User Modeling and User-Adapted Interaction **4:** 253–278, 1995.
© 1995 *Kluwer Academic Publishers. Printed in the Netherlands.*

Knowledge Tracing: Modeling the Acquisition of Procedural Knowledge

ALBERT T. CORBETT[1] and JOHN R. ANDERSON[2]
[1]*Human Computer Interaction Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
e-mail: corbett@cmu.edu
[2]*Psychology and Computer Science Departments, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
e-mail: ja@cmu.edu

(Received 1 December 1994; in final form 13 March 1995)

Abstract. This paper describes an effort to model students' changing knowledge state during skill acquisition. Students in this research are learning to write short programs with the ACT Programming Tutor (APT). APT is constructed around a production rule cognitive model of programming knowledge, called the *ideal student model.* This model allows the tutor to solve exercises along with the student and provide assistance as necessary. As the student works, the tutor also maintains an estimate of the probability that the student has learned each of the rules in the ideal model, in a process called *knowledge tracing.* The tutor presents an individualized sequence of exercises to the student based on these probability estimates until the student has 'mastered' each rule. The programming tutor, cognitive model and learning and performance assumptions are described. A series of studies is reviewed that examine the empirical validity of knowledge tracing and has led to modifications in the process. Currently the model is quite successful in predicting test performance. Further modifications in the modeling process are discussed that may improve performance levels.

Key words: Student modeling, learning, empirical validity, procedural knowledge, intelligent tutoring systems, mastery learning, individual differences

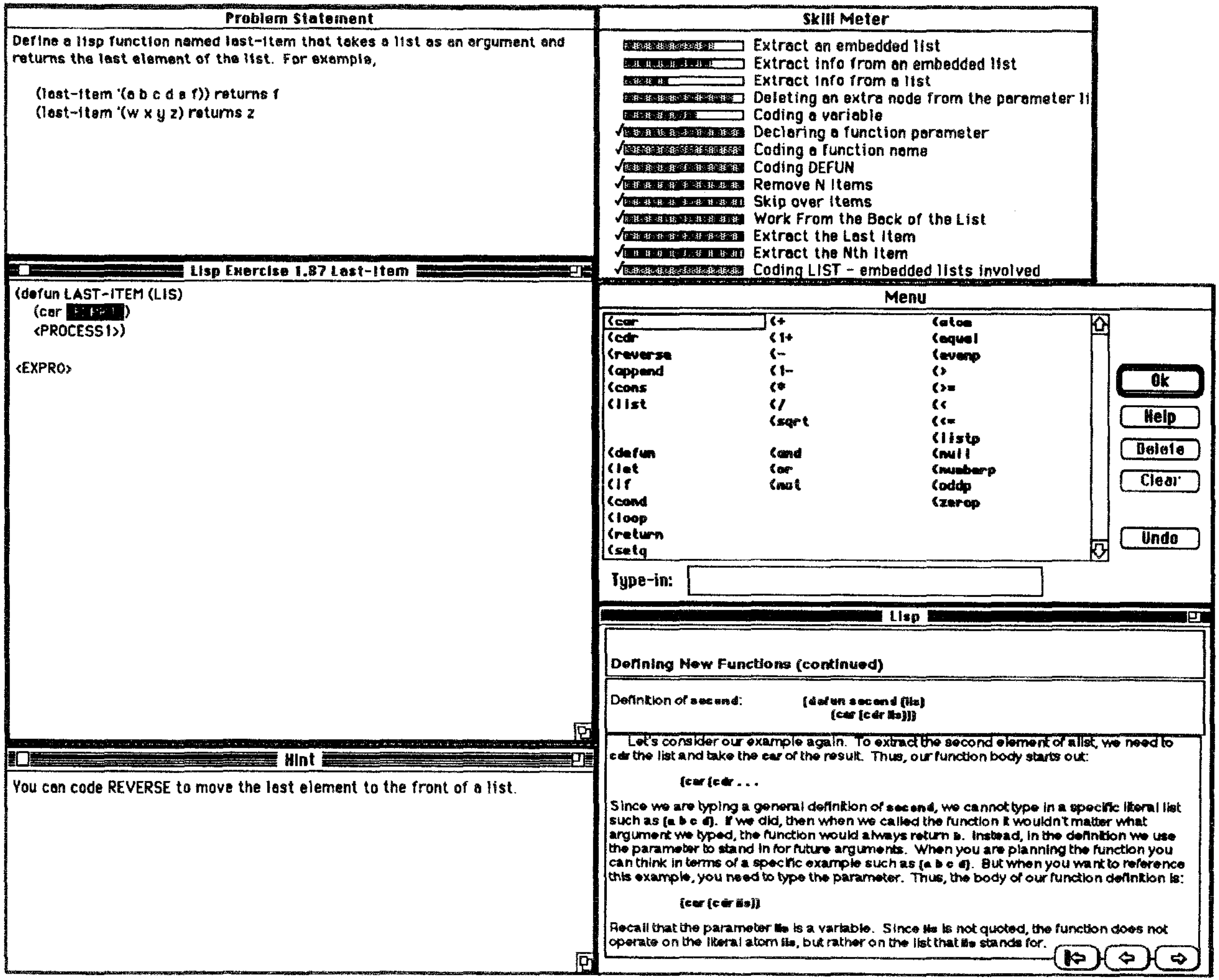
Thirty years ago three influential papers (Bloom 1968; Carroll 1963; Keller 1968) outlined the promise of mastery learning. The core idea is that virtually all students can achieve expertise in a domain if two conditions are met: (1) the domain knowledge is appropriately analyzed into a hierarchy of component skills and (2) learning experiences are structured to ensure that students master prerequisite skills before tackling higher level skills in the hierarchy. This is a tantalizing idea that became entrenched in American education during the 70's, although it can be traced to far earlier times. Review articles overwhelmingly confirm that mastery learning leads to higher mean achievement levels than conventional conditions (Block & Burns 1976; Guskey & Pigott 1988; Hyman & Cohen 1979; Kulik, Kulik & Bangert-Drowns 1990; Kulik, Kulik & Cohen 1979), but critics note that the effect sizes are appreciably smaller than predicted (Resnick 1977; Slavin 1987) and the underlying assumptions concerning skill decomposition are controversial (Resnick & Resnick 1992; Shepard 1991).

In this paper we attempt to bring a cognitive model of skill acquisition to bear on the goals of mastery learning. We attempt to monitor students' changing knowledge state as they practice a complex cognitive skill, to individualize the practice sequence to enable students to master the skill efficiently and to accurately predict students' performance. Our interest in student modeling and individualized instruction has arisen in our work with intelligent programming tutors over the past ten years (Anderson, Boyle, Corbett & Lewis 1990; Anderson, Conrad & Corbett 1989; Anderson, Corbett, Fincham, Hoffman & Pelletier 1992; Anderson, Corbett, Koedinger & Pelletier, in press). These tutors are practice environments in which students write short programs. Each tutor is constructed around a cognitive model of the underlying skill that allows the tutor to solve the exercises along with the student and provide feedback on student actions. These tutors have proven to be effective learning environments; students can work through a fixed set of exercises in substantially less time than students working on their own and score as well or better on posttests. The present research attempts to build on this success. By incorporating learning and performance assumptions with a complete model of the cognitive skill in an environment that demonstrably speeds learning, we hope to enable all students to achieve mastery within a practical time frame.

In the following sections we describe the intelligent tutoring environment, the curriculum, the underlying cognitive model and the learning and performance assumptions. Following that we review a succession of empirically driven refinements in the modeling process. These refinements have brought us part way to our goal and we discuss future directions.

1. The ACT Programming Tutor

The ACT Programming Tutor is a practice environment in which students write short programs in Lisp, Prolog or Pascal. The Lisp and Prolog modules are presently employed in a self-paced introductory programming course at Carnegie Mellon University and the Pascal module is used in an introductory programming course in a Pittsburgh high school. Figure 1 depicts the tutor's screen display midway through a Lisp exercise. The student has previously read through a multi-page section of text in the window at the bottom right and is now completing a corresponding sequence of exercises. The exercise description appears in the window at the upper left and the student's solution appears in the code window immediately below. The tutor interface is similar to a structure editor. The student selects operator templates from the menu on the right and enters identifiers and constants through the type-in buffer beneath the menu. In this example, the student has already selected the *defun* template for defining a new Lisp operator. The tutor recognized this as a correct step and has expanded the template for this operator in the code window, *(defun <name> (<parameter>) <body>)*. The three angle-bracket symbols in this



Problem Statement

Define a lisp function named last-item that takes a list as an argument and returns the last element of the list. For example,

```
(last-item '(a b c d e f)) returns f
(last-item '(w x y z) returns z
```

Skill Meter

- Extract an embedded list
- Extract info from an embedded list
- Extract info from a list
- Deleting an extra node from the parameter li
- Coding a variable
- ✓ Declaring a function parameter
- ✓ Coding a function name
- ✓ Coding DEFUN
- ✓ Remove N Items
- ✓ Skip over Items
- ✓ Work From the Back of the List
- ✓ Extract the Last Item
- ✓ Extract the Nth Item
- ✓ Coding LIST - embedded lists involved

Lisp Exercise 1.87 Last-Item

```
(defun LAST-ITEM (LIS)
  (car [illegible])
  <PROCESS1>)

<EXPRO>
```

Hint

You can code REVERSE to move the last element to the front of a list.

Defining New Functions (continued)

Definition of second: (defun second (lis)
(car (cdr lis)))

Let's consider our example again. To extract the second element of a list, we need to cdr the list and take the car of the result. Thus, our function body starts out:

(car (cdr . . .

Since we are typing a general definition of second, we cannot type in a specific literal list such as (a b c d). If we did, then when we called the function it wouldn't matter what argument we typed, the function would always return b. Instead, in the definition we use the parameter to stand in for future arguments. When you are planning the function you can think in terms of a specific example such as (a b c d). But when you want to reference this example, you need to type the parameter. Thus, the body of our function definition is:

(car (cdr lis))

Recall that the parameter lis is a variable. Since lis is not quoted, the function does not operate on the literal atom lis, but rather on the list that lis stands for.

Fig. 1. The APT Lisp Tutor interface.

template are editor nodes that the student replaces with Lisp code. In Figure 1 the student has already (1) expanded the first node, <*name*>, by typing the name of the function *last-item*, (2) expanded the <*parameter*> node with a single variable, *lis*, and (3) expanded the body of the function <*body*> with the operator template *(car <expr1>)*. As this template indicates, *car* takes a single argument, represented by the node <*expr1*>. This node is selected (highlighted) in the figure and the student can expand the node with Lisp code or select a different node to work on.

The Lisp tutor is constructed around a set of several hundred language-specific rules for writing programs called the *ideal student model*. The ideal student model is a complete, executable model of procedural knowledge in the domain. These rules represent programming knowledge at the grainsize of individual symbols and are used to interpret student actions in a process called *model tracing*. At each step in writing a program the student's action is compared to applicable rules in the ideal model and immediate feedback is conventionally provided. If the student action matches an applicable rule, the tutor accepts the action and fires the rule to update the code window and update the internal representation of the problem state. If the student action does not match the action of any applicable rule in the ideal model, the action does not register in the code window and the tutor provides a brief message in the hint window at the lower left. Thus, the student always remains on a recognized solution path. The skill meter in the upper right displays the tutor's model of the student's knowledge state. This window represents the focus of this paper. Each of the rules in the ideal model is represented by a bar graph and the shading represents the probability that the student knows each rule. The probability for the top two rules in the window is around 0.75 and around 0.35 for the third rule. The checkmarks at the bottom of the screen indicate that the student has 'mastered' the corresponding rules.

2. The Cognitive Model

The tutors reflect the ACT-R theory of skill knowledge (Anderson 1993). This theory assumes a fundamental distinction between declarative knowledge and procedural knowledge. Declarative knowledge is factual or experiential. For example, the following sentence and example in the Lisp text would be encoded declaratively:

> The Lisp function *car* takes one list as an argument and returns the first element. For example,
> *(car '(a b c d))* returns *a*

Procedural knowledge, in contrast, is goal-oriented and mediates problem-solving behavior. ACT-R assumes that skill knowledge is encoded initially in declarative form through experiences such as reading. Early in practice the student solves problems by applying general procedural rules to this domain-specific knowledge. As a consequence of this early activity, domain-specific procedural knowledge is acquired. With subsequent practice, both declarative and procedural knowledge are strengthened so that performance grows more rapid and reliable. Like many cognitive theories (cf. Kieras & Bovair 1986; Newell 1990; Reed, Dempster & Ettinger 1985) ACT-R assumes that procedural knowledge can be represented as a set of independent production rules that associate problem states and problem-solving goals with actions and consequent state changes. The following two goal-oriented productions can be derived from the declarative examples above through practice in writing function calls and evaluating function calls respectively:

> IF the goal is to write an expression that returns the first element of a list,
> THEN code the operator *car* and set a goal to code the list as an argument.

> IF the goal is to evaluate an application of *car* to a list,
> THEN write the first element in the list.

Each of the programming tutors contains many rules for writing programs that model problem solving at the grainsize of individual symbols. For example, six productions are employed in writing the full solution to the exercise displayed in Figure 1:

(defun last-item (lis) (car (reverse lis)))

Figure 2 displays the six production rules that govern the six symbols in this definition. The first production matches the goal that is set up by the problem description. When it fires it sets up subgoals which are satisfied in turn by successive rules. We call the set of programming rules built into the tutor the ideal student model, because it embodies the knowledge that the student is trying to acquire. This ideal model plays two roles in the tutor. First, as described above, it allows the tutor to solve exercises step-by-step along with the student in a process we call model tracing. As the student enters each code symbol, the tutor attempts to match the action to an applicable rule in the model. If a match is found, the tutor accepts the symbol and updates its internal representation of the problem state. If not, the tutor requires the student to try another action. Second, in *knowledge tracing*, the student is represented as an overlay of the ideal model (Goldstein 1982). As the student works through the exercises, the tutor maintains an estimate of the probability that the student has learned each rule in the ideal model.

Production Rule	Action
IF the goal is to define a new Lisp operator THEN code defun, and set a goal to code the operator name, set a goal to declare any variables, set a goal to code the function body.	Code ***defun***
IF the goal is to code the function name THEN type the function name.	Type ***last-item***
IF the goal is to declare variables, THEN type one variable for each argument that will be passed to the function.	Type ***lis***
IF the goal is to extract the last element of a list, THEN code car, and set a goal to move the last element to the front of the list.	Code ***car***
IF the goal is to move the last element to the front of a list, THEN code reverse, and set a goal to code the list.	Code ***reverse***
IF the goal is to code a list that is bound to a variable, THEN type the variable.	Type ***lis***

Fig. 2. Six rules that apply in writing the program *last-item* in Figure 1.

2.1. EVALUATING ACT-R PROCEDURAL KNOWLEDGE ASSUMPTIONS

One of the primary goals in our tutoring research has been to evaluate the ACT-R assumption that procedural knowledge maps onto independent production rules. Three types of results provide support for this assumption:

1. A production rule model provides a regular analysis of learning trends. Figure 3 displays the mean learning curve for the production rules that are introduced in

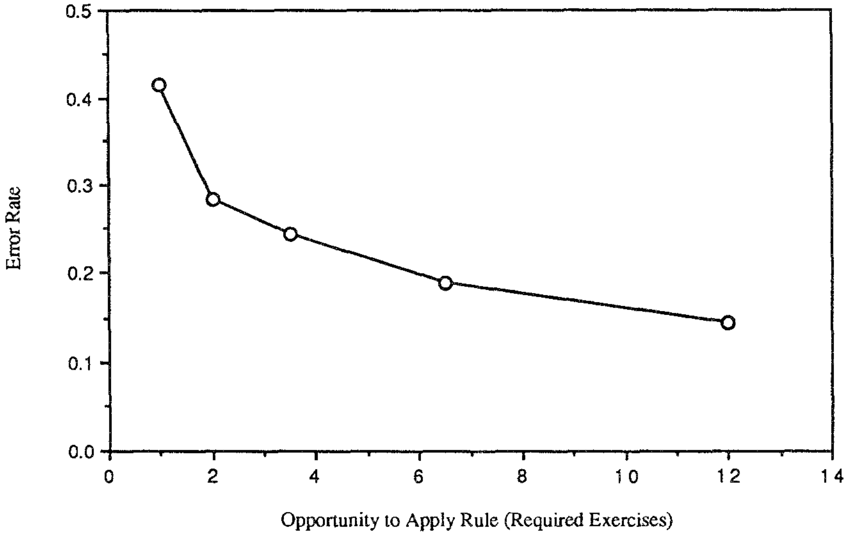


Fig. 3. The mean learning curve for coding rules introduced in the first lesson of the Lisp Tutor curriculum. Mean error rate is plotted for successive opportunities to apply the rules.

the first lesson of the Lisp Tutor curriculum. In this figure, error rate is plotted as a function of practice on the rules. The error rate averages about 0.42 across the first opportunity to apply each rule, then drops monotonically across the successive opportunities to apply the rules. Aggregating the data by other, more superficial units, e.g., over exercises, over problem solving goals within exercises, or over opportunities to employ each surface Lisp construct independent of problem solving context, does not yield systematic learning curves (Anderson, Conrad & Corbett 1989). Kieras & Bovair (1986) provide similar evidence that a production rule analysis yields systematic learning data.

2. Production rule analyses have proven successful in predicting transfer among programming languages (Anderson, Conrad, Corbett, Fincham, Hoffman and Wu 1993) and across text editors (Singley & Anderson, 1989).

3. A variety of results support the assumption that procedural knowledge is goal-specific. For example, consider the two production rules described earlier concerning the simple use of *car*:

IF the goal is to write an expression that returns the first element of a list,
THEN code the operator *car* and set a goal to code the list as an argument.

IF the goal is to evaluate an application of *car* to a list,
THEN write the first element in the list.

They are derived from the same declarative knowledge, but one is specific to writing a function call, while the other is specific to evaluating function calls.

Several studies have shown that practice in one of these two programming skills provides no transfer (Kessler 1988; Anderson, Conrad & Corbett 1989; McKendree & Anderson 1987) or at best limited transfer (Pennington & Nicolich 1991) to the other. Similar evidence of use-specificity has been obtained in calculus (Singley 1986).

3. Knowledge Tracing and Mastery Learning

The programming tutor has proven successful over the past ten years. Students work through practice exercises in as little as one-third the time, while performing as well or better on tests (Anderson, Corbett, Koedinger & Pelletier, in press; Anderson & Reiser 1985). Despite this general success, we recognized early on that some students were floundering. This led us to incorporate knowledge tracing into the tutors in an effort to implement mastery learning. Recall that in model tracing the cognitive model is used to interpret each student action and follow the student's step-by-step path through the problem space. The primary goal in this process is to provide whatever guidance is needed for the student to reach a successful conclusion to problem solving. By contrast, in knowledge tracing we attempt to monitor the student's changing knowledge state during practice. The student model is an overlay of the ideal student model in knowledge tracing. Each time the student has an opportunity to apply a rule in the model, the tutor updates its estimate of whether the student knows the rule, based on the student's action.

As described earlier, the learning assumptions of ACT-R are complex. The student acquires both goal-independent declarative knowledge and goal-oriented procedural rules. With practice, both declarative and procedural knowledge are strengthened in memory and student performance grows more reliable and rapid (Anderson 1993). Modeling these relationships on-line as students practice is computationally expensive and not warranted by the relatively sparse data the tutor provides. Instead, we have substituted a simpler set of learning and performance assumptions in knowledge tracing in our tutors.

Knowledge tracing assumes a two-state learning model. Each coding rule is either in the learned state or in the unlearned state. A rule can make the transition from the unlearned to the learned state prior to practice through reading the text, or at each opportunity to apply the rule in practice. There is no forgetting; rules do not make the transition in the other direction. Performance in applying a rule is governed by its learning state, but only probabilistically. If a rule is in the learned state, the student may nevertheless slip and make a mistake. If the rule is in the unlearned state, there is some chance the student will guess correctly. As the student practices, the tutor maintains an estimate of p(L) for each rule, the probability that the rule is in the learned state. At each opportunity to apply a rule in problem solving, the estimate of p(L) for the rule is updated, contingent on whether the student's action was correct or not. The Bayesian computational procedure is a variation on one described by Atkinson (1972). This procedure employs two

$p(L_0)$	Initial Learning	the probability a rule is in the learned state prior to the first opportunity to apply the rule (i.e., from reading the text).
p(T)	Acquisition	the probability a rule will make the transition from the unlearned to the learned state following an opportunity to apply the rule
p(G)	Guess	the probability a student will guess correctly if a rule is in the unlearned state
p(S)	Slip	the probability a student will slip (make a mistake) if a rule is in the learned state

Fig. 4. The learning and performance parameters in knowledge tracing.

learning parameters and two performance parameters as displayed in Figure 4. These parameters are estimated empirically for each rule.

The following equation is used in knowledge tracing to update the estimate of the student's knowledge state:

$$p(L_n) = p(L_{n-1}|\text{evidence}) + (1 - p(L_{n-1}|\text{evidence})) * p(T) \qquad (1)$$

The probability that a rule is in the learned state following the nth opportunity to apply the rule, p(L_n), is the sum of two probabilities: (1) the posterior probability that the ideal rule was already in the learned state contingent on the evidence (whether or not the nth action is correct) and (2) the probability that the rule will make the transition to the learned state if it is not already there. We use a Bayesian inference scheme to estimate the posterior probability p(L_{n-1}|evidence). Following Atkinson (1972) the probability p(T) of a transition from the unlearned to the learned state during procedural practice is independent of whether the student applies the rule correctly or incorrectly. Verbal learning studies have shown that Bayesian remediation algorithms are successful, for example, in learning sets of independent word pairs (Atkinson 1972; Atkinson & Paulson 1972). Such algorithms lead to higher performance levels than random selection of practice stimuli or student selection of practice stimuli.

Knowledge tracing is employed in the tutor to implement mastery learning. In each section of the tutor curriculum the student reads an accompanying text that introduces a set of coding rules. The tutor follows with a set of exercises that provide practice on the rules. The sequence of exercises is tailored to the student's needs and the student continues practicing exercises in a section until reaching a criterion knowledge probability for each rule in the set. That mastery criterion in the tutor is a knowledge probability of 0.95. This procedure is similar to individualized student-paced mastery learning systems based on Keller's Personalized System of Instruction, which have been shown to be effective in raising mean achievement scores (Block & Burns 1977; Kulik, Kulik & Bangert-Drowns 1990; Kulik, Kulik

& Cohen 1979). In these systems, however, practice and assessment are usually treated as distinct phases. Students work through study/test cycles until obtaining a criterion test performance level. In knowledge tracing, by contrast, assessment is continuously integrated with practice; students simply continue practicing until reaching an hypothetical knowledge state. Tests can be used to validate mastery decisions but do not enter into the mastery decisions.

Note that student modeling is a relatively simple process when formulated this way. First, the learning and memory assumptions of knowledge tracing are the simplest possible. Second, we attempt to avoid the complexities in modeling students' misconceptions (Duncan, Brna & Morss 1994; Self 1988; VanLehn 1990). Third, mastery learning simplifies a potentially complex rule-attribution problem. Since students master prerequisite skills as they move through the curriculum, we need only credit a single production in tracing each student action. In the research described in the following section we are interested in whether knowledge tracing can bring students to mastery of the components of a knowledge-rich skill and in how accurately the model predicts students' performance.

4. Empirical Evaluations of Knowledge Tracing

The goal in knowledge tracing is to estimate the student's knowledge and ensure with a high probability that each rule is in the learned state. This process is only meaningful, however, if the model accurately predicts students' programming behavior. In this section we describe a series of four studies that assess the psychological validity of the student modeling process. The modeling assumptions allow us to predict the probability of a correct action at each problem solving step and the first study examines how well the model predicts students' performance in completing tutor exercises. This study led us to substantially revise the initial cognitive rule set and to adjust the learning and performance parameter estimates. The remaining three studies examined the external validity of the model in predicting test performance. Knowledge tracing is ultimately useful only if it predicts students' performance when they are working on their own. In these studies we focus on the students' ability to write the programs correctly without tutorial assistance. The first of these three studies revealed the need to incorporate individual differences in learning and performance parameters in the model. The final two studies assessed a procedure for doing this.

4.1. General Experimental Procedure

Each of the four studies focuses on the first chapter in the Lisp Tutor curriculum. This chapter introduces two data types, atoms (symbols) and lists (hierarchical groupings of symbols), function calls and function definitions. The chapter introduces three unary functions, *car*, *cdr*, *reverse*, that extract information from or transform a list, three constructor functions, *append*, *cons* and *list*, that build new lists, and the operator *defun*, that is used to define new functions. The curriculum structure has evolved across studies, but in each study, the chapter is divided into sections. In each section the student reads text and then completes a fixed set of required exercises. The tutor monitors the student's growing knowledge of the applicable programming rules as the student completes these required exercises. Then the tutor presents remedial exercises as needed for the student to master the rules introduced in the section. Of course, the number and sequence of remedial exercises varies across students. Students also complete one or more quizzes during a study. Each quiz requires the students to complete additional programming exercises similar to the tutor exercises, but with no tutorial assistance. The primary goal of the first study is to examine how well the model predicts goal-by-goal performance in the tutor environment and to adjust the model accordingly. The ultimate goal is addressed in the remaining three studies: predicting students' performance when working on their own outside the tutor environment.

4.2. Internal Validity: Predicting Tutor Performance

Students receive immediate feedback on each programming action while working with the tutor. Since the tutor does not accept an incorrect programming action, the student is always on a recognizable solution path. Under these circumstances, the model can predict the probability of a correct action at each successive goal (step) in writing programs. For example, the model can predict the probability that a student will perform a correct action at each of the six goals in coding the function *last-item* (see Figure 2). The model's prediction at each step depends on the student's knowledge of the applicable programming rule at the goal and the performance parameters for that rule as shown in the following formula:

$$p(C_{is}) = p(L_{rs}) * (1 - p(S_r)) + (1 - p(L_{rs)}) * p(G_r) \tag{2}$$

That is, p(C_{is}), the probability that a student *s* will perform a correct action at goal *i* is the sum of two products: (1) the probability that an appropriate rule *r* is in the learned state for student *s* times the probability of a correct response if the rule is in the learned state (p(S_r) is the slip parameter in Figure 4), and (2) the probability that an appropriate rule *r* is not in the learned state for student *s* times the probability of a correct guess if the rule is not in the learned state (p(G_r) is the guess parameter in Figure 4).

To assess the validity of the model in fitting the tutor data, we compute two measures at each coding goal in the fixed set of required exercises: (1) the actual probability of a correct response averaged across students and (2) the model's predicted probability of a correct response averaged across subjects. In computing these measures we trace through the tutor protocol files that contain a complete record of each students' responses. At each coding goal in the fixed set of required exercises we first apply Equation 2 to predict the probability the student will correctly apply an appropriate rule, then record whether or not the student did respond correctly on the first attempt and finally apply Equation 1 to update the probability estimate that the student knows the appropriate rule based on the response evidence. Recall that each student has different remedial exercises interspersed among the sequence of required exercises. These exercises are excluded from the measures of actual and expected accuracy, since only a subset of students would contribute to each point. However, the remedial exercises are employed to update the individual student's knowledge state.

We employ three statistics to assess the predictive validity of the model. First, we correlate the actual accuracy and predicted accuracy estimates across goals. If the model accurately reflects both the initial difficulty of each coding rule and the learning rate for each rule across successive opportunities to apply it, these two measures should be highly correlated. Second, we compute the mean error in prediction (expected accuracy minus actual accuracy) across goals to check whether the model is systematically overestimating or underestimating accuracy levels. Finally, we compute the mean absolute error in prediction (average absolute value of predicted accuracy minus actual accuracy) across goals to determine how well the model predicts the absolute differences in accuracy levels among goals.

4.2.1. *Experiment 1: Internal Validity*

The goal of the initial evaluation study (Corbett & Anderson 1993) was to examine the adequacy of the rule set and parameter estimates. In this study the cognitive model consisted of twenty-one ideal coding rules for the dozen Lisp constructs being introduced. The two learning and two performance parameter estimates were held constant across the twenty-one rules and the four values were estimated from an earlier group of students. Students completed 158 programming goals across twenty-five required exercises in this study. They completed an average of 14 remedial exercises in this study, with a range from 1 to 38. As described above, we computed the average actual and expected probabilities of a correct action at each of the 158 required programming goals and compared the two measures across goals. The expected and actual probabilities were moderately correlated ($r = 0.47$, $p < 0.05$), the mean error rate was 0.06 and the mean absolute error rate was 0.16. The absolute error rate is relatively high and visual inspection of the data revealed that the predicted and actual values clearly deviated in systematic ways from the observed values. This led us to revise the knowledge tracing model in two ways.

First, it was apparent that the data is not well fit when learning and performance parameters are held constant. Not surprisingly, some rules are harder to learn than others. Perhaps less obviously, the probability of slips and guesses also varies across rules. If we allow the learning and performance parameter estimates to vary across rules, best fitting estimates provide a substantially better fit, $r = 0.85$. The mean error in prediction, -0.01, and the absolute error in prediction, 0.07, are both reduced in the revised fit.

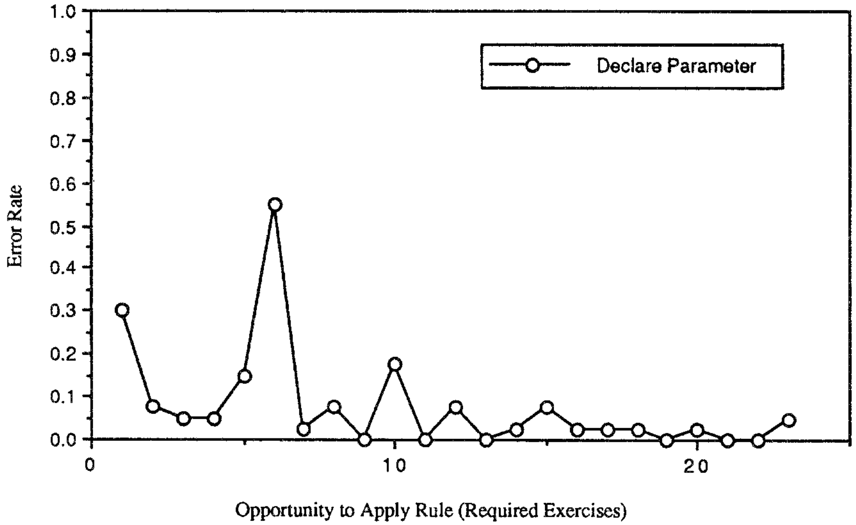


Fig. 5. The learning curve for a hypothetical rule that declares a variable in a function definition. Error rate is plotted for successive opportunities to apply the rule.

Second, it became obvious that some of the rules in the ideal model were too general, at least for some students, and were not well fit by the model. Figure 5 depicts the learning curve for the following rule in the ideal student model:

> In defining a Lisp function, declare one parameter variable in the parameter list for each argument (input value) that will be passed when the function is called.

This rule is expected to fire each time a variable is declared in a function definition. As can be seen, error rate decreases monotonically over the first four applications, but then jumps for the fifth and sixth applications. The first four points are drawn from the first four exercises in which functions are defined. Just one variable is declared in each of these four function definitions. The fifth and sixth points represent the fifth function definition, but the first in which two parameter variables are declared. The seventh and eighth points represent two more function definitions with a single parameter and the ninth and tenth points represent the next exercise that requires multiple parameters. The 55% error rate at goal six in this figure suggests that just under half the students had encoded the ideal rule initially and were able to apply it in the new situation. The other students initially encoded a non-ideal rule that was sufficient to complete the first four exercises but failed to generalize. There are several possible non-ideal rules:

> Always code one variable to stand for the whole list of arguments (too general).
>
> Code one variable when there is one argument (too specific – no thought of

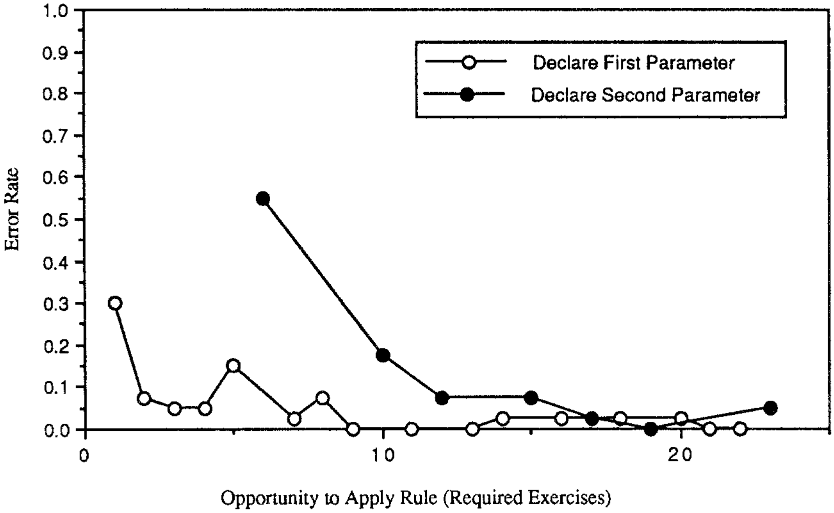


Fig. 6. A partitioning of the single learning curve for variable declarations in Figure 5 into separate learning curves for two distinct variable declaration rules.

more arguments).

Code one variable because that's what the example has (no functional understanding).

Similar patterns were observed for a variety of ideal rules in the model. For example, some students who can apply the constructor functions *append*, *cons* and *list* in building simple lists have difficulty generalizing their knowledge to embedded lists. Similar difficulties have been observed in other domains, such as algebra (Wenger 1987) and obviously represent a serious threat to knowledge tracing. Students may get credit for having mastered an "ideal" rule in simple contexts that, in fact, does not transfer to more complex contexts.

We revised the cognitive model in response to this problem. Ultimately the cognitive model for this curriculum has grown to 55 rules. In the case of variable declarations, we represent the procedural knowledge as two rules:

In defining a Lisp function, declare one variable for the first argument that will be passed.

In defining a Lisp function, declare one additional variable for each additional argument that will be passed.

Figure 6 re-plots the data points for declaring variables. The points from the original learning curve in Figure 5 are plotted as two separate learning curves for these rules. Under this analysis the sixth, tenth, twelfth, fifteenth, seventeenth, nineteenth and twenty-third opportunities to declare a variable are really the first through seventh opportunities to apply the second rule.

4.3. External Validity: Predicting Test Performance

The more important issue is whether the model predicts students' performance in working on their own after having achieved "mastery". To assess this predictive validity we have students complete cumulative tests at several points in the curriculum. In these tests students complete more of the same types of programming exercises and in the same interface as the tutor, but without the tutor's assistance. These tests are not used to decide when to promote students but exclusively to validate the model's performance predictions. Our external validity assessments differ from the internal validity assessments in two ways. First, the unit of measurement is the complete programming exercise rather than the individual programming goal. Predicting students' test performance at each programming goal is more complex than predicting their goal-by-goal tutor performance, since students are not constrained to remain on a successful solution path in testing. Assuming production rules are independent as ACT-R does, though, we can derive a straightforward accuracy prediction for each test exercise as a whole. The probability of completing an exercise with no mistakes is given by

$$\Pi p(C_{is}) \tag{3}$$

the product of the probabilities that the student will respond correctly at each successive goal in solving the exercise.

Second, we are interested in how well the model predicts individual differences across students rather than across programming tasks. To assess the model's external validity we compute two measures for each student across all the exercises in a test: (1) the actual probability of completing an exercise correctly and (2) the predicted probability of completing an exercise correctly. We again compute three statistics to compare these two measures: (1) the correlation of actual and expected accuracy across students, (2) the mean error in prediction across students and (3) the mean absolute error in prediction across students. To the extent that the model accurately captures individual differences, actual and predicted accuracy should be highly correlated across students and the mean prediction error and mean absolute prediction error (absolute value of predicted vs. actual accuracy) across students should be low.

4.3.1. *Experiment 2: External Validity*

The goal of this study was to examine the predictive validity of the model for test performance. Twenty college students participated in the first study of external validity (Corbett, Anderson & O'Brien, in press). They worked through six curriculum sections in this study and completed cumulative tests following the first, fourth and sixth sections. The curriculum was expanded to 64 required exercises (345 total programming goals) to provide coverage of the revised rule set. Students worked to mastery in the first five curriculum sections, but not in the sixth and final section. That is, in the first five sections students completed an individualized

TABLE I. Actual and expected proportion of exercises completed correctly across subjects in each of the three tests in Experiment 2

	Mean proportion correct				
	Actual	Expected	MAE[a]	r_{AE}[b]	r_{AR}[c]
Test 1	0.89	0.90	0.09	–	−0.25
Test 2	0.89	0.84	0.11	–	−0.64
Test 3	0.55	0.64	0.18	0.69	−0.68

[a] MAE = mean absolute error in accuracy prediction across students

[b] r_{AE} = correlation of actual and expected accuracy across students

[c] r_{AR}= correlation of error rate on required tutor exercises and test accuracy across students

sequence of remedial exercises governed by the knowledge tracing process. In the final section, students completed a fixed set of exercises with no remediation. This allows us to examine the model's validity after students had reached mastery in the first two tests, and to examine the model's validity at intermediate learning levels in the third test. On average students completed 18 remedial exercises in the first five sections, with a range across students of 3 to 63. The test exercises were completed with the same interface as the tutor exercises, except that no tutorial support was provided and students could freely edit their code.

The first step was to verify the internal validity of the revised cognitive model and parameter estimates. The correlation of expected and actual mean error rates across the 345 goals in the required exercises was 0.75, $p < 0.0001$. The mean error in prediction across the 345 goals was 0.01 and the absolute error in prediction was 0.07. In an effort to maximize internal validity of the model, we generated best fitting parameter estimates for this group of students and refit the tutor data. With these best fitting estimates, actual and expected accuracy values correlated 0.90, the mean error was 0.003 and the mean absolute error rate was 0.04. We employed these best fitting parameter estimates to establish an upper bound on the validity of the model in predicting test performance in this study.

Recall that the probability of completing a test exercise correctly is the product of the probabilities of satisfying each goal successfully. To assess test predictive validity, we computed (1) the proportion of exercises each student completed correctly and (2) the average predicted probability of completing the exercises correctly. The results are displayed in Table I.

The first column displays the mean actual accuracy level across students and the second column displays the mean predicted accuracy of the model. As can be seen, the model is quite accurate in predicting overall test accuracy for the twenty students. The third column displays the mean absolute error in these predictions across students. The fourth column, r_{AE}, displays the correlation across students of the actual probability and expected probability of writing correct programs. This correlation could not be computed for the first two tests in this study because the model was completely insensitive to individual differences among students in the first two posttests. These two tests followed the first and fourth curriculum sections and, according to the model, students had mastered all the rules. There was no variability in the knowledge estimates across students in these sections, and therefore no variability in the model's predictions of test performance. The third test, however, included a large number of exercises from the final tutor section in which students did not work to mastery. As a result, students' absolute performance level is substantially lower on the third test, and there is substantially more variability in the model's estimate of the students' knowledge state. Under these conditions, the knowledge tracing model is quite sensitive to individual differences in test performance. The correlation of actual and expected valued, $r = 0.69$ is reliable. (The sample size is 19, since one student's test data were lost. A correlation of 0.46 or higher is reliable at the 0.05 level in this study).

While the model performed well at predicting student performance at intermediate learning levels of the third test, its predictions for the first two tests are suspect. The model suggests that all students are in the same learning state for these two tests and any variation in the actual test scores reflects random noise. This seems implausible because there were discernible differences among students upon entering the study, as reflected in the wide range in number of remedial exercises completed by different students, from as few as 3 to as many as 63. While remediation should attenuate these differences it is highly unlikely that it would completely eliminate them. To examine this issue further, we computed a raw measure of student entry characteristics: the tutor number of errors each student made in completing the fixed set of required exercises. We correlated this measure of raw error rate in the tutor with actual test accuracy for each of the three tests. These correlations are presented in column 5 of Table I (r_{AR}).

In each case, there is a negative correlation between raw error rate in the tutor and test accuracy. In general, the more difficulty students had in completing the fixed set of tutor exercises, the worse they performed on the test, despite the tutor's efforts at remediation. The correlation for the first test, $r_{AR} = -0.25$, is not reliable, though, so we cannot reject the model's contention that there are no genuine differences among students on this test. The reliable correlation of tutor errors and test accuracy for the third test, $r_{AR} = -0.68$, also poses no problem for the model. Students did not master the full curriculum for test three, so test accuracy would be expected reflect tutor difficulty and the model is quite sensitive to residual individual differences, $r_{AE} = 0.69$. The correlation of tutor errors and test accuracy for the second test, $r_{AR} = -0.64$ is problematic, however. While the model indicates there are no differences among students following remediation, tutor error rate is a strong and reliable predictor of test performance. Students who struggled in completing the tutor exercises continued to struggle on the second test.

As described in the next section, this pattern of results can arise when individual differences among students in learning and performance are not represented in the model.

4.4. Individual Differences in Learning and Performance

The learning and performance parameter estimates in the previous study varied across rules and thereby reflected average differences in rule difficulty. They did not reflect individual differences among students in learning and performance. As a result, the model mis-estimates the learning state of students to the extent they fall above or below average in learning and performance. The model underestimates the true learning and performance parameters for above-average students. As a result, these students who make few errors receive more remedial exercises than necessary and perform better on the test than expected. In contrast, the model overestimates the true learning and performance parameters for below-average students who make many errors. While these students receive more remedial exercises than the above average students, they nevertheless receive less remedial practice than they need and perform worse on the test than expected. This results in the observed negative correlation of tutor errors and posttest accuracy.

To explore this issue, we incorporated differences among students into the model in the form of individual difference weights. Four weights are estimated for each student, one weight for each of the four parameter types ($p(L_0)$, $p(T)$, $p(G)$, $p(S)$). While the parameter estimates vary across rules, a single adjustment weight for each parameter should be sufficient for each student, since an earlier study of individual differences in Lisp learning failed to find any interaction of individual student differences and differences in rule difficulty (Anderson, Conrad & Corbett 1989). In the revised model, each of the four probability estimates for each rule is converted to odds form ($p/(1-p)$), multiplied by the corresponding subject-specific weight and the resulting odds form is converted back to a probability. If we let i represent the parameter type, ($p(L_0)$, $p(T)$, $p(G)$, $p(S)$), r represent the rule and s the subject, then the individually weighted parameter for each rule and subject is given by the following equation:

$$p_{irs} = p_{ir} * w_{is}/(p_{ir} * w_{is} + (1 - p_{ir})) \qquad (4)$$

where P_{ir} is a best fitting parameter estimate for the rule across all students and w_{is} is the corresponding individual difference weight for the student.

We initially examined this model with the tutor and test data from Experiment 2. We assumed that each of the four individual difference weights is constant over time for a student and derived a best-fitting set of weights for each student given the group parameter estimates and student's tutor performance data. We used these weights to individualize the four parameter estimates for each rule for each student then we ran the knowledge tracing procedure over the students' tutor data to derive revised test predictions. The resulting test predictions are displayed in Table II.

TABLE II. Individual differences: Actual and expected proportion of exercises completed correctly across subjects in each of the three tests in Experiment 2, based on best fitting individualized parameter estimates

	Mean proportion correct				
	Actual	Expected	MAE[a]	r_{AE}[b]	r_{AR}[c]
Test 1	0.89	0.91	0.10	−0.02	−0.25
Test 2	0.89	0.86	0.10	0.51	−0.64
Test 3	0.55	0.68	0.16	0.72	−0.68

[a] MAE = mean absolute error in accuracy prediction across students

[b] r_{AE} = correlation of actual and expected accuracy across students

[c] r_{AR}= correlation of error rate on required tutor exercises and test accuracy across students

Of course, actual test performance in the first column has not changed, and the mean accuracy predictions are quite similar to the prior analysis. However, this individualized model is considerably more sensitive to individual differences in the second test. The correlation of actual and expected accuracy for that test is now reliable, r = 0.51.

4.4.1. *Experiment 3: Dynamic Estimation of Individual Differences*

The preliminary analysis of individual differences was promising, but the procedure employed does not lend itself to dynamic decision making as the student works through the tutor, since best fitting weights were derived *after* the students had completed work. However, the logarithms of the best-fitting weights for each student in this analysis correlated reliably with the student's raw error rate in completing the required tutor exercises and this relationship can be used to estimate individual differences dynamically as students work. As the student completes a minimal set of required exercises at the beginning of each curriculum section, the student's sequence of actions can be stored by the tutor. The student's error rate on these required exercises is then used to estimate a set of student-specific learning and performance weights by means of regression equations and weighted parameter estimates are derived. The students' stored actions for the required exercises are then knowledge-traced to estimate the student's current knowledge state. Remedial exercises are subsequently selected and knowledge tracing continues until the student has mastered each of the rules introduced in the curriculum section.

We have examined this estimation procedure in two studies. The first is reported in Corbett, Anderson, Carver & Brancolini (1994). In this study, twenty students worked through the same curriculum as in the preceding study and completed an average of 11 remedial exercises, with a range of 0 to 98. Thus, while the mean

TABLE III. Dynamic estimation of individual differences: Actual and expected proportion of exercises completed correctly across subjects in each of the three tests in Experiment 3, based on dynamically estimated individualized parameter estimates

	Mean Proportion Correct				
	Actual	Expected	MAE[a]	r_{AE}[b]	r_{AR}[c]
Test 1	0.92	0.91	0.12	−0.02	0.05
Test 2	0.76	0.87	0.15	0.31	−0.63
Test 3	0.66	0.75	0.15	0.81	−0.78

[a] MAE = mean absolute error in accuracy prediction across students

[b] r_{AE} = correlation of actual and expected accuracy across students

[c] r_{AR}= correlation of error rate on required tutor exercises and test accuracy across students

number of remedial exercises is similar to the prior study, the range is wider, as would be expected when parameter estimates are individualized. This model provided a good fit to the tutor data. The correlation of expected and actual mean error rates across the 345 goals in the required exercises was 0.79, $p < 0.0001$. The mean error in prediction across the 345 goals was 0.001 and the absolute error in prediction was 0.06. Table III displays the test results of this study. As can be seen, the model accurately predicts overall performance level on the first test, while overestimating performance on the final two tests by about 10%. The final two columns report the correlation of actual quiz performance with predicted performance and raw tutor error rate respectively. The results for the first and third tests are comparable to those obtained with the earlier non-individualized knowledge tracing process. There is no evidence of individual differences in the relatively easy first test, while the model is very sensitive to remaining individual differences at the sub-mastery levels of performance in the third test, $r = 0.81$ (a correlation of 0.45 or greater is reliable at the 0.05 level). The second test provides the crucial assessment of the revised individualized knowledge tracing model. In the present study, there is again a strong correlation of Posttest 2 performance with raw error rate, $r = -0.63$, so genuine individual differences persist, although students have nominally worked to mastery. Unlike the standard model, however, the individualized knowledge tracing model is moderately sensitive to these residual differences, $r = 0.31$, $p < 0.10$.

The model's sensitivity to individual differences on this second test depends primarily on estimated differences among students in performance, rather than knowledge state. Since all students have worked to mastery in this test, the probability that a given student has learned each of the rules ranges from 0.95 to 1.00. The small differences in these probabilities have little impact on accuracy predictions. Instead, accuracy predictions are strongly related to differences in the

TABLE IV. Current Assessment of Knowledge Tracing: Actual and expected proportion of exercises completed correctly across subjects in each of the three tests in Experiment 4, based on dynamically estimated individualized parameter estimates

	Mean Proportion Correct				
	Actual	Expected	MAE[a]	r_{AE}[b]	r_{AR}[c]
Test 1	0.88	0.94	0.11	0.24	−0.28
Test 2	0.81	0.89	0.11	0.36	−0.52
Test 3	0.81	0.86	0.10	0.66	−0.73

[a] MAE = mean absolute error in accuracy prediction across students

[b] r_{AE} = correlation of actual and expected accuracy across students

[c] r_{AR}= correlation of error rate on required tutor exercises and test accuracy across students

slip parameter estimates across students, that is, to persistent differences among students in the probability of making a mistake when an appropriate rule is in the learned state. The nature of the slip parameter is discussed more fully below.

4.4.2. *Experiment 4: Current Assessment of Knowledge Tracing*

We have recently completed a second evaluation of this individualized knowledge tracing process. The curriculum in this study was extended to include a section of more complex constructor/extractor algorithms, while the total number of required exercises was reduced from 64 to 38. A new test was also constructed to follow the new curriculum section. Twenty-five students worked through this curriculum to mastery. Individualized parameters were again estimated along the way (based on about half as much data as in the previous study) and students completed an average of 29 remedial exercises. The model's predictions again fit the tutor data well. The correlation of expected and actual mean error rates across the 214 goals in the required exercises was 0.71, $p < 0.0001$. The mean error in prediction across the 345 goals was 0.001 and the absolute error in prediction was 0.06.

Test performance is displayed in Table IV. Note that students worked to mastery on the full curriculum for all three tests in this table, and not just the first two tests as in the prior studies. Overall the pattern of results is similar to the previous study. Average actual and predicted accuracy are again displayed in the first two columns of the table and actual accuracy is 5%–10% lower than predicted. The correlation of actual test accuracy with predicted accuracy and with raw tutor error rate is displayed in the final two columns. No individual differences are apparent in the first test, while the correlation of actual and expected accuracy across students is marginally reliable for the second test ($r = 0.36, p < 0.10$) and reliable for the third test ($r = 0.66, p < 0.01$).

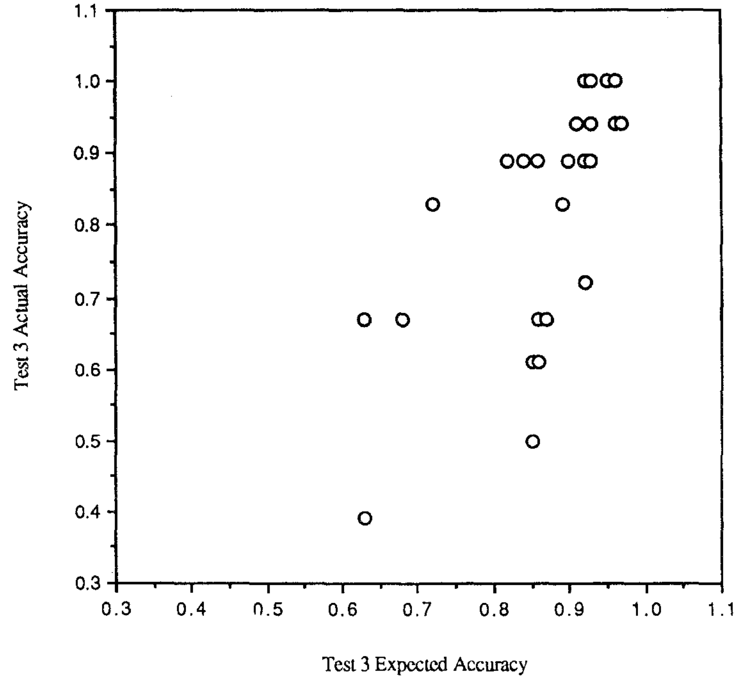


Fig. 7. Actual accuracy on the third test of Experiment 4 plotted as a function of expected accuracy for twenty-five students.

Figure 7 displays the correlation of expected and actual test accuracy for the twenty-five students on the third test. In addition to the quality of the fit, we can ask how successful the model is in enabling students to master the curriculum based on their actual test performance. If we adopt a strict operational definition of mastery as 90% correct on the test, Figure 7 indicates that 56% of the students in this knowledge tracing condition reached the mastery performance level. A second group of twenty-one students in this study who completed just the 38 required exercises provides a contrast. Only 24% percent of these students reached this high level of test performance. This difference in the probability of success between groups is reliable, $z = 2.21$, $p < 0.05$. Students in the knowledge tracing condition average 29 remedial exercises, so they are completing about 76% more tutor exercises than the comparison group. Since the tutor speeds practice by as much as a factor of three, though, total time to complete the exercises in the knowledge tracing tutor condition would compare very favorably with total time for students working through just the fixed set of exercises on their own.

5. Future Directions

One interesting observation concerning Figure 7 is that while the model estimates that all students have essentially learned the underlying programming rules, it predicts that some students will make almost 40% errors on the quiz problems. As described earlier, the model's mastery judgments are based on inferences about the student's knowledge state, but test performance predictions depend both on the student's knowledge of the rules and on the performance parameters. When students have worked to mastery in the tutor, the variability in the model's predictions depend primarily on one of these performance parameters, the slip parameter.

The importance of the slip parameter in test predictions can be seen if we try predicting performance just from the student's knowledge state without considering performance, i.e., with the equation $p(C_{is}) = p(L_{rs})$. According to the model, the probability that a given student has learned each rule ranges from 0.95 to 1.00. If test predictions are based solely on these knowledge probabilities without considering the performance parameters the model greatly overestimates average accuracy. The predicted accuracy for the third test under this model is 0.99 compared to an average actual accuracy of 0.86. Note that incorporating the guess performance parameter (the probability of a correct response if an appropriate rule is unknown) in the predictions tends to raise the predicted accuracy level. In reality, however, the guess parameter has almost no impact on the test predictions, since the estimated probability of having learned each rule is so close to 1.0. Instead, the variability in the model's predictions of test performance primarily reflect persisting differences among students in the slip parameter. This can be seen in the correlation of ln(slip weight) and test accuracy across students, $r = -0.72$.

There are several interpretations of slips. One is that individual differences in the slip weight reflect a pure performance effect, i.e., that there are persistent differences in the care with which students perform the task. This interpretation suggests that there may be a limit on the extent to which 'operational' mastery can be achieved, that is, a limit on the proportion of students who will reach a strict test performance criterion. Certainly it suggests that additional gains will be achieved by manipulating the student's incentive to perform carefully on the test rather than through additional learning. Given the simplicity of the learning and performance assumptions in knowledge tracing, though, it may be that individual differences in the slip parameter actually reflect learning-state differences. One plausible interpretation is that the slip parameter reflects differences in the strength of underlying production rules. The ACT-R model assumes that each rule has a strength property that varies as a function of practice and that influences both the speed and reliability with which the rule fires. Strength is not directly represented in the learning and performance assumptions of the simple knowledge tracing model, but would be reflected in the slip parameter. Consider the parameter-declaration rule described earlier. Students who form an ideal rule early will have an opportunity to apply the rule in every function definition exercise. Students who initially form a suboptimal rule ultimately must form at least one more rule to replace or supplement the initial rule and the resulting rule(s) will receive less practice. This interpretation suggests that providing more practice exercises even after the student has reached the criterion learning state can improve test performance. Such a framework can be implemented by allowing each student's slip weight to vary as a function of practice and continuing practice until it reaches a sufficiently low value.

Alternatively, the slip parameter may reflect the content of the rules the student has formulated. That is, students who form a rule or rule set that is highly correlated with the ideal rule will appear to know the ideal rule, but appear to slip from time to time. Under this interpretation, there is no guarantee that additional practice exercises will have a pronounced impact on test performance, once the knowledge probability estimate has reached the 0.95 mastery criterion. This suggests that it is necessary to monitor the content of students' rules more closely. Under this interpretation it may prove more cost effective to directly assess and remediate students' knowledge of key declarative concepts prior to practice, rather than trying to tease apart alternative rule formulations during practice.

6. Conclusion

We started this research effort with an executable cognitive model of Lisp programming skill. This model consisted of a set of ideal production rules and formed the core of a successful intelligent programming tutor. The goal of the research was to implement a simple student modeling process that would allow the tutor to monitor the student's knowledge state and tailor the sequence of practice exercises to the student's needs. The resulting knowledge tracing process models the student as an overlay of the production rules and the mastery-based curriculum structure allows us to associate each programming action with a single production rule. A simple two-state learning model enables us to estimate the student's knowledge state from performance and predict performance from that knowledge state. Successive evaluations led us to (1) abandon an initial ideal student model and to model a sufficient set of rules, (2) to model differences in rule difficulty and (3) to model individual differences among students in learning and performance. The resulting model predicts student performance quite well and enables most students to reach a high level of task performance. It may be possible to improve on this level of performance and enable more students to reach mastery by manipulating incentive in testing, by providing additional procedural practice or by monitoring and remediating students' knowledge of key declarative concepts.

Acknowledgements

This research was supported by the Office of Naval Research grant N00014–91–J-1597. We would like to thank Valerie Carver, Scott Brancolini, Alison O'Brien, Chris Taylor and Holly Trask for assistance in data collection and analysis.

References

Anderson, J. R.: 1993, *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R., C. F. Boyle, A. T. Corbett, and M. W. Lewis: 1990, Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, **42**, 7–49.

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

Anderson, J. R., F. G. Conrad, A. T. Corbett, J. M. Fincham, D. Hoffman, and Q. Wu: 1993, Computer programming and transfer. In J. Anderson (ed). *Rules of the mind.* Hillsdale, NJ: Lawrence Erlbaum.

Anderson, J. R., A. T. Corbett, J. M. Fincham, D. Hoffman, and R. Pelletier: 1992, General principles for an intelligent tutoring architecture. In J.Regian & V. Shute (eds.) *Cognitive approaches to automated instruction.* Hillsdale, NJ: Erlbaum.

Anderson, J. R., A. T. Corbett, K. R. Koedinger, and R. Pelletier: in press, Cognitive tutors: Lessons learned. *Journal of the Learning Sciences.*

Anderson, J. R. and B. J. Reiser: 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.

Atkinson, R. C. and J. A. Paulson: 1972, An approach to the psychology of instruction. *Psychological Bulletin*, **78**, 49–61.

Block, J. H. and R. B. Burns: 1976, Mastery learning. In L. S. Shulman (ed.) *Review of research in education, Volume 4.* Itasca, IL: F. E. Peacock (AERA).

Bloom, B. S.: 1968, Learning for mastery. In *Evaluation Comment, 1.* Los Angeles: UCLA Center for the Study of Evaluation of Instructional Programs.

Carroll, J. B.: 1963, A model of school learning. *Teachers College Record*, **64**, 723–733.

Corbett, A. T. and J. R. Anderson: 1993, Student modeling in an intelligent programming tutor. In E. Lemut, B. du Boulay & G. Dettori (eds.) *Cognitive models and intelligent environments for learning programming*. New York: Springer-Verlag.

Corbett, A. T., J. R. Anderson, V. H. Carver, and S. A. Brancolini: 1994, Individual differences and predictive validity in student modeling. In A. Ram & K. Eiselt (eds.) *The Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Lawrence Erlbaum.

Corbett, A.T., J. R. Anderson and A. T. O'Brien: in press, Student modeling in the ACT Programming Tutor. In P. Nichols, S. Chipman and B. Brennan (eds.) *Cognitively Diagnostic Assessment.* Hillsdale, NJ: Erlbaum.

Duncan, D., P. Brna, and L. Morss: 1994, A Bayesian approach to diagnosing problems with prolog control flow. In B. Goodman, A. Kobsa & D. Litman (eds.) *User Modeling: Proceedings of the Fourth International Conference*. Bedford, MA: The MITRE Corporation.

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.

Guskey, T. R. and T. D. Pigott: 1988, Research on group-based mastery learning programs: A meta-analysis. *Journal of Educational Research*, **81**, 197–216.

Hyman, J. S. and A. Cohen: 1970, Learning for mastery: Ten conclusions after 15 years and 3,000 schools. *Educational Leadership*, **37**, 104–109.

Keller, F. S.: 1968, "Good-bye teacher ...." *Journal of Applied Behavioral Analysis*, **1**, 79–89.

Kessler, K.: 1988, *Transfer of programming skills in novice LISP learners.* Unpublished doctoral dissertation, Carnegie Mellon University, Pittsburgh, PA.

Kieras, D. E. and S. Bovair: 1986, The acquisition of procedures from text: A production system analysis of transfer of training. *Journal of Memory and Language*, **25**, 507–524.

Kulik, C. C., J. A. Kulik, and R. L. Bangert-Drowns: 1990, Effectiveness of mastery learning programs: A meta-analysis. *Review of Educational Research*, **60**, 265–299.

Kulik, J. A., C. C. Kulik, and P.A. Cohen: 1979, A meta-analysis of outcomes studies of Keller's Personalized System of Instruction. *American Psychologist*, **34**, 307–318.

McKendree, J. E. and J. R. Anderson: 1987, Effect of practice on knowledge and use of basic Lisp. In J.M. Carroll, (ed.) *Interfacing thought.* Cambridge, MA: MIT Press.

Newell, A.: 1990, *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Pennington, N. and R. Nicolich: 1991, Transfer of training between programming subtasks: Is knowledge really use specific? In J. Koenemann-Belliveau, T. Moher & S. Robertson (eds.) *Empirical studies of programmers: Fourth workshop.* Norwood, NJ: Ablex.

Reed, S. K., A. Dempster, and M. Ettinger:1985, Usefulness of analogous solutions for solving algebra word problems. *Journal of Experimental Psychology: Learning, Memory and Cognition,* **11**, 106–125.

Resnick, L. B.: 1977, Assuming that everyone can learn everything, will some learn less? *School Review*, **85**, 445–452.

Resnick, L. B. and D. P. Resnick: 1992, Assessing the thinking curriculum: New tools for educational reform. In B. Gifford & M. O'Connor (eds.) *Changing assessments: Alternative views of aptitude, achievement and instruction.* Boston: Kluwer.

Self, J. A.: 1988, Bypassing the intractable problem of student modeling. In C. Frasson (ed). *Intelligent Tutoring Systems: The proceedings of the of ITS-88.* Montreal: The University of Montreal.

Shepard, L. A.: 1991, Psychometrician's beliefs about learning. *Educational Researcher*, **20**, 2–16.

Singley, M. K.: 1986, *Developing models of skill acquisition in the context of intelligent tutoring systems.* Unpublished doctoral dissertation, Carnegie Mellon University, Pittsburgh, PA.

Singley, M. K. and J. R. Anderson: 1989, *The transfer of cognitive skill.* Cambridge, MA: Harvard University Press.

Slavin, R. E.: 1987, Mastery learning reconsidered. *Review of educational research*, **57**, 175–213.

VanLehn, K.: 1990, *Mind bugs: The origins of procedural misconceptions.* Cambridge, MA: The MIT Press.

Wenger, R. H.: 1987, Cognitive science and algebra learning. In A. Schoenfeld (ed.) *Cognitive science and mathematics education.* Hillsdale, NJ: Lawrence Erlbaum.

Authors' Vitae

Albert T. Corbett

Human Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Albert T. Corbett is a Research Scientist in the Human Computer Interaction Institute at Carnegie Mellon University, Pittsburgh, PA. Dr. Corbett received his B.A. in Psychology from Brown University and his Ph.D in Psychology from the University of Oregon. His primary research interests are in human memory, comprehension and problem solving. Over the past decade he has brought these interests to the development and evaluation of intelligent computer tutors. These tutors have proven to be rich environments for pursuing basic research in human cognition that has applied significance.

John R. Anderson

Departments of Psychology and Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

John R. Anderson is Walter VanDyke Bingham Professor of Cognitive Science at Carnegie Mellon University, Pittsburgh, PA. Dr. Anderson received his B.A in Psychology from the University of British Columbia and his Ph.D in Psychology from Stanford University. The unifying goal of his research is the development of a systematic theory of of human cognition. His interest in intelligent tutoring reflects the joint contribution of artificial intelligence and empirical analysis to this research program. Dr. Anderson has authored nine books and more than 150 articles in cognitive science.