The Process of Learning LISP

Frederick G. Conrad and John R. Anderson

Department of Psychology

Carnegie-Mellon University

Modern research on skill acquisition is dominated by two broadly defined schools. One of these schools analyzes verbal protocols produced by a small number of subjects as they perform complex tasks in unconstrained situations. The other school measures the time and accuracy of larger numbers of subjects performing simplified tasks in controlled experiments. While protocol studies can provide rich detail about the acquisition of useful skills (Ericsson & Simon, 1985), the small numbers of subjects, the free ranging quality of their reports, and the theory laden interpretation of the data make reliability and generality unrealistic. The experimental approach, in contrast, yields precise, objective findings, but the distilled tasks given to subjects jeopardize the everday relevance of these findings (Neisser, 1976). In the current paper we report a series of analyses that exploit the power of both protocol and experimental methods while, we hope, transcending their limitations. We have tracked the acquisition of a complex skill, LISP programming, by examining student performance with the LISP Intelligent Tutoring System (Anderson & Reiser, 1985). This tool provides us with detailed, continuous records of student performance, much like protocol data; however, the nature of these data allows us to analyze them with the quantitative methods typically applied to experimental results. By converging these research methods, our goal is to present a more complete and accurate picture of learning LISP than is possible through either approach alone.

Students interact with the LISP tutor as they solve programming exercises in a one semester, introductory LISP course. Every exercise in the curriculum is decomposed into a sequence of production rules (Anderson, 1983), most of which correspond to typing LISP code. The student's input stream is segmented into separate productions when he or she types a delimiter key (space bar, carriage return or closing parenthesis). The input bracketed by each pair of delimiters is treated as the action of an underlying production rule. For each student action, the tutor tries to generate a corresponding piece of code by applying a relevant production rule. If the student and tutor code match, the student progresses through the problem. For each such match the tutor records the student's coding time, measured from delimiter to delimiter. A discrepancy between student and tutor code is recorded as the student's error and triggers a tutorial dialogue. The student then receives subsequent opportunities to produce the correct code. In the analyses that we report below, we restrict our discussion to latency measures. We further narrow our data by considering latencies from only those trials on which productions are executed without errors.

We are taking the production model in the LISP tutor as the simulation of each student and are using the LISP tutor to automatically analyze the student's protocol and determine a correspondence between that protocol and the production model. This paper will be concerned with how systematic the data appear under such an analysis. We expect that

students will improve as they go through the LISP tutor. To the extent that this improvement is systematic, this will both support the production model and provide evidence about the nature of skill acquisition.

A single LISP symbol (piece of code) can corrspond to several different production rules. What differs are the conditions on the rules. For example, one rule corresponds to *car* when used simply to retrieve the first element of a list; another rule is applied when *car* is called in conjunction with *cdr* to produce the second element of the list. The mean coding times for these rules in the first lesson are about 19 seconds and 45 seconds respectively. This example and many similar ones support the idea that psychologically, there is a fundamental difference between distinct rules with identical actions. While the mapping between surface behavior and underlying rule is usually one to one, the student must still acquire on the order of 200 production rules throughout the term. The various combinations of rules throughout the coding exercises produce about 3400 data points per student. We would like to identify the factors that influence performance at essentially each of these points.

Regression Analysis

We will restrict our analyses to 43 students who used the LISP tutor in courses taught in the fall of 1985 and spring of 1986. Because the data supplied by the tutor are continuous and quantitative, they lend themselves to analysis by multiple regression. Our goal is to formulate a regression model that allows us to predict coding latency at any point in the 125 problems that our students are required to solve. We considered a large number of potential predictors but only a few made independent contributions to predicting the variance We have been most successful in accounting for coding latencies using three predictor variables: (1) lesson number -- the material covered by the tutor is presented in 12 lessons. (2) log number of opportunities to code a production within a lesson, and (3) log absolute serial position of a production within a problem. Our criterion is actually log seconds. We chose a log scale for the dependent measure and two of the independent variables and a linear scale for the remaining independent variable because this combination of scales accounted for significantly more variance than other possible combinations.

We have subdivided our data into "Old" and "New" productions and performed separate analyses on each group. An observation is analyzed as an Old production if the associated rule is introduced in an earlier lesson. If the observation is made within the lesson in which the rule is introduced, it is a New production. So for example, conditionals are introduced in Lesson 3. The production *code-cond* is therefore New when used in Lesson 3, but Old when used in Lessons 4 - 12. Our regression model for Old productions is presented in Equation 1 below and our model for New productons in Equation 3. To make these more interpretable, we have derived equations from 1 and 3 in which the same independent variables, scaled in linear units, predict latencies in actual seconds. These are presented in Equations 2 and 4 for Old and New Items respectively.

$$\text{log latency} = 1.31 - .01(L) - .25(\log O) - .26(\log P) \quad (1)$$

$$\text{latency} = 20.6\ (.97^{L})\ (O^{-.25})\ (P^{-.26}) \quad (2)$$

$$\text{log latency} = 1.36 - .03(L) - .30(\log O) - .15(\log P) \quad (3)$$

$$\text{latency} = 23.0\ (.94^{L})\ (O^{-.30})\ (P^{-.15}) \quad (4)$$

where L is lesson number, O is opportunity number and P is position.

Both regression equations account for highly significant proportions of the variance (r^2 = .17, $p < .001$, for Old productions; r^2 = .18, $p < .001$, for New productions) and all of the regression coefficients are significant ($p < .001$). The first thing to notice is that the intercept term, expressed in log seconds, is greater for New productions than for Old productions. On intuitive grounds, rules that are in the process of being learned should be used more slowly than well known rules. Overall, mean log latencies are in fact slower for New productions than Old (F [1, 840], = 8.87, $p < .001$). A second feature of these equations is that all the coefficients are negative. This means that as the values of our predictor variables increase, students' coding latencies decrease. It is interesting that both regression analyses identified the same independent variables. Among the predictors that we considered and rejected are the rated prior familiarity of productions, the number of goals pending in the problem for a particular observation, the depth of the symbol being coded within embedding function calls, and the position of the symbol on its line of pretty printed code. We will now consider each of the three independent variables that we have included in the models.

Lesson Effect: Learning to Learn.

Our students clearly improve over the course of the 12 lessons. The linear trends due to Lesson alone are negative and close in size to the Lesson coefficients in the three term models presented above (t[6409] = -7.29, $p < .001$, for Old productions; t[3350] = -12.73, $p < .001$, for New productions). Mean coding time in seconds is presented for each lesson in Figure 1. Although the curves show somewhat of an upturn in the final few lessons, their important feature is a general negative trend. The speedup due to lessons cannot be a simple practice effect. If it were, New productions would not show improvement: By definition, performance on these items is measured in only one lesson, namely the one in which they are first presented. Yet, the advantage of additional lessons is greater for New productions than Old items (t[9758] = 7.47, $p < .001$). A second possibility is that the Lesson effect is an "interface effect." The more experience students have with the tutor's display properties, for example, the more rapidly they can enter code. While the improvement on Old items may well be due to increased familiarity with the interface, we interpret the greater lesson effect for New items as evidence of yet another learning process.

Instead of an interface effect, we may be observing students' acquisition of schemas for coding LISP functions. For example, students may come to know that a LISP function must start with the function name followed (typically) by a fixed number of arguments. As the

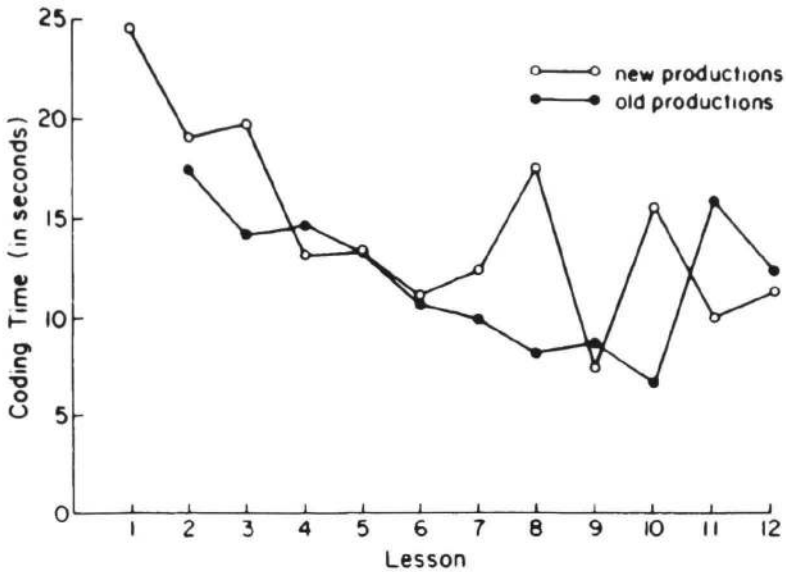

Figure 1: Mean coding time for Old and New productions as a function of lesson.

student encounters new functions, new productions can be created as instantiations of this general tendency. The benefit of assimilating new information with the help of schemas is well documented (*e.g.* Bransford, Barclay & Franks, 1972; Bower, Black & Turner, 1979). In the current context, we refer to this as the "learning to learn" phenomenon. One implication of the phenomenon is that learning certain LISP constructs will require students to update and elaborate their function schemas. If there is no longer a correspondence between the schema and the code students must write, then the benefits of the schema for learning New rules should be reduced until students revise their schemas. This should occur when the conditional structure and recursive structures are introduced. This material is presented in Lesson 3 (conditionals), Lesson 7 (numeric recursion), Lesson 8 (*cdr-recursion*) and Lesson 10 (*car-cdr* recursion). It is at these points in Figure 1 that we see the peaks for New productions, suggesting that the Lesson effect is modulated by lesson content and presumably the impact of content on student schemas.

Opportunity Effect: Knowledge Compilation.

The second term in both of our regression models is log opportunity to code a production within a lesson. An opportunity is essentially a trial in experimental terminology. It is therefore not surprising that students apply particular rules more quickly with practice. However, we have theoretical reasons for expecting that the improvment due to practice is not strictly linear. On the first opportunity to execute a New production, the student has only read the information that will form the basis of the rule. Within the framework of ACT* (Anderson, 1983), this *declarative* knowledge must be *proceduralized*, i.e. compiled into a production rule. Proceduralization explicitly builds certain long term information into the production that previously required maintenance in working memory. Because of the reduced demand on working memory, the student can apply the rule more efficiently. If this is so, we should see a sharp improvement in coding latencies early in the history of a production, followed by more gradual speedup. Once the new information is proceduralized, the resulting rule can be *tuned*, either facilitating or impeding its use, but the greatest change in a rule's performance is associated with its transformation from declarative to procedural knowledge

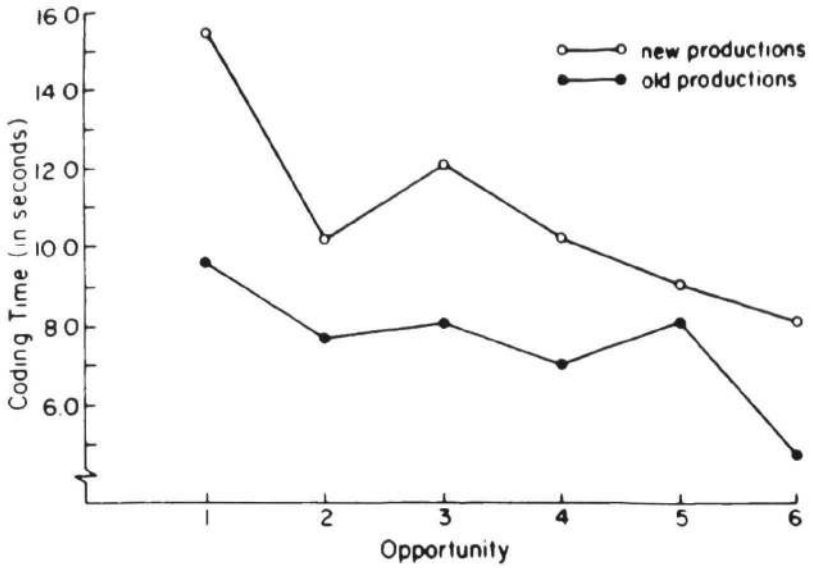

Figure 2: Mean coding time for Old and New productions as a function of opportunity.

This is largely the picture that emerges from our data. Students improve more on New productions than on Old ($t[9758] = 5.88$, $p < .001$) as you would expect if the New items reflect the transition from declarative to procedural kowledge. While we have performed our regression analyses on log coding latencies and log opportunity, the dramatic shifts in performance that we are looking for should be most evident in the untransformed data. In Figure 2, we plot coding time in seconds against actual opportunity count for Old and New productions. The most salient aspect of the Figure is the steep drop in coding time between the first and second opportunity. The improvement is on the order of 33% for New rules versus 20% for Old rules. It seems reasonable to us that this is the difference between proceduralization and tuning. It is also worth noting that at their slowest, Old productions fire about as fast as well practiced New productions. The suggestion is that there is a fundamental difference between the knowledge driving performance on the first opportunity for New productions and the remaining points on the graph. We would expect this if students are using declarative knowledge on this first observation alone.

Serial Position Effect: Plan Rehearsal.

Absolute serial position is the final performance factor that we will consider. This is simply the ordinal value of a production in the sequence of rules comprising a function definition. So for example, *defun* will typically be in the first serial position, *specify-function-name* in the second, and so on.[1] The negative regression coefficients for log absolute serial position tell us that the further into an exercise a student proceeds, the faster he or she will apply production rules. However, a closer examination of the data reveals systematic deviations from the linear trend captured by the regression models.

---

[1] Of course most rules can appear in a variety of positions.

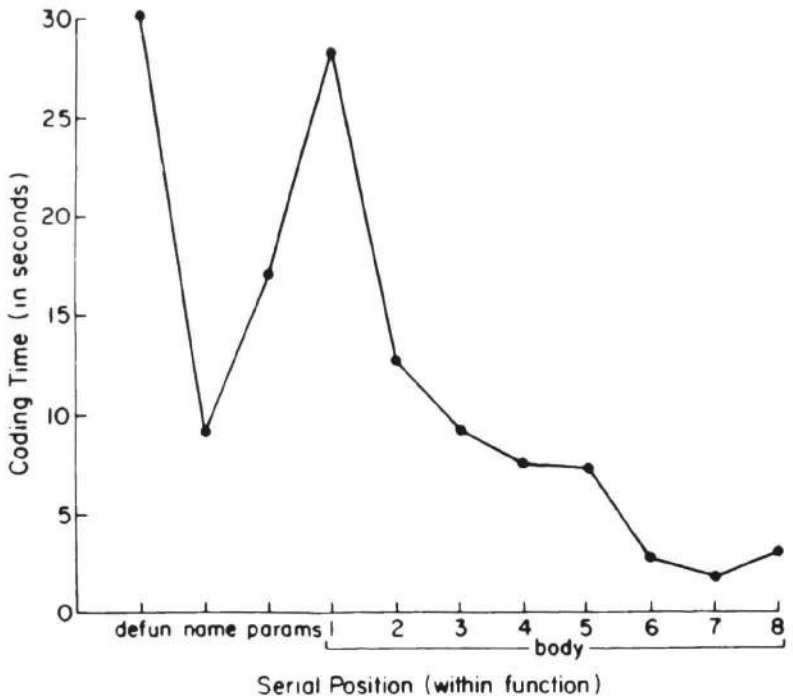

Figure 3: Mean coding time across serial positions in Lesson 2 functions.

We have assembled a profile of coding performance over serial position for the 12 exercises of Lesson 2. Both Old and New productions are included in this analysis which is presented in Figure 3. Latencies and serial position are untransformed in the plot to accentuate the non-linearity of the curve. We have distinguished the first three serial positions, which correspond to the same productions across problems, from the remaining positions, which are used to code the function body. What is striking about this curve is the twin peaks. The first peak likely reflects the final comprehension of the problem description and planning of the solution that precedes coding. The drop over the next two positions is not surprising since these productions are the only permissable actions at these points. We then see the second peak followed by a fairly systematic and negatively-accelerated speed up across serial positions as our regression analysis implies. We think this reflects an effect of formulating and rehearsing the plan. Repeated access and use of the plan should strengthen its encoding in working memory and so speed access to it. It should be noted that in our search for predictors log serial position proved to be a better predictor than other possible variables such as number of pending goals or depth of embedding of the code.

Conclusion

The LISP tutor has enabled us to analyze a complex skill in greater detail than is possible with more conventional methodologies. We feel there is a bright future for tutoring systems in psychological research. Despite the volumes of data and the fine grain size of our analyses, the picture we come away with is actually rather simple. Students' mastery of LISP is best captured by their progress through the curriculum (the lesson effect), the amount of

practice they receive on a particular production rule (the opportunity effect), and the location of that rule within the particular function that they are coding (the serial position effect). This simplicity is particularly striking.

We think that each of the three variables is in the main quite interpretable although they may also be picking up subtle complexities. The lesson effect seems to reflect the schema for working in LISP both with respect to the interface and the structure of LISP functions. The opportunity effect is a simple learning effect. It is noteworthy that the linear relationship between log latency and log opportunity indicates we have another instance of a power-law learning function (Anderson, 1982). The prediction is not as good if we use either linear time or linear practice. Finally, the Serial Position effect indicates the development of an initial plan which becomes rehearsed as it is used in writing the function. Each of these phenomena are hardly news in cognitive psychology. It is this fact that we regard as the news of our paper: Under appropriate decomposition (*i.e.*, as provided by the LISP tutor) acquisition of a complex skill is a matter of simple learning processes.

References

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

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

Anderson, J. R. & Reisser, B. J. (1985, April). The LISP tutor. *Byte*, pp. 159 - 175.

Bower, G. H., Black, J. B. & Turner, T J. (1979). Scripts in memory for text. *Cognitive Psychology, 11*, 177 - 220.

Bransford, J. D., Barclay, J. R. & Franks, J. J. (1972). Sentence memory: A constructive versus interpretive approach. *Cognitive Psychology, 3*, 193 - 209.

Ericsson, K. A. & Simon, H. H. (1984). *Protocol analysis: Verbal reports as data.* Cambridge, Ma.: The MIT Press.

Neisser, U. (1976). *Cognition and reality.* San Francisco: W. H. Freeman and Company