HUMAN-COMPUTER INTERACTION, 1985, Volume 1, pp. 107-131
Copyright © 1985, Lawrence Erlbaum Associates, Inc.

Novice LISP Errors: Undetected Losses of Information from Working Memory

John R. Anderson
Carnegie-Mellon University

Robin Jeffries
Hewlett-Packard Research Laboratories

ABSTRACT

Four experiments study the errors students make using LISP functions. The first two experiments show that frequency of errors is increased by increasing the complexity of irrelevant aspects of the problem. The experiments also show that the distribution of errors is largely random and that subjects' errors seem to result from slips rather than from misconceptions. Experiment 3 shows that subjects' errors tend to involve loss of parentheses in answers when the resulting errors are well-formed LISP expressions. Experiment 4 asks subjects, who knew no LISP, to judge the reasonableness of the answers to various LISP function calls. Subjects could detect many errors on the basis of general criteria of what a reasonable answer should look like. On the basis of these four experiments, we conclude that errors occur when there is a loss of information in the working memory representation of the problem and when the resulting answer still looks reasonable.

Authors' present addresses: John R. Anderson, Department of Psychology, Carnegie-Mellon University, Pittsburgh, PA 15213; Robin Jeffries, Hewlett-Packard Research Laboratories, 1501 Page Mill Road, Palo Alto, CA 94304. Request for reprints should be sent to John R. Anderson.

CONTENTS

1. INTRODUCTION

In numerous observations of students learning to program in LISP, we have noted that many errors are made in applying basic functions. These errors persist long after the student can demonstrate understanding of and mastery over the functions involved. In fact, in protocols of novices programming (Anderson, Farrell, & Sauers, 1984), these errors turn out to be a major determinant of the time it takes to write LISP code. Students write code that contains bugs due to errors in function application, and a great deal of time is spent debugging that code. Often, more time is spent tracking down errors in function application than correcting errors due to conceptual misunderstandings about the problem or the algorithm involved. The digressions caused by these errors not only consume huge amounts of relatively unproductive time, but they also sidetrack learners from the intended pedagogic points. The 15 min spent tracking down a bug caused by giving the wrong kind of argument to CONS may well overshadow the original point of t he exercise. Thus, reducing the frequency of errors in function application should both improve students' productivity and enable them to attend more fully to concepts being taught.

One possible interpretation of these elementary errors is that they are due to misconceptions about how the functions work. However, we frequently see the same subject use the same function correctly in another very similar context. We prefer an alternative interpretation based on processing load. When the processing demands of the current situation are large (where the meaning of large will change as the student gains experience), information will be lost from working memory. When the lost information includes partial results relevant to the current computation, errors will occur.

This paper represents our initial attempts to discover the distribution of errors students make when applying basic LISP functions and to explore the variables that contribute to their occurrence. In Experiments 1 and 2, we catalog the errors made by students in a LISP class on a fairly broad range of problems and show that variables that ought to contribute to processing load have the expected effect on error frequency. Experiment 3 extends these results to students

who are trained in the laboratory, and thus whose learning histories are more carefully controlled. In the first three experiments, we encounter several cases in which students are able to block errors when the answer they produce has certain properties (e.g., it is ill-formed). In Experiment 4, we explore the mechanisms subjects use to detect such errors and show that much of this knowledge is independent of and precedes any experience with LISP.

For readers who may be unfamiliar with the LISP programming language, we can describe its syntax in a few sentences. The basic building blocks of LISP are *atoms* and *lists*. An atom is a sequence of alphanumeric characters; a list is a sequence of atoms and other lists, enclosed in parentheses. LISP encodes both data and programs as lists. A LISP *function*, or program, is a list, where the first element is interpreted as a function to be evaluated and succeeding elements are the function arguments, which may involve function calls in turn. Each argument is evaluated in the same way as the outermost function, unless an argument is preceded by a single quote mark (which blocks evaluation and causes the element which follows it to be interpreted literally). The semantics of the individual functions that were used in our experiments are described as needed in the text.

2. EXPERIMENT 1: EFFECTS OF COMPLEXITY

In our observations of students writing LISP functions, we often saw them make errors in applying functions that they were able to use correctly in similar situations. Such errors seemed to occur more frequently when students were writing comparatively difficult functions. This led us to hypothesize that these errors might be caused by excessive demands on students' working memory. While we originally observed this pattern of errors in the context of students writing their own functions, we decided to explore the relationship of increased processing load to performance using the simpler task of solving LISP equations.

We chose to look first at two basic combining functions in LISP: CONS and LIST. Both functions can take two arguments and create a new list that is the concatenation of their arguments. CONS creates a list whose first element is the first argument and whose remaining elements are the elements of the second argument; for example, (CONS '(a b) '(c d)) ⇒ ((a b) c d). LIST also creates a list whose first element is its first argument and whose second element is the second argument; for example, (LIST '(a b) '(c d)) ⇒ ((a b) (c d)). We, and most LISP instructors, have observed many errors involving these functions and confusions between them.

We created three different kinds of problems for subjects to solve. For a particular LISP equation involving a function (LIST or CONS), two arguments, and a result, a student would be asked: (a) to provide the function, given the arguments and the result (a *function-naming* problem); (b) to provide the second

argument, given the function, the first argument, and the result (an *argument-provision* problem); or (c) to provide the result, given the function and a pair of arguments (an *evaluation* problem). We never explicitly told subjects that the correct answers to function-naming problems were limited to LIST and CONS. Examples of the three types of problems are:

Function naming: (?'(a) '((b) (c) (d))) ⇒ ((a) (b) (c) (d))
Argument provision: (CONS 'x ?) ⇒ (x (y z))
Evaluation: (CONS 'a '(b c d)) ⇒ ?

We chose two methods to increase the complexity of the problems. One was to add an extra level of parentheses around the lowest level elements of both arguments. Thus, the first problem just listed would become:

(? '((a)) '(((b)) ((c)) ((d)))) ⇒ (((a)) ((b)) ((c)) ((d)))

The second method was to embed the second argument in a call to the function REVERSE. REVERSE reverses the order of the highest level elements of its argument; for example, (REVERSE '((a b) c d) ⇒ (d c (a b)). In this case, the second problem would become:

(? '(a) (REVERSE '((b) (c) (d)))) ⇒ ((a) (d) (c) (b))

Notice that although both complexity manipulations should increase the amount of work needed to produce parts of the answer, they do not change the procedure to be used to apply CONS and LIST to calculate the overall structure of the answer. Consider the following as a possible procedure for solving a LIST evaluation problem: (a) find and copy down the first argument, (b) find and copy the second argument to the right of the first, (c) write a pair of parentheses around the two arguments just copied. Adding additional parentheses to the elements of the second argument adds to the amount of work involved in producing a copy of the second argument. Similarly, embedding the second argument in a call to REVERSE also increases the amount of work in calculating what that argument should be. However, in both cases the same steps, (a) - (c), must be executed to evaluate LIST and put its arguments together. We distinguish between errors involved in producing the parts and errors involved in putting the parts together in an overall answer. Except for capacity limitations, there is no reason to expect that manipulation in the complexity of the parts should affect number of errors in putting the parts together.

The three types of problems (evaluation, argument provision, and function naming), two functions (LIST and CONS), the extra level of parentheses, and the use of REVERSE were factorially combined to create 24 problems. These were broken into two sets of 12; 30 subjects were tested with one set and 29 with the other. A complete list of the 24 problems used is given in Figure 1.

The subjects were students in a course taught in the fall of 1982 on artificial intelligence and human information processing. They had been assigned the

Figure 1. Problems used in Experiment 1. The problems in Set A and Set B were seen by different groups, each group being one half of the class. Approximately half of each of these groups saw the problems in reverse order.

Set A:

(? '(A) '((B) (C) (D))) = ((A) (B) (C) (D))

(LIST '((P)) (REVERSE '((Q) (R) (S)))) = ?

(LIST '(M) ?) = ((M) ((N) (O)))

(CONS 'X (REVERSE ?)) = (X (Y Z))

(? '(I) (REVERSE '((J) (K) (L)))) = ((I) ((L) (K) (J)))

(CONS '((E)) '((F) (G) (H))) = ?

(CONS '(V) (REVERSE ?)) = ((V) ((W) (X)))

(LIST 'R ?) = (R (S T))

(? '((Q)) '(((R)) ((S)) ((T)))) = (((Q)) ((R)) ((S)) ((T)))

(LIST '(D) (REVERSE '(E F G))) = ?

(? '((K)) (REVERSE '(((L)) ((M)) ((N))))) = (((K)) (((N)) ((M)) ((L))))

(CONS '(A) '(B C D)) = ?

Set B:

(? '(A) (REVERSE '((B) (C) (D)))) = ((A) (D) (C) (B))

(LIST '((P)) '((Q) (R) (S))) = ?

(LIST '(M) (REVERSE ?)) = ((M) ((N) (O)))

(CONS 'X ?) = (X (Y Z))

(? '(I) '((J) (K) (L))) = ((I) ((J) (K) (L)))

(CONS '((E)) (REVERSE '((F) (G) (H)))) = ?

(CONS '(V) ?) = ((V) ((W) (X)))

(LIST 'R (REVERSE ?)) = (R (S T))

(? '((Q)) (REVERSE '(((R)) ((S)) ((T))))) = (((Q)) ((T)) ((S)) ((R)))

(LIST '(D) '(E F G)) = ?

(? '((K)) '(((L)) ((M)) ((N)))) = (((K)) (((L)) ((M)) ((N))))

(CONS '(A) (REVERSE '(B C D))) = ?

first four chapters of Winston and Horn (1981) as well as a series of exercises based on the material in the chapters. They all should have been quite familiar with LIST and CONS. In case they were not familiar with REVERSE, it was explained to them before the test. They were told that they had no more than 8 min to solve the problems; they were to solve each as quickly as was compatible with giving correct answers. In fact, all students solved their 12 problems in the 8 min allocated.

Because we want to show that increasing the complexity of the argument will affect unrelated aspects of the solution, we did not score as errors responses whose only mistake was a misapplication of REVERSE. That is, if the answer a subject gave differed from the correct response only in that the subject either failed to apply the REVERSE operation or applied it incorrectly, we counted that as a correct answer. Similarly, we did not score as errors any problem associated with reproducing the parentheses around the individual elements of each list. Thus, none of the errors we report can be specifically attributed to the complexity of the problem to be solved. If there is an increase in errors due to problem complexity, the increased complexity must be influencing other aspects of the solution process.

Figure 2 presents the results of the experiment in percentage correct, classified according to the 24 problems. First, subjects are much better with LIST (73% correct) than with CONS (45% correct). This difference holds for all types of problems but is largest for argument provision and smallest for function naming. Subjects are more accurate when they do not have to deal with REVERSE (63% vs. 55%), $\chi^2(1, N = 708) = 4.93$ which is significant at the .05 level. The effect of number of parentheses is marginal (61% vs. 57%) and is not statistically significant, $\chi^2(1, N = 708)$. The extreme effort of complexity, comparing the condition involving both extra parentheses and REVERSE to the one with neither, is 65% versus 51%, a fairly substantial difference. Thus, it does seem that there is a basis for our observation that errors increase with increases in other independent but concurrent information-processing demands.

The kinds of errors made by subjects on these problems were far from random. Figure 3 categorizes the errors found on each problem type for each function. For function-naming problems, subjects show a strong bias for LIST and CONS in their responses, but they mention the only other combining function they know, APPEND, fairly often. For the other problems, almost all errors are due to adding or dropping balanced sets of parentheses. Even the errors categorized as *other* primarily involve misparenthesizations (e.g., unbalanced parentheses). Each problem type for each function shows a different pattern of preferred errors. Notice that for evaluation problems, there are several common errors. On CONS problems, the errors are divided approximately evenly between dropping parentheses and adding parentheses. Almost all of the added parentheses involve the first argument. On LIST evaluation problems,

Figure 2. **Proportion correct in Experiment 1.**

CONS				
	Recognition	Evaluation	Argument Provision	Mean
Base condition	.55	.41	.48	.48
Reverse	.68	.37	.28	.44
Extra parentheses	.55	.34	.52	.47
Extra parentheses and reverse	.41	.52	.28	.40
Mean	.55	.41	.39	.45
LIST				
	Recognition	Evaluation	Argument Provision	Mean
Base condition	.74	.78	.86	.79
Reverse	.59	.66	.93	.73
Extra parentheses	.70	.63	1.00	.78
Extra parentheses and reverse	.34	.52	1.00	.62
Mean	.59	.65	.95	.73

most of the errors are due to dropping parentheses, and most of these are due to dropping parentheses surrounding one or both elements.

The fact that a small number of error types accounts for the vast majority of errors on each problem type seems consistent with the notion that misconceptions about how the functions work might underlie the pattern of errors we see. In fact, one could make the stronger claim that there must only be a few such misconceptions, each of which is held by a fairly large number of subjects, to produce such a consistent pattern of errors. However, what is a consistent pattern at the aggregate level may look quite different at the individual subject level. If subjects are harboring a misconception about a function, they ought to make the same error on all, or almost all, problems of a given type involving

Figure 3. **Percentages of particular errors as a function of problem type.**

	CONS	LIST
Function-naming problems:		
LIST	63%	—
CONS	—	67%
APPEND	33%	27%
other	4%	7%
Argument-provision problems:		
dropping parentheses	94%	14%
adding parentheses	0%	86%
other	6%	0%
Evaluation problems:		
dropping parentheses	45%	73%
adding parentheses	37%	13%
adding and dropping	6%	0%
other	12%	15%

that function. On the other hand, if these errors are due to processing overload, then we expect a more variable pattern of errors; subjects are likely to make an error on one problem and solve an equivalent problem correctly or to make different errors on equivalent problems.

Each subject solved six pairs of equivalent problems; we consider two problems to be equivalent if they involve the same function (LIST or CONS) and if the task is to provide the same component (function, argument, or result). The two problems for each pair, of course, differ on one of the complexity manipulations. Figure 4 shows the number of times each possible pattern of errors occurred for each problem type (function naming, argument provision, evaluation). For a given pair of problems, a subject could: (a) answer both correctly; (b) answer one right and one wrong; (c) answer both incorrectly, but make two different kinds of errors (e.g., drop a set of parentheses on one and add an extra set of parentheses on the other); or (d) make the same error on both problems. Notice that each subject contributes two observations to each column: once for

Figure 4. **Number of times subjects gave consistent or inconsistent responses to a pair of equivalent problems.**

	Function Naming	Argument Provision	Evaluation
Both correct	51	74	44
1 right/1 wrong	36	12	41
2 wrong/different errors	9	0	9
2 wrong/same error	22	32	24

the pair of equivalent CONS problems of that type and once for the pair of LIST problems.

For function-naming problems, there were 67 pairs in which one or more errors were made. In 45 (67%) of them, the subject gave different forms of answers for the two equivalent problems, either making two different errors or giving the correct answer to one and making an error on the other. For evaluation problems, a similar result holds. In 50 of the 74 cases (68%) where at least one error was made, the subject gave two different types of response. For argument-provision problems, there is a consistent pattern of errors that might imply that some difficulty is responsible for the errors observed. Of the 44 times subjects made one or more errors on a pair, only 12 (27%) showed a different pattern of responses for the two problems.

Closer examination of the specific errors made on argument-provision problems leads to an alternative interpretation of these data. The errors made on argument-provision problems are almost totally confined to CONS problems. For LIST argument-provision problems, there were six cases where subjects made an error on one problem of a pair, and only one case where two errors were made. Although these LIST data are too sparse to infer any trend, they are not inconsistent with the pattern observed on the other problem types. Thus, if there is a misconception that leads subjects to make a consistent error, it is restricted to CONS problems. The pattern of results on the CONS argument-provision problems is: 22 cases of no error, 6 cases of one error, and 31 cases of two (equivalent) errors, producing a different pattern of responses for the two problems of a pair on only 16% of the 37 cases where an error was made.

If we look at the specific error made by the subjects who made the same error on both CONS argument-provision problems, we find that 27 of the 31 subjects dropped a pair of parentheses from the answer. That is, given a problem of the form (CONS 'x ?) ⇒ (x (y z)), subjects wrote '(y z) rather than '((y z)), which is the correct answer. This is potential evidence of a systematic misconception

about CONS that was common to a large number of our subjects. However, whatever the nature of this misconception, it did not generalize to other types problems involving the function CONS. Figure 5 categorizes the errors made by these 27 subjects on CONS function-naming and evaluation problems. These subjects are no more consistent in the errors they make on these problems than the subjects as a whole are. Considering the cases where at least one error is made, for 60% of the CONS function-naming problems and for 58% of the CONS evaluation problems, these subjects give a different type of response for each of the two problems of a pair.

Furthermore, even when the same error is made on both problems of a pair, there is no particular consistency across subjects as to what this error is. Of the 10 subjects who made the same error twice on the CONS evaluation problems, 5 dropped a pair of parentheses from the answer and 5 added a pair of parentheses. Only 8 of the 30 subjects made the same error twice on CONS function-naming problems, but they all gave LIST as their response. This is not surprising; although we did not tell subjects that LIST and CONS were the only functions they were being tested on, we expect that many of them figured it out.

To what do we attribute the pattern of errors we see on CONS argument-provision problems? First, the subjects who make errors on both these problems are relatively weak overall. They made an average of 4.9 errors on the other 10 problems, whereas the subjects who made 1 or fewer errors on CONS argument-provision problems made only 2.4 errors on the rest of the problems. Second, when we look at the possible errors that subjects might reasonably make on CONS argument-provision problems, the error of dropping a pair of parentheses is the only candidate. The filtering process to be discussed later should eliminate the errors that we see on other problems or that we can imagine subjects might make. Thus, our interpretation of these data is that the subjects who made two consistent errors on CONS argument-provision problems are simply particularly weak in their understanding of LISP; thus, perhaps they were more likely to experience processing overload and were taken in by the only "sensible" error for these problems.

3. EXPERIMENT 2: REPLICATION

In order to establish the generality and robustness of the effect of problem complexity on errors made in solving LISP equations, we decided to replicate and extend the results of Experiment 1. Another experiment was done at the same point in the class during the fall of 1983. This time, we also included problems using APPEND, the third major LISP combining function; we also used different specific problems. APPEND differs from LIST and CONS in that it combines its arguments into a new list containing the elements of the first argument followed by the elements of the second argument, for example, (APPEND '(a b) '(c d)) ⇒ (a b c d)). With three functions, three problem types, two

Figure 5. **Number of times subjects who made a consistent error on CONS argument-provision problems gave consistent or inconsistent responses to other CONS problems.**

	CONS Function Naming	CONS Evaluation
Both correct	7	3
1 right/1 wrong	7	8
2 wrong/different errors	5	6
2 wrong/same error	8	10

levels of parenthesization, and the presence or absence of REVERSE, there are 36 possible problems. A new set of problems was created and sorted into two random orders; booklets of all 36 problems were handed out to a class of 75 students. The students were again encouraged to work quickly but accurately. Because of class constraints, there were only 12 min to solve the problems, and many students failed to solve all 36. Therefore, the number of subjects contributing to any observation varies from 42 to 65. The 36 problems used for this experiment are given in Figure 6.

Figure 7 presents the results of this experiment organized according to the same format as Figure 1 from Experiment 1. The table shows an effect of adding REVERSE (78% correct without REVERSE vs. 74% correct with), which is marginally significant and in the same direction as the significant difference found in the previous experiment. This time, the effect of an additional level of parentheses is significant (79% vs. 73%), $\chi^2(1) = 8.89$, $p < .01$. Comparing the extremes of problems involving neither REVERSE nor extra parentheses to those containing both, the difference is 82% versus 72%. Thus, the reasonable conclusion from the two experiments combined is that either method of increasing complexity will reduce performance.

Looking at overall performance, subjects do best with APPEND problems (84% correct), less well with LIST problems (76%), and worst with CONS problems (69%). However, performance with CONS problems is substantially better than it was in Experiment 1 (69% vs. 45%). This is due to exceptionally good performance on CONS argument-provision problems (91% in Experiment 2 vs. 39% in Experiment 1). The difference seems to be due to a difference in the form of these problems between the two experiments. Here are equivalent example problems from the two experiments:

Experiment 1: (CONS 'x ?) ⇒ (x (y z)); correct answer is ((y z))
Experiment 2: (CONS '(x) ?) ⇒ ((x) y z); correct answer is (y z)

Figure 6. Problems used in Experiment 2. Approximately half of the subjects saw these problems in the reverse order.

(APPEND '((V)) ?) = ((V) (W) (X))
(CONS '(X) ?) = ((X) Y Z)
(LIST '((P)) (REVERSE '((Q) (R) (S)))) = ?
(CONS '((E)) (REVERSE '((F) (G) (H)))) = ?
(? '((Q)) '(((R)) ((S)) ((T)))) = (((Q)) ((R)) ((S)) ((T)))
(LIST '(D) '(E F G)) = ?
(? '(I) '((J) (K) (L))) = (((I) ((J) (K) (L)))
(CONS '((E)) '((F) (G) (H))) = ?
(APPEND '((V)) (REVERSE ?)) = ((V) (W) (X))
(CONS '(X) (REVERSE ?)) = ((X) Y Z)
(LIST '((M)) ?) = (((M)) ((N) (O)))
(APPEND '(J) ?) = (J K L)
(APPEND '((L)) (REVERSE '((M) (N) (O)))) = ?
(CONS '((V)) ?) = (((V)) (W) (X))
(? '(E) '((F) (G) (H))) = (E (F) (G) (H))
(APPEND '((L)) '((M) (N) (O))) = ?
(LIST '(R) ?) = ((R) (S T))
(? '((K)) (REVERSE '(((L)) ((M)) ((N))))) = (((K)) (((N)) ((M)) ((L))))
(? '((N)) (REVERSE '(((O)) ((P)) ((Q))))) = ((N) ((Q)) ((P)) ((O)))
(? '(A) (REVERSE '((B) (C) (D)))) = ((A) (D) (C) (B))
(? '(E) (REVERSE '((F) (G) (H)))) = (E (H) (G) (F))
(? '((K)) '(((L)) ((M)) ((N)))) = (((K)) (((L)) ((M)) ((N))))
(APPEND '(C) '(D E F)) = ?
(CONS '(A) '(B C D)) = ?
(LIST '(R) (REVERSE ?)) = ((R) (S T))
(APPEND '(J) (REVERSE ?)) = (J K L)
(? '(A) '((B) (C) (D))) = ((A) (B) (C) (D))
(LIST '((M)) (REVERSE ?)) = (((M)) ((N) (O)))
(CONS '((V)) (REVERSE ?)) = (((V)) (W) (X))
(? '(I) (REVERSE '((J) (K) (L)))) = ((I) ((L) (K) (J)))
(LIST '(D) (REVERSE '(E F G))) = ?
(? '((N)) '(((O)) ((P)) ((Q)))) = ((N) ((O)) ((P)) ((Q)))
(? '((Q)) (REVERSE '(((R)) ((S)) ((T))))) = (((Q)) ((T)) ((S)) ((R)))
(CONS '(A) (REVERSE '(B C D))) = ?
(LIST '((P)) '((Q) (R) (S))) = ?
(APPEND '(C) (REVERSE '(D E F))) = ?

Figure 7. **Proportion correct in Experiment 2.**

CONS				
	Recognition	Evaluation	Argument Provision	Mean
Base condition	.77	.63	.94	.78
Reverse	.67	.52	.88	.69
Extra parentheses	.63	.38	.93	.65
Extra parentheses and reverse	.59	.46	.89	.65
Mean	.67	.50	.91	.69
LIST				
	Recognition	Evaluation	Argument Provision	Mean
Base condition	.73	.73	.91	.79
Reverse	.65	.77	.93	.78
Extra parentheses	.76	.58	.93	.76
Extra parentheses and reverse	.77	.38	.93	.69
Mean	.73	.62	.93	.76
APPEND				
	Recognition	Evaluation	Argument Provision	Mean
Base condition	.80	.87	.96	.88
Reverse	.76	.78	.95	.83
Extra parentheses	.66	.89	.96	.84
Extra parentheses and reverse	.71	.81	.94	.82
Mean	.73	.84	.95	.84

These two examples differ in the way the arguments to CONS are parenthesized. The relevant difference is the extra pair of parentheses around the second argument in the version from Experiment 1. The error made 94% of the time for these problems in Experiment 1 was to drop one pair of parentheses around the answer. If subjects had done this in Experiment 2, they would have generated y z rather than (y z). Presumably, they could reject this answer because it is an ill-formed (syntactically invalid) LISP expression. Thus, it appears that subjects can detect errors in their procedures if the answer generated produces a nonplausible result.

4. EXPERIMENT 3: CONTROLLED LEARNING SITUATION

The first two experiments explored the types of errors typically made by students solving LISP equations after a few weeks exposure to the language. We saw that the great majority of mistakes involves errors in parenthesization and that these errors are more common when students are working under higher processing demands. We also saw evidence that students were less likely to make errors when the resulting expression violates the syntactic rules of LISP. In the next experiment, we attempted to extend these conclusions to a more controlled learning environment and some additional LISP functions. We brought 20 students into the laboratory and taught them four basic LISP functions: CAR, CDR, CONS, and LIST. We then tested them on a large number of problems involving these four functions.

CAR and CDR are the basic LISP functions for extracting component parts of lists. CAR returns the first element of its argument; CDR returns the list containing all the elements of its argument other than the first. To wit:

(CAR '(a b c)) ⇒ a
(CDR '(a b c)) ⇒ (b c)

CAR and CDR are closely related to CONS:

(CONS (CAR '(a b c)) (CDR '(a b c))) ⇒ (a b c)

That is, the first argument to CONS becomes the CAR of the result, and the second argument becomes the CDR.

We generated a large number of problems for our subjects to solve using the following algorithm. For CONS and LIST problems, we first generated four candidate-first arguments:

a
(a)
((a))
(a b)

We then factorially combined them with the following six second arguments:

(m)
((m))
(((m)))
(m n)
(m n o)
((m n))

This produces 24 equation templates for CONS and 24 for LIST, generated by evaluating the function with each possible argument pair. From each equation, we generated four problems, each one asking the subject to supply a different component part: the function, the first argument, the second argument, or the result. This proceduces 96 CONS problems and an equivalent 96 LIST problems.

For CAR and CDR problems, we generated 24 equation templates for each function by taking the 24 results from the CONS problems we had generated and using them as arguments to CAR and CDR problems. Thus, the result for each CAR problem was one of the four candidate-first arguments to the CONS and LIST problems, and the result for each CDR problem was one of the six candidate-second arguments. From each of these templates, we created three problems asking the subject to supply the function, the argument, or the result. This produced 72 CAR problems and 72 CDR problems.

A total of 20 subjects were brought into the laboratory, taught the four basic functions, and then presented with the 336 problems in random order on a CRT display. The problems were presented in blocks of 48, with rest breaks between each block. Subjects typed their responses using the CRT keyboard; they were given immediate feedback as to the correctness of the answer. Subjects were told that they were being timed (which was true) and were encouraged to generate their answer as rapidly as was compatible with maintaining a high level of accuracy. They were able to correct mistakes as they made them by canceling the input and retyping the entire answer. The entire experiment took less than 2 hr to complete. We excluded the first 48 problems for each subject from the analysis so as to minimize warm-up effects.

Overall, 11.5% of the responses were in error. Performance on LIST problems was, as before, the best, with 8.3% of the responses containing errors. CONS and CAR were of intermediate difficulty with 10.2% and 11.0% errors, respectively. Subjects had the most difficulty with CDR problems, making 17.3% errors.

Performance also varied with the type of problem. Function-naming problems were the easiest, producing only 7.7% errors. Argument provision was next in difficulty with 10.5%. The argument-provision problems varied greatly in difficulty depending on whether the problem involved a separating or combining function. Subjects made only 7.0% errors when providing argu-

ments to the combining functions LIST and CONS, but made 17.5% errors when providing arguments to the separating functions CAR and CDR. We attribute the exceptional difficulty of these latter problems to the fact that subjects needed to make up material to include in the answer (i.e., they had to create a list that contained the required CAR or CDR). The cognitive demands of generating new list components led to many errors, often unrelated to the made-up material.

We attempted a detailed analysis of the errors subjects made in problems that involved writing lists (i.e., everything by the function-naming problems). Due to a disk crash, we have detailed data for only 11 of the original 20 subjects. In this subset of the data, 17.3% of the responses were errors. Of all the errors, 17% involved a mistyping of the atoms given in the problem. The remaining 83% were due to problems in the placing of parentheses around the terms. Dividing up the 83% parentheses errors, 30% of the errors were due to misbalanced parentheses, and the remaining 53% were cases of balanced parentheses. Dividing up the 53% balanced parentheses, 14% of the errors involved too many parentheses, and 39% involved too few. Thus, the overall results are quite consistent with the previous experiments, both in relative difficulty of the problems and in the fact that the largest number of errors involves dropping a balanced pair of parentheses.

Individual subjects showed approximately the same patterns of errors as the aggregate data, but with more variability, of course. There were no cases where a subject missed all or most of the problems in a particular group of equivalent problems, even when we separate problems on the basis of the form of their arguments, as well as function and problem type. This could be attributed to the fact that subjects received feedback on the correctness of each answer and thus could modify their understanding of the correct procedure for solving these problems as the experiment progressed.

When we look more specifically at the patterns of errors as a function of problem type, function, and form of argument, large variations in difficulty are apparent. Error rates range from 0 to 37%. To better understand the sources of the higher error rates, we classified problems according to the role played by the form of the second argument of the original problem template, which we call the *target structure*. In some problems, subjects never even see this target structure; in others, they see it, but it is not part of the answer they have to generate; in still others, they have to generate either an expression containing the target structure or exactly the target structure itself. The form of the target structure has differing effects on problem difficulty depending on how it it used.

We can classify the 14 problem types into four classes, varying in the role the target structure plays in the answer. Class 1 includes only CAR argument-provision problems. For these problems, the value of the target structure is irrelevant because the subject neither sees it nor has to generate it. In Class 2

problems, the subject sees the target structure but does not have to include it as part of the answer. The four function-naming problems, CONS and LIST first argument-provision problems, and CAR evaluation problems fall in this category. Class 3 problems require the subject to embed the target structure inside another expression. In these problems, the answer given will be a valid LISP expression both with and without the outermost parentheses of the target structure. Class 3 problems are CONS and LIST evaluation problems and CDR argument-provision problems. Finally, Class 4 problems require the subject to give the precise target structure as the answer. Thus, for some forms of the target structure, such as, (m n) and (m n o), dropping the outermost parentheses will result in a syntactically illegal LISP expression. The problems in Class 4 are CONS and LIST second argument-provision problems and CDR evaluation problems. We might expect these four classes problems to differ in their sensitivity to the complexity of the target structure.

Figure 8 presents the data of the experiment classified according to problem type and target structure, and separated into the four classes of problem types. The averages in Figure 8 are based on approximately 80 observations per cell.

It is only for Class 4 problem types that there appears to be an effect of target structure. Those target structures for which dropping parentheses would result in atoms—(m), (m n), and (m n o)—result in 92% correct; in contrast, those for which dropping parentheses would still result in a list structure—((m)), (((m))), ((m n))—result in 80% correct. None of the other classes show this effect. For Class 1, it is 83% versus 84%; for Class 2, it is 93% versus 91%; for Class 3, it is 84% versus 86%. We had predicted higher performance on the problems involving (m n) and (m n o) for Class 4 because subjects could reject as ill-formed an answer with dropped parentheses. Apparently, the fact that an atom results from dropping the parentheses from (m) is sufficiently strange to cause subjects to catch this error. Therefore, (m) behaves like (m n) and (m n o).

Thus, the seductiveness of errors that produce reasonable-looking LISP expressions leads to higher error rates for certain target structures (i.e., those for which dropping a pair of parentheses, the most frequent error, leads to a sensible-looking expression). We can further substantiate this claim by examining the effect of the target structure variable on parentheses-dropping errors specifically. We compared the frequency of such errors for problems that involve single-level lists—target structures of (m), (m n), or (m n o)—versus multilevel lists—target structures of ((m)), (((m))), or ((m n))—for the various classes of problems. We found that 6% of all Class 1 and Class 2 problems (excluding Class 2 function-naming problems) involved dropped parentheses, and this did not vary with the level of the target structure; similarly, 4% of all Class 3 problems involved dropped parentheses, and this did not vary with level. However, for Class 4 problems, 6% of single-level target structures involved dropped parentheses, whereas 12% of multiple-level problems involved dropped parentheses.

Figure 8. **Proportion correct in Experiment 3.**

	Target Structure						
	(M)	((M))	(((M)))	(M N)	(M N 0)	((M))	Mean
Class 1:							
Argument to CAR	.86	.90	.78	.79	.83	.83	.83
Class 2:							
Naming CONS	.90	.82	.82	1.00	.97	.96	.91
1st argument to CONS	.93	.92	.93	.91	.96	.97	.94
Naming CAR	.97	.98	.95	.95	.97	.93	.96
Evaluating CAR	.82	.97	.79	.97	.86	.86	.88
Naming CDR	.95	.96	.95	.99	.89	.78	.92
Naming LIST	.81	.91	.84	.93	.90	.99	.90
1st argument to LIST	.98	.95	.88	.98	.98	.86	.94
Mean	.91	.93	.88	.96	.93	.91	.92
Class 3:							
Evaluating CONS	.77	.79	.92	.85	.85	.81	.83
Argument to CDR	.86	.90	.78	.79	.78	.83	.82
Evaluating LIST	.85	.86	.88	.91	.85	.95	.88
Mean	.83	.85	.86	.85	.83	.86	.85
Class 4:							
2nd argument to CONS	.92	.93	.86	.92	.97	.88	.91
Evaluate CDR	.80	.67	.63	.80	.92	.63	.74
2nd argument to LIST	.98	.87	.93	1.00	.96	.84	.93
Mean	.90	.82	.81	.91	.95	.78	.86

Overall, the results of this experiment are consistent with the results found earlier. Errors could not be attributed to enduring misconceptions; the most frequent error was loss of a pair of parentheses; and in cases where the result of dropping parentheses resulted in a reasonable-looking LISP expression, there was a particularly high rate of dropping.

5. EXPERIMENT 4: PRIOR KNOWLEDGE ABOUT ACCEPTABLE ANSWERS

The errors in Experiments 1-3 followed a consistent pattern. They almost always involved the gain or loss of parentheses; loss or gain of a pair of parentheses was more common than errors involving unbalanced parentheses; parentheses were added or deleted from entire components (the argument or the entire expression), rather than from individual elements of the component; and errors were much less common when the loss of a pair of parentheses led to an ill-formed LISP expression. This led us to hypothesize that students have mechanisms for rejecting certain kinds of answers as implausible. Furthermore, we believe that at least some of the knowledge about what makes an answer sensible should precede and be independent of knowledge of the LISP programming language. The purpose of Experiment 4 was to test these assumptions.

We asked subjects to look at a large collection of LISP equations and to find those they thought contained errors. Six LISP functions were used: three combining functions, CONS, LIST, and APPEND, and three extracting functions, CAR, CDR, and LAST. The function LAST returns the list containing the last element of its argument list, for example, (LAST '(a b c)) ⇒ (c). At the beginning of the experiment, none of the subjects knew anything about LISP. Half of them were given 15 min of instruction on what a valid list is (i.e., that it has an outermost set of parentheses; that all its elements are atoms or other lists), including examples and exercises.

The equations that subjects saw were of three types. The first group contained what we call *filterable errors*, that is, errors that subjects ought to be able to detect due to the implausibility of the answer, independent of any knowledge of the semantics of the LISP functions involved. The filterable errors were of two kinds: *list-structure-relevant* and *list-structure-irrelevant*. The list-structure-irrelevant errors included such things as: in a two argument function, having only one of the arguments appear in the answer, or adding or deleting a (non-top-level) component of an argument. The list-structure-relevant errors either produced an answer that was not a valid list expression or added or deleted parentheses within the structure of an argument. A complete list of the filterable errors, with examples, is given as Figure 9. Notice that some errors apply only to extractor or combiner functions; there are 10 distinct filterable errors for each function.

Figure 9. **Filterable errors used in Experiment 4, with examples.**

List-structure Irrelevant:

New atom added to answer: (CAR '((a) b)) ⇒ (a c)

Atom in argument changed in answer: (CAR '((a b) c)) ⇒ (a d)

Atoms rearranged in answer: (CAR '((a b) c)) ⇒ (b a)

Arguments used in reverse order (combiners only): (CONS '(a) '(b)) ⇒ ((b) a)

Only one argument used (combiners only): (CONS '(a) '(b)) ⇒ ((a))

Answer same as argument (extractors only): (CAR '(a)) ⇒ (a)

Answer unrelated to argument (extractors only): (CAR '((a) b)) ⇒ (c (d))

List-structure Relevant:

Missing parentheses within component of answer: (CDR '(a b (c d) e)) ⇒ (b c d e)

Extra parentheses within component of answer: (CDR '(a b (c d) e)) ⇒ (b ((c d) e))

Extra right parenthesis: (CDR '(a (b c d))) ⇒ ((b c d)))

Missing right parenthesis: (CDR '(a (b c d))) ⇒ ((b c d)

Missing outermost parentheses: (CDR '(a b c d)) ⇒ b c d

The second set of problems, the nonfilterable errors, consisted of the major errors that we have seen subjects generate in our earlier experiments. For the extractor functions, we either added a set of outer parentheses (four instances), deleted a set of outer parentheses (four instances), or used a result that was a valid answer to another function with the same argument (e.g., giving the CAR of the argument as the answer to a CDR problem, two instances). The problems were constructed so that deleting the parentheses always resulted in a syntactically valid expression.

For the combining functions, the nonfilterable errors resulted from including or removing the outermost parentheses of each argument. Because there are two arguments, the four possibilities are: (a) keep parentheses around both arguments (the correct result for a LIST function), (b) keep parentheses around the first argument only (correct result for a CONS function), (c) keep parentheses around the second argument only, and (d) delete outermost parentheses around both arguments (correct result for an APPEND function). For a given function, three of these cases resulted in errors. We included three instances of each of these errors. The tenth error involved adding an extra set of parentheses around the entire answer. Examples of all the nonfilterable errors appear as Figure 10.

Figure 10. **Nonfilterable errors used in Experiment 4, with examples.**

Extractors:

Missing outermost parentheses: (CAR '((a) b c)) ⇒ a

Extra outermost parentheses: (CDR '((a) b c)) ⇒ ((b c))

Wrong function: (CAR '(a b c)) ⇒ (b c)

Combiners:

Parentheses around both arguments: (CONS '(a) '(b)) ⇒ ((a) (b))

Parentheses around first argument: (APPEND '(a) '(b)) ⇒ ((a) b)

Parentheses around second argument: (CONS '(a) '(b)) ⇒ (a (b))

Parentheses around neither argument: (CONS '(a) '(b)) ⇒ (a b)

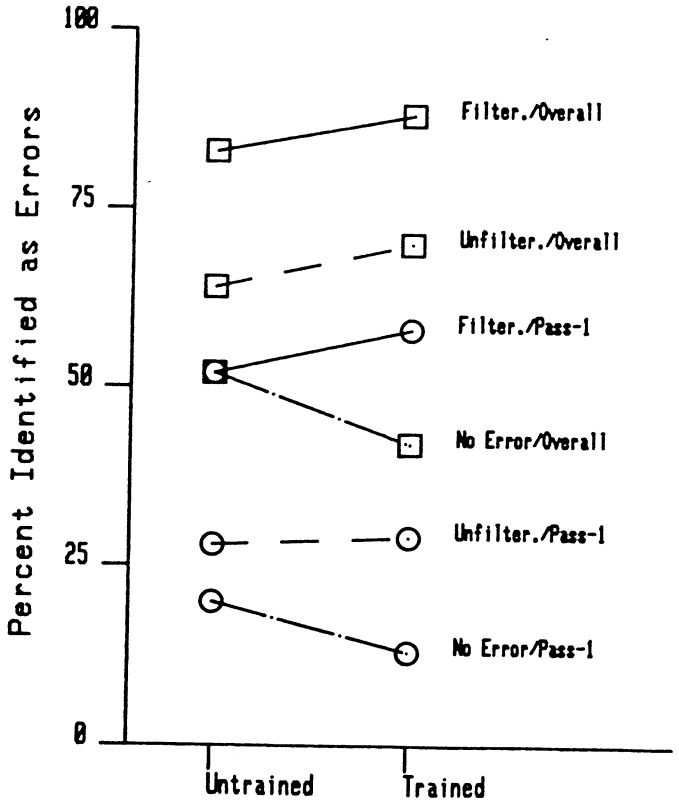
Extra set of parentheses around result: (CONS '(a) '(b)) ⇒ (((a) b))

The third type of problems were correct problems, containing no errors. There were, again 10 instances of these for each function.

With six functions, three problem types, and ten instances of each type, there were 180 problems in all. These were randomly arranged six to a page, with the restriction that there be one instance of each function and two of each problem type on every page. Two groups of 13 subjects each, one totally LISP-naive and one that had just finished a short introduction to list structure, went over these problems and marked those that they believed contained errors. They did so in two passes. On the first pass, they marked two problems on each page; after completing the first pass, they were told to go back and mark two more on each page with a different colored pen. The marking part of the experiment took a little over an hour.

Figure 11 gives the percentage of each type of problem identified as an error by the two subject groups on the first pass and for the two passes combined. Subjects marked more of the filterable errors than the nonfilterable errors—55% versus 29%, first pass, $\chi^2(1) = 129.0$; —86% versus 67%, overall, $\chi^2(1) = 52.2$; 67% versus 47%, overall, $\chi^2(1) = 55.0$. Subjects appeared to do somewhat better in the trained group, but the overall difference between the trained and untrained groups did not approach significance. A more detailed comparison between the two groups reveals that the list-structure trained group was better able to detect the list-structure-relevant filterable errors, particularly on the first pass. This interaction is seen in Figure 12. The list-structure-relevant errors were marked as errors by all subjects much less often than were the other filterable errors on the first pass (40% vs. 70%), but the trained group marked more list-structure-relevant errors than did the untrained group (48% vs. 32%).

Figure 11. **Frequency of answers called errors as a function of answer type, instruction condition, and pass in the experiment.**

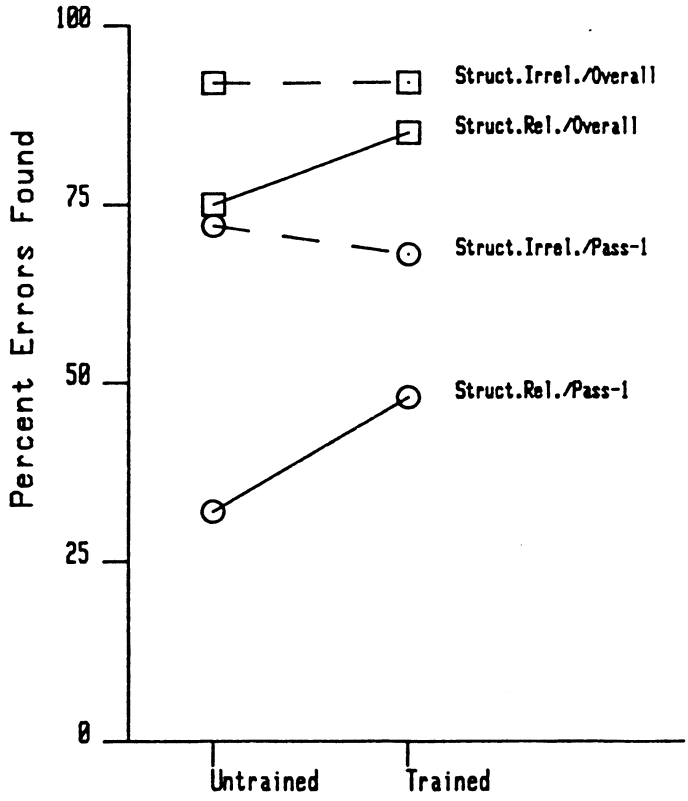


Thus, we see that many errors in LISP functions can be detected by subjects who have no knowledge of the semantics of the functions involved beyond what little insight they get from the names of the functions (which vary widely in their mnemonic value, from nonsense words like CDR to reasonably close semantic matches like LAST). General, common-sense, domain-independent heuristics can be used to reject answers that fail to have an orderly relationship to the function arguments. With minimal knowledge of the syntax rules of LISP, subjects can reliably detect even more errors, both those involving ill-formed expressions and those that alter the internal structure of a list.

6. CONCLUSIONS

How can we describe the errors that are typically made in solving LISP equations involving the most common list manipulation functions? First, these errors are very common; depending on the function and the circumstances, we found that as many as half of all solutions contain errors. Second, most of the errors involve mistakes in the parentheses of the answer; it is unusual for subjects to add or delete atoms. Third, parenthesization errors mostly

Figure 12. **Frequency of classifying filterable errors as a function of whether they are structure relevant or not.**



involve the gain or loss of a pair of parentheses; errors due to unbalanced parentheses represent a relatively small fraction of the errors. Fourth, the most common error is the loss of a pair of parentheses around an entire component, although parentheses are added moderately often and combined errors (e.g., losing a pair of parentheses around one component and gaining a pair of parentheses around another component) are not unusual.

We can also characterize the error patterns made on the sets of problems we gave our subjects. We found that error frequency is affected by the complexity of the components of the problem to be solved. Furthermore, a given subject will not make the same error on all problems of a given kind; he or she is much more likely to give different kinds of responses to equivalent problems. Finally, subjects are less likely to make errors when the error they would generate violates some general principles about what a solution should look like. Some of these principles have to do with the syntax and aesthetics of LISP, but many of them are independent of specific LISP knowledge.

We believe that most errors of the sort made by subjects in our experiments (and by most LISP programmers) are due to processing overload rather than misunderstandings about how the functions operate. The most common result

of excessive processing demands seems to be the loss of a component of the answer, generally a pair of parentheses. At this point, we have only cataloged the types of errors that most commonly occur and show that the hypotheses of processing overload is consistent with the error patterns we see. Much more work remains to be done; we especially need to specify more concretely how and when errors get introduced into the problem-solving process.

Why do we hypothesize processing overload as the primary explanation for the errors made in LISP function application when other researchers (e.g., Brown & Burton, 1978; Matz, 1982), looking at other problem-solving domains, have found evidence for misconceptions? First, other studies have been able to account for only a fraction of the total errors made via the misconceptions interpretation. Clearly, additional sources of errors are necessary to account for all mistakes in any of these domains. Processing overload is a reasonable candidate for one such source. In fact, we do not claim that processing overload can account for all of the LISP errors we have seen. We find it extremely likely that some of the mistakes were due to subject-specific misconceptions. Rather, our claim is that only a small number of errors can be accounted for as misconceptions in the LISP domain; processing overload pays a much more important role.

We can imagine several reasons for the dominance of errors due to processing overload in the domain of LISP. For one, applying the LISP functions we studied is a much less complex, more straightforward task than, for example, solving algebra equations or doing multicolumn subtraction problems. There are only a limited number of steps, and they admit of only a few variations. Second, the sources of the misconceptions in the other areas seem to be incorrect generalizations of procedures that are valid in another circumstance (e.g., rules that cover subtraction without borrowing are inappropriately generalized to the borrowing case). Probably because of the extreme simplicity of the syntax of LISP, such special cases seldom come up, at least in the types of problems considered here.

Matz (1982), in fact, discusses both misconceptions (which she calls *conceptual errors*) and processing errors (which she calls *execution errors*) in algebra equation solving. Execution errors are caused by steps getting lost, which is consistent with the analysis we provide. Her division into the two classes of errors appears to be done on the basis of intuition and on the premise that an error that persists with skilled problem solvers must be due to short-term failure of the problem-solving executive. Our interpretation is again similar; we were led to the processing demands explanation originally because we saw that these errors did not disappear as our students became more proficient.

As Matz shows, and as Norman (1981) discusses, the errors that occur due to processing overload (what Norman calls *slips*) are far from random. Errors creep into previously learned procedures from a limited number of general sources. We believe that the slips observed in our LISP problem solvers follow

the same kinds of general rules. In fact, we hope that we can use what we have discovered about the nature and frequency of these errors to construct a model of the procedures students use to solve such problems.

Our analysis of these errors has had important implications for a design of an intelligent computer tutor for LISP (Farrell, Anderson, & Reiser 1984). First of all, it provides a mechanism for predicting the kinds of errors that will be seen and also provides a diagnosis for such errors. The diagnosis is that parenthesis errors often involve cognitive slips and should be corrected without a great deal of new instruction. Our analysis also implies that anything in the tutor that can reduce working memory overload should reduce the frequency of such errors. Therefore, in the tutor we provide the student with an editor that keeps track of much of the syntax of LISP, thus reducing the number of minor details that a student must keep track of in learning LISP. This enables the student to concentrate on higher level conceptual issues. Although it is difficult localizing the credit to a particular component of the LISP tutor, it is the case that the tutor based on this principle, among others, has proven to be an effective instructional device.

Support. This research was supported by Office of Naval Research contract N000014-84-K-0064 to John Anderson.

REFERENCES

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

Anderson, J. R., Pirolli, P., & Farrell, R. (in press). *Learning recursive programming.* In M. Chi, R. Glaser, & M. Farr (Eds.), *The nature of expertise.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

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

Farrell, R., Anderson, J. R., & Reiser, B. J. (1984). An interactive computer-based tutor for LISP. *Proceedings of the 1984 National Conference on Artificial Intelligence,* Austin, TX.

Matz, M. (1982). Towards a process model for high school algebra errors. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 25–50). New York: Academic.

Norman, D. A. (1981). Categorization of action slips. *Psychological Review, 88,* 1–15.

Winston, P. H., & Horn, B. K. P. (1981). *LISP.* Reading, MA: Addison-Wesley.

HCI Editorial Record. First manuscript received December 3, 1983. Revision received November 25, 1984. Accepted by Elliot Soloway. Final manuscript received April 12, 1985. —*Editor*