BAD ANALOGIES AS THE SOURCE OF SYSTEMATIC ERRORS IN PROBLEM SOLVING SKILLS

Technical Report AIP-18

John R. Anderson

Department of Psychology
Carnegie-Mellon University
Pittsburgh, PA 15213 U.S.A.

The Artificial Intelligence and Psychology Project

Departments of
Computer Science and Psychology
Carnegie Mellon University

Learning Research and Development Center
University of Pittsburgh

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AIP - 18	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Carnegie-Mellon University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Computer Sciences Division Office of Naval Research (Code 1133)
6c. ADDRESS (City, State, and ZIP Code) Department of Psychology Pittsburgh, Pennsylvania 15213	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, Virginia 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Same as Monitoring Organization 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-86-K-0678
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS p400005up201/7-1-90 PROGRAM ELEMENT NO: N/A; PROJECT NO: N/A; TASK NO: N/A; WORK UNIT ACCESSION NO: N/A

11. TITLE (Include Security Classification)
Bad Analogies as the Source of Systematic Errors in Problem Solving Skills

12. PERSONAL AUTHOR(S)
John R. Anderson

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 86Sept15 TO 91Sept14	September 29	48

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Bugs, Systematic errors, learning, Cognitive Psychology,

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This paper starts with a sketch of the basic learning processes that we think are at work in the subjects we have tutored. This basically involves students doing a causal analysis of the examples and trying to extend that analysis analogically. The most straight forward ways errors can occur in this analogy process is for students to choose an inappropriate example to map to the current solution. This is certainly not the only way misunderstandings can arise and it is of interest to get a sense of what fraction of student errors may be expalined in this fashion. Therefore, I will present a list of some of the dominant misconcpetions that students have displayed interacting with our tutors and see how many can be interpreted in this fashion. Then, I will speculate on how many of the subtraction errors that were the focus of Brown and VanLehn's analysis can be so explained. Finally, I will compare this analysis of the source of errors with the analysis offered by VanLehn (1983, 1986).

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Alan L. Meyrowitz	22b. TELEPHONE (Include Area Code) (202) 696-4302 22c. OFFICE SYMBOL N00014

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

We have been working on intelligent tutors to teach students problem solving skills in the domains of generating proofs in geometry (Anderson, Boyle, & Yost, 1985), writing LISP programs (Reiser, Anderson, & Farrell, 1985), and solving algebraic manipulation problems (Lewis, Milson, & Anderson, 1987). Student errors are the most important events in terms of guiding the tutorial dialogs. They are very good signals that the student is in need of help. Other cues do not seem so reliable. Latency of response and requests for help are the two other cues that we have considered in addition to errors. Long latencies may only mean that the student has been distracted and students are notoriously unwilling to ask for help. On the other hand an error comes close to being a necessary and sufficient condition for tutorial intervention. They are the primary signal used by almost all tutorial systems that have been built.

Errors can have different etiologies. One class of error is the slip which is characterized by the fact that the subject does not reliably make that error and can self correct when the error is signaled. In our work on tutoring, slips appear to decrease in frequency with practice and increase with working memory load. We (Anderson & Jeffries, 1985) have argued that slips can be traced to losses from working memory of critical information for solving a problem. Thus, when memory load goes up, slips increase. The decrease in slips with practice reflects a growing effective working-memory capacity with expertise (Chase & Ericsson, 1982). The decrease in such errors tends to follow a power function typical of practice effects. Figure 1 shows the decrease in error rates for various steps of problem-solving in the LISP tutor as a function of number of opportunities to practice these. Note after the first trial the functions are linear in a log-log scale. Thus, our view is that after the first trial in the LISP tutor most of the decrease in errors reflects a decrease in slips. Since slips do reflect lack of expertise, they are useful for monitoring student progress and assigning remedial problems even if they do not reflect any

fundamental misunderstandings that have to be corrected.

Insert Figure 1 about here

A second category of errors that have been discussed (eg. McCloskey, 1983) are importations of prior misconceptions into a new domain. Physics is one domain that has many such errors. However, they have not been a significant category of error in our work given that we are working in domains for which students do not have an abundance of prior conceptions.

A third category of errors, which is the focus of this paper, is within-domain misconceptions. These are misconceptions which arise not because of prior beliefs that the student has, but as a consequence of the learning that takes place in the domain. These are perhaps the most interesting errors because they reflect on the nature of the learning process itself. Also in contrast to the first category of errors they challenge the tutor. It is not enough to simply point out the error because the student cannot self correct easily. The tutor must explain to the student something about what the error is and what is required for a correct answer.

A very elegant set of analyses have been done by Brown & Van Lehn (1980) and Van Lehn (1983) on within-domain misconceptions in multi-column subtraction. Brown & Van Lehn's basic argument was that these errors arose when students tried to bridge points in their problem-solving where they had inadequate knowledge. These points are called impasses and the bridging process produces what are called repairs. Students can come to believe their repairs in which case a permanent misconception appears or they can not in which case we see an inconsistent pattern of "bug migration". The analysis to be offered here has a lot in common with this analysis. Its major point of departure is that it

conceives of the repair process as rather simpler than what Brown and Van Lehn have proposed. This paper will argue that these errors are caused by making analogies to misleading examples of problem solution.

This paper will start with a sketch of the basic learning processes that we think are at work in the subjects we have tutored. This basically involves students doing a causal analysis of the examples and trying to extend that analysis analogically. The most straight forward way errors can occur in this analogy process is for students to choose an inappropriate example to map to the current solution. This is certainly not the only way misunderstandings can arise and it is of interest to get a sense of what fraction of student errors may be explained in this fashion. Therefore, I will present a list of some of the dominant misconceptions that students have displayed interacting with our tutors and see how how many can be interpreted in this fashion. Then, I will speculate on how many of subtraction errors, which were the focus of Brown and VanLehn's analysis, can be so explained. Finally, I will compare this analysis of the source of errors with the analysis offered by Van Lehn (1983, 1985) .

The PUPS Theory of Learning

PUPS (Anderson & Thompson, in press) is somewhere between an elaboration and a successor of the ACT* theory that I developed a few years ago (Anderson, 1983). It differs from that theory mainly in the emphasis it gives to analogy in the knowledge acquisition process.

In the PUPS theory, learning progresses from concrete examples to abstract principles through four stages:

1. Examples are encoded into declarative structures which record their perceptually available form and attributes. Thus, if we see someone type (+ 2 3) into a CommonLISP

system and the system respond 5, PUPS would encode this with something like the following structures:

event1: isa typing
form (type person1 message computer)
context CommonLISP

message: isa list
form (list + two three)

two: isa integer
form (text 2)

three: isa integer
form (text 3)

event2: isa response
form (print computer five screen)

five: isa integer
form (text 5)

2. The example is "understood". By this I mean the components of the structure are placed into a causal framework. Thus, the student might infer that the first event caused the second. This information gets encoded in function slots of the examples.

3. When an appropriate problem arises an attempt is made to analogically extrapolate the understanding of the example to produce a solution to the current problem.

4. Successful extrapolations are encoded as production rules.

The first step of this process I am simply assuming--that our perceptual system (perhaps including the linguistic system) can deliver an encoding of the forms and attributes of things in our environment. Steps 2 and 3 are far from trivial and will occupy most of our attention. Step 4 will receive some analysis.

Step 2: Causal Induction

People have an almost irresistable urge, when they encounter an event or object, to ascribe a cause to it where that cause is some other event or object. The majority of our

adult causal ascriptions flow from theories we have already acquired, but people are quite capable of making these ascriptions in the absence of a domain theory. It is from these pre-theoretical inferences that domain theories eventually arise. Note I am making no claims in this paper about what causality is like in the real world or indeed if there is such a thing as causality. I am only asserting that people naturally perceive causality whether it is really there or not. The beauty of human causal perception is that it ignores the philosophical dimemnas about causality and so produces knowledge.

There are at least three well-documented factors inducing people to perceive one thing as causing another in the absense of an existing theory (Lewis, 1986; Shultz, 1982; Siegler, 1976). Each of these can be fairly easily justified as a rational basis for making causal ascriptions:

(a) Contiguity--People tend to perceive something as the cause the closer in time and space it is to the effect with the strong discontinuous provision that effects cannot preceed their causes.

(b) Similarity--People tend to perceive something as a cause the more similar it is to the effect. It is difficult to specify an all-encomposing metric for similarity, but for our purposes the important feature is that two things are more similar if they overlap in a number of components. For instance, suppose we observe two events involving an unknown computer system--the user points to an icon of an apple and he points to an icon of a dog. After both of these events a third event hapens--the icon of the apple disappears. We are more likely to think the cause of the third event is the pointing to the apple icon than the dog icon. This is because both cause and effect involve the apple icon.

(c) Statistical--If a cause has been accompanied fairly regularly by an effect and the effect has seldom occurred in the absence of the cause, we are likely to perceive a causal

relationship. Note perception of causality does not demand a perfect predictive relationship. There can always be extenuating circunstances.

It is something of an open question just how these three factors should be computed and combined to produce an attribution of a causal relationship. One can imagine doing psychological research by creating somewhat artificial situations to test for refined predictions of one scheme versus another. However, typically causal attributions are highly overdetermined. For instance, suppose we have no knowledge of computers and that we see (+ 2 3) typed into the computer and see 5 as a response. We would decide that the typing caused the 5 on the basis of contiguity, similarity (5 is the sum of 2 and 3), or by statistical trials noting the regularity of the relationship.

Causal information is stored in special cause slots of PUPS knowledge structures. Thus if we were to take our earlier example and embellish it with causal information it would look like:

example: isa typing
form (type person1 message computer)
cause (event)
precondition (example context CommonLISP)
context CommonLISP

message: isa list
form (list + two three)

event: isa response
form (print computer five screen)

two: isa integer
form (text 2)
position (first fact)

three: isa integer
form (text 3)
position (third fact)

five: isa integer
form (text 5)
position (fifth fact)

fact: isa addition-fact
form (sequence two plus three is five)

I have also included the addition fact which is critical to being able to extrapolate

this event.

The form slot is used to record the physical form of the object or event. position slots record the physical positions of these objects in relation to other objects, and cause slots record the causal position of these objects. Preconditions can be attached to cause slots. This example has the precondition that it produces its cause in the context of CommonLISP. Context is just another slot associated with the example knowledge structure. Precondition information is the result of second-order causal analysis which comes about when one wants to predict when a causal relationship will hold or not. In our hypothetical situation of someone observing an interaction with LISP for the first time it could only arise if the person knew that typing (+ 2 3) did not always result in 5 being output by a computer.

In many situations it is useful to compose causal relationships into higher order relationships for compact representation. This composed information is stored under function slots. Thus. rather having the example cause an event and the event be the printing of the message on the screen. we might represent example as:

example:
isa typing
form (type person1 message computer)
function (display computer five)
precondition (example context CommonLISP)
context CommonLISP

where (display X Y) means (cause (print X Y screen)). The development of analogy by Anderson & Thompson (in press) was with respect to function slots and not cause slots. In the remaining discussion we will continue this practice. However. if the reader becomes puzzled about the meaning of these function slots he should remember that they are compositions of causal relationships.

Knowledge Extrapolation

Knowledge extrapolation involves trying to extend a causal analysis to a new situation. Suppose, for instance, one wanted to predict what would happen when (+ 6 7) was typed into CommonLISP. Analogical extrapolation in PUPS (Anderson & Thompson, in press) allows one to map the past example onto the current example provided the categories (isa slots) of the objects are the same. In this case, 2 from the past example would map onto 6, 3 onto 7, and 5 onto 13. Thus, PUPS could predict the computer would display 13. In effect, PUPS has extracted from the example the following rule:

```
IF    =structure:   isa typing
                    form (type =person =message =computer)
                    context CommonLISP
      =message:     isa list
                    form (list + =num1 =num2)
      =num1:        isa integer
                    position (first =fact)
      =num2:        isa integer
                    position (third =fact)
      =fact:        isa addition-fact
                    form (sequence =num1 plus =num2 is =num3)
THEN  =structure:   function (display =computer =num3)
```

The rule above is a production rule which predicts the function of =structure given its form. We can also create problem-solving productions in which the form necessary to achieve a function is specified:

```
IF    goal:         isa typing
                    function (display =computer =num3)
                    context CommonLISP
      =event:       isa response
                    form (text =num3)
      =num3:        isa integer
                    position (fifth =fact)
      =fact:        isa addition-fact
                    form (sequence =num1 plus =num2 is =num3)
THEN  goal:         form (type =person =message =computer)
      =message:     isa list
                    form (list + =num1 =num2)
```

These analogical extrapolations depend on two basic assumptions:

(1) One can map one member of a category to another. Thus, we are able to replace 2, 3, and 5 by variables. This has been called the no function in identity principle in that it asserts that elements from the same category which appear in analogous positions in form slots will appear in analogous positions in function slots. The assumption is that elements of the same category and in the same position in the form will have the same functional relationships.

(2) One is able to trace paths through the function and form slots of structures to find paths of connections from the condition side to the action side of a production. Thus, in the rule above we find a path from 5 to the addition fact that relates it to 2 and 3 in the action side. This has been called the principle of functional elaboration.

For more discussion of both principles, see Anderson & Thompson (in press).

While we often talk about knowledge extrapolation in terms of the rules extracted from an example, it is only evoked when there is an example and a target problem to map it to. The rule is just a specification of the mapping from the example to the target. Thus, availablility of an appropriate example is key. Much of the research of Ross (1984) on analogy turns on manipulating variables that make the example more or less available in the target context. Key to our analysis of errors will be this notion that examples which are misleading are highly available in the problem context.

Knowledge Compilation

The processes of causal inference and knowledge extrapolation produce general rules that can be used for prediction and problem solving. Those rules that prove successful get permanently recorded as production rules. Figure 1 earlier illustrated the marked improvement in error rate after the first trial followed by slow power-law improvement. Figure 2 illustrates the same thing for time to perform the action both for the LISP tutor

and the geometry tutor. Thus, there is also a marked speed up after the first successful application of a piece of knowledge. The next application typically takes half as long. In our view this reflects the greater efficiency of rule recognition relative to knowledge extrapolation. Once a rule has been codified as a production it can accrue strength as it proves repeatedly successful. There is a gradual continued improvement in speed of rule application with practice.

Insert Figure 2 about here

This process of permanently storing successful rules as productions we call knowledge compilation. It really does not change the behavioral potentials of the system, abstractly defined. For instance, an analogy can take precedence over a compiled rule if it offers a better fit to the problem situation. The only effect of knowledge compilation is to reduce the resource costs of mainfesting knowledge. Of course, by changing the resource costs, one can change the effective behavior of a system even if not the abstract behavioral potentials. Paths of problem-solution which were too consuming of time or working memory now become feasible.

Sources of Errors in PUPS

Given this general picture of learning where can systematic errors arise? There are in fact multiple ways that PUPS can make errors, but this paper is devoted to exploring what seems to me to be the most obvious way of making errors--which is to map an example inappropriately to the problem. Typically this is because the example that is chosen is inappropriate for solving the target problem, but it is possible an appropriate example is selected but mapped inappropriately. Either of these cases will be put under the heading of "misleading example". As an example of how obvious this is as an error mechanism I can report that the most frequent criticism I get from people I describe

PUPS to goes like this: "Yes, it works all right if you choose just the right example and encode it just right but how can you assume people can always do this?"

The major goal of the remainder of the paper will be to look at systematic errors in our three tutors and see how many can be interpreted as mistaken examples. I have culled from our protocols cases where at least 30% of the students make the same error at the same point (the 30% is calculated as a percentage of all responses, not just errors). This consistency of the error suggests it is not a slip. Unfortunately for the current purposes, errors tend not to be repeated across problems in the tutors. This is because subjects are immediately corrected. Thus, we do not see in our tutor interactions the kind of consistent bugs that Burton and Brown (1982) found in subtraction. The reason is that they did not intervene tutorially while we do. Had we not intervened, the error we see might become a permanent part of the student's reportoire. However, since we do intervene, we cannot use consistency of error pattern as a basis for diagnosing systematic errors. This is why we resort to a criterion like 30% same errors because it is highly unlikely that this consistency would occur through slips. However, this criterion means we will not see the bizarre and rare errors Burton and Brown were able to document.

Probably it would be possible to conjure up some example under some encoding would serve as the basis for any error we observed. Thus, the mere fact that we can produce such an explanation is not very compelling. However, I think the explanations we will offer are far from strained and indeed are quite compelling--perhaps even obvious. Our analysis will be more compelling if it turns out that examples which could serve as the source of the analogy occur in the vicinity of the error. Our observations of analogical problem solving in these domains is that it makes use of close-by examples. While it is always possible to conjure up some example that could produce the error analogically, it is certainly not always possible to use one of the recent examples except under the most

extraordinarily bizarre encodings. So finding these examples in the vicinity of the error should add considerable force to our analysis. This is where our data base is at an advantage over the one used by Burton and Brown. They have no record of the learning context in which the error first appeared. With our tutors we have a very good record.

What we will do is show that the majority of the errors we see can be interpreted as due to bad analogies to close-by examples. What we do not have a handle on yet is whether PUPS predicts analogy errors that in fact are not observed--what Brown and Van Lehn (1980) would call star bugs. We are currently engaged in large scale simulation efforts to find an exhaustive list of what errors PUPS predicts in our students interactions with the tutors.

Errors with the LISP tutor

In the case of the LISP tutor I will present a list of all errors meeting the 30% threshhold which occur in lessons 2 and 3 of the tutor. I am looking at lessons 2 and 3 because lesson 1 has some peculiar properties due to start-up with the LISP tutor.

First

Lesson 2 concerns how to write LISP functions. The very first problem is specified to the students as: "Define a function called first. Given any list, it returns the first element of that list. For example, (first '(a b c)) returns a." Given that this function is totally redundant with the LISP function CAR, this is really just an exercise in function definition. The following is the correct code:

```
(defun first (lis)
       (car lis))
```

A full 30% of the subjects make the error illustrated below:

```
(defun first (lis)
       (car (lis)))
```

Anderson, Farrell, & Sauers (1984) discuss at length a simulation of a protocol of a subject making this error. In that simulation it came from an inappropriate analogy to the following function definition:

```
(defun f-to-c (temp)
        (quotient (difference temp 32) 1.8))
```

where the first argument to the function quotient was in parentheses. This is quite clearly an example of an error in the representation of the example. The subject is not representing the fact that the parentheses in the f-to-c example are to hold an embedded function call not just an argument. I point the reader to Anderson et al for a more thorough discussion of this error. The important observation for current purposes is that the example which is causing error is in the close vicinity of the problem.

In informal interviews with students, we have also noticed some who make this error out of analogy to the parameter list in the first line of the function definition. In fact, many teachers have complained about the fact that the parameters in LISP definitions are placed in parenthesis and the arguments to functions are not. This is another case of an error by analogy to an inappropriate example. The student either does not encode or ignores that a list is used for specifying parameters and not for holding arguments to functions.

It is worth looking at how the paremeter list and argument to a function would be correctly represented in PUPS to get a sense of where the error could be. The example would be correctly represented something like:

```
example:    isa parameter-list
            form (list lis)
            function (assign lis value)
```

where "assign lis value" is shorthand for something like "cause LISP to cause lis stand for value." And the goal would be correctly represented:

problem: isa argument
function (evaluate-to value)
form ???

Clearly, under this representation there is no reason for the example to be mapped to the problem. It is possible that the subject does not discriminate between the two, perhaps representing both as having a function something like "hold value". We have found that careful textbook instruction emphasing the difference between a variable in a parameter list and a variable as an argument to a function reduces this error. It is also possible that the student represents the difference but tries to force one onto the other because they both involve value which is bound to lis. Unfortunately, we do not have a model in PUPS of which a forced extrapolation process. Nonetheless, given the proximity of the error and subject comments, I think it is pretty clear that this is another way that analogy can lead to the error of embedding the variable in parentheses when it is an argument. Thus, this is an error that has at least two seperate sources as a bad analogy.

Replace

The function that follows first is called second and it is supposed to return the second element of the list. This does not show up a comparably consistent error. The next function does however. It is specified to the student as: "Write a function replace that replaces the first element of a list with a new element. The function takes two parameters--the new element and the list. For example, (replace 'rings '(ties hats pants)) returns (rings hats pants)." The correct code is:

```
(defun replace (item lis)
       (cons item (cdr lis)))
```

A full 55% of the students start out trying to define the function:

```
(defun replace (variable)
```

where variable stands for whatever variable name they choose. The relevent fact is that the two previous function definitions both involved a single list parameter. This is further

evidence that some subjects are not understanding what the parameter list is about and inappropriately map it here. Our guess is that subjects incorrectly conjecture that each parameter must separately be placed in parentheses. We would predict that subjects would have encoded a second parameter in a second list. That is, their code would have taken the form:

```
(defun replace (item) (lis)
```

with a list for each parameter. Unfortunately, the tutor stops them after their first error and so we do not see how they would have continued the code. The important observation about this analysis of the error and others like it is that it attributes the error to incorrectly understanding an appropriate example and consequently inappropriately mapping that example.

Sqr.

The next function to display a high consistent error pattern is specified to the student as "Define a function called sqr that returns a list of the perimeter and the area of a square, given the length of one side. For example, (sqr 2) returns (8 4)" The correct code is displayed below:

```
(defun sqr (side)
        (list (times side 4) (times side side)))
```

A full 35% of the subjects started their code as follows:

```
(defun sqr (side)
        (times
```

In which "list" is missing. This error has no obvious analog and so stands as the first example of an error that cannot be explained by analogy to a close-by example. It is an instance of what we have called a part error. That is, subjects never had to code a list answer before and we assume this 35% could not figure out how to achieve that. What they did was to proceed to code that part of the answer which they did know how to

code--namely the circumference of a circle. Subjects have a tendency to do part of a problem if they cannot do it all.

Ends.

The next function that produces a high error rate is specified to the students as "Define a function called ends, that has one argument and returns a list containing the first and last items in that argument. For example, (ends '(a b c d)) returns (a d)". One form of the correct code is given below:

```
(defun ends (lis)
        (list (car lis) (car (last lis))))
```

A full 50% of the subjects try the following code:

```
(defun ends (lis)
        (list (car lis) (last lis)))
```

This argument could be classified as a part error--subjects did not know how to code the first of a list and so they coded just the list--(last lis) returns a list of the last element of the list. However, this does not seem very plausible. Subjects have coded car successfully many times and this is the first time they used last. More likely it represents a misunderstanding of what last does. Our guess is that subjects do not know what last does--that they believe it returns the last element rather than a list of the last element. We have observed many students firmly assert that last returns the last element despite the fact that they have read instruction and seen examples to the contrary. Our guess is that this misunderstanding of last comes from interpreting it as analogous to car which does return the first element rather than the first list. This error is an interesting case. Under this analysis it has its origin in analogy, but it is not an analogy of the problem to a nearby example. Rather it is an analogy of the instruction about last to the knowledge about car. Thus, we cannot count this as a supporting case for our hypotheses although the error may stem from analogy.

Snoc.

The next function to produce a high error rate is snoc which is specified to students as: "write a function called snoc that is the opposite of cons. Instead of inserting an item into the front of the list, it inserts the item at the end. For example, (snoc 'd '(a b c)) returns (a b c d). Write this function without using append". The correct code is given below:

```
(defun snoc (item lis)
        (reverse (cons item (reverse lis))))
```

A full 70% of the students start their code for this function:

```
(defun snoc (item lis)
        (cons
```

The interesting feature of this example is that the instruction itself implicitly references an example of cons. We think the error derives from analogy to this implicit example. We assume students represent the body of the function they are to write as follows:

```
problem: isa lispcode
           function (insert item lis end)
```

and their representation of the implicit cons example is:

```
example:  isa lispcode
           function (insert item lis front)
           form (cons item lis)
```

The implicit example has the insert at the front where the problem has the insert at the end. Perhaps they intended to reverse the arguments to cons or perhaps they choose to ignore the mismatch between cons and their current problem.

Samesign

The third lesson is on predicates, logical operators, and conditionals. The first function to produce a high consistent error was described to the students as: "Define

samesign. It takes two numbers as arguments, and returns t if both arguments have the same sign. That is, if both arguments are zero, both positive, or both negative, the function should return t. For example, (samesign 0 0) returns t, (samesign -2 -5) returns t, and (samesign -2 3) returns nil." At this point students should have known about the logical functions and and or but not cond. Thus the code we expected was:

```
(defun samesign (num1 num2)
      (or (and (greaterp num1 0)(greaterp num2 0))
          (and  (greaterp  0  num1)(greaterp  0  num2))
          (and  (zerop  num1)(zerop num2))))
```

Or some equivalent variant. 80% of the students started their coded displaying the following error:

```
(defun samesign (num1 num2)
       (greaterp...
```

That is they ignored the or and the and. Since this was the first exercise in or and and we assume they just did not know how to use them. This is the third case where there is no obvious analog to a nearby example for making the error. This seems like another part error where the subject is again trying to write that part of the code they know how to do--namely coding predicates.

Carlis.

The first function to use conditional structure was carlis. It was specified to the subject "Define carlis. It takes one argument. If the argument is a nonempty list, then carlis returns the first element of that list. But if the argument is the empty list, then carlis returns the empty list. If the argument is an atom, carlis returns just that atom. Hint: Be careful how you order your tests. Remember that nil is both an atom and a list. For example, (carlis '(cat rabbit)) returns cat, (carlis 'george) returns george, and (carlis nil) returns nil." The correct code for carlis is:

```
(defun  carlis (object)
```

```
(cond ((null object) nil)
      ((atom object) object)
      (t (car object))))
```

or some variant. This problem produced two high frequency errors which are illustrated below:

```
(defun  carlis  (object)
        (cond (null object) nil)
              ((atom object) object)
              ((listp object) (car object))))
```

The first error, made by 30% of the subjects is to type just a single left parenthesis before null. This is the first time they have had to code two left parentheses in a row and we assume the many examples of a single left parenthesis dominate. The second error, made by 80% of the students, is to use a predicate in the test for the final clause--(listp object). While not semantically a error, the tutor treats it as a styllistic error and the example students studied involved a t for the last clause. However, the two previous lines in this function provide more recent examples of coding tests and we assume these are the sources of the mistake. Basically, their status as appropriate for only non-final tests in not being represented or ignored by students.

Numline

The next function to produce a high error rate is numline. It is specified to the student as "Define a function called numline. It takes one arguement that is a number and returns a two element list. The first element of the list is t if the number is 0 and nil otherwise. The second element of the list is t if the number is negative and nil otherwise. For example, (numline -5) returns (nil t). The correct code for this function is:

```
(defun numline (item)
       (list (zerop item) (< item 0)))
```

This function just occurs after a series of functions all coded with cond. 50% of the students produce the same error which is to try to code the function as follows:

(defun numline (item)
(c ~nd...

This might lead to a function that works, but it is clearly non-optimal (which is what the tutor tells them). It seems that it is again analogy to the recent previous functions which produce this mistake.

LISP errors: A concluding remark

We have now reviewed the 9 dominant high frequency errors in lessons two and three with the LISP tutor. Two of these seem explainable as part errors and another 6 as inappropriate selection of a recent example. The ninth error seems to arise from a misanalogy in the understanding of the instruction about last. I think the relatively simple genesis of these errors is interesting given what a complex domain LISP programming is supposed to be.

Errors with the Geometry Tutor

In the case of the geometry tutor I will present an analysis of all above 30% errors that occur in the first chapter. This is a chapter mainly devoted to relating algebraic operations on geometry measurements to some basic geometry concepts like congruence. The high school students who go through this material often find it frustratingly subtle. If we were designing a geometry course we would not begin it with such material, but our tutor was compelled to follow the textbook sequence.

Addition Postulate

The first geometric rule introduced to the student is the addition postulate which is that if a = b and c = d, then a + c = b + d. Figure 3 shows the three problems on which students practice this postulate. The first two cause no difficulties--students combine the two premises to establish the conclusion. However, 80% of the students make the same error on the third problem. They try to apply the postulate to all three premises at

once. Whether this really should be counted an error is problematical but the tutor and the text restrict the addition postulate to two premises. What is remarkable is the number of students who make the opposite inference. I think the error quite definitely comes from analogy to this training sequence but the interesting observation is that PUPS, as it is currently implemented, does not make the analogy from two of a kind to three of a kind. This points in a direction that we have to develop the PUPS analogy system.

Insert Figure 3 about here

Reflexive Postulate

The postulate to follow addition is subtraction and it does not promote a similarly consistent pattern of errors. However, the next postulate, the reflexive postulate, asserting a quantity is equal to itself, does. Figure 4 show the first problem involving this rule. It requires subjects to first establish mDC = mDC and then use subtraction. The peculiar feature about establishing that

$$m\overline{DC} = m\overline{DC}$$

through the reflexive postulate is that this postulate requires no premises. Almost all subjects first choose the one premise given and try to apply some rule to it. This seems a clear analogy to their pattern of responding with the tutor up until this point where they have always had to choose a premise.

Insert Figure 4 about here

Definition of Congruence

The fourth rule is the definition of congruence that asserts segments with equal measure are congruent. Figure 5 shows a problem that produces two common errors. Almost all subjects once again fall into the habit of selecting the premise on the screen

rather than using the no-premise reflexive rule. Secondly, when subjects establish that mDC=mDC, they then combine this and the given premise by the subtraction postulate but choose as the conclusion the statement on the screen which involves congruences rather than the fact that mCA=mED which has to be converted into the congruence. This is a type of error that we see repeated over and over again; so it is worth offering a detailed PUPS analysis of it. The obvious analog to this problem is the problem in Figure 4 which occured but three problems earlier. Below is a PUPS encoding of the critical subtraction step of inference in it.

Insert Figure 5 about here

```
step:           isa inference
                function (prove statement)
                     precondition (= in form of statement)
                                  (= in form of premise1)
                                  (= in form of premise2)
                form (sequence premise-pick rule-type conclusion-pick)

premise-pick:   isa mouse-action
                function (get-premises step)
                form (sequence pick1 pick2)

pick1:          isa mouse-action
                function (pick premise1)
                     precondition (premise1 on screen)

pick2:          isa mouse-action
                function (pick premise2)
                     precondition (premise2 on screen)

statement:      isa geometry-statement
                function (equal CA ED)
                form (list mCA = mED)
                location screen

premise1:       isa geometry-statement
                function (equal sum1 sum2)
                form (list sum1 = sum2)
                location screen

premise2:       isa geometry-statement
                function (equal DC DC)
                form (list mDC = mDC)
```

location screen

rule-type: isa typing
function (name substraction step)
form (type s u b t r a c t i o n)

conclusion-pick: isa mouse-action
function (pick statement step)
precondition (statement on screen)

A number of these structures make reference to the mousing actions involved in selecting premises. I have ommitted some of the detail which is not neccessary for current purposes. Short of this, it is a correct coding of the rule in the sense that it should not map into the observed error. The reason is that the precondition that statement involve equality does not map. However, short of this Figure 4 is a perfect analog for the error in Figure 5. It even involves the same symbols and diagram. Our observation of students is that they do not even encode whether it is an equality of congruence in their conclusion. Thus, they have not encoded the information to enable them to apply the precondition that would filter out this error. At this point in the course there is also some question whether they actually encoded the precondition on the subtraction postulate. Given such encodings, PUPS could predict this misanalogy for multiple reasons.

Substitution

The next rule that students are introduced to is one that lets them substitute an equal term in another equality. Figure 6 shows the first problem that students have to solve with this rule. It involves using the definition of congruence to go in the other direction and convert

$$\overline{HN} \cong \overline{SM}$$

into mHN=mSM. 70% of the students pick both statements, without first converting, and try substituting the congruence statement into the measure statement. Again it is to be explained as students not respecting the equality precondition. This time the analogy is to

the example that is used to illustrate substitution in the instruction which does not require such a conversion.

Insert Figure 6 about here

Transitivity

The next rule is transitivity and Figure 7 shows the first problem on which 40% of the students make the same error. Perhaps the reader can guess what it is: The student combines the two equalities by transitivity and concludes the congruence.

Insert Figure 7 about here

Midpoint

The next rule is definition of midpoint. Without burdening the reader with the problems, let me just say students do well on the first two problems but once again overextend their knowledge on the third problem which requires discriminating between congruence and equality.

Betweenness

The next rule is the definition of betweenness which says that three points A, B, C are collinear with B in between if and only if mAB + mBC = mAC. The first four problems involving this rule are rather uneventful and then subjects come upon the problem in Figure 8a which high school students always find very difficult. Most students are able to analogize to their past four solutions and make the inferences that mAB + mBC = mAC and that mBC + mCD = mBD. Then the majority of student select these two premises. Having selected them, what they next try to do is unsystematic. Some students abort the inference; some try subtraction, some try addition, some try substitution, etc. This is the only high frequency error in our geometry protocols which does not seem to

have an explanation as an analogy to some example. In some sense subjects feel that these two premises must be enough to establish the conclusion. (The most common eventual solution is one where subjects show mBC = mBC and then mAB + mBC = mBC + mCD and then use transitivity or substitution.)

Insert Figure 8 about here

After this problem, the tutor repeats a number of similar problems. Figure 8b shows the third in the sequence (i.e. there is one problem between Figure 8a and Figure 8b). Students learn the inference pattern required, but I bet the reader can guess the error that 30% of the students are still making on 8b. Yes, ignoring the difference between congruence and equality they try to directly apply an algebraic rule to the congruence $\overline{NY} \cong \overline{RO}$ without first converting it to an equality.

Concluding Remarks about Geometry

We have looked at some 9 common errors in geometry and found that all but one of them can be explained as a analogy to a close-by example. The majority(5) of these analogy errors turn on subjects failing to block the analogy by observing the distinction between equality and congruence. It should be stressed that this confusion is well known in high school geometry and is not a product of the tutor. In fact, it is to the tutor's credit that it eventually gets students to respect the difference between equality and congruence. Many high school teachers have just given in and do not require students to make the distinction. The reader may in fact wonder why there is such a distinction. It turns out that congruence and equality of measure become distinct notions when we get to more complex figures like triangles. Whether this really justifies enforcing the distinction in the case of segments is an issue for mathematicians and mathematics educators.

Errors with the Algebra Tutor

The algebra tutor currently teaches a course which reviews pre-algebra (fractions, signed numbers, distribution), solving linear equations in one unknown, polynomials, and solving quadratics. We have analyzed the 9 errors that occurred above the 30% threshhold in the pre-algebra curriculum. Interactions with the algebra tutor involve selecting an operator, passing its arguments and then producing a result. Most of our high-frequency errors involve calculating the result. Almost always it appears that the error can be explained as an analogy to the preceding problem. Therefore, we will present our results in terms of correct answer on the preceding problem and then common error on the current problem.

Reciprocal

One of the early prealgebra skills that students are taught involves finding the reciprocal of a fraction. The following two are the key problems

correct prior: reciprocal (-3/4) = -4/3
common error: reciprocal (1/4) = 4/1 (rather than 4)

53% of the students make this error although they have been told to enter their answer in simplest possible form. The analogical source of this error is apparent. Again, it is an interesting question in just what sense this is an error. However, our task is not to judge the rules of the game but to observe where students deviate from these rules.

Greatest Common Factor

The next operation to produce a high frequency common error involves finding the greatest common factor (GCF) of two numbers

correct prior: GCF (44, 28) = 4
commor error: GCF (81, 54) = 9 (rather than 27)

36% of the students make this error. In the prior examples when the two numbers have a common factor in the multiplication table (44 = 4 X 11; 28 = 4 X 7) the common

factor is the greatest common factor. Subjects appear to be extrapolating this erroneous pattern to the current case and coming up with 9 since 81 = 9 X 9 and 54 = 9 X 6.

Fraction Addition

Another pre-algebra skill reviewed by the tutor is fractional arithmetic. The example that produces the high error rate is illustrated below:

correct prior: ADD (3/28, 5/7) = 23/28
common error: ADD (1/21, 7/6) = 51/42 (rather than 17/14)

42% of the students make this error. This is the first case of an example from fraction addition where the result needed simplification. We assume that by analogy to the previous problems some subjects were deleting the simplification step when it was needed.

Variable Combination

Another lesson involves representing the product of a constant and a variable. This produces the two problem sequence illustrated below:

correct prior: Varcombine (-10, X) = -10X
common error: Varcombine (-1, Y) = -1Y (rather than -Y)

53% of the students make this error. This is the first case of combining a variable with 1 or -1.

From this example students learn the special case rule involving 1 and apply it correctly until they come across the following pair which involves multiplying parenthesized expressions by constants.

correct prior: Varcombine (-31, (-1X + 17)) = -31(-1X + 17)
common error: Varcombine (-1, (4/5Z - 3)) = -1(4/5Z - 3) (rather than -(4/5Z - 3))

43% of the students made this error. Perhaps the reader can see what happened here. By accident the prior example had gotten into the tutor in a form where the special case rule for 1 was violated. Many students promptly copied this pattern for the next

example. This is probably the case where the student's behavior least deserves the classification "error." However, for our purposes it reinforces how much of the students' behavior is analogically based.

Factor

The next operator that produces a common error involves factoring out a common product. It is illustrated below:

correct prior: factor (5 * 3X + 5 * 1) = 5(3X +1)
common error: factor (5 * 3Z + 5) = 5(3Z +5) (rather than 5(3Z + 1))

35% of the students make this error. This sequence of two was an informally constructed sequence designed to get students to extract the pattern by analogy. However, in making the mapping from the prior example to the target problem subjects map both the 5 and the 1 into the 5 producing the error observed.

Expandexpression

Expandexpression is an operator which rewrites an expression in terms of products involving a specific term. It produces the sequence illustrated below:

correct prior: Expandexpression (5 + 20X, 5) = 5 * 1 + 5 * 4X
common error: Expandexpression (18 + 6Y, 6) = 6 * 3 + 6 * 1Y
(rather than 6 * 3 + 6 * Y)

35% of the students make this error. This is another case of subjects failing to take into account the special case nature of 1 in their analogies.

Factor

The next operator to produce a consistent error pattern involves factoring a sum. This error pattern involves choosing a wrong suboperator rather than a wrong result. It is illustrated below:

prior example: Factor (-14Z + 7)
GCF (-14,7) = 7

Expandexpression (-14Z + Z, 7)
common error: Factor (-6 - 8 X)
GCF (-6, - 8) = -2
Expandexpression (-6 - 8X, 2)
(rather than Expandexpression (-6 - 8X, -2))

The first step in factoring is to find the greatest common factor of the two integers and the second step is to use expandexpression with the expression and the greatest common factor. The problem that produces the error is the first case where the greatest common factor is negative. 50% of the students choose to pass a positive integer as the argument to expandexpresion by analogy to the previous problems.

Adding Terms

The final problem to produce a consistent error pattern is given below:

prior example: ADD (-3/2Z, 5/2Z) = Z
common error: ADD (2X, -2X) = X rather than 0

38% of the students make the error illustrated. It seems unlikely that this is just an analogy error. This error has been analyzed by other researchers as a failure to properly parse the problem (Matz, 1982). Students analyze the 2 and the -2 cancelling each other leaving the X. Their problem is that they do not know how to combine the 0 that results and the X. This is not a problem due to analogy to the prior example.

Summary

Table 1 presents a summary of our analysis of the high frequency errors in the three domains. Clearly there is a preponderance of analogical errors. This classification was necessarily informal given that the PUPS analogy mechanism could not make some of the proposed analogies. Given this informal nature, there is room for disputing particular analyses of particular errors. However, such disputes would not take away from the conclusions of a heavy proportion of analogical errors. On the other hand, in absolute

frequency such errors are rare. This suggests that in most cases where analogy is at work it produces the right result.

Insert Table 1 about here

Einstellung

The reader may have noted the similarity between the errors we are observing and those errors which have been called Einstellung errors (Luchins, 1942). This is the phenomenon that when students have had a run of success with a particular solution pattern they are likely to try to repeat that pattern on a problem where it is no longer appropriate. In fact, in the PUPS theory, Einstellung errors are to be attributed to choosing inappropriate examples for analogy. This contrasts with the explanation that had been offered in ACT* and by others (eg. Lewis, 1978) that saw Einstellung errors as being produced by composing together sequences of production rules. The problem with this explanation has always been that the tendency to make this error is very much under conscious control. For instance, Luchins was able to cut these down by 50% just with the admonition "Don't be blind". The demonstration also never works in my class if I introduce it as an example of where people get tricked in problem solving. Composed production rules are not the sorts of things in most theories which are subject to such conscious control. On the other hand, it is perfectly plausible that subjects interpreted Luchins' "Don't be blind" instructions as instructions not to us the obvious recent example pattern which had been working.

Comparison's with Van Lehn

As a final point it is worth comparing what I am saying here with Van Lehn's analysis of errors. He has produced two theorues of the origins of bugs. The earlier is the repair theory that he developed with Brown (Brown & Van Lehn, 1980) and the more

recent is the step theory he developed for his dissertation (Lehn, 1983). These are complimentary hypotheses not alternative hypotheses. Step theory attributes bugs to an inductive process of learning from examples while repair theory attributes bugs to repairs that students invent when they come to impasses in their problem-solving.

Van Lehn has already compared my analogy-based explanation with inductive-based explanation of step theory. He comments "Although I have not investigated example-exercise analogy in detail, I expect it to behave indistinguishably from learning by generalizing examples" (Van Lehn, 1985, p. 19). I think he is right in that there is no difference between analogy from examples generally considered and induction from examples generally considered. It is the case that PUPS and his step theory are not identical, but then I think we both admit that our theories are not sufficient to produce the full class of inductive/analogical errors. The one thing that the analogy perspective of the PUPS theory emphasizes is that one should be able to observe students making mappings between examples and problems and learning from these mappings. A dominant feature of protocols from students learning is the presence of these analogical mappings.

On the other hand, repair theory does contrast with the predictions of PUPS analogy. It is interesting to look at the domain most associated with Van Lehn, subtraction bugs, and classify those according to whether they seem to have an analogical explanation or a repair explanation. There are bugs which Brown and Van Lehn cannot explain with their repair model but which are naturally explained as analogy errors due to wrong selection of an example. There are errors which can be explained either way. There are errors for which the Brown and Van Lehn explanation seems much more plausible than any analogy-to-wrong-example explanation I have been able to think of. Finally, there are errors which neither model can explain. I will discuss examples of each.

Add instead of subtract

The error of addition instead of subtraction has an obvious analogical explanation where the student relaxes the constraint the sign and uses an addition example as an analogy for a subtraction example. On the other hand, it is not possible to explain this bug as a repair. I think this error is particularly plausibly explained in analogical terms because it is known that children who can do subtraction perfectly when presented with a sheet of all subtraction problems will make add-instead-of-subtract errors when those subtraction problems are intermixed with addition problems. Such mixed addition and subtraction problems provide students with erroneous analogs in close proximity to the subtraction problems.

Carry instead of borrow

Equally explainable in terms of analogy is the error where students perform subtraction correctly except that they carry when they should borrow. Again it cannot be explained by repair theory.

Smaller-from-larger

Certainly the most common subtraction bug, and one which I have anguished much over with my son, is the smaller from larger bug illustrated below:

```
  93                                  7
- 37  which has the obvious analog  - 3
 ---                                 --
  64                                  4
```

In PUPS terms one can represent the problem of finding the digit in the units column as:

Goal: isa number
function (difference 3 7)
form ????

and the example:

example: isa number
function (difference 7 3)
form (text 4)

Cast this way there is actually a problem explaining the error in PUPS because PUPS respects the order of the arguments to a relation like difference and will not make the analogy. Thus, the feature that has to be relaxed here is the argument order. However, despite the fact that PUPS will not, it is more than plausible that children are willing to relax this order. Most kids and I suspect most adults (certainly the ones I have tested), when asked "what the difference between 3 and 7" will respond 4, not -4 or impossible. In fact, memory of examples where it heard "the difference between 3 and 7 is 4" could serve for PUPS as the analogs that would allow it to make the argument reversal.

The Brown and Van Lehn explanation of this is also that students reverse the order of the arguments when they hit an impasse. The difference is that they do not tie it to any specific example of subtraction. Presumably, the two points of view could be separated by careful experimental data.

Borrow-no-decrement

This error is illustrated below:

```
  62
- 44
  --
  28
```

This is produced in repair theory by deleting the decrement rule. It would be produced in PUPS by not encoding the decrement in an example and hence not learning the rule. This is really a very subtle difference and, if the two accounts could be separated at all, it would require taking protocols to find out what students attend to in the

examples they learn from.

Stutter-Subtract

When there are blanks in the bottom number. the student subtract the leftmost digit of the bottom number in every column that has blank. This is illustrated below:

```
  4369                    539
-   22  by analogy to   - 122
------                  -----
  2147                    417
```

We can represent the goal as follows:

```
Goal:  isa number
       function (position third answerrow)

three: isa number
       function (position third toprow)
                (closest goal toprow)

 two:  isa number
       function (position second bottomrow)
                (closest goal bottomrow)
```

We can represent the example as follows:

```
Example: isa number
         function (position third answerrow)
                  (difference five one)

  five:  isa number
         function (position third toprow)
                  (closest example toprow)

   one:  isa number
         function (position third bottomrow)
                  (closest example bottomrow)
```

The example allows the following rule to be extracted which can apply to the example to set the goal of subtracting 2 from 4.

```
IF   =goal: isa number
            function (position =n answerrow)
THEN =goal: function (difference =num1 =num2)
```

```
=num1   isa number
        function (closest =goal toprow)
=num2   isa number
        function (closest =goal bottomrow)
```

The repair in Brown and Van Lehn which produces this error is one where students move their attention to the right when they hit a blank. Again short of taking protocols at the origin of the error it seems hard to separate the two accounts.

Zero istead of borrow

This error is illustrated below:

```
   42                          42
 - 16  could be analogy to   - 12
   ----                        ----
   30                          30
```

However, I think it is implausible to suppose there is such an analog in the environment. The standard repair explanation, that the student starts to count down from 2 and hits zero, is much more plausible. I think if the student were going to look for an analog to deal with this dilemna, the smaller-from-larger bug would be produced.

Thus, this is one example of a number where I find the repair explanation definitely more compelling.

Subtract-top-from-bottom

Just to document an error that neither analysis can handle well consider the following subtraction error which apparently has been documented in at least one kid's behavior:

```
   81
 - 27
   ----
   46
```

In this error the student chooses to subtract the top number from the bottom. Thus 6 written as the difference of 7 and 1. The student borrows a mysterious 1 to convert the

2 to 12 and the subtracts 8 from it to get 4. Apparently, this error has stumped all attempts at explanation.

Conclusions

Although I hate chapters that end with calls for more research, it is essential to end on such a note. This chapter has really been a plausibility argument. We need to extend the PUPS simulation to establish that it can generate the full class of analogy errors. More important, once it does, we need to expose it to the full curriculum that students see to determine if there are over-generation problems. That is, will the PUPS analogy mechanism produce errors that we do not see in students protocols? The point of such an exercise is not so much to establish that PUPS per se is correct but to establish more precisely the sense in which these are analogy errors.

Figure Captions

Figure 1 Decrease on errors with practice: Mean number of errors per step in the LISP tutor. Data is plotted for three separate lessons and averaged across the lessons. Note maximum number of errors per step is 3.

Figure 2 Decrease in time to perform a step of problem solving: (a) LISP tutor and (b) geometry tutor.

Figure 3 The first three problems in the geometry tutor. (a) and (b) are problems that cause few errors. (c) produces the high frequency error.

Figure 4 In this problem, students try to apply a rule to $m\overline{DC} + m\overline{CA} = m\overline{ED} + m\overline{DC}$ rather than first establishing $m\ \overline{DC} = m\overline{DC}$ by the reflexive postulate.

Figure 5 A problem that produces two high frequency errors in the geometry tutor.

Figure 6 In this problem, students fail to convert $\overline{HN} \cong \overline{SM}$ into mHN = mSM before applying the substitution rule.

Figure 7 Subjects tend to combine the two equalities by transitivity and conclude the congruence statement.

Figure 8 (a) Most students conclude mAB + mBC = mAC and mBC + mCD = mBD and then try unsuccessfully to use these conclusions as premises for a rule. Students make an error on (b) by using (a) as an analogy. They fail to establish congruence as a separate step from equality of measure.

Table 1

Classification of

High Frequency Errors

LISP	6	3
Geometry	8	1
Algebra	8	1
Total	22	5

References

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

Anderson, J. R., & Jeffries, R. (1985). Novice LISP errors: Undetected losses of information from working memory. *Human-Computer Interaction, 22,* 403-423.

Anderson, J. R., & Thompson, R. (in press). Use of analogy in a production system architecture. In A. Ortony, et al. (Eds.), *Similarity and analogy.* .

Anderson, J. R., Boyle, C. F., & Yost, G. (1985). The geometry tutor. In *Proceedings of International Joint Conference on Artificial Intelligence-85, pp. 1-7.* Los Angeles: International Joint Conference on Artificial Intelligence.

Anderson, J. R., Farrell, R., & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science, 8,* 87-129.

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

Burton, R. R., & Brown, J. S. (1982). An investigation of computer coaching for informal learning activities. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems, pp. 79-98.* New York: Academic Press.

Chase, W. G., & Ericsson, K. A. (1982). Skill and working memory. In G. H. Bower (Ed.), *The psychology of learning and motivation.* New York: Academic Press.

Lewis, C. H. (1978). *Production system models of practice effects.* Doctoral dissertation, University of Michigan, Ann Arbor, Ph.D. Dissertation.

Lewis, C. H. (1986). Understanding what's happening in system interactions. In D. A. Norman & S. W. Draper (Eds.), *User centered system design: New perspectives in human-computer interaction.* Hillsdale, NJ: Erlbaum.

Lewis, M. W., Milson, R., & Anderson, J. R. (1987). Designing an intelligent authoring system for high school mathematics ICAI: The Teachers Apprentice Project. In Greg

Kearsley (Ed.), *Artificial intelligence and instruction: Applications and methods*. Addison-Wesley.

Luchins, A. S. (1942). Mechanization in problem solving. *Psychological Monographs, 54*, No. 248.

Matz, M. (1982). Towards a process model for high school algebra. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems*. New York: Academic Press.

McCloskey, M. (1983). Intuitive physics. *Scientific American, 24*, 122-130.

Reiser, B. J., Anderson, J. R., & Farrell, R. G. (1985). Dynamic student modelling in an intelligent tutor for LISP programming. In *Proceedings of International Joint Conference on Artificial Intelligence-85, pp. 8-14.* Los Angeles: International Joint Conference on Artificial Intelligence.

Ross, B. H. (1984). Remindings and their effects in learning a cognitive skill. *Cognitive Psychology, 16*, 371-416.

Shultz, T. R. (1982). Rules of causal attribution. *Monographs of the Society for Research in Child Development*, Vol. 47(1, Serial No. 194).

Siegler, R. S. (1976). The effects of simple necessity and sufficiency relationships in children's causal inferences. *Child Development, 47*, 1058-1063.

VanLehn, K. (1983). *Felicity conditions for human skill acquisition: Validating an AI-based theory* (Tech. Rep. CIS-21). Palo Alto, CA: Xerox Parc.

Van Lehn, K. A. (1985). *Arithmetic procedures are induced from examples* (Tech. Rep. ISL-12). Palo Alto, CA: Xerox Palo Alto Research Center.

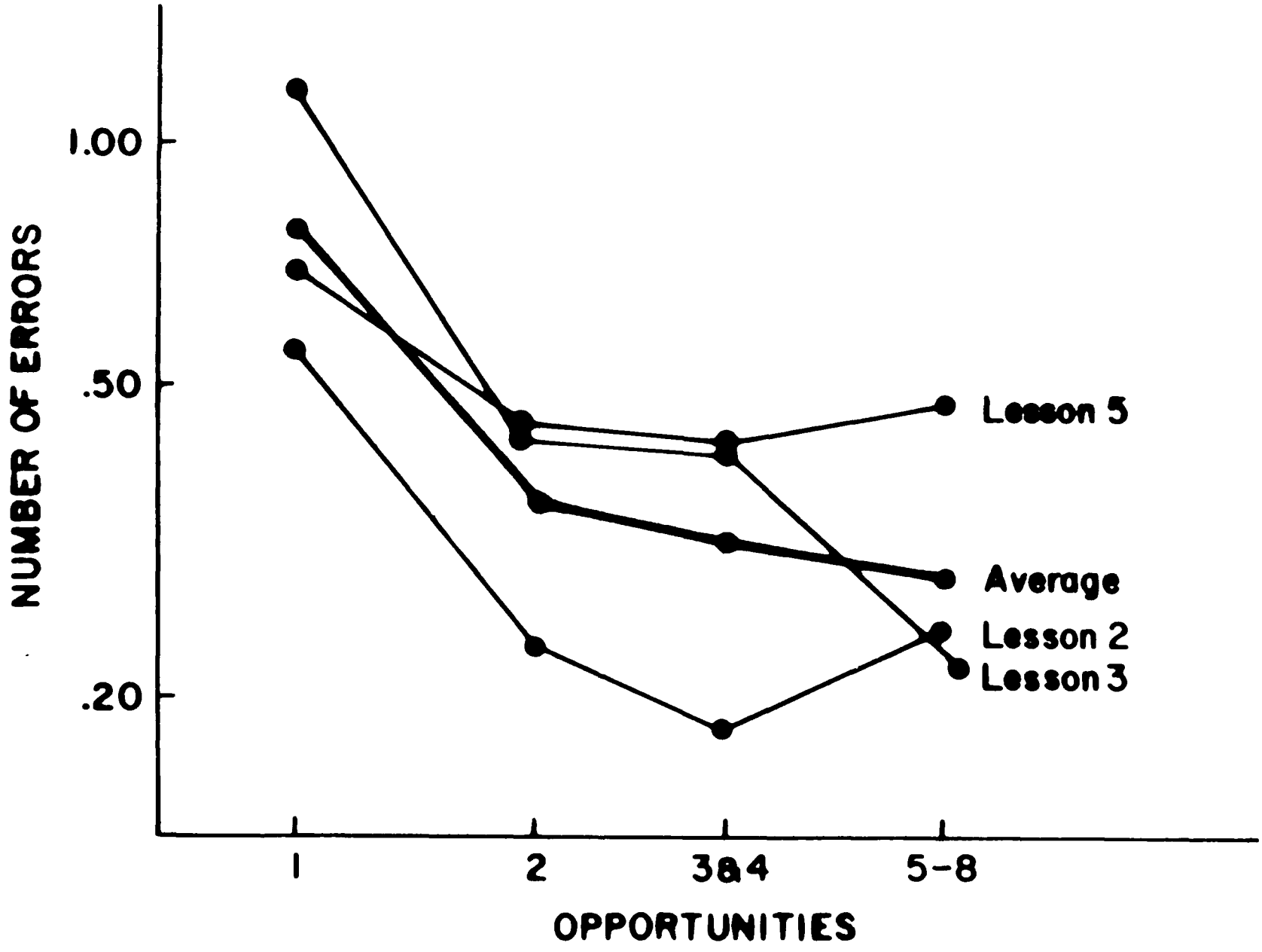

NUMBER OF ERRORS
1.00
.50
.20
Lesson 5
Average
Lesson 2
Lesson 3
1
2
3&4
5-8
OPPORTUNITIES

Figure 1

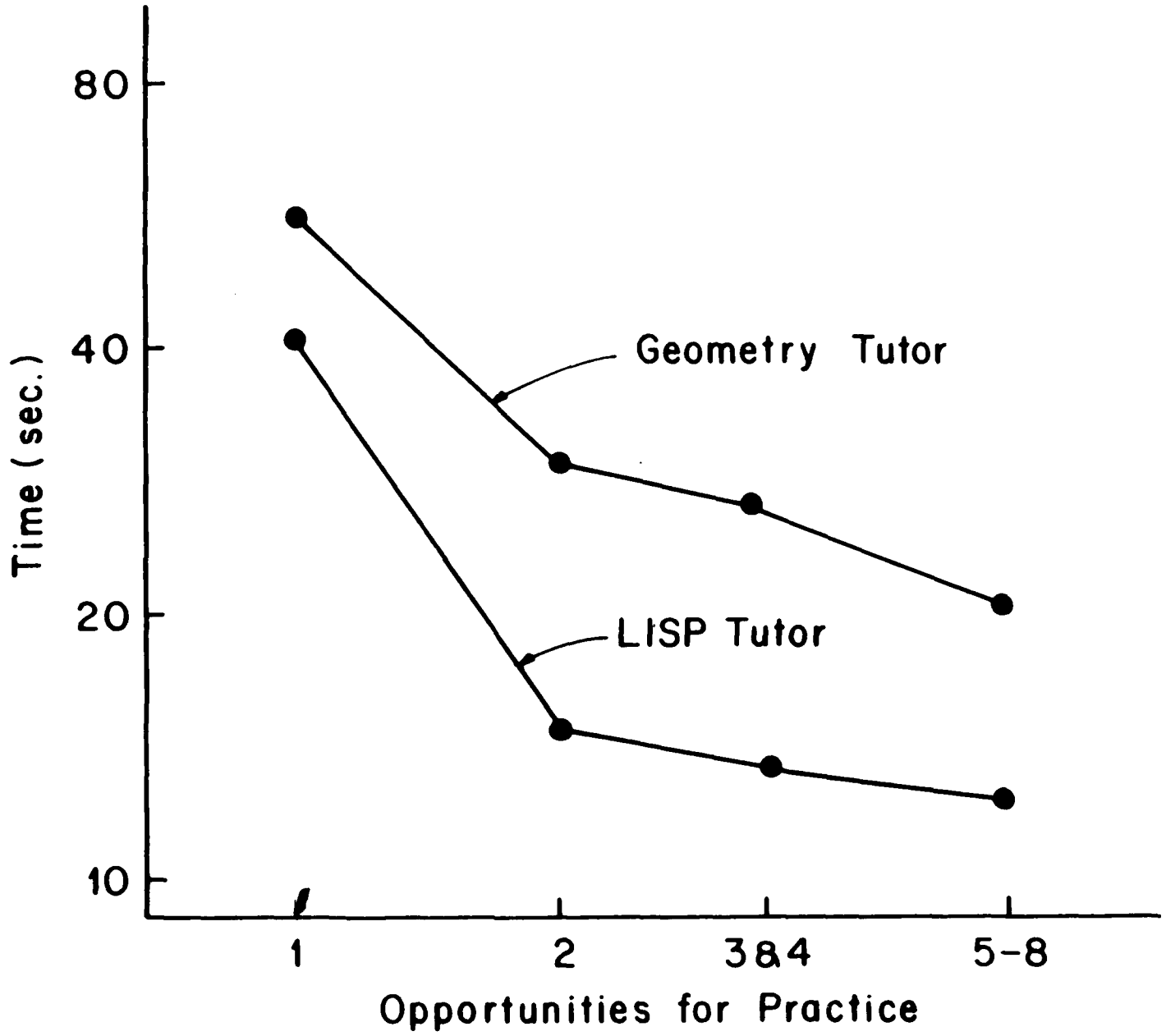

80
40
20
10
Time (sec.)
Geometry Tutor
LISP Tutor
1
2
3&4
5-8
Opportunities for Practice

Fig. 2

(a)

Prove: $12 = m\overline{DC} + m\overline{CA}$

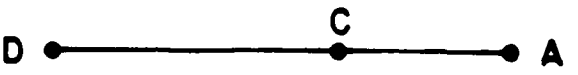
Given:

$5 = m\overline{CA}$

$7 = m\overline{DC}$

(b)

Prove: $m\overline{DC} + m\overline{CA} = m\overline{GF} + m\overline{FE}$



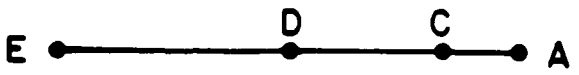
Given:

$m\overline{DC} = m\overline{FE}$

$m\overline{CA} = m\overline{GF}$

(c)

Prove: $24 = m\overline{ED} + m\overline{DC} + m\overline{CA}$



Given:

$3 = m\overline{CA}$

$9 = m\overline{DC}$

$12 = m\overline{ED}$

Fig. 3

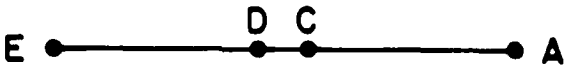
Prove: $m\overline{CA} = m\overline{ED}$

Given:

$$m\overline{DC} + m\overline{CA} = m\overline{ED} + m\overline{DC}$$

Fig. 4

Prove: $\overline{CA} \cong \overline{ED}$

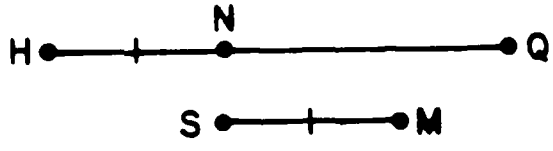


Given:

$m\overline{DC} + m\overline{CA} = m\overline{ED} + m\overline{DC}$

Fig. 5

Prove: $12 = m\overline{SM} + m\overline{NQ}$



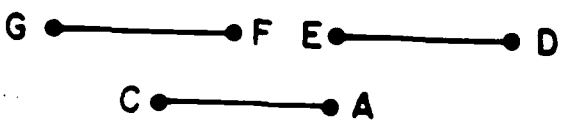
Given:

$\overline{HN} \cong \overline{SM}$

$12 = m\overline{NQ} + m\overline{HN}$

Fig 6

Prove: $\overline{CA} \cong \overline{GF}$



Given:

$m\overline{CA} = m\overline{ED}$

$m\overline{ED} = m\overline{GF}$

Fig. 7

(a)

Prove: $m\overline{AC} = m\overline{BD}$

Given:

$m\overline{AB} = m\overline{CD}$

(b)

Prove: $\overline{OY} \cong \overline{RN}$

Given:

$\overline{NY} \cong \overline{RO}$

Fig. 9