ITS92 Submission

Student Modeling and Mastery Learning in a Computer-Based Programming Tutor

Albert T. Corbett
John R. Anderson

Psychology Department
Carnegie Mellon University

Abstract

The CMU General Programming Languages 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 program to solve exercises step-by-step along with the student. This paper evaluates the tutor's student modeling procedure. This procedure employs an overlay of the tutor's programming rules. The tutor maintains an estimate of the probability that the student has learned each rule, based on the student's performance. These estimates are employed to guide remediation and implement mastery learning. The predictive validity of these probability estimates for posttest and tutor performance is assessed.

Please address correspondence to:

Albert T.Corbett
Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213
phone: 412-268-2815
fax: 412-268-2844
email: corbett @psy.cmu.edu

Key words:

Student Modelling
Mastery Learning
Knowledge Representation
ITS Evaluation

This paper describes the student modelling process in the CMU General Programming Languages Tutor (Anderson, Corbett, Fincham, Hoffman & Pelletier, in press) and our efforts to implement mastery learning in the tutor. The tutor is a practice environment for students learning to program. It presents exercises requiring students to write short programs and monitors the students' performance symbol-by-symbol as they enter code. It informs the students of errors and provides advice upon request. The tutor contains modules for Lisp, Prolog and Pascal; this report focuses on students learning Lisp.

We have been using such tutors to teach programming courses for the past eight years. The tutors have proven effective; students generally work through exercises more quickly with the tutor and perform as well or better on posttests (Anderson & Reiser, 1985; Corbett & Anderson, 1990). Despite this general effectiveness, however, some students flounder. This was particularly true the first time we taught a course to programming novices. Our impression was that the tutor and student together made an effective team in solving exercises, but that the tutor was providing assistance at key junctures and the students were sometimes unaware of the holes in their knowledge. As a result, we incorporated a student modelling and remediation mechanism into the tutor. This mechanism has also proven effective; posttest scores are higher when the remedial mechanism is in operation (Anderson, Conrad & Corbett, 1989). In this paper we describe a more detailed assessment of the internal and external validity of this mechanism. Before reviewing these results, we will describe the tutor, the underlying cognitive theory and the student modelling procedure in more detail.

The Programming Tutor

Figure 1 depicts the computer screen shortly after the student has begun an exercise. The problem description appears in the window at the upper left and remains on the screen as the student completes the exercise. The student's solution appears in the code window immediately below the problem description window. In this figure the student has just entered the operator **car**. Students can enter code by selecting operators from the menu immediately to the right of the code window, or by typing. The highlighted symbol **<EXPR0>** is not lisp code. It is a syntactic symbol which represents the argument to **car** (the value that **car** will operate on). The student will replace this goal reminder with a Lisp expression. Feedback from the tutor appears in the hint window at the bottom of the screen. If the student makes a mistake the tutor immediately notifies the student of the error and requires the student to repair the error. The student can also ask for help at each step in a solution. Three levels of help are available. The first hint reminds the student of the current goal, the second level provides an explanation of how to achieve the goal and the third level describes exactly what code to enter. Finally, the skill meter in the upper right corner of the screen displays the tutor's model of the student. As will be described below, this model consists of a set of programming rules. The skill meter displays the rules that have been introduced so far in the curriculum. The bar graph represent the tutor's estimate of the probability that the student has learned each rule. In the diagram, this probability stands at 50% for three of the rules and at about 80% for the rule the student has just exercised.

Insert Figure 1 about here

Model Tracing: The Cognitive Model

The tutor was developed an an environment in which to test the ACT* theory of skill acquisition (Anderson, 1983). The central assumption of this theory, for present purposes, is that a cognitive skill such as programming can be modeled as a set of independent production rules. Each step in performance of a skill is governed by one of these if-then rules, relating the current goal and problem state to an appropriate action. In Figure 1, for example, the student has coded the operator `car`, which takes a list of symbols as an argument and returns the first one. Thus, one of the earliest and simplest rules a student would learn is:

> If the goal is to extract the first element of a list X,
>
> Then code CAR and set a goal to code X as the argument to CAR.

The tutor is constructed around a set of several hundred such rules for writing programs, called the *ideal student model*. This model, which represents the knowledge that the student needs to acquire, allows the tutor to monitor the student's behavior in real time. At each step in an exercise solution, the tutor attempts to match the student's input to an applicable rule in the ideal model in a process we call *model tracing*. If a match is found, the assumption is made that the student has applied an analogous mental rule and the tutor's representation of the problem state is updated accordingly. If no match is found, the tutor notifies the student and requires the student to try again.

Knowledge Tracing: The Learning Model

Within this framework, the task in student modelling is to track the student's knowledge of each of the rules. The programming tutor employs an overlay of the ideal student model for this purpose (Goldstein, 1982). Each time the student has the

opportunity to apply a rule in the ideal student model, the tutor updates a probability estimate that the student has learned the rule, contingent on the accuracy of the student's response. We refer to this student modelling process as *knowledge tracing*.

The computational procedure is a variation of one described by Atkinson (1972). Our procedure assumes a simple two-state learning model with no forgetting. Each rule is either in a learned or unlearned state. A rule can make the transition from the unlearned to unlearned state when an opportunity arises to apply the rule, but rules cannot make the transition in the opposite direction.

At each goal in an exercise solution, the tutor updates its probability estimate that an applicable rule is in the learned state on the basis of the student's first response at the goal. The Bayesian expressions used to compute these probabilities are displayed in the Appendix. The equations employ four parameters, which we estimate empirically:

$p(L_0)$	=	the probability a rule is in the learned state prior to the first opportunity to apply the rule (i.e., from reading text).
$p(T)$	=	the probability a rule will make the transition from the unlearned to the learned state following an opportunity to apply the rule
$p(C\|U)$	=	the probability a student will guess correctly if the applicable rule is in the unlearned state
$p(E\|L)$	=	the probability the student will slip and make an error when the applicable rule is in the learned state

Mastery Learning

The tutor uses the knowledge tracing mechanism in an attempt to implement mastery learning. Each lesson in the tutor is divided into sections in which a handful of programming rules are introduced, organized around a theme. Students continue working on exercises in a section until the learning probability of every rule in the set has reached at least 95%. All students complete a minimal set of required exercises in each lesson, selected to cover all the rules introduced. After a student has completed the required exercises, the tutor examines the list of rules to determine if more practice is needed on any. If so, the tutor will present additional exercises that provide practice on those rules, until all rules finally reach criterion.

Design of the Study

This study assesses the internal and external validity of knowledge tracing. We are focusing on the first five of sections of the Lisp curriculum and are interested in how well our student modelling procedure predicts both posttest performance and performance internal to the tutor.

The Students

Forty-one students worked through this curriculum in the course of completing an introductory programming course. This was the first college-level programming course and first exposure to Lisp for all students. Half the students had some prior exposure to programming.

The Curriculum

Table 1 displays the constructs introduced in the first five sections of the Lisp curriculum. Students are introduced to two basic data types *atoms* (symbols) and *lists*

(symbols grouped in parentheses). They learn that *function calls* (operations) in Lisp are represented as lists in which the first symbol is the *function* (operator), and the remaining elements are *arguments* (values the function operates on). They learn about three Lisp functions, **car**, **cdr** and **reverse**, that extract information from lists and three functions that are used to construct new lists, **append**, **cons** and **list**. Finally, they are introduced to the operator **defun**, which is used to define new functions in Lisp. A minimum of twenty-five tutor exercises is required to complete this curriculum.

Insert Table 1 about here

The Model

The cognitive model for this portion of the curriculum consists of a set of 21 rules in the ideal student model. Table 2 diplays the rules that are introduced in each of the five sections and a sample exercise from each section. Sections 1, 2 and 4 primarily introduce operators, while sections 3 and 5 focus on algorithms. In the first two sections, students practice basic extractor and constructor functions. In the third section, no new operators are introduced. Rather, students practice algorithms employing multiple extractors. In the fourth section they learn to define new functions that perform such extraction algorithms and in the fifth section they define functions that employ both constructors and extractors.

Insert Table 2 about here

In knowledge tracing, the following set of parameter estimates were held constant across the 21 rules in the cognitive model:

$$p(L_0) = 0.50$$

$$p(T) = 0.40$$

$$p(C|U) = 0.20$$

$$p(E|L) = 0.20$$

These estimates were derived from our work with earlier programming tutors. The first two values are averages of parameter estimates obtained for a similar set of rules in the CMU Lisp Tutor. The latter two estimates are derived from an earlier menu-driven programming tutor.

Procedure

The students worked through the tutor exercises at their own pace. They read about Lisp in a text that is coordinated with the tutor. After completing each section of text, students completed a section of exercises with the tutor. At the conclusion of the first lesson (which also included later sections on arithmetic operations) the students completed an on-line quiz.

Results

Summary Statistics

Students completed an average of 39 exercises, with a range of 26 to 63 exercises. Since there are twenty-five required exercises across the five sections, students averaged 2.8 remedial exercises per section. The knowledge tracing mechanism ensures that the learning probability of each production exceeds 95% for each student, so there was very little variability in the final probability estimates across

students. We computed the average learning probability estimate across the 21 rules for each student and these averages ranged from 98% to 99%.

External Validity

The learning probabilities assigned to the programming rules in the course of knowledge tracing should predict programming performance. Moreover, the number of mistakes made along the way, and by extension, the number of exercises required to reach mastery should not predict programming performance. Since there is virtually no variability in the final probability estimates across students, the simplest prediction is that there should be no variability in posttest performance. This is an implausible prediction, since the quiz is in part a transfer task and since students' preparation for the quiz after having completed the tutor exercises may have varied. Whatever variability there may be in posttest scores, we would not predict a correlation with number of exercises required to reach mastery in the tutor.

In fact, the knowledge tracing procedure failed this validity test. Quiz scores correlated strongly with the number of exercises required to reach mastery, $r = -0.52$, $p < 0.05$. The more exercises students required to achieve "mastery" the worse they did on the quiz. Quiz performance, on the other hand, was not correlated with average learning probability estimate ($r = -0.09$), although this is to be expected, since there is little variability in these estimates.

In an earlier study, the knowledge tracing procedure passed a minimal external validity test. Students who received extra practice through the remediation algorithm performed better on posttests than students who only completed the required exercises. Presumably the extra practice in the present study (an average

of 2.8 exercises per section) was also helpful. However, the external validity of the learning probability estimates computed in this study turned out to be poor. We decided to pursue the issue further by examining the internal validity of knowledge tracing.

Internal Validity

While it is not the direct goal of the knowledge tracing mechanism, we can use the underlying model to predict performance internal to the tutor. Students complete 158 coding goals (solution steps) across the twenty-five required exercises in the first five sections of the curriculum. The error rate across the 41 students for each of these goals in displayed in Figure 2. The first six points in this figure, for example, represent solutions to the first three tutor exercises:

```
(car '(c d e))
(reverse '(x y z))
(cdr '(a b c))
```

Thus, the first, third and fifth points in Figure 2 represent the students' efforts at coding the functions **car**, **reverse** and **cdr** respectively. The second, fourth and sixth points reflect the students first three attempts to code literal lists as arguments. We can predict students' accuracy at each of the 158 points with the following equation:

$$p(C_i) = p(L) * (1 - p(E|L)) + p(U) * p(C|U).$$

That is, the probability of a correct response at goal i is the sum of two probabilities:

(1) the probability that the applicable programming rule is in the learned state times the probability the student will respond correctly if it is, and

(2) the probability that the applicable programming rule is in the unlearned state time the probability that the student will guess correctly in that state.

To assess the internal validity of the knowledge tracing process we traced each student's performance goal-by-goal through the curriculum. At each of the 158 goals in the required exercises, we first predicted the probability of a correct response, then applied the the knowledge tracing procedure to update the probability that the applicable rule was in the learned state. Not shown in Figure 2 are the remedial exercise goals interspersed among the required exercise goals. Since the number and position of remedial goals varies across students, we did not try to fit these goals (i.e., predict their accuracy). However, we did apply the knowledge tracing procedure to remedial goals to update learning probabilities, just as the tutor would.

Insert Figure 2 about here

Constant Parameter Estimates. Figure 2 displays the fit that was obtained with the constant set of parameters employed in the course. Overall, the predicted values fit reasonably well, $r = 0.47$, $p < .05$. However, it is apparent that the predicted values deviate systematically from the observed values. The slip parameter, p(E|L) sets a floor on the error estimates. Since this parameter is set to 0.2, the model predicts at least a 20% error rate, even if all productions are learned. Similarly, the estimate of learning from the text $p(L_0)$, effectively sets a ceiling on error rates at 50%. It is not

surprising that the external validity of the model is low when the internal validity is weak.

Variable Parameter Estimates. In an effort to improve the internal validity of the model, we performed a second fit in which we allowed the four parameter estimates to vary across the 21 programming rules. We employed a curve fitting program to generate the best fitting parameter values. As before, we fit the goals in the required exercises and traced both the required and remedial goals. Allowing parameter estimates to vary yields a substantially better fit to the data, $r = 0.87$. Final learning probability estimates also varied more widely in this fit. Average learning probabilities ranged from 0.91 to 0.99. This enhanced internal validity had a negligible impact on external validity, however. The learning probability estimates correlated 0.11 with quiz scores.

We performed one more fit to the tutor data that was restricted to the required exercises. That is, we fit and traced the goals in the required exercises as before, but did not trace the goals in the remedial exercises - as if the remedial exercises had not occurred. This procedure yielded a fit to the required exercises that is very similar to the prevous one, $r = 0.87$. (The fits are similar because the remedial exercises in each section follow the required exercises). However, the final learning probability estimates vary more widely, since they are not being driven up by remedial exercises. Average learning probability estimates ranged from 0.81 to 0.95. This fit improved external validity dramatically; average learning probability correlated 0.43 with quiz performance.

Insert Figure 3 about here

Discussion

By allowing the four parameter estimates in our statistical model to vary across productions we were able to obtain a good fit to student's performance with the tutor. However, the knowledge tracing mechanism only achieved reasonable predictive validity for quiz scores when it was restricted to required exercises. We think there are two reasons for the latter result.

The Cognitive Model

The curve fits suggest that there are inaccuracies in the cognitive model. As a rule of thumb, if we have identified psychologically valid rules, we would expect performance to improve systematically with practice (Anderson, Conrad & Corbett, 1989). A straightforward violation of this expectation is displayed in Figure 4. This figure traces the learning curve for the rule that declares a variable in a function definition. Error rates generally decline over the first four opportunities to apply the rule, but then rise for the fifth point and more dramatically for the sixth point and eleventh points. The first four points represent four exercises, each of which requires a single variable. The fifth and sixth points are drawn from the fifth exercise that requires two variables. Similarly, the tenth and eleventh points are drawn from the next exercise requiring two variables. In short, error rate jumps when students need to declare a second variable in an exercise. While the ideal model assumes that the same rule applies at all twelve goals, many students seem to be learning a more specific rule that applies in the first four exercises (e.g., always declare one variable in a function definition) and as a result have trouble in the fifth and ninth exercises.

Insert Figure 4 about here

More serious violations of the cognitive model concern the three rules involving extractor algorithms in sections 3 and 5. The learning curves for these rules are quite irregular, suggesting that they are too general. Moreoever, these points are not fit well by the learning model. If students had an ideal understanding of the extractors **car**, **cdr** and **reverse**, it might be reasonable to trace various algorithms involving multiple extractions with a small set of general rules. More realistically, it appears that we need to decompose these rules into a larger set of algorithm specific rules.

Mastery Learning

The learning probability estimates derived from just the required exercises predict posttest performance moderately well. The estimates derived from required and remedial exercises together are unrelated to posttest performance. Thus, the tutor's knowledge tracing mechanism appears to be successful at identifying difficulties students are having, but less successful at remediation. We suspect that this pattern of results reflects inaccuracies in the underlying cognitive model as described in the previous section. We anticipate that modifying the cognitive model will enhance the validity of knowledge tracing. However, this pattern may also reflect a certain insensitivity in knowledge tracing. In knowledge tracing we can track a student's ability to successfully manipulate symbols in specific contexts, but we cannot directly track the student's understanding of those manipulations. Knowledge tracing may insure that students are learning rules that enable them to

complete the exercises, but they may not be the rules assumed in the ideal student model.

Figure 5 provides evidence that students who are performing comparably may, in fact, be learning different rules. This figure displays error rates for simple extractor rules. The first six points in the figure are drawn from the first six exercises in the tutor. In each of these exercises students need to select one of the three extractor functions. (The second goal in each exercise, in which students code a literal list, is not displayed in this figure). In this figure, the 41 students have been partitioned into three groups, based on overall error rate in the tutor. As can be seen, error rate at each of these six goals is quite similar across the three groups. The last three points in the figure are drawn from three exercises in the final section of the lesson. In each case, students need to code a simple extractor as an argument to a combiner function. At this point, there are sizeable differences among the groups. Thus, while students in the three groups have demonstrated comparable "mastery" of the extractor rules in the first section, they apparently have a different understanding of the rules that leads to differential transfer to new contexts.

More generally, it may be that the number of opportunities required to master a rule is inversely related to the probability of acquiring an optimal understanding. In that case, the present pattern of results may emerge, particularly when posttests involve substantial retention intervals and/or transfer tasks. The relationship of learning probability estimates to posttest performance will be weakened while the number of opprtunities required to achieve mastery will be inversely related to posttest performance.

These observations do not lead us to dismiss the utility of "mastery learning" as described here. Mastery of symbol manipulations, even with suboptimal semantics, should enhance skill acquisition, since it is at least a co-requisite of optimal learning. These observations do pose two challenges: to create an environment that fosters appropriate understanding with practice and to enhance the tutor's student modeling capability. We expect that three types of modifications can enhance that capability: (1) revising the cognitive model as described in the previous section, (2) revising the statistical model, so that p(T), the probability of transition to an optimal learning state, changes over practice and (3) monitoring retention and transfer directly.

References

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

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*.

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. *Proceedings of Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA.

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.

Appendix

Each time the student has the opportunity to apply a rule, we need to estimate the probability that the rule is in the learned state, contingent on the accuracy of the student's response.

The probability $p(L_n|C_n)$, that a production is in the learned state following a correct response at the nth opportunity to apply the rule is expressed in equation (1). This probability is the sum of two probabilities: (a) the probability that the rule was already in the learned state, given a correct response and (b) the probability the rule was acquired, if it was not already in the learned state. The first of these probabilities is expanded in equation (3).

Similarly, the probability $p(L_n|E_n)$ that the production is in the learned state following an error is expressed in equation (2). This probability is the sum of two probabilities: (a) the probability that the rule was already in the learned state, given an error and (b) the probability the rule was acquired, if it was not already in the learned state. The first of these probabilities is expanded in equation (4).

Equations

[1] $p(L_n|C_n) = p(L_{n-1}|C_n) + (1 - p(L_{n-1}|C_n)) * p(T)$

[2] $p(L_n|E_n) = p(L_{n-1}|E_n) + (1 - p(L_{n-1}|E_n)) * p(T)$

[3] $p(L_{n-1}|C_n) = p(L_{n-1}) * p(C|L) / (p(L_{n-1}) * p(C|L) + p(U_{n-1}) * p(C|U))$

[4] $p(L_{n-1}|E_n) = p(L_{n-1}) * p(E|L) / (p(L_{n-1}) * p(E|L) + p(U_{n-1}) * p(E|U))$

(See next page for symbol definitions).

Definitions

$p(L_n)$	The probability a production rule is in the learned state following the nth opportunity to apply the rule
$p(L_{n-1})$	The probability a production rule is already in the learned state at the nth opportunity to apply the rule
$p(U_{n-1})$	The probability a production rule is in the unlearned state at the nth opportunity to apply the rule
C_n	A correct response at the nth opportunity to apply a rule.
E_n	An error at the nth opportunity to apply a rule.
p(C|L)	The probability of a correct response if the rule is currently in the learned state.
p(E|L)	The probability of a slip (an error) if the rule is currently in the learned state.
p(C|U)	The probability of a correct guess if the rule is currently in the unlearned state.
p(E|U)	The probability of an error if the rule is currently in the unlearned state.
p(T)	The probability a rule will transit from the unlearned to unlearned state given the opportunity to apply the rule.

Table 1

12 Constructs Introduced in the Curriculum

CONSTRUCT	EXAMPLE	RESULT
Data Structures		
Literal Atoms (symbols)	king	-
Literal Lists	(king queen bishop)	-
Extractor Functions		
car	(car '(a b c))	a
cdr	(cdr '(a b c))	(b c)
reverse	(reverse '(a b c))	(c b a)
Combiner Functions		
append	(append '(a b) '(c d))	(a b c d)
cons	(cons 'a '(b c d))	(a b c d)
list	(list 'a 'b 'c 'd)	(a b c d)
Function Definitions		
defun	(defun second (lis) (car (cdr lis)))	-
function name		
variable declaration		
variable reference		

Table 2

21 Rules in the Ideal Student Model

Section 1: Extractors

Rules: Code-Car-Extract-First-Element

Code-Cdr-Remove-First-Element

Code-Reverse-Flip-List

Code-Literal-List-Argument

Sample Exercise: (car '(a b c))

Section 2: Combiners

Rules: Code-Append-Merge-Lists

Code-Cons-Insert-Element

Code-List-Group-Expressions

Code-Literal-Atom

Delete-Extra-Editor-Node

Sample Exercise: (cons 'a '(b c))

Section 3: Extractor Algorithms

Rules: Code-Extractor-Start-Algorithm

Code-Extractor-Complete-Algorithm

Sample Exercise: (car (cdr '(a b c)))

Section 4: Function Definitions - Extractor Algorithms

Rules: Code-Defun

Code-Function-Name

Declare-Variable

Code-Variable-Reference

Delete-Variable-Declaration-Node

Delete-Top-Level-Editor-Node

Sample Exercise: (defun second-of-first (lis) (car (cdr (car lis))))

Table 2 (cont).

Section 5: Function Definitions - Combiner/Extractor Algorithms

Rules:	Code-Append-Combine-Extractions
	Code-Cons-Combine-Extractions
	Code-List-Combine-Extractions
	Code-Extractor-Argument-to-Combiner
Sample Exercise:	(defun ends (lis) (list (car lis) (car (reverse lis))))

Figure Captions

Figure 1. The computer screen near the beginning of an exercise.

Figure 2. Actual and predicted error rates across 41 students at each goal in the required tutor exercises. (Remedial exercises are traced, but not fit).

Figure 3. Actual and predicted error rates across 41 students at each goal in the required tutor exercises. (Remedial exercises are not traced).

Figure 4. The Learning Curve for the production rule that declares a variable in a function definition.

Figure 5. Error rates for simple extractor functions in the first six exercises and in three more complex exercises drawn from the last section of the lesson.

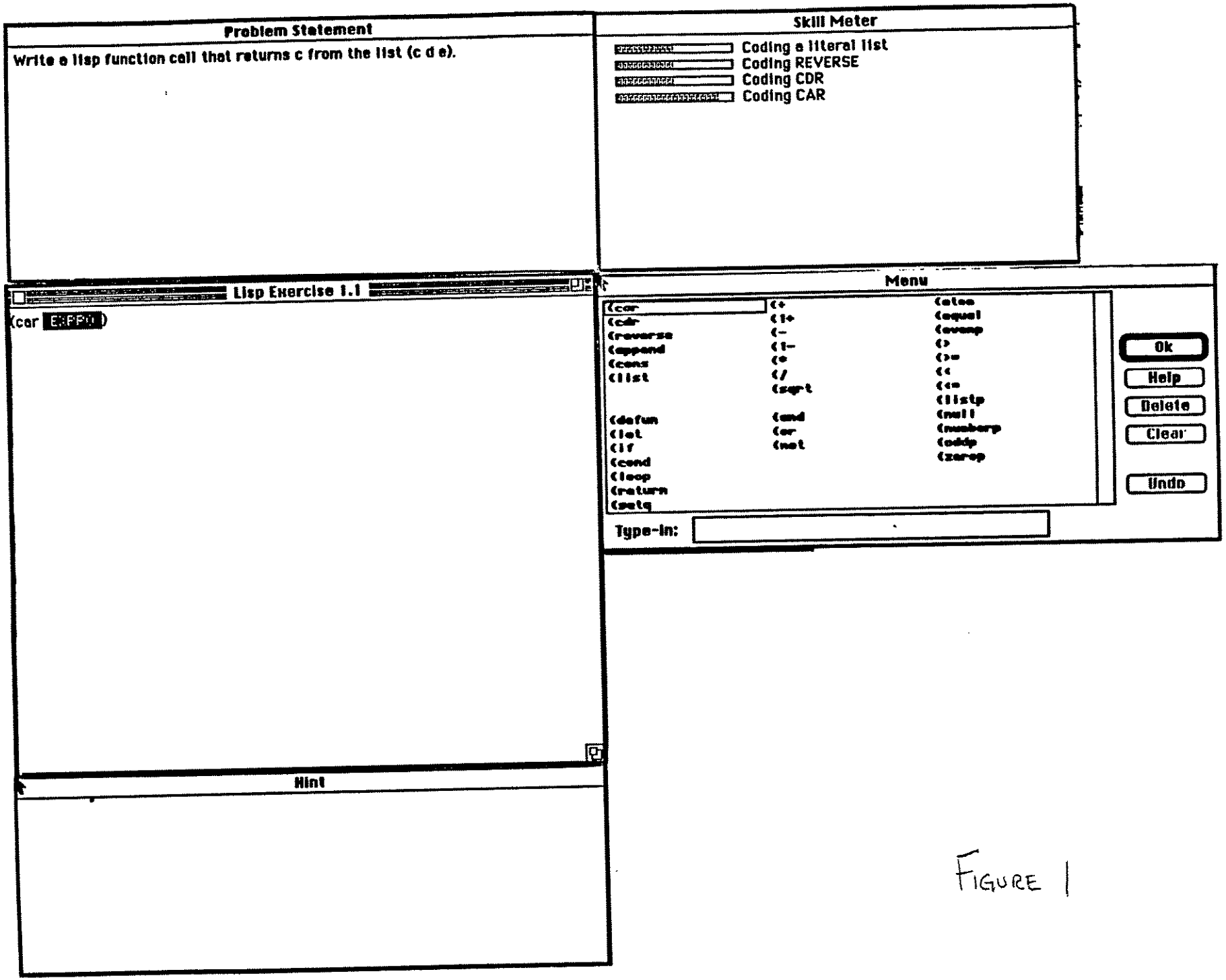
Problem Statement
Write a lisp function call that returns c from the list (c d e).
Skill Meter
Coding a literal list
Coding REVERSE
Coding CDR
Coding CAR
Lisp Exercise 1.1
(car
Menu
(car
(cdr
(reverse
(append
(cons
(list
(defun
(let
(if
(cond
(loop
(return
(setq
(+
(1+
(-
(1-
(*
(/
(sqrt
(and
(or
(not
(eln
(equal
(evenp
(>
(>=
(<
(<=
(listp
(null
(numberp
(oddp
(zerop
Ok
Help
Delete
Clear
Undo
Type-in:
Hint

FIGURE 1

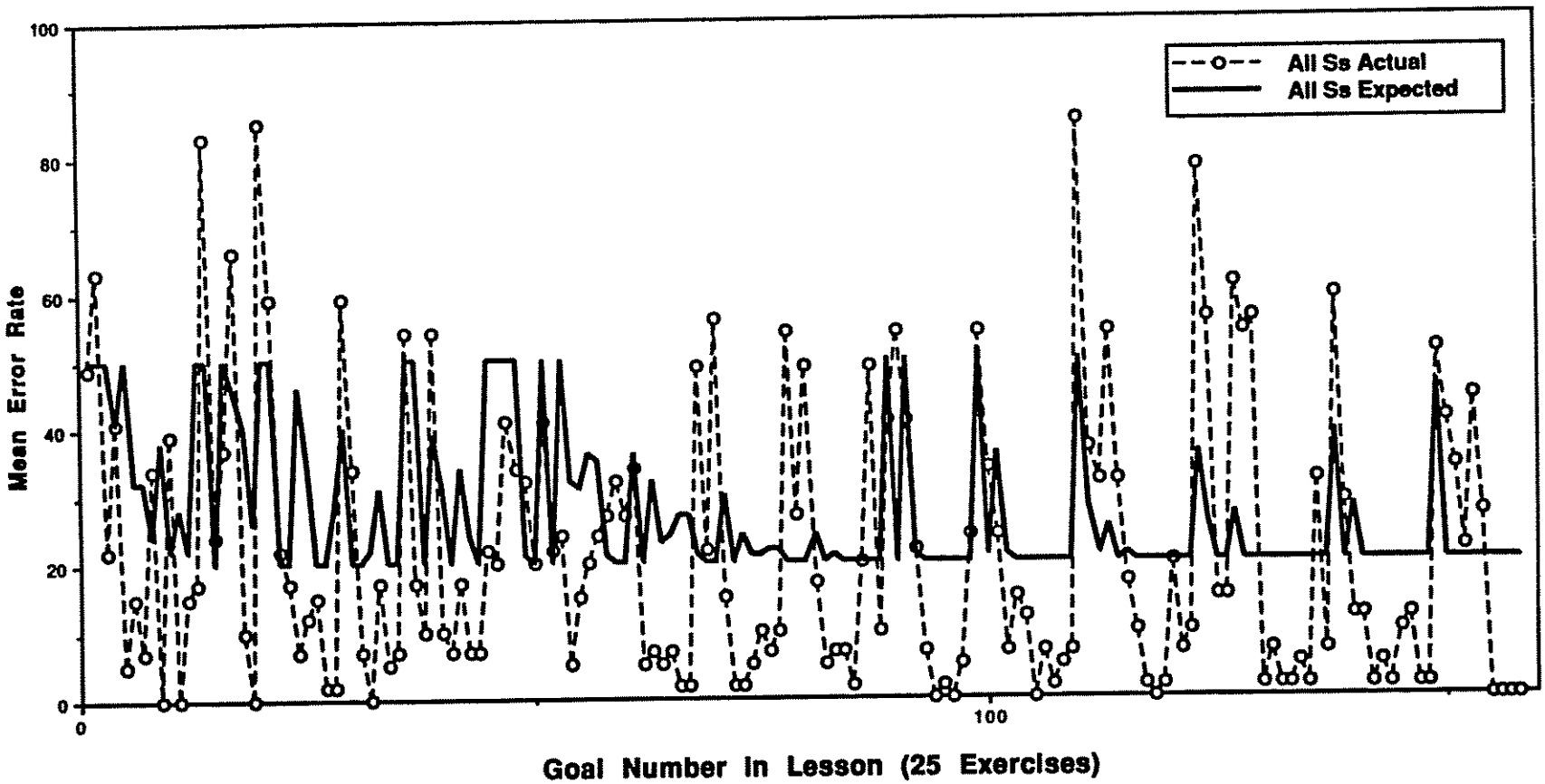
Constant Parameter Estimates
All Ss Actual
All Ss Expected
Mean Error Rate
100
80
60
40
20
0
0
100
Goal Number in Lesson (25 Exercises)

FIGURE 2

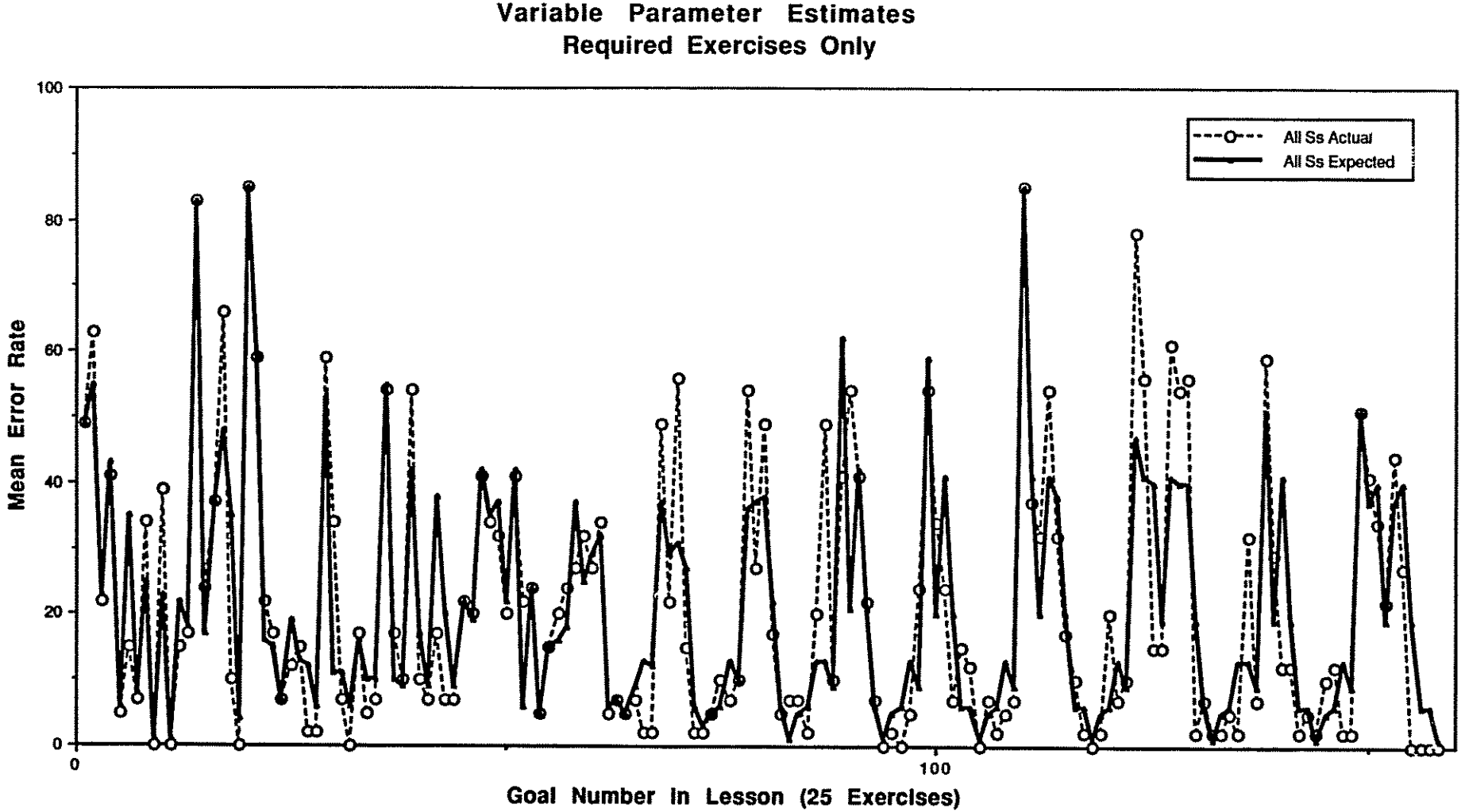
Variable Parameter Estimates
Required Exercises Only
All Ss Actual
All Ss Expected
Mean Error Rate
100
80
60
40
20
0
0
100
Goal Number In Lesson (25 Exercises)

FIGURE 3

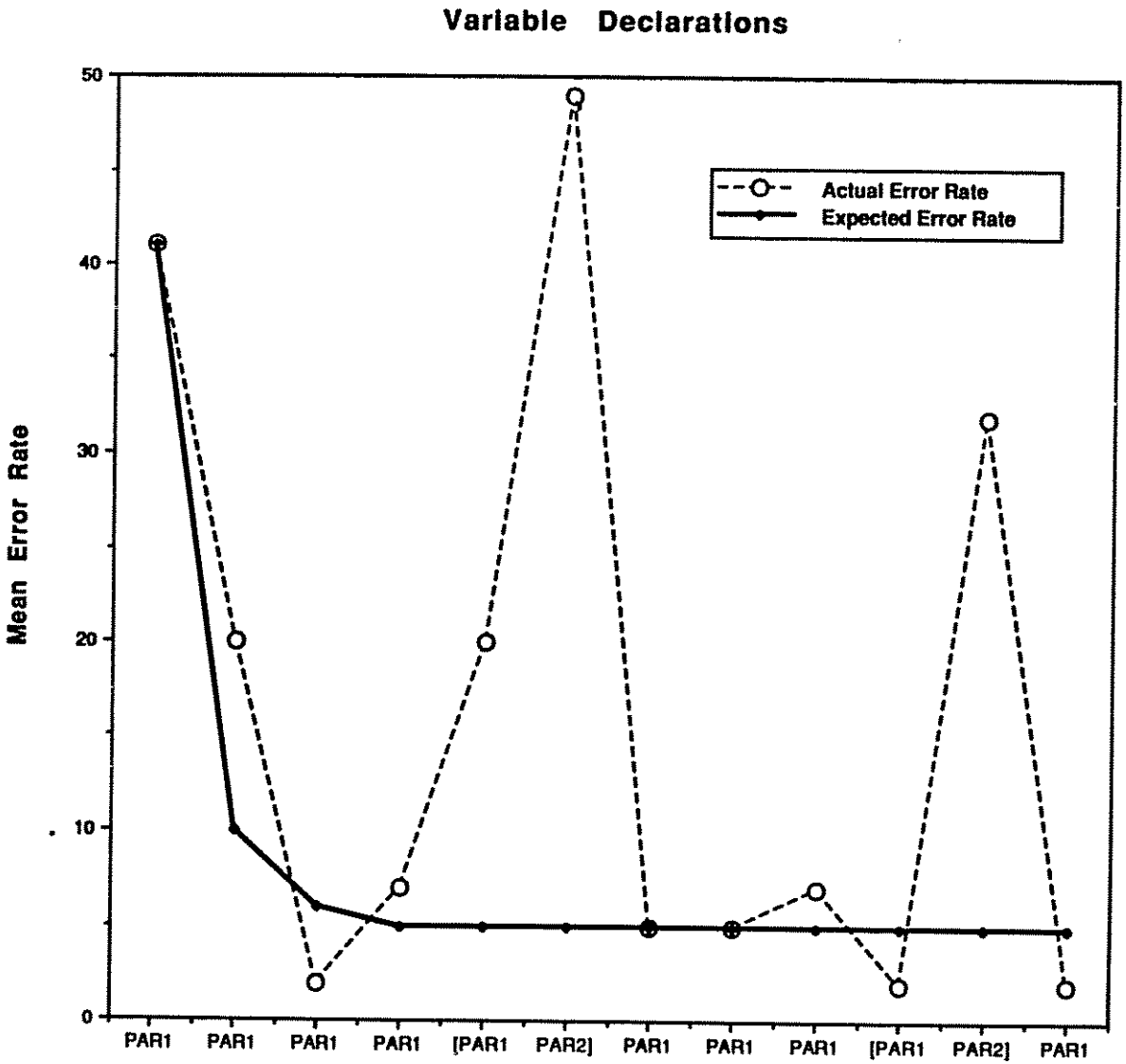
Variable Declarations
Mean Error Rate
50
40
30
20
10
0
Actual Error Rate
Expected Error Rate
PAR1
PAR1
PAR1
PAR1
[PAR1
PAR2]
PAR1
PAR1
PAR1
[PAR1
PAR2]
PAR1

FIGURE 4

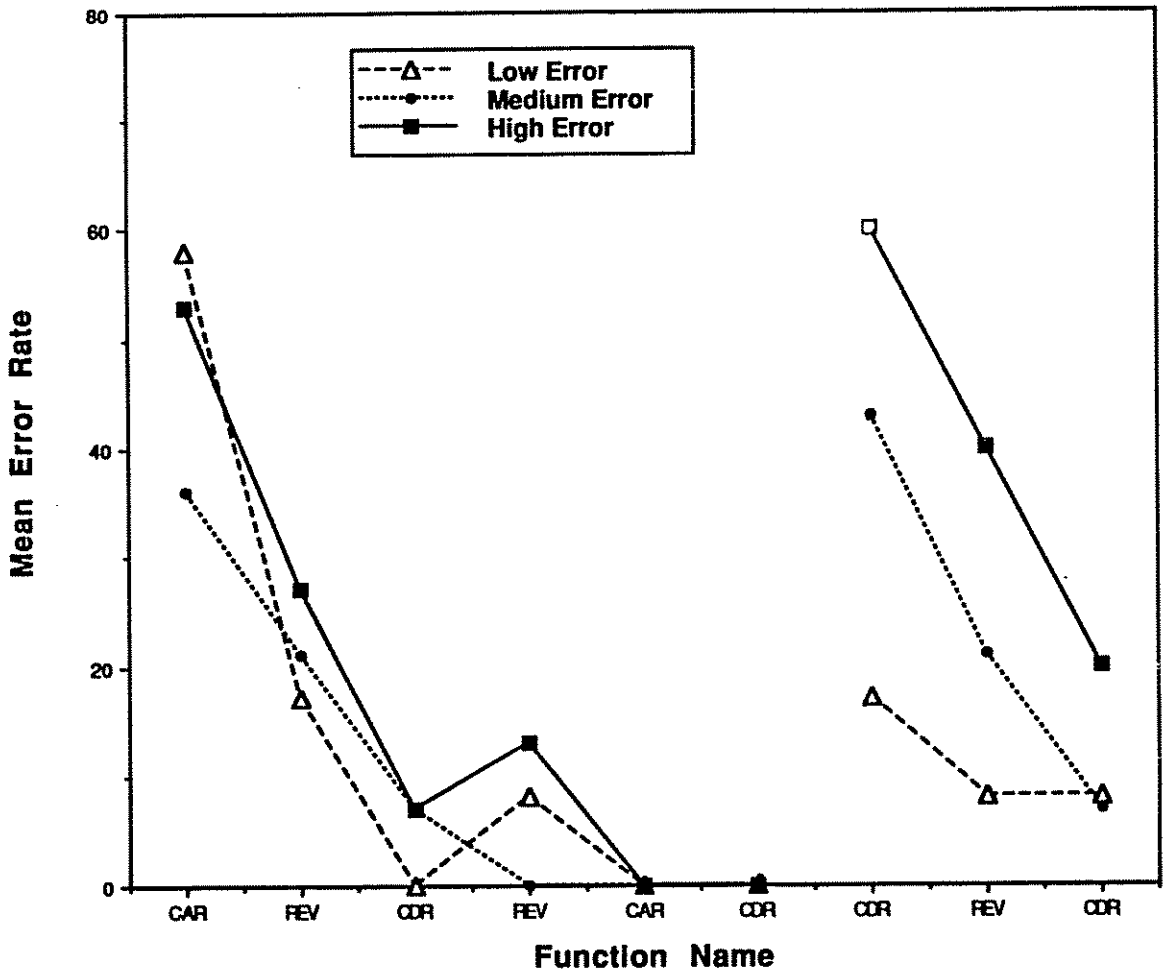
Simple Extractors: CAR, CDR, REVERSE
Low Error
Medium Error
High Error
Mean Error Rate
80
60
40
20
0
CAR
REV
CDR
REV
CAR
CDR
CDR
REV
CDR
Function Name

FIGURE 5