct opposition to the semantic demands of the situation.

6. Zero-Instead-Of-Borrow. Like Smaller-From-Larger, this bug simply avoids the borrowing operation altogether, while observing all of the important syntactic constraints of operating within columns, writing only one small digit per column, and the like. This bug, however, does not violate the semantics of the digit structure as blatantly as the Smaller-From-Larger bug. In fact, a child producing this bug may be following a semantics of subtraction that generally precedes any understanding of negative numbers. In this inferred semantics of subtraction, when a larger number must be taken from a smaller, the decrementing is begun and continued until there are no more left—yielding zero as the answer.

7. Borrow-From-Bottom-Instead-Of-Zero. This bug seems purely syntactic in the sense that the search for something to decrement seems to lead the child to ignore the digit structure and the semantics of exchange that justifies borrowing within the top number. But it does produce a "funny-looking" solution, so it would probably be generated only by a child whose syntactic rules did not specifically require that all increments and decrements be in the top number.

A CLOSER LOOK AT CHILDREN'S SEMANTIC AND SYNTACTIC KNOWLEDGE

If bugs result from weak application of semantic constraints, a first hypothesis is that children are simply unfamiliar with the semantics of the base-ten system. Data we have collected permit us to examine children's semantic and syntactic knowledge quite directly. These data lead me to conclude that this simple hypothesis is false—that instead, children are likely to know a good deal about the base system but still be unable to use their semantic knowledge to support written procedures for arithmetic.

The data are from four children who were followed between November and May of the school year in which they first learned addition and subtraction with regrouping. The children were all in a nongraded, individualized mathematics curriculum at the beginning of the study. Three were second-graders and one was a third-grader. In November these children had recently passed the criterion test for a unit in which they were required to read and write numerals up to 100 and to interpret these as compositions of tens and ones. We reinterviewed the children in February, after each had encountered instruction in addition and subtraction with regrouping but had not yet passed the criterion test for those skills, and again in May, at which time all had passed the curriculum test for addition and subtraction with regrouping.

All interview sessions were individually administered and semistructured, with a planned sequence of problems presented in a standard format. Probes by the experimenter and some attempts to explain or even demonstrate a procedure were permitted. Learning obviously took place in the course of the interviews—sometimes in response to the experimenter's "teaching," sometimes as a result of inventions by the children. We have used the speed and character of these learning incidents to infer the children's knowledge base.

The content varied over the three interview periods. In November, we focused on the children's understanding of the semantics of the base system. Most of the tasks required each child to represent written numerals (10 through 99) in concrete forms and to add and subtract in the concrete representations. Dienes blocks, color-coded chips, bundles of sticks, or pennies and dimes were used for the representations. In February, virtually all the tasks were addition and subtraction problems, presented in both written and concrete form. We paid particular attention to the extent to which the child made correspondences ("mappings") between written and concrete representations of these processes. The May interviews replicated those of February but added a special session in which the child was asked to teach a hand puppet to add and subtract. This allowed the children to give explanations and justifications for their addition and subtraction procedures in a less self-conscious way than by simply explaining to an adult why

certain routines were used. The results of these interviews are summarized in the following section.

Semantics of Concrete Representations of the Base System

Three of the four children—Amanda, Alan, and Anton—demonstrated immediate and strong knowledge of the base system in concrete representations. They showed this by:

1. Immediately using the designated color chip, bundle, or block shape to represent tens.
2. Counting by tens and then ones as they "read" concrete displays or counted out the objects to construct those displays.
3. When comparing concrete displays, sometimes counting only the tens—thus relying on a canonical (i.e., no more than nine items of any one denomination) display in which the representation of the higher number will necessarily have more tens.
4. When counting large numbers of units, grouping them into piles of ten, in order to count by tens.
5. Recognizing the conventionality of the codes, as when Amanda said that the bundles of sticks might have nine or eleven sticks each, but: "Let's say they all have ten."

One child, Alan, apparently learned the power code in the course of the initial interviews. He began by counting all blocks, chips, etc., as units, but switched to counting by tens when the experimenter asked him to represent a large number and provided too few blocks to represent it in units. His speed in picking up our conventions suggests that it was only the conventions that he had to learn, not the base-ten semantics that they represented. By contrast the fourth child, Sandra, seemed to be truly acquiring the semantic knowledge of the base system over the course of the period in which we studied her. In November, although the experimenter's prompts would lead her to count by tens and use this code for a few problems, she would revert to counting all denominations as ones whenever new representations were presented or the experimenter did not remind her of the convention. With the Dienes blocks she used a "compromise" solution, in which each of the individual squares on the ten bar was counted. She thus respected the conventional coding, but did not really benefit from its "ten-ness." Even in May (although by this time she always used the code representations), Sandra still counted the individual squares on the ten bar. Of the children in our group, Sandra clearly had the weakest command of the base-system semantics and the least tendency to use it in constructing shortcuts.

Procedures for Adding and Subtracting With Concrete Representations

In the November interviews the only addition and subtraction problems given to the children were in the context of a store game using the penny-and-dime representation. This was the representation with which all four children had seemed most comfortable. All the children were in command of the basic semantics of addition and subtraction. That is, they knew how to add by combining and then recounting objects that originally represented two numbers, and they knew how to subtract by taking away a specified quantity from a given representation of number. All could do this without error for two-digit representations, as long as there was no need for regrouping.

Addition. The equivalent of the carry in written addition is trading ten pennies for a dime in the money representation. In November, none of the children ever initiated a trade of pennies for dimes. However, when the experimenter said, "The store won't accept so many pennies," each child traded ten pennies for a dime without further prompts. Thus, although the children knew that exchanges were possible, they did not naturally tend to use the concrete representations in a manner that corresponded well to the rules of the written algorithm.

Further evidence from the protocols suggests that a preference for canonical form—which would match the written mode—appeared rather late in the development of the children's understanding of the base system. For example, Amanda, whose mental arithmetic performance and general facility with the various concrete representation tasks suggested a very early and strong command of the base system, seemed to have the *least* preference for canonical displays in November. She instead seemed to be experimenting with the various ways in which a given number could be represented. She constructed noncanonical displays on various problems and then converted them to canonical, or vice versa. In general, she made far more trades and exchanges (always ten-for-one or one-for-ten, in keeping with her strong command of the ten-ness in the base system) than any of the other children. By February, however, Amanda had begun to prefer canonical form. She initiated trades in adding and tended to do her trading sequentially—that is, each time that she accumulated ten blocks in a long addition problem, she traded for the next block size.

Sandra, the child who was still counting by ones in May and not using the ten-ness of the system, did not at any time initiate trades to canonical form on her own. But she did show a response to the experimenter's rule of "no more than nine per column" that suggests that she had adopted the rule—without reference to its rationale—as an arbitrary constraint to be followed at all costs. The following protocol segment illustrates this:

14 + 9 + 33. S. (Sets out one tens block and four units blocks; nine units blocks; three tens blocks and three units blocks.) "Ten, twenty, thirty, forty, forty-

one, . . . fifty-one, . . . fifty-six." *E:* "Now what if I told you there could not be more than ten in a column? How could you get rid of them?" *S:* (Hesitates.) *E:* "Could you trade? You could trade ten ones for a ten." *S:* (Trades, putting the ten in the tens column.) *E:* "Do you have the same as before?" *S:* (Counts by tens, continuing with the ones.) "Fifty-six." *E:* "Still the same?" *S:* "Yes."

75 − 49. S: (Sets out seven tens and five units.) "I don't have nine." *E:* "Can you trade one of these?" *S:* "But then I would have ten" (i.e., more units than allowed in a column).

Sandra accepted the experimenter's rule for addition and then applied it to subtraction as well, showing resistance to the suggestion that she trade to solve a subtraction problem.

This performance suggests why *non*preference for canonicity may be an important developmental stage. Perhaps a strong rule specifying canonicity can interfere with learning about situations other than addition until children have come to understand the situations in which canonicity should be preferred.

Subtraction. The equivalent of borrowing in concrete representations is the process of getting more units by trading a ten for ten units. The problem initially posed to the children in November was to take 61 cents (which all of the children did by taking six dimes and one penny) and then give the experimenter 37 cents. There are two critical aspects of this performance. The first is whether the child *initiates* a trade-down or needs to be prompted. The second is whether the trade is "fair"—i.e., whether the child always makes a one-for-ten trade.

In November, only Amanda (who had the most highly developed understanding of the base system) both initiated the trade and made a one-for-ten trade with no prompting or explanation. Sandra, the weakest of the four children, needed to be explicitly told to trade on almost every trial. When unfair trades were made they almost always followed a pattern of trading the dime for only as many pennies as were needed to give the experimenter the number she requested. For example, in the 61 − 37 problem Alan traded a dime for six pennies which, together with the penny he already had, allowed him to give seven pennies to the experimenter.

The children improved between November and May, so that by May all but Sandra were initiating trades and making only fair trades. Even Sandra needed only an initial and weak ("get more") prompt. Because there is little practice in the children's school curriculum on subtraction using these concrete representations, it seems reasonable to suppose that the much more skillful and semantically correct performance they all showed by May was not the result of practice on the subtraction routine itself, but of the development of general semantic knowledge of the base system and subtraction. This suggests that the later trades are driven by the need to get more units (or tens) and that the requirements of equivalent (one-for-ten) trades are well internalized.

Semantics of the Writing Code

All the children could interpret written numbers as "*x* tens and *y* ones." Performance on other tasks that tapped knowledge of the writing code, however, suggests that this may not reflect a very rich knowledge of the semantics of written numerals. We observed the following:

1. Three of the children correctly represented two-digit numerals with concrete representations on the first trial or after only a few prompts by the experimenter. They could also write the numerals when shown a display in one of these concrete representations. This shows some knowledge of the writing code. However, this knowledge appeared to be only weakly linked to the concrete representations. It was common for the children to count all the chips or blocks whether in canonical form or not, say the numeral aloud, and then write this numeral without matching its digits to the concrete display.

2. Three of the four children were able to use expanded notation cards to construct numerals. For example, 98 was constructed from 90 and 8, with the 8 placed on top of the 0. There is some evidence, however, that the solutions to these expanded notation problems were more syntactic than semantic, as the children seemed to be trying different ways of putting the cards together until they found something that looked right. All the children demonstrated some difficulty with zeros (as in the number 708) when using these cards.

Two of the children (including the one who did not use expanded notation cards correctly) gave us spontaneous evidence of a deeper understanding of the written code. This came in the form of:

3. Occasional comments when working on various problems. For example, Alan, when comparing the numerals 9 and 90, said the 90 was larger because the 9 "doesn't even have ten."

4. Solving written problems mentally and then writing down the answers. Amanda used a mental arithmetic strategy in which she partitioned two-digit numbers into tens and ones and then operated separately on the two sets of values. For example, she solved 37 + 25 as follows: "Thirty plus twenty is fifty. Fifty-seven. Fifty-seven, fifty-eight, fifty-nine, sixty, sixty-one, sixty-two." Then she wrote 62, aligning the digits in the proper columns.

Procedures for Written Subtraction and Addition

Written addition and subtraction problems were first presented to the children in February and were repeated in May. Except for a few special probes, only two-digit problems in subtraction were used, so we were unable to observe any of the zero bugs described earlier. From these protocols, however, we can chart the development of three key components of written algorithms: right-to-left rules of

procedure, the carry procedure, and the borrow procedure. Three of the children—Sandra, Anton, and Alan—can be followed with respect to these components. Amanda's use of mental arithmetic allowed her to avoid the written algorithms altogether.

Right-to-Left Direction. The three children all showed a rather slow and hesitant development of the right-to-left rules of procedure. Each had learned in school that right to left is the "correct" way to do the problems. When asked why they should start at the right they typically answered, "So you get the right answer" or "It's easier," but never, "To make borrowing or carrying easier" or a similar comment. In February each child began by working left to right and switched to a right-to-left direction after encountering difficulty. These switches were spontaneous; the experimenter never suggested working in the other direction. They were not, however, full-fledged inventions, as it is quite clear from the protocols that a procedural rule learned earlier was being invoked to cope with difficulties encountered.

Carry Procedure. Sandra, the weakest of the three in her command of the semantics of the base system as represented in concrete materials, performed the carry procedure perfectly in February. (It had been taught to her in school by then.) Anton, initially worked left to right, but quickly corrected himself. In February, Alan almost completely avoided carrying by counting on his fingers and then writing the answers, and he continued to show some difficulty with carrying in May. Amanda did not use the written carrying algorithm at all, even in May. Thus, strong semantic understanding (as shown by manipulations of the concrete representations and use of mental arithmetic) in no way guarantees ability to use the carry procedure in written addition.

Borrow Procedure. As might be expected, this procedure was harder to learn than carrying. Alan, Anton, and Sandra in the February and May interviews each showed the Smaller-From-Larger bug, but each then spontaneously switched to a borrow operation. By May, Sandra and Anton were using the correct, school-taught written algorithm, notating both increments and decrements. Alan repeatedly stated that he was supposed to borrow, but even in May he had some difficulty remembering the exact procedure. Amanda first performed in the borrow procedure in May, with many false starts and bugs. Again, the two children with the strongest knowledge of base system semantics—Alan and Amanda—seemed to have the weakest control of the written algorithm.

Mapping

The final question to be addressed on the basis of these data is the extent to which the children had connected their knowledge about the base system (as displayed in their work with the concrete materials) and their knowledge about the written

procedures for subtraction and addition. A general response can be made by simply noting again the lack of correlation between knowledge of the base-system semantics and knowledge of the written algorithms.

More specifically, our interview data permit a closer examination of three levels of mapping between the written algorithms and the concrete materials: (1) code mapping—the extent to which the child recognizes that the shape or color of the concrete materials codes the same information as position (column) in the written numerals; (2) result mapping—the extent to which the child expects procedures in the written system to yield the same answers as procedures in the concrete materials; and (3) operations mapping—the extent to which the child can identify equivalent operations in the written and the concrete systems and can model a written algorithm in a concrete mode.

Code mapping was at least weakly present in all the children. This was shown by their quickly acquired ability to represent written numerals in concrete forms and to write numerals that corresponded to the concrete representations. However, none of the children insisted on an exact correspondence between the written and the concrete form.

There were distinct differences among the four children with respect to result mapping. Although we did not systematically probe for this, Amanda and Alan gave clear evidence of expecting blocks and writing to yield the same answers. During the May interview, for example, Alan corrected his written subtraction after doing the same problem with blocks. He trusted his blocks answer, and expected writing and blocks to yield the same answer, showing this by doing a new or amended written procedure. Anton, by contrast, seemed undisturbed by getting different answers in blocks than in written subtraction.

Although Alan expected the same answers from blocks and writing, he did not use detailed operations mapping to correct his written algorithm. Instead, he seemed to search his memory for a previously learned procedure that would correct his errors. There was also other evidence that these children were not doing operations mapping. With the possible exception of Amanda in May, at no time did we see the children examining the steps in a blocks procedure as a way of helping themselves perform a written algorithm.

LINKING SYNTAX AND SEMANTICS

If the preceding analysis is correct, then children's difficulty with place value in addition and subtraction does not necessarily derive from an *absence* of semantic knowledge about the base system. Rather, it results from an inadequate *linking* of the semantics of the base system with the syntax of the written algorithms. This suggests that for many children who have difficulty learning or remembering the rules for written arithmetic, instruction that explicitly links semantic and syntactic knowledge may be useful. In addition, initial instruction that stresses

the semantic properties of the addition and subtraction algorithms should help block the difficulties and buggy routines that might arise later.

We have pilot tested with three other children a remedial teaching procedure that explicitly forces a mapping at the operational level between block subtraction and written subtraction. The procedure requires that the child perform the same problem using Dienes blocks and writing, alternating between the two representations. The writing thus serves as a record of the blocks actions. Conversely, the blocks justify the steps in the written algorithm. Figure 10.3 depicts the alternating blocks and written steps for a simple problem. We demonstrated the procedure to the children along with a great deal of verbalization to explain why each step was taken. Following the demonstration there were repeated trials in which the children did more and more of the work themselves. During this process more complex problems, including three-digit problems, were intro-

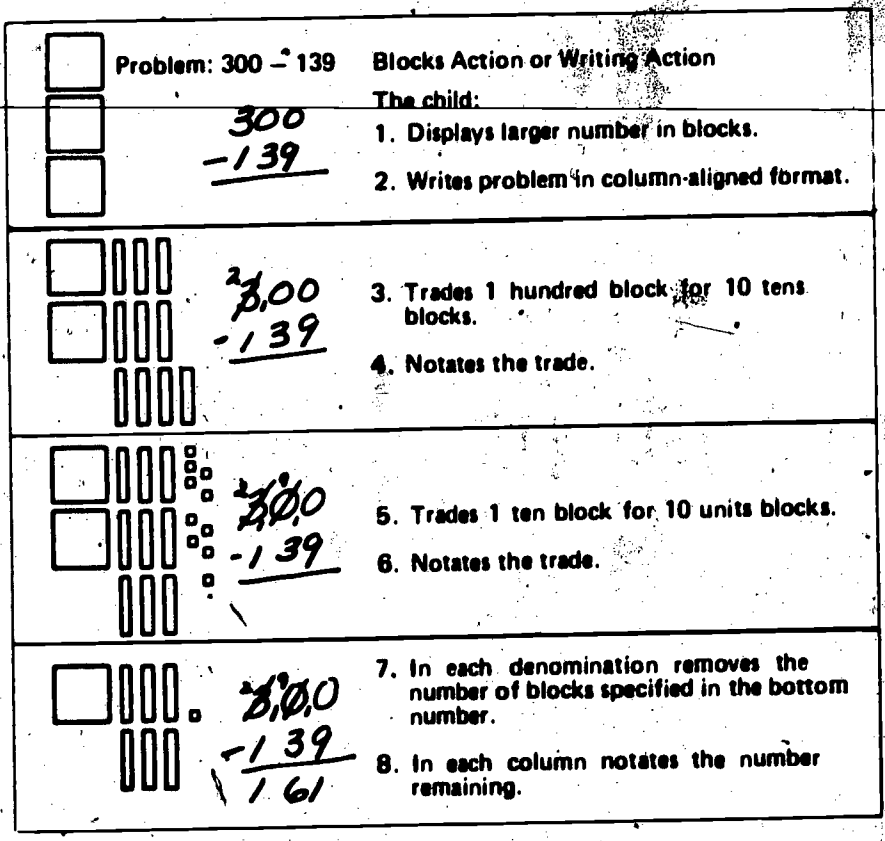


FIG. 10.3. Solution of a subtraction problem with blocks and a written algorithm.

duced. Eventually the child took over the entire process of manipulating blocks and writing the corresponding algorithmic step. Typically, within about forty minutes each student was performing the task smoothly. At this point the blocks could be removed (sometimes after a brief intermediate period in which "imaginary" blocks were manipulated) and purely written work undertaken.

The three children who have been taught this mapping procedure each began with a different diagnosed bug. Laura was doing Smaller-From-Larger; Molly, Don't-Decrement-Zero; and Ann, Borrow-Across-Zero. Each was brought back for a follow-up interview three to six weeks after the instruction. On this occasion each performed a series of written problems with ease. The bug was gone and no new ones had arisen. Further, each child remembered clearly what her original bug was and could explain what she had learned about why the bug was wrong.

All three children were questioned about what the various digits stood for, why they were being crossed out and written in, etc. Their answers suggest that they understood *why the algorithm works*, especially as their wording was not the same as the wording the experimenter used during instruction. They were also able to show correctly which blocks stood for the special borrowing notation digits (for example, a ten block for the small 1 written before the units digit), indicating a mapping between the two representations that had not been present prior to instruction. Finally, each child showed some interesting pattern of transferring semantic knowledge by inventing a procedure or an explanation that had not been taught.

Molly's performance was particularly striking, because she invented a justification for a (correct) written notation that did not exactly match the one produced during mapping instruction. The experimenter had asked her to do the problem, 2003 − 1467, and she had written:

Under questioning, she said she had borrowed one thousand; we then asked her to tell us where that thousand had been placed. She replied, "Well, 100 is right here (pointing to the 1 of the 13 and to the 9 in the tens column), and 900 is right here (pointing to the 9 in the hundreds column)."

WHY MAPPING WORKS

There is, of course, considerably more work to be done to establish the range of conditions under which this kind of mapping instruction will be effective. We will want to explore what kinds of bugs this instruction is capable of eliminating, and what initial knowledge of place value and addition and subtraction semantics is required for mapping to be effective. We will also need to examine the

possibility that certain bugs inherent in the blocks routines may unwittingly be adopted into the writing algorithm along with all the semantic corrections. Finally, we will want to consider the possibility that other representations—sticks, abaci, expanded notation, for example—may be equally or more powerful than Dienes blocks for the purpose discussed here, and the strong possibility that mappings between three representations may be even more powerful than between two in producing understanding of the written algorithms.

There is, however, enough evidence now in hand about the power of mapping instruction to warrant building a systematic theory of how mapping works to build understanding and to correct procedural errors. Although I do not attempt a formal theory here, in the remainder of this chapter I consider the possible shape of such explanatory theories. In the process, I review the kind of empirical evidence that might sharpen the theory-building effort.

There seem to be two possible accounts for the effects of mapping. One, the *prohibition* explanation, focuses on how pairing of each step in the written algorithm with a parallel step in blocks might serve simply to prohibit wrong operations in the writing. A second possibility, the *enrichment* explanation, is that semantic knowledge initially embedded in the blocks algorithm is, by mapping, applied to the rules for writing so that the newly enriched knowledge structure then eliminates bugs.

Prohibition

It is possible that most of the effect of the mapping instruction derives not from acquiring a deep understanding of the semantics of subtraction, but simply from the external constraints imposed by rules of the instructional situation. If the subtraction with blocks is performed correctly and if the rule of alternating operations in the blocks and the written representation is followed exactly, most of the known written subtraction bugs are impossible because they cannot be modeled with blocks without violating basic exchange principles. The way in which mapping between blocks and writing could prohibit writing bugs can best be appreciated by considering individual bugs.

Smaller-From-Larger is prohibited because in the blocks subtraction routine only the top number is represented in the blocks. The task is to remove the number of blocks specified in the bottom digits. As long as all digits of the top number are represented in blocks before any subtraction operations begin, there is no way of reversing top and bottom digits in a particular column, because only the top digit has been displayed.

The various bugs that arise when it is necessary to borrow from a column with zero as its top number are also prohibited in the mapping situation, provided the child knows that a fair trade must be made. When working with blocks the organizing goal is to get enough blocks to permit removing the specified number. These blocks must always come from the top number as it is the only one

represented. This constraint forces the child to go left until blocks are found. The child cannot add blocks to the zero column (Borrow-From-Zero) until something has been "borrowed" from the hundreds column. Similarly, Stops-Borrow-At-Zero is prohibited because the child cannot add blocks to a column without having borrowed from another column. Borrow-From-Bottom-Instead-Of-Zero is prohibited because the bottom number is not even represented. Borrow-Across-Zero is prohibited by details of the exchange rules: having borrowed from the hundreds column, the normal exchange (which we assume the child knows) will be for ten tens. Because the blocks have been arranged in columns for mapping instruction, the new blocks must be placed in the tens column, and this in turn requires some notation in that column. Also, a second trade is needed to fulfill the goal of getting more blocks in the active column. This second trade would have the effect of prohibiting Don't-Decrement-Zero, which changes the 0 to 10 but does not then decrement it to 9.

The point of this analysis is that it demonstrates that, to the extent that the rules of block exchanges and block subtraction are followed along with the rules of mapping, most of the operations associated with buggy algorithms are simply not possible.[1] Perhaps, then, the power of mapping instruction lies largely in providing a high-feedback environment in which the child's normal routine is prohibited and a new, permitted one is heavily practiced.

We are currently testing a form of pure prohibition instruction. Consideration of Brown and VanLehn's theory of the origin of bugs leads us to expect that pure prohibition instruction may eliminate bugs in the short run, but the bugs may reappear later or new buggy algorithms may replace them. Further, we expect that little transfer or invention of the kind shown by our three initial subjects will occur. If, as Brown and VanLehn argue, bugs derive from children's active attempts to invent procedures for dealing with new situations, then prohibition of wrong moves cannot by itself provide any basis for successful rather than maladaptive inventions. An instructor can prohibit an operation in writing, but if no new information is offered to the system, then the system will have no basis for responding with an appropriately modified set of operations.

Enrichment

For these reasons, we are inclined to believe that something more than prohibition took place in our mapping instruction, that there was some kind of enrichment of the knowledge structure that permitted the children to build some new connections and thus make sense of situations newly encountered some weeks

[1] A few bugs are not strictly prohibited by the blocks. Of the bugs shown in Fig. 10.2, Zero-Instead-Of-Borrow is physically possible. If prohibition were the only way in which mapping affected writing performance, one would, therefore, expect this bug to remain intact even after extensive mapping practice. This has not yet been tested in our work, as none of the children taught thus far had these particular bugs at the outset.

later. If we imagine that the children began the mapping instruction with an already well-developed justification for the operations in block subtraction based on the semantics of place value and subtraction, our theory will need to show how this semantic knowledge can be incorporated into the knowledge structure for written arithmetic as a result of mapping activity. Figure 10.4 shows the mapping links between a child's semantic knowledge and a written procedure.

The left side of Fig. 10.4 schematizes the semantic knowledge relevant to subtraction that is instantiated in the blocks routine. There are two aspects to this knowledge: (1) a goal structure for blocks subtraction and (2) knowledge about the kinds of exchanges allowed within the base-ten system. Only knowledge directly relevant to borrowing is shown, not the underlying knowledge about how base-ten information is coded in blocks. The goal structure that controls the exchange/borrowing process is shown at the far left. Constraints on each goal are shown to its right by a single arrow pointing to the goal. In each case a subgoal is generated in order to meet the goal. The main goal (*A*) is to remove the bottom quantity from the top—which, it is assumed, has been represented in blocks according to the rules specified in the base-ten code. This main goal requires that there be at least as many objects in the top of the active column as in the bottom. Otherwise it is necessary to get more blocks for the column (*B*), and this subgoal must be achieved without changing the quantity as a whole (i.e., while maintaining equivalence). This can be accomplished through trading. The dotted single arrow between getting more (*B*) and trading (*C*) reflects the finding, described earlier, that some children need prompting before they initiate trades to solve concrete subtraction problems. Trades are constrained by the requirement that there be both an increment and a decrement; and by the requirement that the trade be "fair" (ten for one). The ten-for-one relationship built into trades means that the only way to decrement from the hundreds column and increment in the units column is to do a double trade (*D*).

The right side of Fig. 10.4 represents the components of the written algorithm stripped down to include only the steps involved in borrowing. The paths for problems containing zeros and not containing zeros are shown in the left branch and the right branch respectively. The operations in the two branches can be done in several orders, but those shown are the orders frequently observed in algorithmic performance. Omission of one or more operations will produce buggy algorithms.

Operations-level mapping between blocks and written procedures would link each operation in the written procedure to a corresponding operation in blocks. The double arrows in Fig. 10.4 represent the kinds of links that mapping might be expected to produce. First, the top-level goal of the blocks routine (*A*) becomes linked to the written algorithm. A particular effect of this is to motivate testing (at *a*) whether the top number in a column is greater than or equal to the bottom, thus reducing the probability that this crucial step will be omitted. The next goal (*B*), getting more blocks, may also be linked to the written algorithm as indicated by the dotted box below *a*. Mapping of the trading goal (*C*), with its

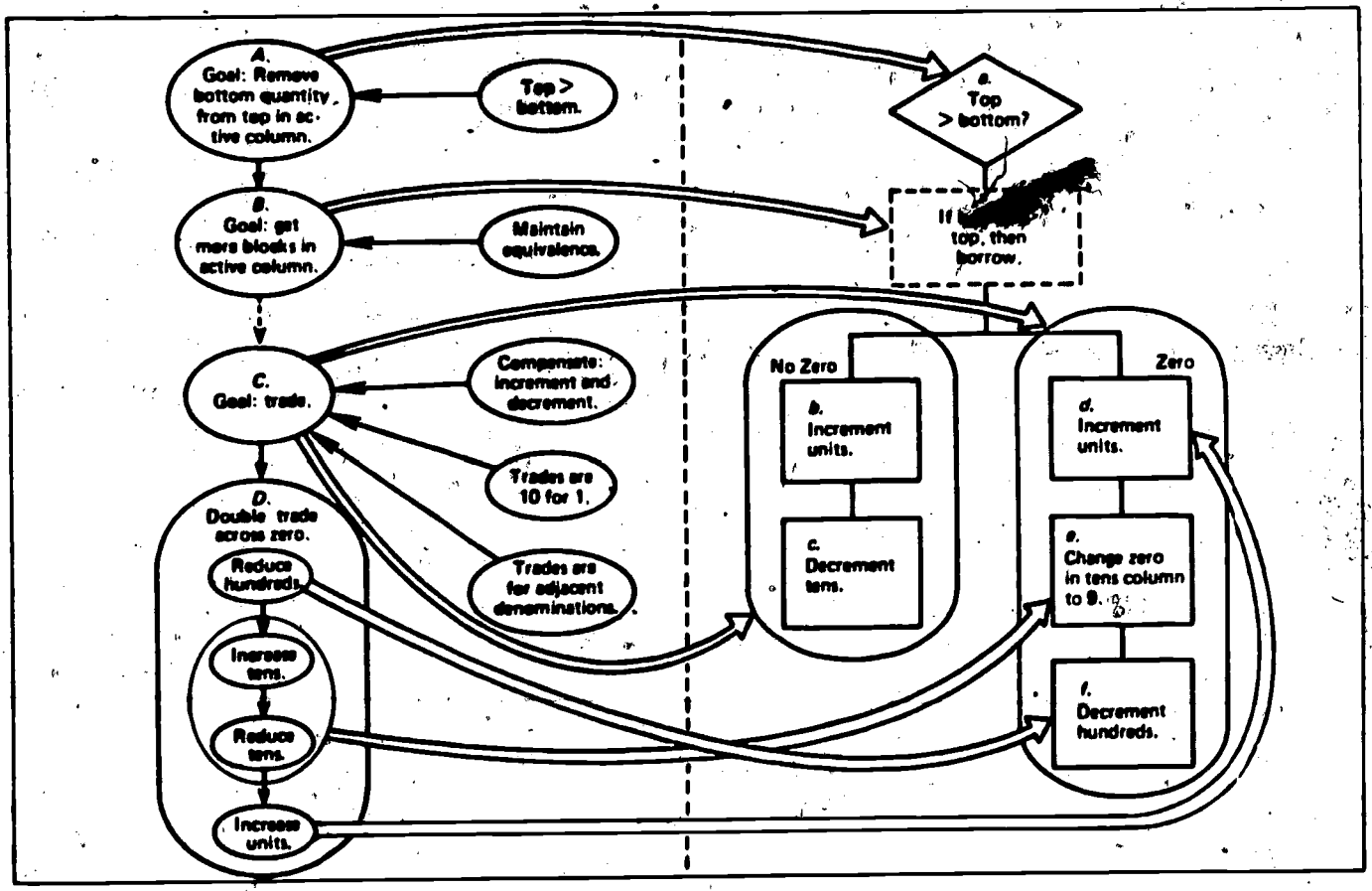


FIG. 10.4. Conceptual links facilitated by mapping.

constraints, can be expected to produce a "chunking" of incrementing and decrementing in one's mental representation of the written algorithm. This chunking in the algorithm is shown by the circles that now enclose steps *b* and *c* and steps *d*, *e*, and *f*. The chunking ensures that incrementing and decrementing are seen as complementary actions. For problems with zeros in the top number, the operations on zero are tied to the incrementing and decrementing operations that surround them. In addition to organizing the steps of borrowing into chunks—thus probably making omissions of individual steps less likely—the mapping has the effect of imposing the trading constraints (compensation, ten-for-one, and adjacency) on the written algorithm. Borrowing becomes, in effect, understood as an analog of trading, and the rules of trading become available as justification or explanation of the steps in borrowing. The links from the subop-erations within *D* to specific steps in the *d*-*e*-*f* sequence suggest how this may work. Probably the most important aspect of the detailed mapping is that it "explains" why 0 should be changed to 9—something that mystifies many children, according to our interview data.

This kind of enrichment of the knowledge structure can be expected to make the various steps in a correct writing algorithm easy to remember, because they are embedded in a knowledge network that justifies and explains them. Perhaps more important, the enriched knowledge structure should provide a basis for modifying or building routines that would make future repairs on subtraction algorithms likely to produce correct rather than buggy routines. The effect of enriched knowledge might be expected both at the point of *generating* possible operations and at the point of *testing* them. The semantic knowledge available from mapping would suggest potential incrementing and decrementing moves. Constraints that are part of the semantic knowledge would also serve to block possible incorrect operations that children might generate. The parallel between this informal account of the effects of mapping and Brown and VanLehn's (this volume) formal account of the origin of bugs suggests the possibility of a formalization of enrichment as an extension of their present repair theory. What is needed to guide such an effort is direct observation of the invention and repair of algorithms by children.

REFERENCES

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

Young, R. M., & O'Shea, T. Errors in children's subtraction. *Cognitive Science*, 1981, *5*(2), 153-177.