9

Effect of Practice on Knowledge and Use of Basic Lisp

Jean McKendree and
John R. Anderson

Practice Effects

Cognitive psychology is being called upon to guide designers of computer systems toward more effective interfaces and training methods. Basic research on learning of skills has the potential to influence the direction of these projects and to have an impact on the productivity of old and new users. Research in human-computer interaction has generally concentrated on characterizing various tasks and on the methods available for accomplishing these tasks via the computer. The enticement of the technology often has been the guide to the design and direction of these research projects. Many domains and questions have been addressed, but fundamental issues in theory or application are seldom made clear. The need now is for an explicit modeling of the processes involved in performing the given tasks and in interacting with the chosen interface. Detailed performance models should be able to provide guides for further research and to make predictions about relevant dimensions in man-machine interaction.

Card, Moran, and Newell (1983) assert that the usefulness of psychology to the design of human-computer interfaces "centers on this notion of performance models." They note that such a model must explain the transition from problem solving to cognitive skill that is seen in the progression from novice to expert. The expert has accumulated knowledge that can guide problem solving to the point that the search for possible answers is vastly reduced. They conclude that "examination of the transition . . . shows that the mechanisms whereby search is reduced are (1) accumulation of control knowledge and (2) the formation of new, large scale operators, which effectively partition the problem space into a reduced problem space and a skill space."

Control knowledge serves to guide decisions so that solutions become more efficient. The learner attempts to find more optimal methods and

to use knowledge to constrain the search for possible operators. When enough knowledge and experience are accumulated to define a partial method for a problem, search is effectively eliminated. The newly acquired method, discovered by search guided by control knowledge, becomes part of the skill space of the evolving expert. Card, Moran, and Newell discuss the acquisition of this control structure by one user. This was an expert text editor given a novel task. Over the course of the first 35 trials spent repeating the same task, the user came to use the optimal sequence of operators and became much faster as a result. While this is certainly a factor in the earliest stages of skill acquisition, it largely ignores the concurrent and subsequent speedup due to the second of their hypothesized mechanisms: the formation of larger operators.

Their model of expert users assumes that they have acquired the methods already for accomplishing a task and have reached the minimal time per operator. In other words, the model does an admirable job of predicting expert use of methods and at least addresses the novice's acquisition of these methods through discovery. It does not address the transition from the time of discovering the method to the point when expert performance time is attained. There may be only a small amount of control knowledge to be found in a repeated task. After the "reduced problem space" of methods is acquired, the important component influencing performance for the intermediate-level user becomes the formation of larger operators. The mechanisms that produce and then gradually speed up these operators are the subject of this chapter.

The present study investigates these basic questions about learning through the acquisition of simple programming skills. Programming skill has recently begun to be a subject of research in skill acquisition. It involves a complex interaction of skills, such as problem representation, algorithm design, code generation, and debugging. All of these cognitive tasks should be better understood and integrated in order to assess human-computer interactions and designs more effectively. Many of the studies of programming have looked at aspects of languages or at structures used in programming in an attempt to enhance performance of programmers or to discover expert-novice differences (Adelson, 1981; Gannon, 1976; Green, 1977; Mayer, 1976). However, it has proved notoriously difficult to establish these differences using current experimental methodology (Sheil, 1981). Perhaps an understanding of more basic units from which these skills are produced would aid in focusing on the relevant dimensions in the more complex tasks.

We have set out to investigate the effects of practice on learning beginning with a theoretical framework. One definition of learning is any change

in a system that allows it to perform more efficiently and effectively on subsequent repetitions of the same or similar tasks (Simon, 1983). Practice on a task generally results in faster and more accurate performance. Therefore, practice somehow induces learning in the system seemingly without need for conscious effort or awareness on the part of the performer. There have been numerous studies describing phenomena associated with practice, particularly in instructional areas, such as effects of spaced versus massed practice on retention and recall (Gay, 1973; Reynolds and Glaser, 1971; Hintzman, 1974), and effects of practice on performance time (Neisser, Novick, and Lazar, 1963; Winkleman, 1974). Learning theories naturally included the fact that repetition improved memorability, but few suggested any explanatory mechanisms until recently.

One of the most robust effects of practice found has been the reduction in time to perform a task. The decrease in time for performance on a task has been found to follow a power function of the number of times a task has been performed. This equation is of the general form

$$T = AP^{-b},$$

where T is the performance time, P is the number of units of practice, A is a coefficient reflecting the initial speed when $P = 1$, and b determines the rate of speedup. This power law has been found to hold true over a variety of tasks from basically perceptual-motor skills, like the Seibel task (Rosenbloom and Newell, 1982), to complex problem solving skills, like playing solitaire (Newell and Rosenbloom, 1981).

The present study attempted to find empirical support for predictions about practice effects made by the ACT* production system model of learning (Anderson, 1983). This production system model assumes a representation of a problem that is present in working memory during problem solving. This representation is compared to existing knowledge structures that take the form of conditions to be matched in order to carry out an associated action. This type of information processing model of learning and skill acquisition suggests some mechanisms that can predict basic results of previous practice experiments. Specifically, we apply the ACT* model to questions of repetition and the resulting speedup in performance time. The ACT* model indicates that increasing familiarity with a series of procedures speeds performance because of the actor's ability to consolidate a sequence of operations into a single cognitive step. Sequences which previously required meticulous recall during solution of a problem become collapsed into a single cognitive action. This *composition of knowledge* produces a much faster solution time by creating efficient operators for handling particular problems.

An example from basic algebra will illustrate the process of composition. Given the task of solving $3x + 4 = 10$, the learner might have productions like these:

IF the goal is to solve an equation of the form
    ax + b = c
THEN subtract b from both sides
    AND solve ax = d

IF the goal is to solve an equation of the form
    ax = d
THEN divide both sides by a.

The solver will work through the steps serially, producing the intermediate step $3x = 6$, which will then be used in the next step. There may be other steps, such as performing the subtraction of $10 - 4 = 6$, particularly when the student is first learning the skills of math. After solving the problem successfully, the learner can eventually eliminate the explicit steps and can form a new, composed production like

IF the goal is to solve an equation of the form
    ax + b = c
THEN subtract b from both sides
    AND divide by a.

A second example involving generation of simple Lisp functions will show this process in the domain of the current study. If the learner were given the task of returning the second element in a list, he might start out with a set of productions like the following:

IF the goal is to find the second element of a list
THEN eliminate the first element
    AND return the first element of the new list.

IF the goal is to eliminate the first element of a list
THEN apply the function CDR to the list.

IF the goal is to return the first element of a list
THEN apply the function CAR to the list.

Having figured out the initial algorithm (by whatever means), the system applies the productions serially, setting the new subgoals and explicitly recalling the separate Lisp functions for achieving them. Having solved the problem, the system can eliminate the explicit steps and will eventually

form a composed production. This new production might look like

IF the goal is to find the second element of a list
THEN apply the function CDR
AND apply the function CAR

In these examples, the subgoals have been subsumed in the top-level goal and the actions can be performed without the explicit recall necessary before composition took place. This will allow the correct actions to be executed more quickly and with fewer memory requirements.

A second process interacts with the composition of operators to enhance speedup. This process, called *proceduralization*, involves incorporating factual knowledge into productions. This serves to apply knowledge directly that previously had to be recalled explicitly into working memory and then applied interpretively. This is apparent in math when the student can recall the answer to addition facts without having actually to perform the addition operations. In Lisp, this occurs when a person no longer has to recall or read the definition of a function in order to apply it. Instead, the actions associated with applying the function are executed automatically when the goal is set and the production conditions are matched by the current problem representation. These two mechanisms, composition and proceduralization, produce a speedup with practice particularly in the initial stages of learning. At later stages, the speedup will continue at a gradually lessening rate as working memory capacity increases with strengthening of accumulated knowledge.

The third process that serves to govern the rate of speedup in this model is *strengthening of productions* when they are used. As production strength increases for the newly built productions, they will be able to be applied more consistently and more rapidly. This process, in fact, turns out to be the limiting factor for rate of speedup of performance (see Anderson, 1982, for a more detailed discussion).

This combination of learning processes predicts that while subjects should speed up on all tasks involving the components of a skill, they should speed up most rapidly on the tasks faced most frequently. Each task involves a specific sequence of mental operations. As these operations are practiced they will become faster due to strengthening. However, the specific combinations of operations that are practiced repeatedly will be composed into larger operators and will gain an extra boost in speed. Therefore, frequency of specific combinations is going to have an impact on performance time and accuracy, in addition to the general effect of practice.

In order to test these predictions empirically, the task environment must create a number of conditions. First, the task must be complex enough to require extended practice before reaching its asymptote in order to assure that effects will be observable. Second, the task should contain discrete components that can be placed in combinations requiring sequences of several steps. This will allow observation of the effect of repeating sequences of steps in contrast to applying individual ones. This may reveal the process by which individual steps in various combinations become collapsed with practice. Finally, the task must be composed of sufficient components to allow variation in frequency as well as combination. The combinations seen frequently can be directly compared to the ones seen infrequently since they will be built of the same basic components placed in different relations.

This paper presents a study that systematically tested the learning of a skill in a limited domain. The domain used in this study was the evaluation of basic Lisp functions seen alone and in combinations. The Lisp domain is complex enough to require a reasonable amount of practice to reach a level of proficiency at evaluation. Also, there is sufficient structure to restrict the number of possible responses and to allow an analysis of the steps used in formulating an answer. Because larger functions are easily built out of basic ones, they provide a means of evaluating effects of combinations independent of individual step differences.

By requiring subjects to evaluate Lisp functions seen with varying frequencies over a period of several days, the quantitative effects of practice on this domain can be studied. The frequent and infrequent combinations built of identical components allow an investigation into the mechanisms behind the proposed composition of operators. The processes suggested by the ACT* production system model of learning predict that high-frequency problems should have shorter solution times and higher accuracy than low-frequency problems. By testing these sorts of predictive theories, cognitive psychology can begin to build models that can guide design and evaluation of interfaces and interactive systems.

1 Method

1.1 Subjects

Subjects were 20 Carnegie-Mellon undergraduates with no previous experience with Lisp. They were given one class credit and $15.00 plus

bonuses. These bonuses were given for points awarded on the basis of both speed and accuracy.

1.2 Procedure

The subjects reported to the experiment for four consecutive days. On the first day, they were given a brief introduction to four basic Lisp functions. Since one of the four functions in Lisp has a mnemonic name but the other three do not, we chose to change the three so that all four had mnemonic names. Two of the functions, FIRST and REST, return part of a list. FIRST returns the first element in a list and REST returns the list without the first element. (These functions are called CAR and CDR, respectively, in Lisp.) These will be referred to as *extract functions*. The other two, INSERT and LIST, (corresponding to CONS and LIST in Lisp) join elements. These will be called *combine functions*. INSERT puts an element given as its first argument into the beginning of the list given as its second argument. For example, (INSERT (a) (b c)) would return ((a) b c) as its answer. LIST makes a new list out of the two arguments. Thus, (LIST a (b c)) would give (a (b c)) as the answer.

The instructions also explained that the experiment would contain combinations of two and three of the basic functions. After reading this, the subjects went through 10 practice trials with the experimenter. These trials contained one problem with each of the basic four functions alone and six problems involving pairs of functions. No triple combinations were included in the practice trials.

The trials were presented on a terminal via a DEC system PDP 11/34 computer. The problems were given and the subjects were required to type in the answer. There was no time limit on the trials and subjects were given immediate feedback about the correctness of their answers. If the answer was incorrect, the correct answer was shown and the subject could study it as long as desired. The subject received 30 points at the beginning of each trial. For every 1/2 second that passed, 1 point was subtracted until the "Return" key was pressed or until no points remained. Thus, the subject could receive no less than 0 points for a correct answer. Forty points were subtracted for a wrong answer. At the end of four days, the points were totaled and the subjects were given a bonus of $1.00 for every 500 points.

There were several measurements collected on each trial. Every keystroke was timestamped by the computer and these times were recorded. The total number of keystrokes was also recorded along with a subject's answer and the correctness of the answer. After the first and last sessions, subjects

were given a transfer task that involved generating functions similar to those they had practiced evaluating. They were given the variables and the answer that the function should yield and were instructed to write a function to give the desired answer. The nature of the stimulus materials for the main task and the transfer task is explained in greater detail and with examples in the next section.

1.3 Materials

The trials consisted of four possible single functions and various pair and triple combinations. There were 150 trials each day for four days. The subject saw the function to be evaluated along with its arguments on the screen and had to type in the answer. A typical problem involving one of the four basic functions FIRST, REST, INSERT, or LIST would look like

(FIRST v1)

v1: (h d s)

The subject should apply the function, retrieve the first element in the list, and type in h.

Pairs were constructed by embedding one function within another. When the embedding function was a *combine function* like INSERT, the embedded function was placed as the first argument. For instance, a typical pair problem would be

(INSERT (FIRST v1) v2)

v1: (x w)
v2: (y z)

Applying FIRST would give x and INSERTing it into (y z) would give the answer (x y z). By crossing the four possible embedding functions with the four possible embedded functions, there are 16 possible pairs. However, some possible pairs were eliminated because applying these pairs results in performing a step and then undoing it. For example, (FIRST (INSERT V1 V2)) would involve inserting V1 into V2 and then extracting V1 again to return it as the answer. These *extract-combine* functions were eliminated, leaving 12 pairs.

The basic functions, FIRST, REST, INSERT, and LIST, were all seen by themselves an equal number of times. Pairs of functions were seen either *frequently* or *infrequently*. For instance, the FIRST-REST combination might

be seen frequently, while the REST-FIRST pair would be infrequently presented. Half of the subjects saw one set of functions frequently and half saw the opposite pairs frequently.

The triples involved taking an embedded pair and embedding it within a third function. An example triple is

(LIST (INSERT (FIRST v1) v2) v3)

Such a function can be seen as being composed of two pairs; in the example given, FIRST is embedded in INSERT and INSERT is embedded in LIST. The triples were presented with frequencies that preserved the relative frequency of the pairs. That is, if LIST-INSERT were a high-frequency pair and INSERT-LIST were low-frequency, the triples including the first pair would be seen more often. The triples can be categorized into four types based on the frequency of their constituent pairs. These were both pairs high frequency, just the first high, just the second high, or both low frequency. For example, a high-low frequency triple would be constructed by taking a high-frequency pair, like FIRST-REST, and combining it with a low-frequency pair such as REST-REST. The triple constructed would be

(FIRST (REST (REST v1)))

The four basic functions can be combined into four different sequences of triples. These are *extract-extract-extract*, *combine-extract-extract*, *combine-combine-extract*, and *combine-combine-combine* functions. Within each of these four types, there are eight triples: two made of high-frequency pairs, two made of low-frequency pairs, and four made of one high- and one low-frequency pair. This results in 32 triples.

As with the pairs, the first argument of any *combine* function in a triple was more complex than the second argument. For example, a typical one of these problems was

(LIST (LIST (FIRST v1) v2) v3)

v1: (d (g n))
v2: v
v3: (b)

The subject should apply the FIRST function to get the d from the first argument. Then this, LISTed with the second argument, gives (d v) and finally, LISTing this with the last argument, the (b), gives the final answer. The subject should type ((d v) (b)).

A complete list of the functions to be evaluated is given in table 9.1

Table 9.1
Lisp functions and frequencies

FIRST	16	INSERT-FIRST-FIRST	9
REST	16	INSERT-FIRST-REST	3
INSERT	16	INSERT-REST-REST	3
LIST	16	INSERT-REST-FIRST	1
FIRST-FIRST	12	LIST-REST-REST	9
FIRST-REST	4	LIST-FIRST-FIRST	3
REST-REST	12	LIST-REST-FIRST	3
REST-FIRST	4	LIST-FIRST-REST	1
INSERT-FIRST	12	INSERT-LIST-REST	9
INSERT-REST	4	INSERT-LIST-FIRST	3
LIST-REST	12	INSERT-INSERT-FIRST	3
LIST-FIRST	4	INSERT-INSERT-REST	1
INSERT-LIST	12	LIST-INSERT-FIRST	9
INSERT-INSERT	4	LIST-INSERT-REST	3
LIST-INSERT	12	LIST-LIST-FIRST	3
LIST-LIST	4	LIST-LIST-REST	1
FIRST-FIRST-FIRST	9	INSERT-LIST-INSERT	9
FIRST-FIRST-REST	3	INSERT-INSERT-LIST	3
FIRST-REST-REST	3	INSERT-LIST-LIST	3
FIRST-REST-FIRST	1	INSERT-INSERT-INSERT	1
REST-REST-REST	9	LIST-INSERT-LIST	9
REST-FIRST-FIRST	3	LIST-INSERT-INSERT	3
REST-REST-FIRST	3	LIST-LIST-INSERT	3
REST-FIRST-REST	1	LIST-LIST-LIST	1
		Random	12
Total			300

along with the frequencies of each seen over two days by one group. The second group had the same stimuli, but with reversed frequencies for the pairs and corresponding changes for the triples. Note that although the frequencies of combinations varied, the numbers of times that each of the basic functions was used are equal.

Each subject solved 150 problems per day. It took two days (300 trials) to complete the set of problems in table 9.1. In these 300 trials, subjects solved problems involving 64 basic functions, 96 pairs of functions, and 128 triples. In addition, they saw 12 "random" function combinations. These were combinations of two or three functions that were not repeated. Each of these combinations was seen only once during the entire experiment. The major difference with these combinations was that the complex argument occurred in the second position. For example, one of these

problems would have looked like

(LIST v1 (FIRST v2))

v1: b
v2: (n x)

The subject should have responded with (b n). These functions were included as a baseline to determine the speedup pattern for combinations of functions that were not repeated at all. Also, these functions prevented subjects from making the overgeneralization that complex arguments in Lisp always occurred in the first position. Each subject saw the 300 trials in random order. The arguments had been randomly generated for each problem so that no two problems were identical, but each subject within a group saw the same set of problems and arguments.

The subjects were given a transfer task after the first and fourth days of evaluation problems. The transfer task involved generation of simple functions given the arguments and the desired answer. An example problem was

v1: (l s)
v2: k

answer: (l k)

The subject should have written (LIST (FIRST v1) v2). There were 16 of these problems on each of the two days: 4 requiring only one of the basic functions and 12 requiring generation of one of the pairs of functions.

2 Results and Discussion

The two groups of subjects differed in terms of the combinations seen for high- versus low-frequency conditions. Since the individual functions were seen equally often in each group and there were the same types of combinations, no differences were expected between the groups. In fact, there were no significant differences between two groups seeing the counterbalanced combinations. The number of keystrokes ($F(1,19) = .41$, $p < .53$), the time to first keystroke ($F(1,19) = .04$, $p < .85$), the time to final keystrokes ($F(1,19) = .13$, $p < .73$), and the error rate ($F(1,19) = .32$, $p < .62$) all failed to indicate a difference between the groups.

The total time spent on the task decreased each day, reflecting the general nature of the power law. This time was found by summing the

solution times for all 150 trials each day. The total time for the first day was 3534 seconds, or 58.9 minutes, and by the last day was 1716 seconds, or 28.6 minutes. The equation fitted to the time across the four days is

$$TT(\text{sec}) = e^{8.16}P^{-.55} = 3498P^{-.55}, \quad (1)$$

which gives the total time in seconds to evaluate all 150 problems in a session. The exponent $-.55$ is similar to the one found for the geometry proof justification task studied by Neves and Anderson (1981). This indicates a faster rate of learning than found on simpler tasks, such as the Siebel task used by Rosenbloom and Newell to study chunking of operators.

The analysis of the power law speedup by Anderson (1982) would predict this difference in rate of speedup between simple versus complex tasks. According to that analysis, speedup in performing a complex task like Lisp evaluation is produced by a combination of strengthening of individual productions and collapsing of multiple productions into a single one. Thus, the $-.55$ exponent found in this experiment reflects both collapsing and strengthening of productions used. In contrast, performance of a simple task speeds up primarily because of the strengthening process alone. These simple tasks would not involve complex sequences of productions that could be collapsed in the compilation process.

The "random" functions can be used to indicate the effect of general speedup. These combinations are also built from the basic functions, but each is seen only once over the course of the experiment. The final times for these functions are included in table 9.2. These combinations are solved more quickly with practice, indicating a general effect of practice. The

Table 9.2
Summary of final times in seconds for function types by frequency

	Day			
Function type	1	2	3	4
BASIC	11.50	7.72	5.42	5.07
Pairs				
HIGH FREQUENCY	17.86	12.63	9.57	9.35
LOW FREQUENCY	18.42	12.98	11.00	10.75
Triples				
HIGH FREQUENCY	24.63	18.68	14.06	14.18
LOW FREQUENCY	24.79	20.62	15.07	15.71
RANDOM	21.28	20.32	16.33	17.07

rate of the speedup is not as great for these functions as for the other combinations seen repeatedly; the decrease is from 21.3 sec to 17.1 sec, a 19.7% decrease, over four days, compared to a decrease from 24.8 sec to 15.3 sec or 38.3% for the low-frequency triple combinations.

The speedup for the combinations should be affected both by the general practice and also by the differing frequencies with which they were seen. This is a reflection of the combined effects of strengthening and collapsing of productions.

2.1 Frequency of Combinations

Combining basic functions into pairs and triples allows frequency of combinations to vary while keeping the frequency of the components equal. This allows a separation of effects due to collapsing of productions from effects due to strengthening of productions. Any advantage of the high-frequency combinations would be a reflection of the collapsing of production steps.

A repeated measures ANOVA on total solution time for high- versus low-frequency combinations across the four days indicated that the overall frequency effect is significant for both pairs ($F(1,16) = 9.78$, $p < .006$) and triples ($F(1,19) = 11.26$, $p < .003$). The times for the low-frequency triples were found by collapsing the intermediate- and low-frequency triples and comparing these composite scores to the high-frequency triples. The frequency effect, when broken down into the various types of combinations for post hoc comparisons, is in the predicted direction but is not significant for the *combine-extract* pairs ($F(1,16) = .01$, $p < .91$). The frequency effect is significant for both the *extract-extract* pairs ($F(1,16) = 4.75$, $p < .04$) and the *combine-combine* pairs ($F(1,16) = 21.38$, $p < .0003$). For the triples, the *extract-extract-extract* combination ($F(3,57) = 4.32$, $p < .008$), the *combine-combine-extract* combination ($F(3,57) = 11.53$, $p < .0001$), and the *combine-combine-combine* combination ($F(3,57) = 4.15$, $p < .01$) all showed a significant frequency effect on total solution time. Again, the times for the *combine-extract-extract* triples were in the predicted direction, but the effect was not significant. A summary of the solution times for high- and low-frequency combinations is shown in table 9.2.

The number of keystrokes does decrease from the first to the last day ($F(3,57) = 8.71$, $p < .0001$) and is different between high- and low-frequency combinations ($F(1,19) = 48.62$, $p < .0001$). However, the number of keystrokes is greater for the high-frequency functions, which would tend to decrease the frequency effect. This difference in keystrokes is

significant only across correct trials; the number of keystrokes did not differ significantly if all trials were used. Correct trials only were used in the analysis. It is possible that that the greater number of keystrokes is a result of the subject's realizing more often that an answer to a high-frequency problem was wrong initially and hitting the "Delete" key several times to correct the answer. This would increase the number of keystrokes somewhat without influencing the time as much as would normal typing of keys. However, it was not possible to extract this information from the original data.

The frequency effect holds also for the error rates across high- and low-frequency combinations for both pairs ($F(1,19) = 7.17$, $p < .01$) and triples ($F(1,19) = 7.86$, $p < .01$). By the last day, the subjects are getting 87.7% of the high-frequency pairs correct and 80.8% of the low-frequency pairs. They are correct on 78.1% of the high-frequency triples and on 72.5% of the low-frequency triples on the last day. As the subjects are exposed to the frequent problems, they not only become faster over the days, but more accurate than they are on the less frequent function combinations.

2.2 Error Data

A more qualitative analysis of the types of errors was done for the basic and paired functions. This yielded 473 errors on the first day and 211 on the fourth day. These were categorized into error types, and the results are summarized in table 9.3.

Anderson and Jeffries (1983) have modeled novice Lisp errors in terms of undetected working memory failures. As functions increase in complexity, it appears that subjects lose partial results when applying functions. Some errors can be detected by "critics" if the result is not a sensible Lisp answer, but the errors are often overlooked if they produce a potentially legitimate form. The errors that are filtered in a manner consistent with the "repair theory" of Brown and VanLehn (1980) are fairly low-level errors, such as unbalanced parentheses or the use of an element that was not present in the problem statement. Even subjects with no Lisp experience were able to detect 75–80% of these types of errors in a study by Anderson and Jeffries. It is consistent with these observations that very few of the errors found in our study were due to these ill-formed expressions. Only 9.2% of the errors analyzed were due to unbalanced parentheses on the first day. This went up to 14.1% on the last day primarily because of the overall

Table 9.3
Percentage and types of errors on basic and paired functions on the first and last days

	Day 1 (n = 473)	Day 4 (n = 211)
BASIC		
Overall error rate	22.5	11.5
	Extract	
Parentheses added	2.1	3.0
Parentheses dropped	3.5	1.5
Misapplication	5.6	7.9
	Combine	
APPENDed elements	9.2	7.8
INSERT-LIST confusion	2.8	3.2
Parentheses added	1.4	0.0
Parentheses dropped	.7	1.5
Unbalanced parentheses	1.4	3.2
Typographical errors	3.5	6.2
PAIRS		
	High frequency	
Overall error rate	36.7	12.3
APPENDed elements	9.2	7.8
INSERT-LIST confusion	7.0	6.3
Parentheses dropped—REST	11.3	6.2
—other	1.4	0.0
Parentheses added	4.9	4.7
Unbalanced parentheses	5.7	4.7
Misapplied extract function	4.2	4.7
Typographical errors	6.3	7.9
	Low frequency	
Overall error rate	39.5	19.2
APPENDed elements	7.0	3.2
INSERT-LIST confusion	1.4	3.2*
Parentheses added	1.4	0.0
Parentheses dropped—REST	4.2	4.7
—other	0.0	1.5
Unbalanced parentheses	1.4	4.7
Misapplied extract function	2.3	4.7
Typographical errors	2.1	1.5

decrease in other types of errors. None of the errors involved the use of elements not found in the problem statement.

Most of the errors were of a more conceptual nature. These errors primarily reflected a misapplication of a function or the use of an incorrect argument in a function. This type of error became more frequent as the complexity of the problem increased from a single, basic function to a combination of two; also more errors were being made on infrequently seen functions.

Overall error rates on the basic functions were 22.5% for the first day, dropping to 11.5% by the last day. The most common error on both days was a specific misapplication of INSERT or LIST. The subjects treated both these functions like the Lisp function APPEND (which they had not learned). Subjects exhibiting this error would simply create a list of all the individual elements in the arguments. For instance, given the problem

(INSERT (LIST v1 v2) v3)

v1: a
v2: (b)
v3: (c)

the subject would produce (a b c). This error accounted for 25.4% of all errors on the first day and 18.8% of all errors on the fourth day. The classic confusion in Lisp between INSERT and LIST was apparent in 11.2% of errors on the first day and 12.7% of those on the last day. It appears that the subjects have not formed a clear picture of what the *combine* functions do. They often simplify application of INSERT or LIST into a single, incorrect action. Error rates on INSERT were 35% for the first day and 14% on the fourth day; for LIST, 26% on the first day and 18% on the fourth day. These rates were for the problems where these functions were seen alone.

The *extract* functions seemed to be confused less often by the subjects. Overall error rates on these were for FIRST, 15% on day 1, dropping to 5% on day 4, and for REST were 14% on day 1 and 9% on day 4. Of the errors made, the most common on single *extract* equations was returning the last element for REST instead of all but the first element. Given a problem like

(REST v1)

v1: (r j v z)

a subject showing this error would answer z. This accounted for 3.5% of the errors made on the first day and 4.7% on the last.

However, when REST is included in a problem with a pair of functions, the error pattern changes. The most common error becomes dropping the outer pair of parentheses from the part of the answer produced by REST before applying the second function. For example, given

(INSERT (REST v1) v2)

v1: (a (b c))
v2: (d)

the subject would answer ((b c)d) rather than (((b c))d). This error accounts for 15.7% of all errors made on the pair problems, but only for 2.1% of errors on REST seen alone. By the last day, these error rates are 10.9% for the pairs and 1.5% of errors on REST seen alone. This indicates that the subjects knew the proper application of the function, but as the complexity of the evaluation task increased and partial results had to be retained in memory, features began to be lost. Since the resulting answers are syntactically correct Lisp expressions, the errors are less likely to be caught than if the same features were missing for a simpler function evaluation. The embedding of inside elements is not a predictable feature of the problems and therefore is not as easily proceduralized during practice.

2.3 Transfer Task

Subjects were given a transfer task in which they were to write the functions to produce a desired answer. All the problems involved basic functions or pairs that they had seen in the evaluation task immediately before. This transfer task provided indications of the scope of the learning acquired in the first task. Performance on the transfer task showed practically no improvement from the first day to the last day. On the first day, 29.3% of the problems had errors versus 26.6% on the fourth day. A third of the errors each time were due to a confusion between the functions INSERT and LIST, a classic difficulty for Lisp novices. A summary of the types of errors on both days is shown in table 9.4. In contrast to the relative lack of change in the error rates for the generation task, error rates on the evaluation of single and paired functions began at 34% errors on the first day and dropped to 15% by the fourth day. The ACT* model again would predict that practice effects should be very specific to the current task. The transfer task, though involving the same abstract knowledge as the evaluation task, is clearly not the same task for the subjects. Though they understand quite well how to evaluate the functions by the fourth day, subjects are not better able to generate simple functions than they were on the first day.

Table 9.4
Summary of errors made on transfer task requiring function generation

	Percentage of errors	
Type of error	Day 1	Day 4
INSERT/LIST confusion	33.9	23.5
Reversed order of application	14.6	17.6
Missing extract function	14.6	21.6
Extra extract function	4.8	5.9
Missing combine function	15.6	15.6
Extra combine function		
LIST	8.7	11.7
INSERT	1.0	0.0
Miscellaneous	6.8	3.8
Overall error rate	29.3	26.6

2.4 Mathematical Model

The observed solution times for evaluation of basic Lisp functions and combinations showed systematic changes with practice over days and over varying frequencies. We then attempted to characterize these changes using a mathematical model based on predictions made from a theory of skill acquisition. The ACT* model would predict a general component of practice due to production strengthening and a specific component due to the frequency of particular combinations affecting the collapsing of production steps. We felt that the effects of these two processes would have different mathematical forms. As argued in Anderson (1982), the speedup due to strengthening should take the form of a power function. Thus, assume that there is a processing time associated with each function that will decrease according to a power function. In contrast, the speedup due to production composition should conform to an exponential function. That is, we assume that on each trial there is a constant probability, p, that a composition will occur, resulting in a procedure requiring a fraction, c, of the previous time needed to perform the operations. Thus, the expected proportional improvement with each trial is $p(1 - c)$. The expected time

after f trials is

$$RT = (1 - p(1 - c))^f T, \quad (2)$$

where T is the time on the first trial. On setting

$$d = (1 - p(1 - c))$$

the equation becomes

$$RT = d^f T. \quad (3)$$

In addition to the mental time associated with function evaluation, there will be some time associated with each keystroke in the final answer. The average number of keystrokes for each problem type was found and multiplied by the time per keystroke parameter in the model.

The total time to evaluate a function will be a physical time involving number of keystrokes and a mental time reflecting speedup due to composition and strengthening:

$$RT = \text{physical} + \text{mental}$$

The physical time will be the product of the number, n, of keystrokes and the time per keystroke, k. The mental time will be the product of the number of mental operations, N, and the time per operator, K. This gives this equation

$$RT = nk + NK. \quad (4)$$

The subjects' typing speed will decrease with the number of units of practice, u. This decrease will take the form of a power function with exponent of speedup a. In addition, the number of mental operations will go down with the frequency, f, of a particular combination. We assume that each time a combination is repeated, the expected number of steps goes down by the fraction c. The time per operation is a power function of units of general practice, u. Thus, the form of the equation we chose to fit becomes

$$RT = nku^{-a} + c^f(mu^{-b}). \quad (5)$$

We took the units of general practice, u, to be days and the frequency of a particular function or combination to be the number of times it had been seen by the end of each day. In the equation, b is the exponent of general speedup due to practice on the Lisp task, and m is the sum of the setup times for each basic function making up the combination.

The STEPIT program (Chandler, 1965) was used to find the best-fitting values for the parameters. These values were found to be

Parameter	Value
Cost per keystroke, k (sec)	1.39
Exponent of speedup on typing, a	− .28
Setup times for basic functions (sec)	
FIRST	3.59
REST	2.66
INSERT	2.87
LIST	3.70
Proportion of reduction due to forming a composition	.019
Exponent of speedup on Lisp, b	− .25

The correlation of the observed to the predicted values in this model is .96. The chi-square measure computed between the model and the data was $X^2(379) = 183.1$.

The values for the setup times for the four basic functions are very similar, as are the exponents of speedup for typing and for the Lisp evaluation. Therefore, we refigured the model by collapsing these separate parameters. The equation becomes

$$\begin{aligned} RT &= nku^{-b} + c^f(mu^{-b}) \\ &= (nk + mc^f)u^{-b} \end{aligned} \quad (6)$$

The loss in accuracy of the fit was minimal; the X^2 measure increased from 183.1 to 198.2. The correlation coefficient was .95. This simplification is a reasonable approximation of the previous equation. The values found using only these four parameters were

Parameter	Value
General exponent of speed-up, b	− .26
Cost/keystroke, k (sec)	1.42
Cost/function, m (sec)	3.16
Proportion reduction due to forming a composition	.020

A comparison of the data and the model is shown in figure 9.1.

The value of cost per keystroke, 1.42 seconds, seems rather large if this measure were simply an indication of typing speed. This is the value for the first day, but on applying the exponent of speedup for typing, this value

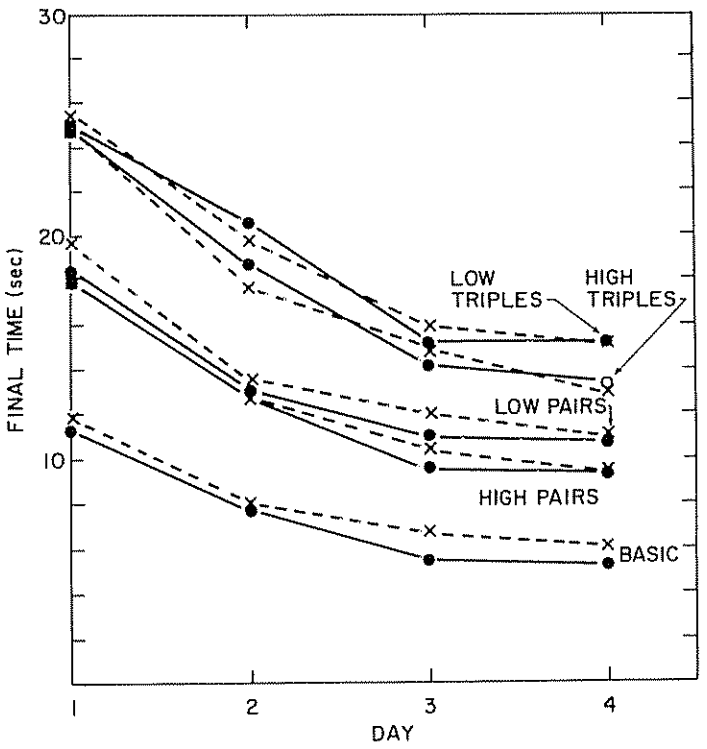

Figure 9.1
Observed and predicted final times for high and low combinations: solid line (with •), observed; dashed line (with ×), predicted

becomes .99 seconds by the last day. This parameter is being influenced by the time to encode information to be typed and to prepare a motor plan, indicated by the fact that the actual typing episodes occurred in bursts with interkeystroke times of much less than .99 seconds. Card, Moran, and Newell (1983) found average values of between .5 and .75 seconds per keystroke for practiced subjects typing random letters. The task in the current study involves reproducing parts of random letter strings, so the value calculated by our model for the fourth day is consistent with their results.

The speedup in mental time has components due to composition and due to strengthening. The composition component involves a time reduction of .02 each time the function combination is repeated. While this number seems small, the cumulative effect can be quite large. For instance, high-frequency pairs are practiced $4*6 = 24$ times over the four days. This means that they are reduced to $98^{24} = .616$ of their original time. In contrast, the speedup from the first day to the last day due to strengthening is $4^{-.26} = .70$. Thus, in some cases, the speedup due to composition to specific problems can exceed the speedup due to general practice.

3 Summary

Systematic effects of frequency of practice were found for the task of evaluating simple Lisp problems. These problems were composed of the same basic four functions, but combinations of them were seen with varying frequency. The difference in number of presentations of analogous pairs was not great, 24 versus 8 times, and yet there are differences found in performance time measures and in error patterns. These differences carry over at least to some extent to the combining of pairs to make triples.

The general and specific practice effects were fitted well by a mathematical model based on the predictions of the ACT* theory of skill acquisition. This model indicated that the composition of operators is very problem specific and can account for differences in performance on problems that seem very similar on the surface.

These results lend empirical support to the theoretical speculations about the knowledge level to which compilation processes apply. The steps of the procedures appear to be encoded in productions at a very specific level at the beginning. Practice mechanisms then act to combine these steps into productions involving more information and action as the skill becomes mastered. Our results also indicate that the combining of operator sequences is indeed a process underlying a part of the speeding up of performance time consistently found with practice. This action of collapsing production steps combines with the general practice effect of strengthening existing productions. These processes combine to produce the speedup in performance that has been documented and described for many skills, but that is not well understood in terms of the mechanisms producing it.

We frankly were surprised how well the simple mathematical model did in fitting a complex pattern of data from many different conditions. That model assumed that

1. Evaluation in all cases involves a serial set of basic operations.
2. Each operation speeds up at a fixed rate independent of context.
3. Sets of operators collapse into single operators, producing an additional benefit.

The fact that this model fitted the complex pattern found in the data is strong evidence for the general conception of skill acquisition we have advanced. These theories can then be applied to the problems of complex skill learning involved in human-computer interaction domains.

The model we presented deals with the transition from complete novice to relatively expert performance. We have offered empirical evidence for the ACT* theory of skill acquisition, which has subsequently been used to model acquisition of text editing skills (Singley and Anderson, 1985) and to guide the design of intelligent tutoring systems for Lisp and geometry (Anderson et al., 1984; Anderson, Boyle, and Yost, 1985). We feel, as do Card, Moran, and Newell, that detailed performance models such as these are useful for supporting and verifying general theories of skill acquisition and cognition (Anderson, 1985) and that these types of theories will be essential in building a general theory of interface design.

References

Anderson, J. R. (1983). Acquisition of cognitive skill. *Psychological Review*, 89, 369–406.

Anderson, J. R. (1983). *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.

Anderson, J. R. (1985). Skill acquisition: Compilation of weak-method problem solutions. CMU Technical Report.

Anderson, J. R., and Jeffries, R. (1983). Novice LISP Errors: Undetected losses of information from working memory. CMU Technical Report.

Anderson, J. R., Boyle, C. F., and Yost, G. (1985). The Geometry Tutor. *Proceedings of IJCAI*, Los Angeles, CA.

Anderson, J. R., Boyle, C. F., Farrell, R., and Reiser, B. J. (1984). Cognitive principles in the design of computer tutors. CMU Technical Report.

Atkinson, R. C., and Juola, J. F. (1973). Factors influencing speed and accuracy of word recognition. In S. Kornblum (Ed.), *Attention and Performance IV*. New York: Academic Press.

Brown, J. S., and VanLehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, 4, 379–426.

Card, S. K., Moran, T. P., and Newell, A. (1983). *The Psychology of Human-Computer Interaction*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Chandler, J. P. (1965). STEPIT Modelling Program. Quantam Chemistry Program Exchange, Indiana University Chemistry Dept., Bloomington, IN.

Gannon, J. D. (1976). An experiment for the evaluation of language features. *International Journal of Man-Machine Studies*, 8, 61–73.

Gay, I. R. (1973). Temporal position of reviews and its effect on the retention of mathematical rules. *Journal of Educational Psychology*, 64, 171–182.

Green, T. R. G. (1977). Conditional program statements and their comprehensibility to professional programmers. *Journal of Occupational Psychology*, 50, 93–109.

Hayes-Roth, F., Waterman, D. A., and Lenat, D. B. (1983). *Building Expert Systems*. Reading, MA: Addison-Wesley.

Hintzman, D. L. (1974). Theoretical implications of the spacing effect. In R. L. Solso (Ed.), *Theories in Cognitive Psychology: The Loyola Symposium*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Mayer, R. E. (1976). Comprehension as affected by the structure of problem representation. *Memory and Cognition*, 4, 249–255.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (1983). *Machine Learning: An Artificial Intelligence Approach*. Palo Alto, CA: Tioga Publishing Co.

Neisser, U., Novick, R., and Lazar, R. (1963). Searching for ten targets simultaneously. *Perceptual and Motor Skills*, 17, 955–961.

Neves, D. M., and Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition*. Hillsdale, NJ: Erlbaum.

Newell, A., and Rosenbloom, P. R. (1981). Mechanisms of skill acquisition and the law of practice. In J. R. Anderson (Ed.), *Cognitive Skills and Their Acquisition*. Hillsdale, NJ: Erlbaum.

Reynolds, J. H., and Glaser, R. (1971). Effects of repetition and spaced practice upon retention of a complex learning task. In M. D. Glock (Ed.), *Guiding Learning*. New York: John Wiley.

Rosenbloom, P. R., and Newell, A. (1982). Learning by chunking: a production system model of practice. Technical Report No. 82–135, Department of Computer Science, Carnegie-Mellon University.

Schneiderman, B. (1980). *Software Psychology*. Cambridge, MA: Winthrop.

Sheil, B. A. (1981). The psychological study of programming. *Computing Surveys*, 13, 101–120.

Simon, H. (1983). Why should machines learn? In R. Michalski, J. Carbonell, and T. Mitchell (Eds.), *Machine Learning*. Palo Alto, CA: Tioga Publishing Co.

Singley, K., and Anderson, J. R. (1985). The transfer of text-editing skill. *International Journal of Man-Machine Studies*, 22, 403–423.

Winkleman, J. H. (1974). The repetition effect in mental arithmetic. PhD dissertation, University of Oregon.