# Knowledge Tracing in the ACT Programming Tutor[1]

**Albert T. Corbett and John R. Anderson**
Psychology Department
Carnegie Mellon University
Pittsburgh, PA 15213
corbett@psy.cmu.edu
anderson@psy.cmu.edu

## Abstract

The ACT Programming Tutor provides assistance to students as they write short computer programs. The tutor is constructed around a set of several hundred programming rules that allows the tutor to solve exercises step-by-step along with the student. This paper evaluates the tutor's student modeling procedure that is used to guide remediation. This procedure, termed *knowledge tracing*, employs an overlay of the tutor's programming rules. In knowledge tracing, the tutor maintains an estimate of the probability that the student has learned each of the rules. The probability for a rule is updated at each opportunity to apply the rule, based on the student's performance. The predictive validity of the modeling procedure for tutor performance accuracy and posttest performance accuracy is assessed. Individual differences in learning parameters and cognitive rules are discussed, along with possible improvements in the modeling procedure.

## The ACT Programming Tutor

This paper reports an assessment of student modeling in the ACT Programming Tutor (APT). APT is a practice environment for students learning to program in Lisp, Pascal or Prolog (Anderson, et. al., in press). The tutor presents exercises that require students to write short programs and provides assistance as the students code their solutions. This report focuses on the initial seven sections of the Lisp curriculum. Table 1 displays example exercises from the first and last of these sections. The first section introduces two basic data types, atoms (symbols) and lists (groupings of symbols), and three functions that extract information from a list. By the seventh section, students are learning to define new functions that employ these three extractor functions in combination with three other functions that construct new lists.

In working with the tutor, students enter exercise solutions top-down with an interface that is similar to a structure editor. The first exercise requires two coding cycles: first the student enters *car* (either by typing or through menu selection) then types the literal list '(c d e). The second example requires eight coding cycles. The student enters ***defun*** (to define a new function), then codes the new function name, declares two variables (two cycles) and codes the body of the function (four cycles). The last exercise also requires three additional interface manipulation cycles in which unneeded editor nodes are deleted. The tutor monitors student performance on a cycle-by-cyle basis and provides immediate feedback to keep the student on a correct solution path. If the student makes a mistake, the tutor notifies the student, and allows the student to try again. The tutor does not volunteer any verbal feedback on errors, but the student can request help at each step.

We have been using such tutors to teach programming courses over the past eight years and the Lisp and Prolog modules are currently used to teach a self-paced introductory course. Overall, the tutors have proven effective. Students using the tutor generally work through exercises more quickly and perform as well or better on posttests (Anderson & Reiser, 1985; Corbett & Anderson, 1990, 1991). Despite this general effectiveness, however, some students flounder. As a result, we incorporated a student modeling and remediation mechanism into the tutor. Recently we have begun evaluating the validity of this mechanism.

1 This research was supported by the Office of Naval Research, grant N00014-91-J-1597.

Section 1 Example Exercise

Write a Lisp function call that returns c from the list (c d e).

Answer: (car '(c d e))

Section 7 Example Exercise

Define a function named replace-first that takes two arguments. Assume the second argument will be a list. The function replaces the first element of the list with the first argument. For example,

(replace-first 'rose '(tulip daisy iris))
returns (rose daisy iris).

Answer: (defun replace-first (itm lis)
(cons itm (cdr lis)))

Table 1. Two Lisp Exercises drawn from the initial and final sections of the curriculum under review.

## Student Modeling

APT is constructed around a set of several hundred production rules for writing programs, called the *ideal student model*, which allows the tutor to solve the exercises step-by-step along with the student. The tutor attempts to match the student's action at each step to an applicable rule in the ideal model in a process we call *model tracing*. The ideal student model also serves as an overlay model of the individual student's knowledge state (Goldstein, 1982). As the student works, the tutor maintains an estimate of the probability that the student has learned each rule in the ideal model. At each opportunity to apply a rule, the probability that the student knows the rule is updated contingent on the accuracy of the student's action. This process, which we call *knowledge tracing*, serves as the basis for remediation in the tutor. A small set of coding rules is introduced in each section of the curriculum and after the student completes a minimal set of required exercises, the tutor continues presenting remedial exercises in the section until the student has "mastered" each rule in the set. Mastery is defined in the tutor as a learning probability of at least 0.95.

The knowledge tracing mechanism passed a minimal validity test when it was first introduced. Posttest scores were higher when the remediation mechanism was in operation (Anderson, Conrad, & Corbett, 1989). Recently, we completed a more detailed assessment of knowledge tracing on the basis of both tutor and posttest data (Corbett & Anderson, 1992). While knowledge tracing is intended to infer a student's knowledge state for the purpose of guiding practice, the underlying cognitive and learning models can be used to predict a student's accuracy in completing tutor exercises. At each goal (step) in solving the exercises, we can estimate the probability of a correct response given the student's history. To assess the model, we compared actual and predicted accuracy across subjects at each goal. Knowledge tracing performed moderately well in this validity check. Across the 158 coding steps in the required exercises, actual and predicted accuracy were reliably correlated, $r = 0.47$.

In principle, the final production rule learning probabilities should predict posttest performance just as they predict tutor accuracy. However, the final probabilities are tightly distributed between 0.95 and 1.0 after knowledge tracing, so there is little potential to test this hypothesis. A related prediction is that if mastery learning is successful, posttest performance should not correlate with the number of mistakes students make in achieving mastery. The student modeling mechanism did not do as well by this criterion. The number of exercises required to reach criterion was reliably correlated with posttest performance, $r = -0.52$. The more exercises students completed in reaching criterion, the worse they did on the quiz.

## Revising the Models

While the model predicted tutor performance reasonably well, there were systematic deviations in the fit that could be traced to deficiencies in both the underlying mathematical and cognitive models. As described below, the mathematical model employs four parameters. The values employed by the tutor, which were estimated from prior tutor data and held constant across production rules in the ideal model, resulted in a substantial underprediction of the variability in accuracy across goals in the lesson. We refit the data after the fact, allowing the four parameter values to vary across productions. The best fitting estimates yielded a substantially better fit, r =

0.85. In this fit, predicted and actual accuracy still deviated substantially for some rules, suggesting that the ideal model requires revision. In this paper we report an evaluation of a revised model.

## The Study

The study assesses the internal and external validity of the APT knowledge tracing mechanism over the first seven sections of the Lisp curriculum.

### The Students

Twenty five students worked through the curriculum in the course of completing a class in introductory programming. This was the first college level exposure to programming for all students and the first exposure to Lisp.

### Procedure

Students worked through the exercises at their own pace. In each section of the tutor curriculum students read about Lisp in a text, completed a set of required exercises that covers all the rules introduced in the section, then completed remedial exercises as needed. Remedial exercises are selected by the tutor to bring all productions introduced in the section to a minimum learning probability of 0.95. At the end of the first lesson the students completed a quiz.

### The Curriculum

In the seven sections of the curriculum under investigation, students are introduced to (1) two data types, atoms (symbols) and lists (hierarchical groupings of symbols), (2) function calls (operations) and (3) function definitions. The first section in the curriculum introduces simple lists (flat lists of atoms) and three basic extractor functions, *car*, *cdr* and *reverse*, that return components of or a transformation of a list. The second section introduces three basic functions that form new lists, *append*, *cons* and *list*. In the third and fourth sections students learn to apply the same six functions to hierarchically nested lists. In the fifth section, students are introduced to extractor algorithms - nested function calls applying multiple extractor functions to lists. In the sixth section, students learn to define new functions that perform such extractor algorithms. In the final section students define functions that employ both extractor and constructor functions. A minimum of forty exercises is required to complete this curriculum.

### The Cognitive Model

The cognitive model consists of 35 productions rules. Three rules govern the coding of a single extractor function, either, *car*, *cdr* and *reverse* with a simple list in section 1. A single rule governs the transfer of all these extractors to nested lists in section 2. Five additional rules govern the coding of these three functions in more complex algorithms in section 5. The model distinguishes five additional extraction contexts in the last two sections on function definitions. A single rule governs the coding of any extractor or extractor algorithm in each of these five contexts.

The three constructor functions are employed in three contexts across sections 2, 4 and 7 - simple lists, nested lists and functional arguments. Given students' difficulty with constructor functions, the tutor makes no assumptions concerning the generalization of constructor knowledge. Thus, nine rules are employed to model the three rules in the three contexts. Twelve additional rules model the coding of data structures, the elements of function definitions, notably variable declaration and usage, and some editor manipulations.

This model incorporates three revisions stemming from the prior assessment: (1) separate rules for coding functions with flat and with nested lists, (2) the modeling of extractor algorithms with five functionally defined rules rather than two syntactically defined rules and (3) two distinct rules for declaring the first variable and subsequent variables in a function definition.

### Knowledge Tracing

Knowledge tracing in the tutor assumes a simple two-state learning model with no forgetting. Each rules is either in a learned or an unlearned state. A rule can make the transition from the unlearned to the learned state at each opportunity to apply the rule, but rules cannot make the transition in the opposite direction. The goal in knowledge tracing is to estimate the probability that each rule is in the learned state. After each step in problem solving, the tutor updates this learning probability estimate for the applicable production rule, based on the student's

| | |
|---|---|
| $P_0$ | the probability that a rule is in the learned state before the rule is employed in problem solving for the first time (i.e., from reading) |
| $P_T$ | the probability that a rule will make the transition from the unlearned to the learned state following an opportunity to apply the rule |
| PG | the probability a student will guess correctly if the rule is in the unlearned state |
| PS | the probability a student will slip and make an error when the applicable rule is in the learned state |

Table 2. The four parameters employed in knowledge tracing.

action. The Bayesian computational procedure is a variation of one described by Atkinson (1972). It employs two learning parameters and two performance parameters, as displayed in Table 2. These parameter values are estimated empirically from the previous study and vary freely across the thirty-five rules in the tutor. See Corbett and Anderson (1992) for complete details on estimating learning probabilities and performance accuracy.

## Results

Students completed an average of 23 remedial exercises in addition to the 40 required exercises in working through the curriculum. The number of remedial exercises ranged from 1 to 49. The revised cognitive model and parameter estimates improved the fit of the student model to the tutor accuracy data. A correlation of 0.71 was obtained between empirical and predicted values across the 203 goals in the required exercises. However, a moderate correlation persisted between number of tutor exercises required to reach criterion and posttest performance, $r = -0.44$. We again refit the data after the fact, and the best fitting parameter estimates yielded a better fit to the tutor data, $r = 0.90$. The final learning probabilities in this fit are still tightly distributed and do not correlate reliably with posttest performance.

There are a variety of reasons that the posttest performance may correlate with number of errors required to reach criterion in the tutor. First, the quiz involves transfer to a different coding environment. We might expect such transfer to correlate with the amount of practice required to reach criterion in the tutor. Second, students may be preparing differentially for the posttest, since the assessment draws on data from a course. Again, study habits may correlate with learning rate in the tutor. As a result, some correlation of number of tutor exercises and posttest performance might be expected even if the student model is essentially valid. Nevertheless, we plan to explore alternatives that do reflect on the validity of the student model: individual differences in parameter estimates and the nature of cognitive rules.

**Individual Differences in Parameter Estimates.** While the four parameter estimates vary across productions, they are held constant across subjects. Consequently, we may be underestimating the learning probabilities for students who are performing well and overestimating for students making more errors. As a result, students who are doing few remedial exercises would nevertheless tend to be overlearning while students doing many remedial exercises would tend to be underlearning. This pattern would result in a negative correlation of posttest performance with the number of errors in reaching criterion in the tutor. To assess the magnitude of individual differences, we divided the students into two groups based on posttest accuracy and generated best fitting parameter estimates for their tutor performance. The average estimates for the two learning parameters, $P_0$ and $P_T$, for twelve production rule categories are displayed in Table 3. As can be seen, the mean estimates for the two parameters are roughly 30% and 10% higher respectively, for students who performed well on the quiz. This suggests that we may obtain better fits by individualizing parameter estimates. An immediate goal is to investigate whether applying an individualized multiplicative constant to the group parameter estimates improves the performance of the model.

**Individual Differences in Cognitive Rules.** A second possibility is that different students acquire different cognitive rules. While we can track a student's ability to manipulate symbols in specific contexts, we cannot directly track the student's understanding of those manipulations. Knowledge tracing may insure that students are

| Production Rule Category | High Test Accuracy | | Low Test Accuracy | |
|---|---|---|---|---|
| | $P_0$ | $P_T$ | $P_0$ | $P_T$ |
| Extractors - Flat Lists | 0.78 | 1.00 | 0.68 | 0.97 |
| Constructors - Flat Lists | 0.73 | 0.73* | 0.76 | 0.93* |
| Literal Data Structures | 0.45 | 0.48 | 0.06 | 0.72 |
| Extractors - Nested Lists | 0.87 | 1.00 | 0.68 | 0.69 |
| Constructors - Nested Lists | 0.68 | 0.37 | 0.35 | 0.37 |
| Extractor Algorithms | 0.69 | 0.58 | 0.70 | 0.28 |
| Defun/Function Name | 0.96 | 0.00* | 0.93 | 0.00* |
| Variables | 0.32 | 0.53 | 0.24 | 0.40 |
| Extractor Algorithms Function Body | 0.58 | 0.10 | 0.70 | 0.24 |
| Extractor Algorithms Novel Contexts | 0.63 | 0.31 | 0.25 | 0.09 |
| Constructor Function Body | 0.31 | 0.35 | 0.03 | 0.22 |
| Interface | 0.56 | 0.91 | 0.51 | 0.81 |
| Mean | 0.63 | 0.53 | 0.49 | 0.48 |

*One rule in this set is excluded in computing pT. p0 = 1 for that rule, so pT is inestimable

**Table 3.** Best fitting estimates for $P_0$ (the probability a production is in learned state initially) and $P_T$ (the probability of a transition to the learned state) for twelve production rule categories

learning rules that enable them to complete exercises, but they may not be the rules assumed in the ideal student model. For example, one of the earliest stumbling blocks in learning Lisp is understanding the hierarchical structure of lists. In section 1, however, students are introduced to extractor functions and constructor functions with flat, non-hierarchical lists. As a result, students can apply everyday knowledge of lists to learn the extractor functions without fully grasping the structure of lists. Constructor functions are a second stumbling block in learning Lisp. To master constructors, students must understand the structure of lists and grasp the relationship between the arguments to the constructor function and structure of the resulting list. However, when constructor functions are introduced in the tutor curriculum with flat lists, students can learn rules based on the structure of the arguments alone that do not generalize to later sections.

The parameter estimates in Table 3 suggest that this may be happening. The two learning parameter estimates are quite similar across the two groups for the extractor function rules in section 1 and the constructor functions in section 2. However, when these functions are employed with more complex data structures and in more complex algorithms in later

sections (rows 4, 5, 10 and 11 of Table 3) the learning parameter estimates are generally higher for the high posttest accuracy group. Students read text at the beginning of each section, so these parameter estimates may partly reflect differential initial comprehension of each section. However, such differences would be expected if some students are learning rules that generalize more readily to later contexts.

A final result is also consistent with this possibility. We generated best fitting parameter estimates for just the required exercises that every student completes at the beginning of a section. These estimates fit the required exercise data quite well, r = 0.91. We employed these parameter estimates to generate production learning probabilities on the basis of just the required exercises, as if students had not completed the remedial exercises. The mean probability estimates obtained from the required exercises correlated reliably with posttest accuracy, r = 0.43. This pattern would be expected if the number of opportunities required to master a rule is inversely related to the probability of acquiring an optimal understanding. Some suboptimal rules may capitalize on accidental characteristics of the tutor environment that do not transfer to the posttest environment. Other suboptimal rules may transfer in principle, but may in fact be retained less well. We might be able to model this possibility by decreasing the transition probability $P_T$ with practice. However, it also suggests that students may benefit from explanatory feedback in the context of correct actions as well as in the context of errors.

## Conclusion

On balance, the tutor's knowledge tracing procedure performed fairly well in this assessment. The model accounted for about 50% of the variance in students' performance with the tutor, and about 80% when best fitting parameter estimates were derived. On the other hand, students' learning rate (the number of errors students made in satisfying the model's mastery criterion) correlated negatively with posttest performance. While the student's final knowledge state should predict posttest performance, learning rate in reaching that state should not. We plan to explore possibilities for accomodating individual differences in learning rates by adjusting the model's learning parameters and the underlying cognitive rule set.

The correlation of tutor learning rate and posttest performance in this study may also reflect the fact that the tutor allows students to practice one programming skill, code generation, while the posttest environment allows the students to exercise other skills, e.g., debugging. It should be possible to generalize the knowledge tracing mechanism to other skills, however. Knowledge tracing does not depend on a unique solution path for each exercise, nor on immediate feedback, although these characteristics simplify the task. Rather, what is required is a cogntive model of the task consisting of rules that map onto observable behavior. As a result it should be possible to trace debugging and other skills.

## References


Anderson, J.R., Conrad, F.G. and Corbett, A.T., 1989. Skill acquisition and the Lisp Tutor. *Cognitive Science* 13: 467-505.

Anderson, J.R., Corbett, A.T., Fincham, J.M., Hoffman, D. and Pelletier, R., in press. General principles for an intelligent tutoring architecture. In V. Shute an W. Regian (eds.) *Cognitive approaches to automated instruction.* Hillsdale, NJ: Erlbaum.

Anderson, J.R. and Reiser, B.J., 1985. The Lisp Tutor. *Byte, 10,* (4), 159-175.

Atkinson, R.C., 1972. Optimizing the learning of a second-language vocabulary. *Journal of Experimental Psychology,* 96, 124-129.

Corbett, A.T. and Anderson, J.R., 1990. The effect of feedback control on learning to program with the Lisp Tutor. In The Proceedings of the Twelfth Annual Conference of the Cognitive Science Society. Hillsdale, NJ: Erlbaum.

Corbett, A.T. and Anderson, J.R., 1991. Feedback control and learning to program with the CMU Lisp Tutor. Paper presented at the Annual Meeting of the American Educational Research Association.

Corbett, A.T. and Anderson, J.R., 1992. Student modeling and mastery learning in a computer-based programming tutor. In The Proceedings of the Second International Conference on Intelligent Tutoring Systems. New York: Springer-Verlag.

Goldstein, I.P., 1982. The genetic graph: A representation for the evolution of procedural knowledge. In D. Sleeman and J.S.Brown (eds.) *Intelligent tutoring systems.* New York: Academic.