FOUNDATIONS OF KNOWLEDGE ACQUISITION: Cognitive Models of Complex Learning

THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE

OFFICE OF NAVAL RESEARCH
Advanced Book Series

Consulting Editor

André M. van Tilborg

Other titles in the series:

Foundations of Knowledge Acquisition: Machine Learning, edited by Alan L. Meyrowitz and Susan Chipman
ISBN: 0-7923-9278-7

Foundations of Real-Time Computing: Formal Specifications and Methods, edited by André M. van Tilborg and Gary M. Koob
ISBN: 0-7923-9167-5

Foundations of Real-Time Computing: Scheduling and Resource Management, edited by André M. van Tilborg and Gary M. Koob
ISBN: 0-7923-9166-7

FOUNDATIONS OF KNOWLEDGE ACQUISITION: Cognitive Models of Complex Learning

edited
by

Susan Chipman
Office of Naval Research

Alan L. Meyrowitz
Naval Research Laboratory

SPRINGER SCIENCE+BUSINESS MEDIA, LLC

Library of Congress Cataloging-in-Publication Data

Foundations of knowledge acquisition / edited by Susan Chipman, Alan L. Meyrowitz
p. cm. -- (The Kluwer international series in engineering and computer science ; SECS 194)
Includes bibliographical references and index.
Contents: v. [1] Cognitive models of complex learning.
ISBN 978-1-4613-6390-3 ISBN 978-1-4615-3172-2 (eBook)
DOI 10.1007/978-1-4615-3172-2
1. Knowledge acquisition (Expert systems) I. Chipman, Susan. II. Meyrowitz, Alan Lester. III. Series.
QA76.76.E95F68 1993
006.3'1--dc20 92-36720
CIP

Printed on acid-free paper.

TABLE OF CONTENTS

Foreword

One of the most intriguing questions about the new computer technology that has appeared over the past few decades is whether we humans will ever be able to make computers learn. As is painfully obvious to even the most casual computer user, most current computers do not. Yet if we could devise learning techniques that enable computers to routinely improve their performance through experience, the impact would be enormous. The result would be an explosion of new computer applications that would suddenly become economically feasible (e.g., personalized computer assistants that automatically tune themselves to the needs of individual users), and a dramatic improvement in the quality of current computer applications (e.g., imagine an airline scheduling program that improves its scheduling method based on analyzing past delays). And while the potential economic impact of successful learning methods is sufficient reason to invest in research into machine learning, there is a second significant reason: studying machine learning helps us understand our own human learning abilities and disabilities, leading to the possibility of improved methods in education.

While many open questions remain about the methods by which machines and humans might learn, significant progress has been made. For example, learning systems have been demonstrated for tasks such as learning how to drive a vehicle along a roadway (one has successfully driven at 55 mph for 20 miles on a public highway), for learning to evaluate financial loan applications (such systems are now in commercial use), and for learning to recognize human speech (today's top speech recognition systems all employ learning methods). At the same time, a theoretical understanding of learning has begun to appear. For example, we now can place theoretical bounds on the amount of training data a learner must observe in order to reduce its risk of choosing an incorrect hypothesis below some desired threshold. And an improved understanding of human learning is beginning to emerge alongside our improved understanding of machine learning. For example, we now have models of how human novices learn to become experts at various tasks -- models that have been implemented as precise computer programs, and that generate traces very much like those observed in human protocols.

The book you are holding describes a variety of these new results. This work has been pursued under research funding from the Office of Naval Research (ONR) during the time that the editors of this book managed an Accelerated Research Initiative in this area. While several government and private organizations have been important in supporting machine learning research, this ONR effort stands out in particular for its farsighted vision in selecting research topics. During a period when much funding for basic research was being rechanneled to shorter-term development and demonstration projects, ONR had the vision to continue its tradition of supporting research of fundamental long-range significance. The results represent real progress on central problems of machine learning. I encourage you to explore them for yourself in the following chapters.

Tom Mitchell
Carnegie Mellon University

Preface

The two volumes of *Foundations of Knowledge Acquisition* document the recent progress of basic research in knowledge acquisition sponsored by the Office of Naval Research. This volume you are holding is subtitled: *Cognitive Models of Complex Learning*, and there is a companion volume subtitled: *Machine Learning*. Funding was provided by a five-year Accelerated Research Initiative (ARI) from 1988 through 1992, and made possible significant advances in the scientific understanding of how machines and humans can acquire new knowledge so as to exhibit improved problem-solving behavior.

Previous research in artificial intelligence had been directed at understanding the automation of reasoning required for problem solving in complex domains; consequent advances in expert system technology attest to the progress made in the area of deductive inference. However, that research also suggested that automated reasoning can serve to do more than solve a given problem. It can be utilized to infer new facts likely to be useful in tackling future problems, and it can aid in creating new problem-solving strategies. Research sponsored by the Knowledge Acquisition ARI was thus motivated by a desire to understand those reasoning processes which account for the ability of intelligent systems to learn and so improve their performance over time. Such processes can take a variety of forms, including generalization of current knowledge by induction, reasoning by analogy, and discovery (heuristically guided deduction which proceeds from first principles, or axioms). Associated with each are issues regarding the appropriate representation of knowledge to facilitate learning, and the nature of strategies appropriate for learning different kinds of knowledge in diverse domains. There are also issues of computational complexity related to theoretical bounds on what these forms of reasoning can accomplish.

Knowledge acquisition, as pursued under the ARI, was a coordinated research thrust into both machine learning and human learning. Chapters in *Cognitive Models of Complex Learning* thus include summaries of work by cognitive scientists who do computational modeling of human learning. In fact, an accomplishment of research previously sponsored by ONR's Cognitive Science Program was insight into the knowledge and skills that distinguish human novices from human experts in various domains; the Cognitive interest in the ARI was then to characterize how the transition from novice to expert actually takes place. Chapters particularly relevant to that concern are those written by Anderson, Kieras, Marshall, Ohlsson, and VanLehn.

Significant progress in machine learning is reported along a variety of fronts in the companion volume, *Machine Learning*, also published by Kluwer Academic Publishers. Included is work in analogical reasoning; induction and discovery; learning and planning; learning by competition, using genetic algorithms; and theoretical limitations.

The editors believe these to be valuable volumes from a number of perspectives. They bring together descriptions of recent and on-going research by scientists at the forefront of progress in one of the most challenging arenas of artificial intelligence and cognitive science. Moreover, those scientists were asked to comment on exciting future directions for research in their specialties, and were encouraged to reflect on the progress of science which might go beyond the confines of their particular projects.

Dr. Susan Chipman
ONR Cognitive Science Program

Dr. Alan L. Meyrowitz
Navy Center for Applied Research in Artificial Intelligence

FOUNDATIONS OF KNOWLEDGE ACQUISITION: Cognitive Models of Complex Learning

Acquisition of LISP Programming Skill[1]

John R. Anderson and Albert T. Corbett

Department of Psychology
Carnegie Mellon University
Pittsburgh, PA 15213-3890

Abstract

The acquisition of a complex skill like writing LISP code can be decomposed into the learning of an underlying set of production rules. Each production rule appears to have a simple learning trajectory and its acquisition is independent of the acquisition of other production rules. These basic results do not appear to be dependent on the instructional modality under which one learns. Learning under a more directive modality only reduces the amount of time students spend debugging their errors.

INTRODUCTION

This chapter provides a review of our work on the acquisition of LISP programming skill in the context of various intelligent tutoring systems we have developed. This work can be seen as a general test of the ACT* (Anderson, 1983) theory of skill acquisition. That theory of skill acquisition is actually quite simple. It asserts that

(1) The knowledge underlying a skill begins in declarative form. Knowledge in this form must be interpreted to produce performance. The initial declarative knowledge commonly takes the form of an encoding of illustrative examples of the skill. These examples are used by analogy to guide the problem solving. For example, when first learning how to perform arithmetic in LISP, the student may encode that the sequence:

(* 27 32)

[1]This research was supported by contracts N00014-87-K-0103 and N00014-91-J-1597 from the Office of Naval Research

calculates the product of 27 and 32. The student may proceed to use this as an analog for determining how to compute the sum of 384 and 492.

(2) As a function of its interpretive use, this knowledge becomes compiled into a production rule form. In the example above, the following production might be formed:

IF the goal is to apply an arithmetic operation to X and Y
 and OP is the symbol denoting that operation
THEN enter (OP X Y).

(3) Individual production rules, once formed, acquire strength as a function of practice.

(4) Ultimately, a skill such as programming consists of hundreds of independent production rules. Performance consists of the sequential application of these rules.

The simplicity of the ACT* theory of skill acquisition is its most important theoretical claim. It certainly is a counter-intuitive claim about the nature of complex skill acquisition which appears anything but simple.

Given the importance of the claim that skill acquisition is simple under a production rule analysis, it becomes important to provide empirical evidence for this simplicity. This can be done directly by confirming regularities in the data predicted by the theory and indirectly by failing to find effects of plausible complicating factors. In the following sections, we will describe our efforts to find regularities in superficially complex data and our efforts to find complicating factors in the structure of exercises and in individual differences. The analysis of individual differences is particularly interesting. One might believe that different subjects would display differential success in mastering clusters of subskills. But while the ACT* theory allows for different rates of acquisition across subjects, it does not allow for different styles of learning. Before describing the results, it is necessary to describe the LISP Tutor, the data it generates and our methods of analyzing the data.

We have been engaged in the study of programming in LISP since 1980. Around 1983 it seemed that we had a good enough understanding of programming in LISP (reported in Anderson, Farrell, & Sauers, 1984) that we might actually use it as a basis for instruction. The essential idea was to organize instruction around the individual production rules that we feel the student should acquire. This set of production rules defines what we call the ideal student model. We try to match up in real time the student's problem-solving steps with some solution path that can be generated by this ideal production system model of the skill. This is called a model-tracing methodology. In the version of tutoring we will first describe, called immediate-feedback tutoring, whenever the student makes a problem-solving move that does not match any move allowed in the student model we immediately correct the student and force the student to make a move that matches the student model. Later in this chapter, we will describe what happens with less restrictive modes of tutoring.

AN EXAMPLE INTERACTION WITH THE LISP TUTOR

In this section we will describe an interaction with the original, immediate-feedback tutor we created. Figure 1 depicts the terminal screen at the beginning of an exercise. The screen is divided into two windows, and the problem description appears in the "tutor window" at the top of the screen. As the student types, the code appears in the "code window" at the bottom of the screen. The example exercise is drawn from Lesson 6, in which iteration is being introduced. Students are familiar with the structure of function definitions by this point, so the tutor has put up the template for a definition, filling in **defun** and the function name for the student. The symbols in angle brackets represent code components remaining for the student to supply. The tutor places the cursor over the first symbol the student needs to expand, <PARAMETERS>.

As the student works on an exercise, this tutor monitors the student's input, essentially on a symbol-by-symbol basis. As long as the student is on some reasonable solution path, the tutor remains in the background and the interface behaves much like a structured editor. The tutor expands templates for function calls, provides balancing right-parentheses for students, and advances the cursor over the remaining symbols which must be expanded. If the student makes a mistake, however, the tutor immediately provides feedback and gives the student another opportunity to type a correct symbol. When the student types another response, the feedback is replaced either by the problem description (if the response is correct) or another feedback message (if the student makes another error). The tutor will also provide a correct next step in a solution, along with an explanation if the student appears to be floundering,[2] or if the student requests an explanation.

---

[2]A student is judged to be floundering at a step in the solution if he/she repeats the same type of error three times or makes two mistakes that the tutor does not recognize.

Define a function called "create-list" that accepts one argument, which must be a positive integer. This function returns a list of all the integers between 1 and the value of the argument, in ascending order. For example,

(create-list 8) returns (1 2 3 4 5 6 7 8).

You should count down in this function, so that you can just insert each new number into the front of the result variable.

CODE for create-list

```
(defun create-list <parameters>
   <process>)
```

Figure 1. The appearance of the tutor screen at the beginning of a coding problem.

Table 1 contains a record of a hypothetical student completing the code for the exercise. This table does not attempt to show the terminal screen as it actually appears at each step in the exercise. Instead, it shows an abbreviated "teletype" version of the interaction. As described above, while the student is working, the problem description generally remains in the tutor window, while the code window is being updated on a symbol-by-symbol basis. Instead of portraying each update to the code window in the interaction, the table portrays nine key "cycles" in which the tutor interrupts to communicate with the student. At each of these enumerated cycles the complete contents of the code window are shown, along with the tutor's response. The tutor's response is shown below the code to capture the temporal sequence of events; on the terminal screen, the tutor's communications would appear in the tutor window above the code. In each cycle all the code which the student has typed since the preceding key cycle is shown in boldface. However, in each case, the tutor is responding specifically to the last symbol the student typed.

The Problem Description

Define a function called "create-list" that accepts one argument, which must be a positive integer. This function returns a list of all the integers between 1 and the value of the argument, in ascending order. For example,

(create-list 8) returns (1 2 3 4 5 6 7 8).

You should count down in this function, so that you can just insert each new number into the front of the result variable.

Cycles in which the tutor communicates with the student

1. (defun create-list **(num)**
   (loop)

   You will need local variables in this function. So before you code anything else, there is a particular LISP operator you need to code.

2. (defun create-list (num)
   (let (( )
   <BODY>))

   Which of the following do you want to do?
   PRESS: IF YOU WANT TO
   1 Initialize a variable to hold the list of digits between 1 and the value of the parameter.
   2 Initialize a counter variable to keep track of how many times we go around the loop.
   3 Have the tutor choose.

Menu choice: 2

3. (defun create-list (num)
   (let ((**count 1**)
   <OTHER-INITIALIZATIONS>)
   <BODY>))

In this problem we won't want to start counting at the same value each time we call the function, so we can't initialize the counter to a constant like 1. How is the starting value being communicated to the function?

4.
```
(defun create-list (num)
    (let ((count num)
          (numlist (list num)))
      (lop))
```

Did you mean loop (y or n)? **y**

5.
```
(defun create-list (num)
    (let ((count num)
          (numlist (list num)))
      (loop
        (cond ((equal count 1) numlist))
        <LOOP-ACTIONS2>)))
```

You do want to return the variable "numlist" here, but just typing it into the code won't accomplish that. What LISP operator will allow you to return something from a LOOP?

6.
```
(defun create-list (num)
    (let ((count num)
          (numlist (list num)))
      (loop
        (cond ((equal count 1) (return numlist)))
        <LOOP-ACTIONS2>)))
```

In numeric iteration the first thing you should do after the exit test is update the counter variable. You can use the function SETQ to do this.

7.
```
(defun create-list (num)
      (let ((count)
            (numlist (list num)))
       (loop
          (cond ((equal count 1) (return numlist)))
          (setq count (1- count))
          (cons)))
```

(continued)

You've got the right idea, since you'll need to use the function "cons", but remember you are trying to insert the new value of the counter into the list. What LISP function allows you to assign the results of this function call to a local variable?

8. (defun create-list (num)
```
        (let ((count num)
            (numlist (list num)))
        (loop
            (cond ((equal count 1) (return numlist)
            (setq count (1- count)
            (setq numlist (list)))
```

If you LIST together arguments like 7 and (8) you get (7 (8)). what you want is (7 8). Try another function that combines things together.

9. (defun create-list (num)
```
        (let ((count num)
            (numlist (list num)))
          (loop
            (cond ((equal count 1) (return numlist)))
            (setq count (1- count)
            (setq numlist (cons count numlist)))))
```

-----YOU ARE DONE. TYPE NEXT TO GO ON AFTER-----
---TESTING THE FUNCTION YOU HAVE DEFINED---

THE LISP WINDOW

=> (create-list 10)

(1 2 3 4 5 6 7 8 9 10)

=> next

Table 1. A Teletype Depiction of a Hypothetical Student Completing the Code for the Function **create-list** with the Tutor.

In the first of the cycles displayed, the student has typed in the parameter list and has called **loop** in order to iterate. The tutor reminds the student that it is necessary to create some local variables before entering the loop.

In the second cycle, the student has called **let** and is about to create a local variable. The template for numeric iteration calls for two local variables in this function, so the tutor puts up a menu to clarify which variable the student is going to declare first.

In the third cycle, the student has coded an initial value which would be correct if the function were going to count up. However, this exercise is intended to give the student practice in counting down, so the tutor interrupts the student.

In the fourth cycle, the student has made a typing error which the tutor recognizes, and in the fifth cycle the student is attempting to return the correct value from the loop, but has forgotten to call **return**.

In the sixth cycle, the cursor is on the symbol <LOOP-ACTIONS2> and the student has asked the tutor for an explanation of what to do next. The tutor tells the student what the current goal is and what symbol to type next in order to accomplish the goal. In addition, the tutor puts the symbol, **setq**, into the code for the student.

In the seventh cycle, the tutor recognizes tnat the student is computing the new value for the result variable, but has forgotten that the new value must be assigned to the variable with **setq**.

In the eighth cycle, the student has gotten mixed up on the appropriate combiner function to use in updating the result variable. The tutor tries to show, by means of an example, why **list** doesn't perform quite the right operation and another combiner is needed.

Finally, in the ninth cycle, the student has completed the code.

Note that, for illustration sake, this interaction shows students making rather more errors than they usually do. Typically, the error rate is about 15 percent while it is approximately 30 percent in this dialogue.

After each exercise, the student enters a standard LISP environment called the LISP window. Students can experiment in the LISP window as they choose; the only constraint is that they successfully call the function they have just defined (which the tutor has loaded into the environment for them).

GROUP DATA ANALYSES

Analysis of Computer Records

It is worth identifying how we segment a sequence of interactions, such as those in Table 1, and assign these segments to various production rules. The data from the LISP tutor comes in as a stream of keystrokes and responses by the tutor. This data can be partitioned into cycles in which (1) the tutor sets a coding goal (i.e., places a cursor over a goal symbol on the screen); (2) the student types a unit of code corresponding to a production firing (generally a single atom or "word" of code); and (3) the tutor categorizes the input as correct or incorrect (or as a request for help) and responds accordingly. If the response is correct, the tutor will set a new goal in the next cycle. If it is incorrect, the tutor provides feedback and resets the same goal in the next cycle. If the student asks for an explanation or appears to be floundering at the goal, the tutor will provide the correct answer and set a new goal in the next cycle.

Suppose the student is coding a function called **insert-second**, and imagine that the student has just typed **cons** as the beginning of the body of the function. At this point the screen would look like this:

```
(defun insert-second (lis1 lis2)
       (cons <elem1> <elem2>))
```

In the following cycle, the tutor would place the cursor over the goal symbol <elem1>, the student would type code, for example, "(car" and when the student has typed the final space after car, the tutor would evaluate the input and respond. It is of interest here to extract two measures of production firings from this data: time and accuracy. Firing time is measured only for goals in which the student's first response is correct. The measure of firing time is the time from when the tutor is ready to accept input (cursor over <elem1> in the above example) to when the student has completed the code for that element. Two measures of firing accuracy have been extracted: (1) the probability that a student responds correctly in his/her first attempt at a goal, and (2) the number of extra attempts (cycles) required to achieve a correct answer at a goal. The second measure will be used since it proves to be more sensitive (often initial errors are just slips while repeated errors are signs of real difficulty). As a rule of thumb, the number of extra attempts is about one and a half times the number of errors.

What is happening during the period of time attributed to a production firing? It is clearly not just a single correct production rule firing. There must be an encoding of the screen, the setting of subgoals to type the individual characters, and the actual typing of these characters. Moreover, students can delete characters in order to correct mistypings, or even change their mind about the correct code unit to type. The tutor will also intervene to block syntactically illegal characters. Thus, the time for these segments will involve much more than simply the time for the target production to fire. The target production just sets the top level organization for the episode. However, it is the rule of interest, because it represents the new thing that the student must learn. Also, since typing and interacting with the tutor presumably represent skills at relatively high asymptotic levels of proficiency, learning the coding rules accounts for much of the variation in performance across segments.

Learning Curves

We will present a description of the most systematically and exhaustively analyzed data collected with the LISP tutor. This data was collected in the fall of 1985 and the spring of 1986. While there were more students in these classes, we collected complete data sets from 42 students and these were used for this analysis. There were 12 lessons in the tutor at that time. Each lesson involved students solving a sequence of problems where a problem involves writing a LISP function to do a specified task. Table 2 displays exercise solutions, for a small illustrative portion of the curriculum, the first six exercises in lesson 3. For the sake of illustration, consider how performance varied across these six problems. Figure 2 presents coding time and error rate averaged across productions for each of the six exercises. There is a notable lack of any learning trend when the data is collapsed in this fashion.

Figure 3 presents the same data aggregated according to production identity rather than problem. Note that production rules appear in different patterns across the problems. For instance, the rule that codes an inequality operator ("<" or ">") applies once each in the first and third problems and twice in the fifth, while the rule for coding **cond** appears once each in the fourth through sixth. Thus, there is a poor correlation between problem number and how many times specific productions have been practiced. Figure 3 shows the data organized according to the number of practice opportunities per production. Now we see very systematic learning trends.

Thus, we see that a production analysis can convert an apparently complex pattern of data into a very simple pattern of data. The data in Figure 3 illustrate that amount of practice of specific productions is a strong determinant of performance as expected.

```
3.1   (defun compare (num1 num2)
            (> (+ num1 10) (* num2 2)))

3.2   (defun palp (lis)
            (equal lis (reverse lis)))

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

3.4   (defun carlis (object)
            (cond ((null object) nil)
                  ((atom object) object)
                  (t (car object))))

3.5   (defun checktemp (temp)
            (cond ((> temp hightemp) 'hot)
                  ((< temp lowtemp) 'cold)
                  (t 'medium)))

3.6   (defun make-list (item)
            (cond ((null item) nil)
                  ((listp item) item)
                  (t (list item))))
```

Table 2. Initial Problems in Lesson 3.

Regression Analysis

We (Anderson, Conrad, & Corbett, 1989) have performed a fairly exhaustive analysis of the data from all 12 lessons both to better analyze the practice factor and to identify other complexity factors. Regression analyses were performed on these data in an attempt to find best predicator equations for log coding times and errors. These analyses were performed separately on "new" productions in each of the lessons (production rules introduced in that lesson) and "old" productions in each of the lessons (production rules introduced in previous lessons). There were about 100,000 observations in total, far too many for our regression program, so we collapsed across subjects, generating a mean for all observations in which different subjects applied the same production (according to the tutor's analysis) in the same serial position in the same exercise in a particular lesson (in order to fit the size constraints of our regression program). This produced about a 10 to 1 collapsing in average, yielding 6409 observations for old productions and 3350 observations for new productions.

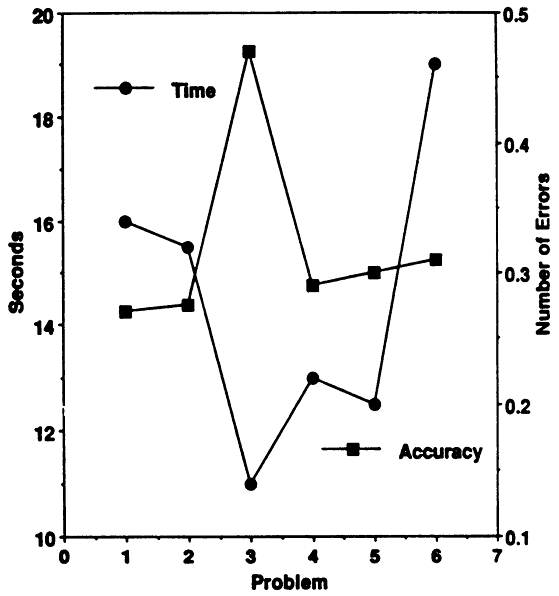


Figure 2. Mean coding time and error rate per production as a function of problem number in Lesson 3.

The following regression equations were determined as the best fitting function for new productions:

$$\log(\text{time}) = 1.35 - .03(\text{lesson number}) - .31 \log(\text{within lesson opportunity}) - .15 \log(\text{absolute position in code})$$

$$\text{mean errors} = .23 - .11 \log(\text{within lesson opportunity}) - .03 \log(\text{absolute position in code})$$

where "lesson number" is just the number 1 through 12, "within lesson opportunity" is the number of times the production had been used in the lesson up to and including the current opportunity, and "absolute position in code" is the serial position of the code in the function definition. A rather similar best fitting function was obtained for the old productions:

log(time) = 1.31 - .01(lesson number) - .25 log(within lesson opportunity)
- .26 log(absolute position in code)

mean errors = .16 - .09 log(within lesson opportunity)
- .02 log(absolute position in code)

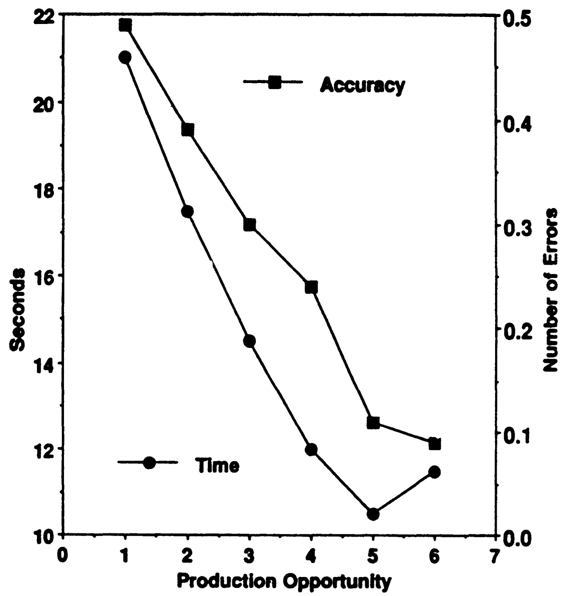


Figure 3. Mean coding time and error rate per production as a function of production opportunity in Lesson 3.

The within-lesson opportunity effect reflects the production specific practice effect and is illustrated in Figure 4. The difference between the intercepts for the old and new productions reflects the advantages subjects have when using the productions during a second or later session. It is interesting to compare the shape of the learning curves for old and new productions in Figure 4. The new productions show much larger speed up from first to second use, but after that they show similar rates of improvement. We attribute the special advantage from first to second opportunity for new productions to knowledge compilation.

The other two variables reflect "complications" to a degree, although they are not inconsistent with the theory. Indeed, the effect of absolute position in the code is a confirmation of a subtle prediction of the theory. Before discussing these two variables, however, it is worth noting the additional variables which did not prove significant when placed in competition with these variables. They included: depth of embedding of the code which was being written, number of pending goals (or unexpanded symbols to the right), left-to-right position in the pretty-printing of the code, familiarity of the concept behind the production (as rated by a panel of four judges), and number of keystrokes in typing the symbol. It is also the case that the logarithm of lesson opportunity and the logarithm of absolute position are better predictors than are untransformed scores.

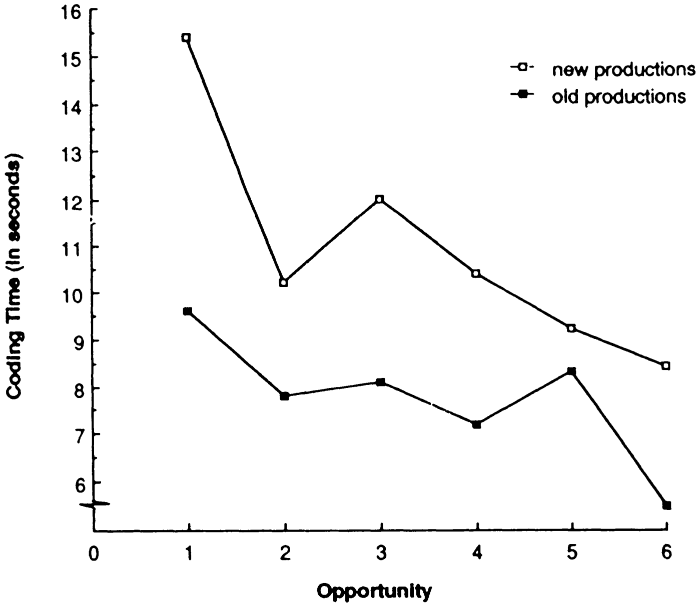


Figure 4. Mean coding time for old and new productions as a function of coding opportunity.

The effect of lesson number, while small, is quite significant for reaction times. This variable may just reflect an increased familiarity with the tutor interface. The fact that the same variable shows up for new productions (that do not appear in earlier lessons) as well as old productions suggests that this variable does not reflect production-specific practice. At least part of the phenomenon is a matter of general interface learning. It is also the case that lesson number is not significantly related to error rate. This is further evidence that the effect may be an interface effect and not reflect any real proficiency in coding.

Problem Practice and Its Interaction with Production Practice

The effect of absolute serial position in the code is interesting because it has been established that the effect is logarithmic, not linear, and not a result of potentially confounded variables such as depth of embedding, number of pending goals, or left-to-right position in a pretty-printing. It is also the case that absolute serial position is a better predictor than relative serial position or total length of the function. Figure 5 illustrates the average serial position effect over the first 33 positions. The initial long pause is at least in part due to reading the problem specifications and planning. The subsequent speed up is thought to be attributable to subjects using, and hence practicing, their problem understanding as they go through the problem. This would strengthen their declarative representation of the problem and so speed their access to it. According to the ACT* theory there should be an effect both of procedural practice and declarative practice, and the overall performance improvement should be the product of two power practice functions. Thus, the effects of procedural practice and declarative practice should be superadditive.

A more direct test of the proposed superadditivity was attempted. Productions were broken into two categories which were above or below the median use. Serial positions were similarly broken into above and below the median. Data were then classified into a 2x2 matrix according to whether the production involved was above or below the median frequency, and the serial position above or below the median frequency. This analysis was done separately for each subject. Then an analysis of variance (ANOVA) was done with three factors: subject, (42 values), production frequency, (2 values), and serial position, (2 values). Separate ANOVAs were performed for old and new productions on mean coding time. These data, as well as mean production frequency and mean serial position, are reported in Table 3. There are main effects for both factors, but critically there is an interaction between the two of them ($F_{1,41}=9.09$; $p<.01$ for new; $F_{1,41}=21.75$; $p<.001$ for old). In both cases, as predicted, the effect of production frequency is greater at lower values of serial position.

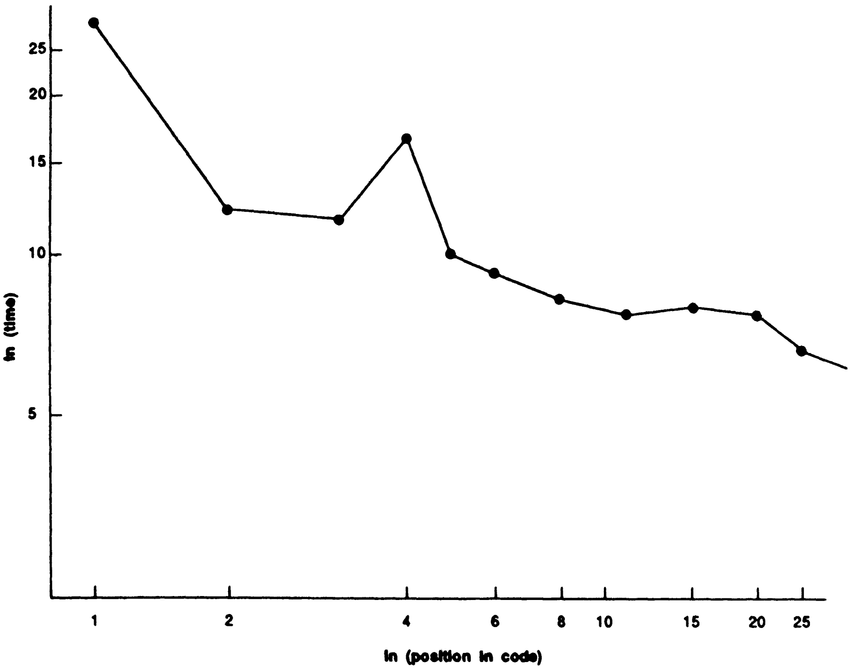


Figure 5. The effect of serial position in code on production time in Lessons 2 and 3.

INDIVIDUAL DIFFERENCES IN LEARNING

There are substantial differences among students in their performance with the LISP tutor both in error rate, time to complete exercises, and in post-test performance. The interesting question is whether there is anything more involved than some unidimensional factor of skill. In one attempt to answer this, a factor analysis was performed to see whether certain groups of subjects found certain productions categories difficult. One factor analysis was performed on the data from the spring of 1985; a second factor analysis was then performed on the combined data for fall of 1985 and spring of 1986. These two separate analyses were performed because the spring 1985 subjects worked through a different curriculum. The details of the factor analysis of the spring 1985 data and the details of the methodology are reported in Anderson (1990). In the spring 1985 data, and more clearly in the combined fall 1985 data and spring 1986 data, factors emerged within each lesson which loaded on thematically related productions. For instance, in Lesson 1 a factor emerged which loaded on arithmetic operations, and a factor emerged in Lesson 3 that loaded on logical operations. This meant, for instance, that in Lesson 1, one group of students tended to do relatively poorly on all arithmetic productions while another group of students tended to do well.

New Productions

		Serial Position Low	Serial Position High
Frequency	Low	17.3 sec 1.0 6.2	10.1 sec 2.1 16.1
	High	13.8 sec 7.9 6.3	8.5 sec 8.9 20.4

Old Productions

		Serial Position Low	Serial Position High
Frequency	Low	13.5 sec 2.0 6.8	8.9 sec 2.1 18.2
	High	9.6 sec 10.1 7.8	6.1 sec 13.3 20.6

Table 3. Results of Superadditivity Analysis. Reported in Each Cell are Mean Time, Mean Frequency, and Mean Serial Position

The initially frustrating feature of these within-lesson factors is that they did not show any across-lesson consistency. Thus, productions which loaded on one factor in one lesson would split up and load on different factors in a later lesson. To help organize these within-lesson factors, a meta-factor analysis was done. That is, students' factor scores from particular lessons were taken and a factor analysis of these was performed. Two meta-factors emerged fairly strongly in the spring 1985 data and in the combined 1985-1986 data. When the spring 1985 data was examined, it was noticed that most of the productions which loaded on factors of one of the meta-factors were new to that lesson (22 out of 34), while most of the productions that loaded on factors in the second meta-factor were old (20 out of 23). This led to a labelling of the first meta-factor as an acquisition factor and the second meta-factor as a retention factor. A similar analysis was done on the 1985-1986 data. Most of the productions associated with one meta-factor were new to that lesson (18 out of 23), while most of the productions associated with the other meta-factors were old (23 out of 31).

Thus, what seems stable across lessons are only very general learning attributes, acquisition and retention. We think we understand why thematic clusters of new productions appeared in individual lessons but disappeared thereafter. These thematically related productions were discussed in the text in close proximity. If a subject's attention waxes and wanes while reading the text, then this will produce a local correlation among thematically related productions. Another factor that would produce this thematic clustering is that many of the new productions in a lesson are thematically related. For instance, a large fraction of the productions in the third lesson on conditionals are concerned with logical operation of some sort or another. Thus, to the extent that there is an acquisition factor, within a lesson it will produce a thematic clustering of productions. Thus, the apparent themacity of productions only reflects the fact that the new productions introduced in any lesson tended to be thematically related.

There is some external validation of these two meta-factors. Although both were defined on behavior internal to the LISP tutor, both were strong predictors of performance on paper-and-pencil midterms and final exams. These factors were also associated with math SATs but not verbal SATs. The correlation of the retention factor with math SATs was .62 for spring 1985 and .38 for 1985-1986. The correlation of the acquisition factor with math SATs was .03 for spring 1985 and .60 for 1985-1986. Except for the 1985-1986 correlation coefficient for the acquisition factor, all coefficients are significant.

These factors or math SATs are equally good predictors of performance on a final paper and pencil test at the end of the course. The failure to find any consistent thematic individual differences in the LISP learning is further evidence for the view that learning is simple. The only stable individual differences are very general acquisition and retention factors. This means that the acquisition of a production rule is not sensitive to its content.

LEARNING UNDER DIFFERENT TUTORING MODALITIES

So far the data we have displayed has come from the immediate-feedback version of the tutor. The suggestion has been made a number of times that the simplicity of the learning results reflect the constraints imposed by the tutor, particularly the constraint that students remain on a correct solution path, and is not representative of learning in general. Over the past few years we have begun to explore other feedback schemes that relax this constraint. We have developed a number of environments that relax the tutor's control over feedback and over the student's behavior. These newer versions of the tutor are:

No Tutor — The student receives no instruction at all and must solve the problems on their own. All they are told is whether their final solution is correct. They can use the LISP environment to try out their solutions which provides a feedback of a sort. Essentially, these students are in the same kind of environment that students have who learn without a tutor. However, they enter their code into the same structured editor as the tutor students. This enables us to perform some analyses of their data.

Demand Feedback — Students are not interrupted by the tutor with feedback but can request feedback on their solution at any time. The tutor will tell them if their solution so far is correct and if not where the first error is. If an error is found, the tutor will provide the same feedback about the error as the immediate feedback tutor does. Our experience is that as much as 80 percent of the time students wait until they have finished their solution before they request feedback. So by their own choice they tend to turn this into a delayed feedback condition.

Flag Tutor — Whenever the student makes an error, the error is immediately highlighted by the tutor. However, students are not required to fix the error immediately, they can ignore the error and continue to code. Indeed if they can come up with a code that works despite "errors" flagged on the screen (i.e., if they generate a solution that the tutor does not recognize) they are credited with solving the problem. This is true in the no-tutor and demand-feedback tutors also. At any time the student can go back and request feedback on an error, in which case the same message is presented as would be in the immediate-feedback tutor. Our experience is that about 10 percent of the time students will ignore the error signal at least initially and continue coding, about 15 percent of the time they will request the tutor's explanation of the error signal, and the other 75 percent of the time they will try to correct their code without any information from the tutor as to the nature of their error or what the correct code would be. Thus, students tend to turn this into an immediate correction tutor but one in which they only receive minimal feedback from the tutor.

Effects of Tutoring Modalities

More information about these various tutors can be found in Corbett and Anderson (1990) and Corbett and Anderson (1989). We find that time to complete a fixed set of exercises is inversely related to the amount of influence exercised by the tutor. Students take longest in the no tutor condition and are fastest with the immediate feedback tutor. The demand-feedback tutor, and error-flagging tutor fall in between, with the demand feedback condition taking somewhat longer of the two. The effects, comparing the extremes, can be as large as a 3 to 1 difference in times. There tends to be no difference among the three tutor conditions in final achievement measured by post-tests although the no-tutor condition sometimes performs worse. This fact is very significant to our understanding of the learning process. Students in the three tutor conditions are going through very different trajectories (and taking very different amounts of time) to reach essentially the same final solutions. Students in the no-tutor condition, however, sometimes fail to reach a correct solution at all. It seems that learning is a function of the solutions students achieve and not the process by which they achieve it—students do achieve higher levels if they solve more problems under any discipline.

This pattern of results provides an important confirmation of the ACT* theory. Recall that initial production formation takes place by analogy to an example. Thus, the ingredient for learning is a product, the solved example, from which the analogy can take place. Once the production is in place, further learning is in response to a process—further use of the production leads to further strengthening. Since learning is both in response to a product (the initial compilation) and in response to a process (subsequent strengthening) it might not be obvious why ACT* predicts that the number of problems should be the critical variable. This requires going into more detail as to what happens in the ACT* theory in particular situations:

When solving their first problems in a lesson, students are in a state in which they have adequately studied the instructional examples so that they can perform about 50 percent of the new production rules without error. This 50 percent will be learned and compiled into production rules in all conditions without hitch. The compilation will occur in the same way in all conditions. The remaining 50 percent will be learned when the student comes to an understanding of one of the problems which involves that production and uses that problem as the example for a later problem. This 50 percent of the learning will therefore be based on the products of later problem solving episodes. Since, in all tutor conditions students come essentially to the same solutions and understanding of these solutions, there would be no difference in the learning involved in this 50 percent either. Once the productions are learned they are strengthened on each subsequent trial that they apply. Since the tutoring conditions only differ in how they respond to errors, strength will accumulate with correct rule applications across conditions in the same way.

The differences among conditions can have large consequences when the student makes an error and has to correct it. In conditions that provide little guidance the student can spend a lot more time finding out how to correct errors. However, according to ACT* students do not learn from errors or error correction.[3] The only thing they learn when an error is made is the final correct code. Thus, the differences among the conditions are not relevant to learning as long as students come to the same understanding of the same correct code at the end of the correction episodes. Students do not always come to the same code and there can be other subtle differences among conditions but the learning consequences of the various conditions are substantially the same and so one would not expect to see substantial differences in learning outcome.

Learning Curves

It is also of interest to consider what the learning curves are like in the various conditions. We have collected some data on this issue. A difficulty in collecting such learning curves concerns the fact that only in the immediate feedback condition are subjects guaranteed to stay on an interpretable path of behavior to which we can apply model tracing. Our solution to this dilemma has been only to analyze that fragment of the data which is on an interpretable path. As soon as a student makes an error in an exercise, we stop trying to analyze any subsequent data for that function. In the flag tutor, 87 percent of the interactions are analyzed, in the demand-feedback tutor 70 percent are analyzed, and in the no-tutor condition 67 percent are analyzed. Also, we have only the data analyzed for three of the lessons; lessons 2, 3, and 7 from the LISP tutor curriculum, and only for 10 subjects per lesson. Therefore, we have a much smaller data base than for the analyses we reported earlier.

Figure 6 presents the change in error rate as a function of condition and Figure 7 presents the change in coding time. Clearly, we are getting learning

---

[3]That is, with respect to writing correct code. They may learn how to debug their code.

functions across all tutoring modalities. In the case of error rates, there appear to be no differences as a function of tutoring modality. With respect to times, there are initial speed advantages for the two delayed feedback conditions (no tutor, demand) relative to the two immediate feedback conditions (flag, immediate). There are several reasons to believe these time differences do not reflect real learning differences. First, this may reflect the selection artifact. We only look at data on a correct path and subjects may be on a roll, so to speak, in the delayed conditions whereas in the immediate conditions we are mixing in more difficult situations. Second, subjects may be more cautious in the two immediate feedback conditions. Finally, since there is no feedback given in the two delayed feedback conditions, subjects do not have the implicit positive feedback to process. It is interesting in this regard that subjects are slowest in the flag tutor condition where they have to discriminate the type case of the feedback to detect errors which are printed in bold.

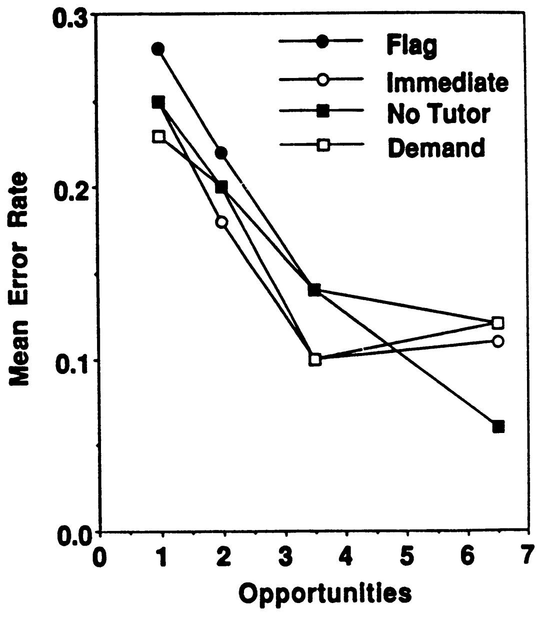


Figure 6. Learning in various tutor modalities as a function of amount of production practice: mean number of errors per coding attempt.

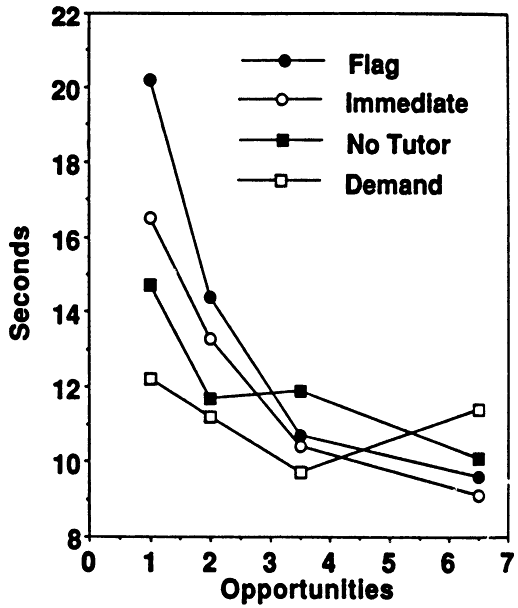


Figure 7. Learning in the various tutor modalities as a function of amount of production practice: mean time per correct coding attempt.

The basic similarity of the learning functions across tutoring conditions offers further support for the ACT* conception of the learning process. As discussed earlier, learning should be a function of the number of solutions the student has passed through and not of how the student passes through these solutions.

CONCLUSIONS

This chapter has reviewed the effects we have found in our research with the LISP tutor. Perhaps more interesting are the effects we did not find. There were no interesting effects of different content of problems, of individual differences, or of instructional style. The picture of learning a complex skill was every bit as simple as advertised at the beginning of the chapter. Learning individual production rules underlying a complex skill does not appear much different than learning simple paired associates. The resulting skill is complex because of the interrelations among the units but the learning of the units themselves is quite straight forward.

REFERENCES

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

Anderson, J. R. (1990). Analysis of student performance with the LISP tutor. In N. Fredericksen, R. Glaser, A. Lesgold, & M. Shafo (Eds.), *Diagnostic Monitoring of Skill and Knowledge Acquisition*. Hillsdale, NJ: Erlbaum, 27-50.

Anderson, J. R., Conrad, F.R. & Corbett, A.T. (1989). Skill acquisition and the LISP Tutor. *Cognitive Science, 13*, 467-506.

Anderson, J.R., Farrell, R. & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science, 8*, 87-130.

Corbett, A. T. & Anderson, J. R. (1990). The effect of feedback control on learning to program with the Lisp Tutor. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, 796-803.

Corbett, A. T. & Anderson, J. R., (1989). Feedback timing and student control in the LISP Intelligent Tutoring System, Artificial Intelligence and Education. *Proceedings of the Fourth International Conference on AI and Education*, 64-72.

Learning by explaining examples to oneself: A computational model*

Kurt VanLehn and Randolph M. Jones
Learning Research and Development Center
University of Pittsburgh

Abstract

Several investigations have found that students learn more when they explain examples to themselves while studying them. Moreover, they refer less often to the examples while solving problems, and they read less of the example each time they refer to it. These findings, collectively called the self-explanation effect, have been reproduced by our cognitive simulation program, Cascade. Moreover, when Cascade is forced to explain exactly the parts of the examples that a subject explains, then it predicts most (60 to 90%) of the behavior that the subject exhibits during subsequent problem solving. Cascade has two kinds of learning. It learns new rules of physics (the task domain used in the human data modeled) by resolving impasses with reasoning based on overly-general, non-domain knowledge. It acquires procedural competence by storing its derivations of problem solutions and using them as analogs to guide its search for solutions to novel problems.

THE TWO MAJOR OBJECTIVES OF THE CASCADE PROJECT

As Tom Dietterich pointed out in the keynote address of the 1990 Machine Learning Conference, one of the biggest challenges in machine

*This research was supported by the Cognitive Science division of the Office of Naval Research under contract N00014-88-K-0086 and the Information Sciences division of the Office of Naval Research under contract N00014-86-K-0678.

learning is to get machines to learn from ordinary instructional material, such as that used to train scientists, engineers and technicians. Not only is this an exciting intellectual challenge, but it might help alleviate the notorious problem of getting expertise out of the culture of experts and into an operable form. The expert systems community recently realized that not all experts are good at explicating and explaining their knowledge, but instructors vary in quality too, so a common practice nowadays is to acquire knowledge for an expert system from an expert who is also a good instructor. Often, there are textbooks written by very good instructors. Utilizing this material requires having the kind of system that Dietterich envisioned.

The program described here, Cascade, is a direct response to Dietterich's challenge, for it can learn how to solve Newtonian mechanics problems from the same materials that undergraduates learn from. However, it is only a partial solution to the problem, because Cascade cannot read. The information in the prose parts of the textbook is given to it in a predigested form. As will be demonstrated later, this information is not as helpful in solving problems as one might think. People and Cascade acquire much of their problem solving skill by solving problems and by studying the textbook's worked example problems.

The second objective of the research presented here is to integrate and deepen the theory of skill acquisition. As theories go, the theory of cognitive skill acquisition is in its infancy. Theories range all the way from the highly integrated, nomological theories of certain natural sciences to loose collections of ideas which can be woven together to explain phenomena. The current theory of cognitive skill acquisition is in the collection-of-ideas stage. Given almost any behavior, a cognitive scientist can often string together ideas from psychology and AI that will offer at least a plausible explanation of the phenomenon. This is certainly an advance over the state of the art 25 years ago. However, the theory is not as integrated as it could be. For instance, no one has built a computational model of skill acquisition that starts as a novice and slowly becomes an expert while being trained on the same material as human students. Several models of pieces of this process have been built, including Sierra (VanLehn, 1990), Pups (Anderson & Thompson, 1989) and X (Pirolli, 1987). The reason that cognitive science has no simulated students is not just because it is technically difficult, but because we do not know which of the many ideas floating around should be woven together. Moving the theory of cognitive skill acquisition out of the collection-of-ideas stage and into a stage of integrated student simulations will require deep thought and significant new empirical work. Development of student simulations should go hand-in-hand with these empirical advances, because such simulations are the only way to demonstrate the computational coherence and empirical coverage of an integrated theory of cognitive skill acquisition. Cascade is intended to be a step further in that it incorporates new empirical evidence from a study by Chi, Bassok, Lewis, Reimann and Glaser (1989). However, Cascade is far from a complete simulation, because some important cognitive processes, such as reading, have been deliberately omitted from the model.

Interesting new educational technology may result from developing the simulated students that are required of an integrated theory of cognitive skill acquisition. For instance, a simulated student might be a valuable tool for training teachers. Simulators have been successful adjuncts in training other skills, ranging from flying airplanes to trading stocks. It may be a worthwhile investment to use simulators to train teachers in the skills of selecting material to teach, organizing it, explaining it, detecting student misconceptions and remediating them. In addition to teacher training, there are other potential applications for simulated students as well (VanLehn, 1991b).

The lack of an integrated theory prevents development of many applications, not just educational ones. The problem is that a theory that is in the collection-of-ideas stage often provides multiple or vague explanations of phenomena, which means that it can make only ambiguous or vague predictions at best. Yet many applications, such as simulation, require the theory to make unambiguous, precise predictions. Until our understanding of cognitive skill acquisition is good enough that we can make such predictions, many applications are beyond our reach.

This chapter is intended as a summary of the results so far from the Cascade project. The project has gone through three major phases. In the first phase, the program was developed and shown capable of learning Newtonian mechanics correctly (VanLehn & Jones, in press). In the second phase, the major findings from the Chi et al. study were simulated (VanLehn, Jones & Chi, in press). In the third phase, protocols of each of the 9 subjects in the Chi study were simulated individually. The third phase is ongoing, so we can present only some of the planned analyses. In particular, we evaluate the overall fit of Cascade to the protocols, which is important for seeing how well Cascade functions as a simulated student and as a knowledge acquisition system that would satisfy Dietterich's challenge. This chapter follows the historical development by first describing the Chi et al. (1989) study, then describing Cascade, then describing how Cascade accounts for the Chi et al. findings, then describing how it simulates individual subjects.

THE SELF-EXPLANATION EFFECT

One of the major open issues in cognitive skill acquisition is understanding what happens when people study examples. (An example is a problem together with a solution that is printed or demonstrated for the student.) Much research has shown that when people are given instruction consisting of theory, examples and explanations, they rely heavily on the examples (e.g., Anderson, Farrell & Saurers, 1984; Sweller & Cooper, 1985). In some cases they seem to ignore the theory and explanations, and in other cases their learning is actually retarded by them (e.g., LeFevre & Dixon, 1986; Charney, Reder, & Kusbit, 1990; Ward & Sweller, 1990). Because examples seem to do much more of the teaching than was previously thought, it is important to understand how they work.

Chi et al. (1989) took a direct approach to understanding how students study examples. They collected protocols as subjects studied examples in classical particle dynamics, the first topic in a typical first-year college physics course. Nine subjects studied the first three chapters of a college textbook, then read the prose part of a chapter on Newton's laws. They took a test on their understanding of the chapter, then studied 3 examples and solved 25 problems. Protocols were taken as they studied the examples and solved the problems. On the basis of the scores on problem solving, the subjects were divided into two groups. The 4 students with the highest scores were called the Good solvers; the 4 students with the lowest scores were called Poor solvers, and one student was not analyzed (see Chi and VanLehn, 1991, for a discussion of the subjects' backgrounds and the median-split procedure). Since the students in both groups scored the same on pre-tests, the Good solvers seemed to have learned more during the experiment. Using protocol analysis, Chi et al. attempted to find out how the Good solvers managed to learn more than the Poor solvers from the same material. They found four differences:

1. The Good solvers uttered more self-explanations as they studied examples, whereas the Poor solvers' comments were mostly paraphrases of the examples' statements.

2. All students commented frequently on whether they understood what they had just read. The Good solvers tended to say that they did not understand what they had just read, whereas the Poor solvers tended to say that they did understand. However, the Poor solvers' scores show that they understood less than the Good solvers. This indicates that the Poor solvers' self-monitoring was less accurate than the Good solvers'.

3. During problem solving, the Poor solvers tended to refer back to the examples more often than the Good solvers.

4. When the Good solvers referred to the examples, they read fewer lines than the Poor solvers. The Poor solvers tended to start at the beginning of the example and read until they found a useful line, whereas the Good solvers started reading in the middle of the example and read only one line.

Similar findings have also been observed in protocol studies of students learning Lisp (Pirolli & Bielaczyc, 1989; Bielaczyc & Recker, 1991), electrodynamics (Fergusson-Hessler & de Jong, 1990) and biology (Chi, de Leeuw, Chiu, & LaVancher, 1991). This cluster of findings is called the self-explanation effect.

THE CASCADE MODEL

A consensus has emerged in both machine learning and cognitive psychology that it is important to distinguish two kinds of learning:

- One kind of learning is responsible for getting knowledge from the environment into the mind of the agent. This is called knowledge acquisition in the cognitive skill acquisition literature and knowledge-level learning in machine learning. There are many possible learning processes, depending on the type of instructional information available in the environment and the type of knowledge to be acquired. The Cascade project, for instance, focuses on how agents can learn college physics by studying worked example exercises and solving problems.

- The other kind of learning increases the effectiveness of knowledge that is already in the mind of the agent. This is called knowledge compilation or knowledge tuning in the cognitive skills literature, and symbol-level learning in machine learning. This class of learning mechanisms includes explanation-based learning (EBL), chunking (Newell, 1990), production composition (Anderson, 1983), and many others. Some of these mechanisms provide explanations for robust findings in the skill acquisition literature (e.g., Anderson, 1987; Newell, 1990).

In order to determine whether the learning in the Chi et al. study was knowledge acquisition or knowledge compilation, and to set the stage for developing a simulated student, we began by developing a problem solver that could solve the problems in the study.[1] The solver was based on past mechanics problems solvers (Bundy et al., 1979; Larkin, 1983; Novak & Araya, 1980) as well as our informal inspection of the Chi et al. protocols. The resulting solver had 62 physics rules and a host of mathematical and common sense rules. These 62 rules became the target knowledge base. The first goal of the Cascade project was to understand when they were learned and how.

In order to find out where the target rules could be learned, two people who were not involved in the development of the knowledge base determined whether each rule was mentioned anywhere in the textbook prior to the point where the examples were introduced. There was 95% agreement between the two judges, and disagreements were settled by a third judge. They determined that only 29 of the 62 rules were mentioned in the text. The other 33 rules would have to be learned during

[1]Actually, it could solve only 23 of the 25 problems. The other two involved kinematics knowledge that we did not bother to formalize. These two problems will be ignored throughout the remainder of the chapter.

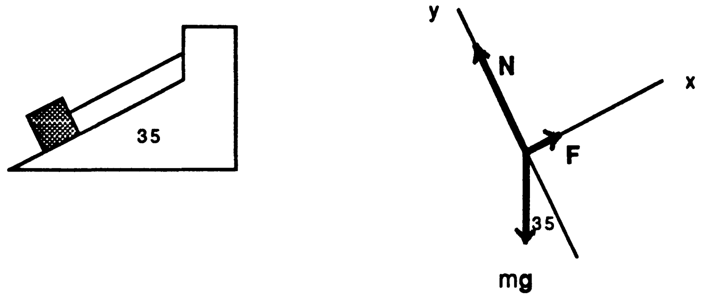
example studying or problem solving.[2] This indicates that knowledge acquisition must be going on during example studying and/or problem solving. Knowledge compilation alone would not suffice to explain the subjects' learning.

Cascade models two basic activities: explaining examples and solving problems. Knowledge acquisition goes on during both. Because the type of physics problems used in Chi's study involve only monotonic reasoning, Cascade uses a rule-based, backwards chaining theorem prover (similar to Prolog) to implement both activities. A physics example is presented to Cascade as a set of propositions representing the givens of the problem, a list of sought quantities, and the lines of the problem's solution. For instance, the example of Figure 1 is represented with the information of Table 1. Cascade explains each line by proving that it follows from the givens and the preceding lines. To solve a problem, Cascade is presented with propositions representing the problem's givens and is asked to prove a proposition of the form "The value of Q is X" for each sought quantity Q. In the process of proving the proposition, Cascade derives a value for the variable X, thus solving that part of the problem. Although this model of problem solving and example explaining is clearly too simple to cover all task domains, it suffices for physics and other task domains dominated by monotonic reasoning.

Cascade includes two kinds of analogical problem solving. Both types of analogy begin by retrieving an example and mapping the example's givens to the current problem's givens. These retrieval and mapping processes usually correspond to overt behavior. The subjects flip through the textbook pages in order to locate an example, then look back and forth between the example and problem, comparing the diagram and text that describe the example's problem with the diagram and text describing the problem they are trying to solve. This behavior generally occurs only once per problem. All the examples and most of the problems are accompanied by diagrams, and usually the subjects would search for an analogous example after looking at the diagram and before reading the problem. Thus, we think that the major process of retrieving an analogous problem is based on recalling, finding and comparing diagrams. This retrieval process was not modeled in Cascade. The sys-

[2]They could also be recalled from earlier training in physics, but there is evidence that this seldom occurred (Chi & Vanlehn, 1991; VanLehn, Jones & Chi, in press).

Problem: The figure on the left below shows a block of mass m kept at rest on a smooth plane, inclined at an angle of 35 degrees with the horizontal, by means of a string attached to the vertical wall. What are the magnitudes of the tension force and the normal force acting on the block?



Solution:
(1) We choose the block as the body.
(2) The forces acting on the block are shown in the free-body diagram on the right.
(3) Because we wish to analyze the motion of the block, we choose ALL the forces acting ON the block. Note that the block will exert forces on other bodies in its environment (the string, the earth, the surface of the incline) in accordance with the action-reaction principle: these forces, however, are not needed to determine the motion of the block because they do not act on the block.
(4) Since the block is unaccelerated, we obtain:

$$F+N+mg=0.$$

(5) It is convenient to choose the x-axis of our reference frame to be along the incline and the y-axis to be normal to the incline (see figure above, right).
(6) With this choice of coordinates, only one force, mg, must be resolved into components in solving the problem.
(7) The two scalar equations obtained by resolving mg along the x- and y-axes are:

$$F - mg \sin 35 = 0 \quad \text{and} \quad N - mg \cos 35 = 0.$$

(8) From these equations F and N can be obtained if m is given.

Figure 1: An example

Table 1: An English version of the representation of the example of Figure 1

Problem givens:

The current situation is named Ix.

Ix is a standard-gravity situation.

Block-ix is a block.

String-ix is a massless string.

Plane-ix is an frictionless inclined plane.

Block-ix slides on Plane-ix.

String-ix is tied to Block-ix.

Block-ix is at rest.

Block-ix is above Plane-ix.

String-ix is above Block-ix.

String-ix is to the right of Block-ix.

The inclination of Plane-ix is 35.

The inclination of String-ix is 35.

The mass of Block-ix is m.

Problem soughts:

The magnitude of the tension force on Block-ix due to String-ix.

The magnitude of the normal force on Block-ix due to Plane-ix.

Solution lines:

The set of bodies of Ix is Block-ix.

The set of arrows on the free-body diagram for Block-ix is {an arrow at inclination 35 pointing up, an arrow at inclination 115 pointing up, an arrow at inclination 90 pointing down}.

The set of axes on the free-body diagram for Block-ix is {an x-axis at inclination 35, a y-axis at inclination 115}.

The magnitude of the tension force on Block-ix due to String-ix is $0 - 0 + (1(mg)\sin(35))$.

The magnitude of the normal force on Block-ix due to Plane-ix is $0 - (1(mg)\cos(35)) + 0$.

tem was simply told which examples the subjects retrieved, and forced to retrieve the same ones.

One of the two kinds of analogy is used to make search control decisions. It comes into play when Cascade has two or more rules for achieving a goal and it needs to select among them. It uses the analogical mapping to see if the example's derivation has a goal that is equivalent to the goal that it is currently working on. If it finds an equivalent old goal, the rule that achieves the old goal is chosen for achieving the new goal. This type of analogy is called analogical search control, because it uses the example as a source of advice on which of several alternatives to try first. For instance, a student might say, "I cannot tell whether I should project this onto the x-axis or the y-axis. At an analogous point in the example, they projected onto the x-axis, so I'll try that too." Analogical search control is also used in the Eureka system (Jones, 1989, this volume).

The second type of analogy is used when Cascade cannot find a rule that will apply to the current goal. It uses the analogical mapping to try to find a line in an old example that it can convert into an appropriate rule. It looks for a line in the example's solution that mentions the current goal (or rather, a goal equivalent to the current goal under the mapping). Most lines are equations, so it is simple to convert a line to a temporary rule which can then be used to try to achieve the goal. For instance, a student might say, "I need some way to get the tension of string A. The example has a line saying that string 1's tension is $mg\sin(30)$. Those two strings are analogous, and 30 degrees is analogous to 45 degrees in this problem, so I bet that the tension of string A is $mg\sin(45)$." This type of analogy is called transformational analogy, after a similar method explored by Carbonell (1986). As Carbonell discovered, transformational analogies often yield wrong answers.

A major difference between the two kinds of analogy is that analogical search control refers to the rules that achieved goals during the solution of an example, whereas transformational analogy only refers to the lines of the solution. When Cascade explains an example, it stores in memory a set of triples, each of which contains the example's name, a goal and the rule that achieved it. These triples are what analogical search control searches through. If an example is not explained, then no derivation is recorded, so analogical search control cannot get any advice from that example. On the other hand, transformational analogy refers only to the solution lines. These are present regardless of whether the example is explained, since they merely represent what the student can see as they look at the page containing the example. Thus, transformational analogy can function even if the example has not been explained.

Cascade's main knowledge acquisition method is called explanation-based learning of correctness or EBLC (VanLehn, Ball & Kowalski, 1990). The basic idea is to divide knowledge into domain knowledge and non-domain knowledge. Domain knowledge represents rules that the student believes to be correct and appropriate for the task domain. Non-domain knowledge represents rules that are believed to be incorrect or relevant only to other task domains. The most important non-domain rules for learning are overly general rules. They can apply to many situations, but they often draw incorrect conclusions. For instance, a domain rule is "If there is a tension force F caused by a string S, and the tension in the string is T, then the magnitude of the tension force is also T." An overly general rule is, "If there is an entity F, with a part S, and a property of part S has value T, then a property of the entity F also has value T." This rule happens to be a generalization of the domain rule, but as argued in VanLehn and Jones (in press), not all domain rules have plausible overly general counterparts.

The basic idea of EBLC is to use overly general rules whenever domain rules fail, then save a specialization of the overly general rule as a new domain rule if all goes well. For instance, the domain rule just mentioned is learned by specialization of the overly general rule. EBLC begins when Cascade reaches an impasse that is caused by missing rules in the knowledge base. An impasse is defined to be an occasion when the current goal matches none of the known domain rules or problem givens. Impasses can be caused by missing domain knowledge or by reaching the end of a dead end path in the search space which could have been avoided by making a better search control decision earlier. Cascade explicitly checks for the latter possibility before deciding that an impasse is caused by missing knowledge. To resolve a missing-rule impasse, Cascade tries to use an overly general rule to achieve the stuck goal. If the use of such a rule ultimately leads to achieving the current top level goal (i.e., to explain a line or to find the value of a sought), then Cascade forms a new domain rule that is a specialization of the overly general one. The specialization is chosen so that it is also a generalization of the particular usage. For instance, on one problem Cascade could not determine the pressure in a part of a container even though it knew the pressure in the whole. Since there was no alternative solution to the problem using its domain rules, Cascade decided that it was at a missing-rule impasse. It applied the overly-general rule, "If an object is composed of parts, then the property values of the parts and the wholes are the same." This rule application ultimately led to a solution of the problem. Cascade then formed a new domain rule, "If a container has a part, then the pressure in the part is equal to the pressure in the whole." Thus, Cascade learned a correct rule of physics by specializing an overly general rule in order to resolve an impasse caused by missing domain knowledge.

Cascade has a second technique for learning new rules. It applies only when it is explaining an example and attempting to prove a proposition that has no variables. If it cannot prove the proposition with either domain rules or overly general rules, then it gives up and simply accepts that the proposition is true. It also builds a rule that sanctions this in future similar cases. The rules say, in essence, that if a later problem is analogous to this problem, then the analog to this proposition can be assumed true for that problem too. This type of learning is called analogical abduction.

From a machine learning point of view, Cascade does both knowledge-level learning (via EBLC and analogy abduction) and symbol-level learning (via the saving of derivations, which are used by analogical search control). EBLC and analogy abduction are both triggered by impasses, so they will often be referred to as impasse-driven learning.

As a summary, Table 2 lists Cascade's main processes. Notice that a new one has been slipped in. Cascade can be told to ignore an example line instead of self-explaining it, a trivial process labeled "acceptance" in the table.

Cascade's learning is similar to those proposed by existing theories of skill acquisition. We believe that analogical search control can eventually provide an account for the practice effects usually explained by knowledge compilation (Anderson, 1983), chunking (Newell, 1990) and other learning mechanisms. EBLC is similar to proposals by Schank (1986), Lewis (1988), Anderson (1990) and others, which also acquire new knowledge at impasses by specializing existing, overly general knowl-

Table 2: Cascade's major processes

Example studying

- Self-explanation: Prove a line via backwards chaining.
- Acceptance: Ignore the example line.

Problem Solving

- Regular problem solving: Find a value for a sought via backwards chaining. At search control choice points, use analogical search control to decide which rule to apply.
- Transformational analogy: Find a line in an example that could be adapted to achieve the current goal.

Impasse-driven learning

- Explanation-based learning of correctness (EBLC): Apply an overly general rule. If that leads to success, save a specialization as a new domain rule.
- Analogy abduction: Like transformational analogy, except a rule is built so that future occurrences of the goal will be handled the same way.

edge. Although all these models of skill acquisition are similar in spirit, they differ in significant ways. For more on the Cascade system and a detailed comparison with its predecessors, see VanLehn and Jones (in press).

MODELING THE SELF-EXPLANATION EFFECT WITH CASCADE

A simple hypothesis for explaining the four major differences between Good and Poor solvers is that Good solvers chose to explain more example lines than Poor solvers. To test this, several simulation runs were made. All these simulations began with the same initial knowledge. The initial domain knowledge consisted of the 29 physics rules that three judges found to be present in the text (see the discussion at the beginning of the preceding section). The rest of the initial knowledge base consists of 45 non-domain rules, of which 28 represented common sense physics (e.g., a taut rope tied to a object pulls on it) and 17 represented over-generalizations, such as "If there is a push or a pull on an object at a certain angle, then there is a force on the object at the same angle." See VanLehn, Jones and Chi (in press) for a list of the overly general rules.

In principle, Cascade can use regular problem solving or transformational analogy at any goal. For the sake of these experiments, we gave it a fixed strategy. It would first try regular problem solving. If that failed due to missing domain knowledge, then impasse-driven learning was applied. Transformational analogy was used only as a last resort.

The first simulation was intended to model a very good student who explains every line of every example. Cascade first explained the 3 examples in the study, then it solved the 23 problems. (The 2 problems that are not solvable by the target knowledge were excluded.) It was able to correctly solve all the problems. It acquired 23 rules: 8 while explaining examples and 15 while solving problems. All but one of the rules was learned by EBLC; analogical abduction learned the other. The new rules are correct physics knowledge, allowing for the simplicity of the knowledge representation. Moreover, they seem to have the right degree of generality in that none were applied incorrectly and none were inapplicable when they should have been applicable. However, some of the rules dealt with situations that only occurred once in this problem set, so they were never used after their acquisition.

The second simulation was intended to simulate a very poor student who does no self-explanation. Because none of example lines were explained, there was no opportunity for EBLC to learn new rules during example studying, nor were any derivations left behind for use by analogical search control during later problem solving. Cascade was given the same 23 problems given to it in the good student simulation. It correctly solved 9 problems. Apparently these problems require only knowledge from the text. As Cascade solved these problems, Cascade learned 3 correct rules via EBLC. On 6 other problems, Cascade found an incorrect solution. EBLC did not occur on these problems. On the remaining 8 problems, Cascade failed to find any solution or its search went on for so long that it was cut it off after 20 minutes. Although EBLC was used extensively on these problems, the rules produced were always incorrect. On the assumption that a poor student would not believe a rule unless it led to a correct solution to a problem, rules acquired during failed solution attempts were deleted. Thus, the poor student simulation acquired only 3 rules and solved only 9 problems correctly.

Explaining the self-explanation correlations

Cascade should be able to explain the four differences observed by Chi et al. (1989) between Good and Poor solvers. Assuming that the number of self-explanatory utterances is directly proportional to the number of lines explained during example studying, the job facing Cascade is to explain why explaining more lines causes better scores on quantitative post-tests (finding 1), more accurate self-monitoring (finding 2) and more frequent (finding 3) and more economical reference to the examples (finding 4).

The contrast between the good and poor student simulations indicates that Cascade can reproduce the positive correlation between the number of example lines explained and the number of problems solved correctly. During the good student simulation, it explained all the example lines and got all 23 problems correct; on the poor student simulation, it explained none of the example lines and got 9 of the problems correct. Knowing the operation of Cascade, it is clear that having it explain an intermediate number of lines would cause it to correctly answer an intermediate number of problems. So the two extreme points (the two simulations) plus Cascade's deterministic design are sufficient to demonstrate the main finding of the self-explanation effect.

One of the major advantages of a simulation like Cascade is that one can run it many times with different components turned off in order to ascertain why it succeeds. In particular, 20 rules were learned by the good student simulation and not by the poor. For each rule, we can find out why self-explanation allowed Cascade to learn it.

First, when more lines are explained, Cascade is more likely to stumble across a gap in its domain knowledge. Such missing knowledge causes impasses, which lead to impasse-driven learning and the acquisition of new rules during example explaining. Of the 20 rules that were learned during the good student simulation and not the poor, 8 (40%) were learned while explaining examples.

Analogical search control also aided the good student simulation's learning. When more lines are explained, more derivations become available for analogical search control. Analogical search control tends to keep Cascade on solution paths during problem solving, and this means that any impasses that occur are more likely to be due to missing domain knowledge. Thus, EBLC is more often applied to appropriate impasses, and thus more often generates correct domain rules. Of the 20 rules, 9 (45%) require analogical search control for their acquisition.

The acquisition of rules during example studying helps produce contexts during problem solving that allow EBLC to learn more rules during problem solving even without the aid of analogical search control. Of the 19 rules, 3 (15%) can be acquired during problem solving even when analogical search control is turned off. These new rules also contributed to the improvement in problem solving. Table 3 summarizes the learning of the two runs.

Cascade provides a simple explanation of the correlation between the amount of self-explanation and the accuracy of self-monitoring statements. The explanation assumes that negative self-monitoring statements (e.g., "I don't understand that") correspond to impasses, and that positive self-monitoring statements (e.g., "Ok, got that.") occur with some probability during any non-impasse situation. When more example lines are explained, there are more impasses, and hence the

Table 3: Rules learned during Good and Poor student simulations

Good	Poor	When acquired
8	0	Example studying
		Problem solving
3	3	No ex. studying rules, no analogical search control
3	0	With ex. studying rules, no analogical search control
9	0	With ex. studying rules, with analogical search control
23	3	Total

proportion of negative self-monitoring statements will be higher. In the extreme case of the poor student simulation, where no example lines are explained, all the self-monitoring statements during example processing would be positive, which is not far off from Chi et al.'s observation that 85% of the Poor solver's self-monitoring statements were positive.

The third and fourth findings involve the frequency and specificity of analogical references during problem solving. The number of references made by analogical search control and transformational analogy were counted. We assumed that only some of the analogical search control references to the derivation were overt, and that the others were mental references that would not show up in the Chi et al. data. This gave us a prediction of the frequency of analogical references. To get a prediction of the specificity of analogical references (i.e., the number of example lines read per reference), we counted the number of lines read by transformational analogy before it found one it could use, and we assumed that someone using analogical search control would go directly to the line whose derivation contained the sought goal. Given these assumptions, the good student simulation produced fewer and more specific analogical references than the poor student simulation, thus modeling the Chi et al. finding (see VanLehn, Jones & Chi, in press, for details).

Discussion

Although we controlled Cascade's behavior during example studying, by either telling it whether to explain the examples or not, its behavior during problem solving was determined solely by how much it learned during example studying. Qualitatively, the behaviors of the Good and Poor runs were quite similar to the behaviors of the Good and Poor students during problem solving. The good student simulation tended to stay on solution paths, use regular problem solving more often than transformational analogy, and learn something from the occasional impasses it encountered. The Poor student simulation tended to wander down unproductive paths, use transformational analogy more often, and learn nothing from the many impasses that it encountered.

These properties of Cascade's problem solving behavior are consistent with a preliminary analysis by Chi, VanLehn & Reiner (1988), who analyzed the protocols of a Good solver and a Poor solver as they solved the same problem. The Poor solver's protocol was divided into 77 episodes, and of these, 30 (39%) resulted in impasses.[3] Many of these impasses seemed to result in acquiring incorrect beliefs. In contrast, the protocol of the Good solver was divided into 31 episodes, of which only 7 (23%) resulted in impasses. In 6 of these, the Good solver seemed to learn a correct piece of knowledge. This preliminary analysis indicates that the Poor solvers had proportionally more impasses (39%) than the Good solvers (23%) while problem solving, and that the resulting knowledge was more often incorrect. This is just what Cascade did, too.

Example studying took up a relatively small proportion of the time that subjects spent during the study. Not only were there only 3 examples compared to 25 problems, the subjects spent less time on average studying an example than solving a problem. The learning strategy of self-explanation was active only during example studying, so it comes as a surprise that such a proportionally small change in work habits caused such a large change in the amount learned. Perhaps the most important result from the Good/Poor simulations is an explanation for this counterintuitive finding. The simulations showed that only 40% of the rules learned by the good student simulation and not by the poor were learned during example studying. The others were learned during problem solving. This came as somewhat of a surprise to us. There were two basic

[3]An impasse was identified as an outcome of an episode whenever the student believes that the next step that should be executed cannot be performed. Most (98%) of the impasses were identified by explicit statements such as "I don't know what to do with the angle," or "So that doesn't work either."

reasons that self-explanation increases learning during problem solving.

- The rules learned earlier allowed Cascade to travel down correct solution paths and reach impasses at places were it was indeed missing knowledge. Without these rules, the poor student simulation could not reach these productive impasses.

- The derivational triples acquired during rederivation of the example lines served as search control advice during problem solving, thus tending to keep the good student simulation on solution paths that led to productive, missing-knowledge impasses. The poor student simulation tended to wander off the solution paths, and reach impasses where there was nothing valuable to be learned.

It is doubtful that these interactions would have been discovered without a simulation as detailed as Cascade.

These results taught us about self-explanation per se, but the use of idealized student simulations leaves open the question of whether Cascade can actually model a real student. The next study tackles this question.

MODELING THE PROTOCOLS OF INDIVIDUAL SUBJECTS

The objective of the study reported in this section was to find out how close Cascade can come to modeling individual subjects. This study was undertaken in the same spirit as the ones in Newell and Simon (1972): Given a protocol, how closely can a simulation be fit to it? One difference between this study and those of Newell and Simon is that the task domain is physics, which is arguably a much richer task domain than the ones they studied. However, a more important difference is that considerable learning took place during our protocols.

A third difference is that our protocols are much longer than Newell and Simon's protocols, which made it impossible to employ their method of analysis. Each of the 9 subjects contributed protocols for 3 examples and 25 problems, so there were 252 protocols to analyze. Each protocol averaged about 12 pages, for a total of 3000 pages. Creating problem behavior graphs for all of them would be far too much work. Thus, part of

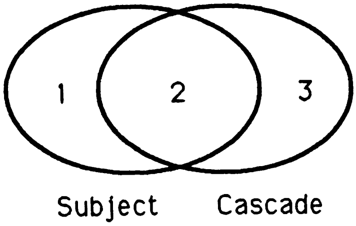


Figure 2: Matching the behaviors of Cascade and a subject

the challenge in this study was to devise feasible methods for measuring the match between the behaviors of Cascade and the subjects.

Figure 2 shows how the match between the behaviors of Cascade and a subject can be viewed. Region 1 represents behavior that the subject exhibited and Cascade did not. Region 2 represents the behaviors that are the same for both agents. Region 3 represents Cascade behaviors that the subject did not exhibit. The behaviors in region 3 have two sources. Some are computational expediencies: We couldn't get Cascade to do exactly what the subject did, so we had it do something else instead. That "something else" shows up in region 3. A Cascade behavior will also be put in region 3 if it is plausibly something that the subject did, but the protocol happens to show no signs of it occurring. For instance, it is known that not all cases of impasse-driven learning show up as hesitations or negative comments in protocols (VanLehn, 1991a). Whenever Cascade's impasse-driven learning is not reflected by overt signs of an impasse in the subject's protocol, that behavior is classified as region 3 behavior.

In all of the analyses presented below, we tried to determine two ratios: the amount of Cascade behavior that is matched by the subject (region 2 divided by the union of regions 1 and 2), and the amount of subject behavior that is matched by Cascade (region 2 divided by the unions regions 2 and 3). In order to make these comparisons, we had to find a way to count behaviors, which implies choosing a unit of analysis.

This was not hard for the first ratio, because Cascade's behavior is well defined. For instance, we usually used a goal as the unit of analysis and counted the number of goals generated by Cascade that were matched or unmatched by the subject. It was not easy to determine a unit of analysis for the other ratio, the percentage of subject behavior matched by Cascade. A variety of units were used, depending on the type of analysis being conducted.

Five analyses were conducted (see Table 4). Because we are more interested in getting Cascade to simulate the subjects' acquisition of physics rules than in getting it to simulate the chronology of their reasoning, four of the analyses ignored the order in which Cascade and the subject made inferences. Both Cascade's behavior and the subject's behavior were reduced to sets of inferences. Set intersections and differences were calculated, just as shown in Figure 2. However, we cannot entirely ignore the chronology of inferencing, since the earlier study indicated that analogical search control affects the location of impasses, which in turn determines what can be learned during problem solving. So a fifth analysis was conducted in order to see if the subjects' choices during problem solving could be predicted by analogical search control. After a description of how Cascade was fitted to the protocols, each of these analyses will be presented.

Fitting Cascade

Fitting Cascade means setting values for parameters so that the program's behavior matches the given subject's behavior as closely as possible. The parameters represent the products of cognitive processes that are not modeled by Cascade, and yet Cascade's performance depends on the outputs of these unmodeled processes, so they cannot be ignored entirely.

There are two major types of parameters. The first controls initial knowledge, which refers to the knowledge possessed by a student or Cascade just prior to studying the examples. The student's initial knowledge comes from reading the first several chapters of the textbook and from their earlier studies of physics and mathematics. Cascade does not model these processes, so it must be given an initial knowledge base. Cascade's initial knowledge base was always a subset of a fixed "rule library." The

Table 4: Analyses comparing Cascade's behavior to the subjects' behavior

1. How many of Cascade's example studying inferences were also made by the subject?
2. How many of the subject's example studying inferences were also made by Cascade?
3. How many of Cascade's problem solving inferences were also made by the subject?
4. How many of the subject's problem solving inferences were also made by Cascade?
5. Do the search control decisions made by the subject match those made by Cascade?

library consists of 3 buggy physics rules,[4] the 62 rules that constitute the target domain knowledge, and the 45 non-domain rules mentioned earlier. Selecting an initial knowledge base can be viewed as setting 110 binary parameters, one for each rule in the library, where 1 means that the rule is included in the initial knowledge base, and 0 means that the rule is excluded.

The second type of parameter controls the depth of self-explanation. When studying examples, subjects choose to explain some lines but not others. Even when they do explain a line, they may explain it only down to a certain level of detail and decide to take the example's word for the rest. For instance, they might explain most of the line, $F_{a_x} = -F_a \cos(30)$, but not bother to explain where the minus sign comes from. Cascade does not model how the subjects decide which lines to explain and how deeply to explain them.[5] To simulate the output of this decision

[4] One buggy rule applies $F = ma$ to any force and not just a net force. Another asserts that the mass of a body is equal to its weight. The third assumes that the sign of all projections is positive.

[5] There are many possible reasons for why subjects do not explain everything. For instance, the subjects may feel that they already know everything that they could

making process, extra propositions were entered into the descriptions of examples. Whenever Cascade is about to explain something, it first checks to see if `accept(P)` is in the example's description, where `P` is the thing it is about to explain. If an `accept(P)` is found, Cascade merely accepts `P` as explained without any further processing. Viewed as parameter setting, this amounts to associating a binary parameter with every explainable object, and setting it to 1 if it should be explained and 0 if it should be accepted.

The accept propositions are set by inspecting the subject's protocol. If the subject merely reads a line and says nothing else about it, then an accept proposition is entered for the whole line. If the subject omits discussion of a detail in a line, then an accept is placed around the Cascade goal that corresponds to that detail. In this fashion, the data completely determine which lines and parts of lines are explained by Cascade.

On the other hand, there is no way to easily determine what the subject's initial knowledge is. The whole protocol must be examined. As will be seen later, we sometimes made mistakes in selecting the initial knowledge. We should have fixed our mistakes, rerun the simulations and redone the comparisons of the program's output with the protocols. This will require months of work, so for this chapter, we are forced to report the analyses with our imperfect choices of initial knowledge left intact.

How many of Cascade's explanations are matched?

This section discusses the behavior of Cascade and the subjects as they explained examples. Goals were used as the basis for dividing Cascade's behavior into countable units. Each goal produced by Cascade was classified according to the method used to achieve it:

- Regular explanation: Cascade used one of the domain rules.
- Impasse with learning: Cascade reached an impasse, successfully applied an overly general rule, and learned a new domain rule via EBLC or analogical abduction.

learn from explaining the line, or they may feel that explaining such details can be left until such time as they really need to know them. Deciding how deeply to explain a line is a fascinating topic for future research.

- Accept without explanation: The goal was not processed any further, but merely accepted as true without proof, because the example's description contained an "accept" proposition for it.

Aggregating across the simulation runs of all 9 subjects, there were 1121 goals. We located each of these goals in the subjects' protocols, and based on the talk surrounding them, classified them into the same three categories plus a new one:

- Impasse with learning: If the subject paused, complained about the goal or in some other way showed signs of being stuck, then we classified the goal as being achieved by impasse-driven learning.

- Regular explanation: If the subject merely mentioned the goal or its conclusions without any fuss, or the subject said nothing at all about this goal but did mention its subgoals, then we classified the goal as being solved by regular explanation.

- Accept without explanation: If the subject said nothing about this goal nor its subgoals, then the goal was classified as being accepted without explanation.

- Impasse and accept: Sometimes subjects clearly tried to explain a goal, but couldn't do it at all, so they just accepted the goal without proof. This is different from the other kind of acceptance, where the subject did not even try to explain the goal. It is different from the other kind of impasse because no learning occurs.

Table 5 shows the 1121 goals and how they were classified. Most ($1061 = 9 + 2 + 654 + 396$) goals were processed the same way by both Cascade and the subject, so 95% of Cascade's behavior was matched by subject behavior, which is highly significant ($p << .001$, Chi-squared test). In order to get a qualitative understanding of the shortcomings in Cascade's model of the protocols, each of the off-diagonal cells is discussed.

There were 7 cases where Cascade learned a rule and the subjects were coded as accepting the goal. All 7 cases occur at the same point, on a line where the example says, "Consider the knot at the junction of the three strings to be the body." Explaining this line causes Cascade

Table 5: Proportions of Cascade actions matched by subject actions

Cascade	Subjects				
	Impasse resolved	Impasse unresolved	Regular explanation	Accept silently	Totals
Impasse + EBLC	9	0	10	0	19
Impasse + Anal. Abduct.	0	2	0	7	9
Regular explanation	0	0	654	39	693
Accept w/o explanation	0	4	0	396	400
Totals	9	6	664	442	1121

to learn a new rule via analogical abduction. However, only 2 of the 9 subjects commented about this rule during example studying. The other 7 said nothing at all about the line wherein this rule would be learned, so they were coded as accepting the goal without proof. We could have made Cascade accept the goal as well, which meant that it wouldn't learn the knot rule. During later problems that had three strings converging on a knot, this would cause Cascade to reach an impasse and use transformational analogy. Unfortunately, Cascade's transformational analogy mechanism is not powerful enough to make use of the knot-is-a-body line in the example. When we increased its power so that it could use this line, it became too powerful and would draw analogies that were so far fetched that no subject would consider them. This led us to invent analogical abduction, which is a novel type of machine learning (see VanLehn & Jones, in press, for discussion). In order to test it out, it was included in Cascade. However, it is clear from this analysis that there are empirical problems with it. If an example's line really does cause analogical abduction, which is a form of impasse driven learning, then more subjects should have shown impasses. We now believe that transformational analogy is actually the source of transfer between the example line and problem solving, and that the two subjects who had impasses here should be classified as "impasse and accept," rather than as learning a new rule. As will be seen later, there are other signs that Cascade's model of transformational analogy is flawed.

There were 10 cases where Cascade learned a rule and the subjects were coded as doing regular explanation. There are two possible explanations of this discrepancy. Either the subjects really were doing impasse-driven learning, but they showed no signs of it in the protocol, or the subjects knew the rule already and were simply applying it here rather than learning it. Since the data are consistent with both explanations, the interpretation depends on the prior probabilities of possessing the rules in question. All but one of the ten rules were not mentioned in the textbook, so they are less likely to be in the subjects' initial knowledge. The rule that is mentioned in the textbook determines the sign of a projection of a vector onto a negative axis. However, three subjects showed clear signs of impasses when this rule was first used, and three used a buggy version of the rule that always assigned a positive sign. The correct sign rule appears to be hard to learn and/or recall from the text for 6 of the 9 subjects, so it was probably not known by the other subjects either. Thus, because none of the rules involved seem likely to be in the subjects' initial knowledge, we suspect that all 10 cases in this cell of Table 5 correspond to impasse-driven learning events that were not displayed by the subjects.

There were 4 cases where the subject clearly tried to explain a goal but failed. On subsequent problems, the subject would explicitly refer back to these points in the examples and use transformational analogy. This is just what the two subjects who were coded as doing analogical abduction do, so that is why we now believe that there is no evidence for analogical abduction.

There are 39 cases where Cascade did regular explanation and the subjects were coded as doing accepts. Of these, 35 occurred when Cascade was trying to explain a force diagram. According to its rules, figuring out which forces exist and determining their directions are subgoals deeply embedded beneath the goal of drawing the force diagram. Some subjects discussed the forces without mentioning the force diagram. Either they were silently explaining many of the details of the force diagram, or they ignored the force diagram and "just knew" that it was important to explain the forces. We think the latter is more plausible, but we cannot easily model it without changing the goal structure embedded in Cascade's rules or Cascade's model of accepting the example's statements without proof.

Stepping back from the details, there are several results from the analysis in Table 5. It is clear now that Cascade needs several kinds of revisions: (1) Analogical abduction should be eliminated and transformational analogy strengthened. (2) The goal structure and/or the acceptance mechanism need to be revised in order to handle some subjects' explanations of free-body diagrams. (3) The subjects showed impasse-like behavior on 4 occasions, but seemed to learn nothing from them. This is currently not an option with Cascade. It should be changed so that one of its responses to an impasse is just to accept the stuck goal as true without learning anything.

Another result concerns the handling of impasses. Cascade had 19 impasses where learning occurred, and of these, the subjects showed impasse-like behavior on 9. If we believe the codings, then 48% of the subject's impasses where learning occurred were visible in protocol data, and the other 52% were the victim of the usual incompleteness of protocol data. In the VanLehn (1991a) study of impasses in strategy discovery, there were 11 learning events, of which 8 (73%) were marked by impasse-like behavior on the subject's part. This is consistent with the impression that one gets from reading the protocols, which is that the subject in the VanLehn (1991a) study verbalizes much more of her thoughts than the subjects in the Chi et al. (1989) study. Thus, although the proportion of "silent" impasses is higher in the present study, it is not inconsistent with the earlier study's proposition.

How many of the subjects' explanations are matched?

The preceding section evaluated the match in one direction only, by seeing how much of Cascade's behavior is matched by subject behavior. This section evaluates how much of the subject's behavior is matched by Cascade behavior.

In order to do this, we extended an analysis by Chi and VanLehn (1991). They first coded every utterance in the example studying protocols as either a physics explanation, a mathematical explanation or one of several other kinds of utterances. They then coded each physics explanation into an if-then rule that presented the gist of the subject's comment in a uniform, more easily understood format. We extended this analysis by including the mathematical explanations as well and by correcting what we felt were a few minor mistakes in the earlier analysis of physics explanations.[6] Finally, we determined whether each rule apparently used by the subjects was also contained in Cascade's knowledge.

---

[6]It was usually clear what the action sides of these rules should be, but inferring the preconditions and the generality of the rules often required making some assumptions. Sometimes we disagreed with the assumptions made in the earlier analysis.

Table 6: Subject's explanations during example studying

Explanations	Categories	
143	Matches Cascade (63%)	
61	Outside task domain (27%)	
	23	Mathematical manipulations
	9	Editting part a to solve part b
	16	Extra example lines
	13	Units or terminology
	61	Total
23	Inside task domain (10%)	
	8	Incorrect explanations, retracted
	6	Acceleration and motion
	3	Abstract, partial plans
	0	Global planning from Chi et al.
	6	Miscellaneous, opportunistic
	23	Total
227	Grand total	

Some of this knowledge appeared explicitly as Cascade rules, while some of it was implicit in Cascade's rule interpreter (e.g., algebraic knowledge).

Of the 227 total explanation episodes found in the protocols, we determined that 143 (63%) were matched by Cascade explanations, while 84 were not. This indicates that Cascade models a large portion of the subjects' explanatory behavior. However, there are also many explanations that Cascade does not appear to account for. We categorized these explanations in order to determine whether Cascade should be expected to make them (see Table 6).

Of the 84 explanations, 61 concerned reasoning that was outside the domain of cognition being modeled. There were four classifications:

- 23 explanations concerned mathematical inferences that Cascade did not need to make because it had either been given the information in the problem statement (e.g., certain geometric information was provided to it), or it did not simplify its answers.

- 9 explanations occurred exclusively on part *b* of one example. Part *a* of this example describes a static situation where a string is holding a block on an inclined plane. Part *b* of the example asks one to find the acceleration of the block when the string is cut. Subjects explained part *b* by "editing" the explanation of part *a*. For instance, one editing rule was "If a force is removed from a static case and there is no friction, the body will move." Cascade does not reason about part *b* of the example in this manner. Rather, the system treats parts *a* and *b* as two distinct examples.

- Sometimes an example would contain a few lines that would emphasize aspects of the problem that were not germane to solving it or would discuss limiting cases (see Figure 1, line 3). Although we did not ask Cascade to explain these lines, the subjects sometimes would, and 16 of their explanations were of this type.

- 13 explanations involved miscellaneous comments about the examples that were judged to represent knowledge outside of the task domain as formalized in Cascade. For instance, we did not bother to model reasoning with units, so the statement, "In the English system, slugs are mass and pounds are weight," is considered outside the task domain.

The remaining 23 explanations are all relevant to the domain modeled by Cascade, so it should probably generate them. They fell into three classifications.

- 8 explanations appear to have been generated tentatively then retracted. They are all incorrect statements about physics, and the subjects seem to have revised their explanation a short time later. An example is, "If the two bodies in an Atwood's machine have the same acceleration, then they are not moving." How the subjects generated these conclusions is a bit of a mystery, although some are clearly the results of overly general rules. For instance, one subject said that one can calculate the tension in a string by adding the tensions in its parts. He probably generated this explanation by applying an overly general part-whole rule that works correctly for quantities such as volume, mass and weight.

- 6 explanations concerned the relationship between motion and acceleration. Of these, 4 explanations stated essentially the same thing: "If a body moves, then it has a non-zero acceleration." Another explanation said, "If a body has motion on an axis, then it has acceleration on that axis," and the last explanation said, "If acceleration is zero, then nothing is moving." These all stem from the same incorrect conception of acceleration as speed, which is very common and hard to remove (Reif, 1987). Cascade should also be equipped with this misconception. Even though these new rules would alter Cascade's model of the subjects' explanation of examples, we do not expect that they would change Cascade's behavior on later problems. This is because the problems all deal with acceleration and do not mention concepts like velocity or "motion."

- 3 explanations articulated an abstract, partial plan for solving the problem. For instance, one explanation said "The weight of the block in the string example can be computed by figuring out the tension in the strings." This rule represents the top level of an abstract plan for determining the weight of the block. Cascade does not do hierarchical planning, but this rule provides evidence that perhaps it should. In particular, we would probably find more evidence for planning in the problem solving protocols. Planning did not have much of a chance to emerge during example studying because subjects are mostly led by the hand through the example solutions, so there is no real need to plan.

- 6 explanations defy classification, so they are simply listed below

  1. The body is the thing that the forces are acting on.
  2. Tension is important because it transmits the force between the blocks.
  3. The acceleration in an Atwood's machine is caused by gravity.
  4. If the right mass is greater than the left mass in an Atwood's machine, then the machine will accelerate downward.
  5. An upward force can act against gravity to keep a body from falling down.
  6. Most forces are gravitational.

> All 6 explanations are true statements. However, they are not relevant to the goal of solving the problem, which is why Cascade did not make them.

This analysis indicates that of the 227 explanations uttered by subjects, 143 (63%) were matched by Cascade's explanations, 61 (27%) were outside the task domain being modeled, and 23 (10%) are explanations that Cascade should make but does not.

Cascade embodies a hypothesis about explanation, which is that explanation of solution lines in physics examples consists of rederiving them. This is a kind of local explanation, in that Cascade focuses only on the current solution line. It does not step back and try to see a global pattern that spans all the solution lines in an example. This may seem somewhat unusual, as other models of example explaining (e.g., VanLehn, 1990; Reimann, in press) emphasize global explanation. However, from one point of view, there is little point in global explanation of physics examples. Because later solution lines use results from earlier solution lines, doing local explanation of the later lines ties them together with the earlier lines, yielding an overall coherent structure. From another point of view, there is great benefit in global explanation, because it turns out that all the examples have a similar chronological structure: one first chooses bodies, then draws a diagram showing all the forces acting on the bodies, chooses coordinate axes, instantiates Newton's law along each axis, and solves the resulting system of equations. The textbook even mentions this procedure. One might expect the subjects to look for such a global, chronological structure, perhaps by first locally explaining all the solution lines, then reflecting on the whole solution to see if the overall structure made sense. However, the subjects produced only 3 global explanation statements. Seeing the overall chronological pattern in solutions does not appear to be a major concern for these subjects, perhaps because the logical structure suffices to make lines cohere. This is consistent with Sweller's work, which has shown that chronological patterns embedded in solutions are often overlooked when subjects are focussed on obtaining a goal (Sweller & Levine, 1982).

Another hypothesis about explanation embodied in Cascade is that all inferences are ultimately directed towards the top level goal of explaining the current solution line. Cascade uses a backwards chaining theorem prover, which means that it starts with the top level goal, chooses and applies a rule, and then focuses on the first of the several subgoals created by the rule's application. When all subgoals have been achieved, it executes the rule, which finally draws a conclusion. This means that all inferences are done in order to satisfy some goal, and that goal is ultimately a subgoal of the top level goal. This makes Cascade a narrowly focused, methodical explainer. It could be that people are more opportunististic and make observations (i.e., drawing conclusions) whenever they notice that they can be drawn. In the extreme, they might do forward chaining, drawing all possible conclusions about a problem while paying no attention whatsoever to the solution lines or the overall goal of the problem. However, only 6 of the 227 explanations appear to be opportunistic, in that they were not matched by Cascade's goal-directed inferences.

How much of Cascade's problem solving is matched?

We turn now to considering problem solving behaviors. The comparison between Cascade and the subjects is made difficult by two factors. First, the protocols are huge. There are approximately 2700 pages of problem solving protocol, as compared to 300 pages of the example studying protocol. Second, problem solving is less constrained than example studying. Rederiving a line in a solution takes at the very most only a few minutes, whereas solving a problem can take almost an hour. Subjects seldom get badly lost while rederiving a solution line, whereas when solving a problem, subjects often wander down several unproductive paths before finding a solution or giving up. Getting Cascade to follow the subjects on a long doomed search path can be difficult.

An important methodological problem is finding a fair way to evaluate the fit of a simulation and a protocol. Suppose one evaluated the fit by counting the actions taken by the simulation that are matched by subject actions, and dividing by the total number of actions taken by the simulation. It often happens that the actions of the simulation and the subject disagree at some point. This often causes them to diverge and follow separate paths for a while, perhaps even a long while. The longer the divergent paths, the worse the fit, even though the blame is due to one false move earlier in all cases. Thus, simply comparing matching to mismatching actions is unilluminating, for it confounds the quality of the simulation with properties of the search space, namely, the lengths of certain paths.

Our approach is to equip Cascade with a variety of parameters, which we call "nudges," whose main purpose is to nudge Cascade back onto the subject's search path whenever it would otherwise wander off. The fit between the simulation and the subject's protocol is measured by counting the number of times the simulation had to be nudged. An additional benefit of nudging for measuring fit is that each nudge represents a piece of unexplained cognition. By taking a census of the nudges, one can rank types of unmodeled cognition and discover which ones are affecting problem solving behavior the most. Here are the types of nudges used:

- When Cascade solves a problem, it normally tries transformational analogy only after it tries regular domain knowledge. However, some subjects apparently prefer to use transformational analogy in some cases even when their behavior on earlier cases demonstrates that they have the appropriate domain knowledge and could potentially use it here. In order to force Cascade to follow the subjects, propositions of the form `trafo-only(G)` were placed in the problem's description whenever `G` is a goal that the subject preferred to achieve via transformational analogy. We call such cases of transformational analogy "forced."

- By default, the top-level goals of a problem require identifying a body and drawing a free-body diagram before finding the sought quantities. If the subject did not draw a free-body diagram, we eliminated these goals from the statement of the problem.

- On rare occasions, subjects came up with analogical mappings that Cascade could not generate. In such cases, we simply gave Cascade the subject's mapping or modified the problem representations so that the subject's mappings could be generated.

- Subjects did not always use the buggy $F = ma$ rule, which lets F be an individual force instead of the net force, when it was applicable (i.e., we could not figure out exactly what preconditions the subjects had for this rule). Perhaps they rightly believed that it was overly general, and thus they would only use it as a last resort. At any rate, we controlled its usage by entering `ignore(F=ma_wrong)` into the descriptions of some problems but not others.

In order to evaluate Cascade's ability to model a given subject, nudges were entered by trial and error. Cascade's behavior then was compared to the subject's by classifying each of the goals in its trace according to how the goal was achieved. Four classifications were used:

- Regular solving: Cascade used one of the domain rules.
- Impasse with learning: Cascade reached an impasse, successfully applied EBLC, and learned a new domain rule.
- Transformational analogy: Cascade could achieve the goal with a domain rule, so it used transformational analogy.
- Forced transformational analogy: We forced Cascade to use transformational analogy even though it could have used a domain rule to achieve the goal.

Next, each of these goals was classified according to how the subject appeared to achieve it. Four classifications were used here as well:

- Transformational analogy. If the subject referred to an example and copied parts of it over, the goal was classified as achieved by transformational analogy.
- Impasse with learning. If the subject paused, complained about the goal or in some other way showed signs of being stuck, but the subject did not refer to an example to achieve the goal, then we classified the goal as being resolved by EBLC.
- Impasse and give up. On a few impasses, the subject just gave up without seeming to resolve the impasse or learn any new rules. Giving up is not one of the options available to Cascade for handling an impasse, although it should be.
- Regular solving: If the subject solved the goal without complaining, referring to an example or pausing for inordinate amounts of time, then we classified the subject as achieving the goal via regular problem solving.

Table 7: How many Cascade goals are achieved the same way by subjects?

Cascade	Subjects				
	Regular solving	Transform. analogy	Impasse with learning	Impasse and give up	Totals
Regular solving	3653	47	0	0	3700
Transform. analogy	16	176	0	4	196
Forced trans. analogy	0	35	0	0	35
Impasse with learning	7	0	9	0	16
Totals	3676	258	9	4	3947

Given these classifications, the results for all nine subjects appear in Table 7. In most cases ($3653 + 176 + 9 = 3838$ or 97%), the subjects handled goals in the same way that Cascade did, which was extremely unlikely to occur by chance ($p << .001$, Chi-squared test). Let us examine each of the other cells to see how serious the mismatching cases are.

There were 47 cases where the subjects did transformational analogy and Cascade did not. All these were due to the simplicity of Cascade's model of transformational analogy. One subject often used vector equations as if they were scalar equations, and applied them in creative, albeit incorrect ways that Cascade could not model. We let Cascade solve those problems in its normal way and counted all 42 goals as cases where the subject did transformational analogy and Cascade did not. The other 5 cases occurred when a subject could not recall some trigonometry rules, so she mixed transformational analogy with regular problem solving in a complex way that Cascade could not model. These 47 cases indicate that Cascade's model of transformational analogy needs improvement.

There were 16 cases where Cascade did transformational analogy and the subject seemed to do regular problem solving. Transformational analogies occur when Cascade is missing the knowledge to do regular problem solving, so there are two possible explanations for each case: Either the subject knew the rules that Cascade lacked, or the subject actually did have an impasse and resolved it with transformational analogy, but they didn't refer overtly to the example because they could remember the line that they needed, and thus were not coded as performing transformational analogy. Of the 16 cases, 8 seemed to be cases of covert transformational analogy because they involve accessing the free-body diagram, which is much easier to remember than the equations. Two more cases seemed to be covert transformational analogy, because the subject had already referred to the example's equation 3 times earlier, and probably had committed it to memory. Five cases seemed to be caused by the subject having a buggy rule about negative signs that Cascade lacked, but Cascade was able to get the same effect with transformational analogies. The last case is similar to the five just discussed, but with a different rule. Thus, of the 16 cases, 10 seem to be covert impasses correctly predicted by Cascade, and 6 seem to result from the subject having initial knowledge that is not in Cascade's standard initial knowledge base.

In 4 cases, the subjects reached impasses and gave up. Since Cascade currently cannot give up at impasses, these cases were approximated with transformational analogy. However, Cascade did succeed in predicting the location of the impasses.

There were 35 cases of forced transformational analogy. Two subjects (9 of 35 cases) always copied the free-body diagrams and never generated them on their own, so these subjects apparently were lacking knowledge about drawing free-body diagrams. In retrospect, these subjects should have been modeled by having their initial knowledge adjusted to remove the rules about drawing free-body diagrams. The other 26 cases occurred with subjects who clearly had the requisite rules, but chose to do transformational analogy instead. Of these 26 cases, 21 involved copying a free-body diagram rather than reasoning it out from physics principles and the other 5 involved copying trigonometry functions rather than figuring out whether the function should be sine, cosine or tangent, and what the angle should be. It is certainly easier to use transformational analogy for these particular cases, and apparently the subjects felt it was safe to do so, even though transformational analogy is fallible.

There were 7 cases where Cascade did impasse-driven learning and the subjects were coded as doing regular problem solving because they showed no signs of impasses. In general, there are two possible explanations for such cases. Either the subject actually did impasse-driven learning but failed to show any signs of it in their protocol, or they already had the rule that Cascade was missing so they just applied it instead of learning it. We examined each of the 7 cases to try and determine which explanation was most plausible for each. In some problems with friction in them, Cascade must learn four rules about friction forces.

One subject showed no signs of an impasse at any of these locations, so we suspect that she already understood friction forces. Another subject showed signs of an impasse at three of these places, but not at the fourth; it is likely that this fourth occasion was a covert impasse. Similarly, one problem required learning four rules about pressure forces. A subject showed signs of an impasse on only two of the four occasions, so it is likely that the other two occasions are silent impasses. Thus, of the 7 cases where Cascade does impasse-driven learning and the subjects appear not to, 3 seem to be silent cases of impasse-driven learning and 4 seem to be cases where the subject already knew the rules that Cascade learned.

From this examination of the mismatching cases, it seems that Cascade would need three augmentations in order to handle all the data. (1) Transformational analogy needs to be made more powerful so that it can model the more creative (albeit incorrect) usages exhibited by subjects. (2) The model should be free to choose transformational analogy instead of regular problem solving when it estimates that transformational analogy would be easier or more reliable than regular problem solving. (3) When Cascade cannot resolve an impasse, it should be allowed to give up. All the other cases of mismatching appear to be covert versions of the predicted events, or cases where rules should have been removed from the initial knowledge base.

In short, it appears that almost everything that Cascade does is matched by subject behavior. The next section analyzes the subjects' behavior in order to see how much of it is matched by Cascade.

How much of the subjects' problem solving behavior is matched?

In order to quantify how much of the subject's thinking during problem solving could be simulated by Cascade, we adopted the same unit of analysis that was used in the preceding analyses by converting the subjects' protocols into Cascade-sized goals. As an illustration of this analysis, Appendix 1 shows a protocol and our encoding of it. Following the tradition of Newell and Simon (1972), the protocol appears in the left column, and the encoding appears in the right column.

This kind of analysis is quite time consuming, so we could not do it for all 252 protocols. Thus, we selected 4 protocols that we felt were typical. Two were from Good solvers, and two were from Poor solvers. Each of these pairs consisted of one protocol that was mostly transformational analogy and one that was mostly regular rule-based reasoning. (The protocol in Appendix 1 is a Poor solver who is doing mostly transformational analogy.) Clearly, this is too small a sample to draw strong inferences, but our point in this section is just to get a rough idea of the match.

Inferences occur whenever a goal is reduced to subgoals, or a goal is achieved. Thus, by literally reading between the lines, one can tell from the encoded protocols what the subjects' inferences were. In the four protocols, there were 151 inferences, excluding trivial arithmetic and algebraic ones. We examined each, and determined that Cascade could do all but 15 of them. That is, if we were to simulate these protocols with Cascade, we would need 15 new rules and would probably have to nudge it 15 times in order to get it to apply these rules. Thus, it appears that Cascade can model about 90% of the subject's inferences during problem solving.[7]

In order to give a qualitative sense of the behavior that Cascade could not simulate, we divided the inferences into ones that seemed outside of the intended task domain of Cascade and those that Cascade really should have modeled. Those that are outside the task domain are:

- In 2 inferences, the subjects checked their work by plugging their answers back into equations and seeing if the equations balanced.
- In 2 inferences, the subjects struggled to find the appropriate units for their calculations.
- In 2 inferences, the subject had difficulty understanding the diagram that accompanied the problem statement. In particular, it was difficult to decide whether a certain line stood for a string or not.

The inferences that were inside the intended domain were:

[7]Frankly, this estimate seems high to us. If we actually tried to add the requisite 15 rules to Cascade and simulate these protocols, we would probably find that the coverage was closer to 60% or 70%.

- 4 inferences were coded for a case where the subject decided he needed to understand freefall better, and went off to read the relevant page of the textbook, then decided that his (incorrect) solution to the problem was right anyway.

- 2 inferences were coded for a case where the subject decided to convert a vertical acceleration to one parallel to an inclined plane, but apparently did not realize that projection could be applied directly, so he "converted" it to a force using $F = ma$, projected the force, then converted it back.

- 2 inferences involved a subject who let $g = -9.8$ for no apparent reason. This could have been an unintentional error, except that the subject noticed later that g was negative and did not correct it. The second inference occurred later, when she dropped a negative sign "for fun" as she put it, thus effectively canceling her earlier error and obtaining a correct answer.

- 1 inference was coded for a subject who invented something he called a "double force" that included both gravitational and frictional influences.

Although these lists would clearly be much longer if more protocols had been analyzed, and the small sample makes any statistical inferences unsound, taking the data at face value indicates that about a third of the unmatched behavior is outside the task domain, and the other two thirds is behavior that Cascade should model. Moreover, most of the behavior that Cascade should model is incorrect reasoning of a wide variety of types. More research is needed before we can conclude anything about the sources of these incorrect inferences.

Control choices

The preceding analyses assessed what was done, but ignored the order in which actions took place. This section concerns the overall control structure as well as the local choices of which rule to try first in achieving a goal. Both these factors control the order in which inferences take place.

Cascade uses a backwards chaining control structure. A goal is processed by selecting a rule, then posting any subgoals required by that rule. After the subgoals have been achieved, the rule's conclusion is asserted. Thus, a goal should show up twice in a protocol: when it is first posted and when it is completed. In our protocols, subjects did not usually talk about their goals when they posted them (see Appendix 1), although they often mentioned the conclusions that were made when a goal was achieved. This could be taken as a sign that they were not following a backchaining control structure. However, a control structure also restricts the order in which actions can take place. For instance, if A and B are subgoals of C, and D is not, then the order A,D,B cannot occur. Thus, the ordering in which goals are achieved is diagnostic of the control structure.

As part of the analysis in the preceding section, we fit a backwards chaining goal structure to the subjects' behavior in the four protocols analyzed. Of the approximately 151 goals, there were three cases where backwards chaining would not fit. In two, the subject performed goals prematurely (i.e., the A,D,B case just mentioned). During the third case, the subject explicitly decided which of two conjunctive (sibling) goals to do first. This kind of search control occurs in some means-ends analysis problem solvers (e.g., Prodigy; Minton et al., 1989) but not in Cascade. In short, the available evidence indicates that Cascade's control structure is not a bad first approximation to the subjects' overall approach.

The only search control decision made by Cascade is which rule to apply given that more than one matches the current goal. Two factors determine Cascade's choice of rule. If analogical search control can find an old goal that is isomorphic to the current goal, then the old goal's rule is chosen. If analogical search control does not apply, Cascade selects rules in the order in which they appear in a file. This file is set up to generate efficient behavior in general, and is not tuned for any particular subject.

The first analysis involves placing all goals in one of two classes: Either the first rule selected for achieving this goal was ultimately rejected and another rule was used in its place in the final solution, or the first rule selected was used in the final solution. This categorization was carried out for all Cascade goals and for all subject goals corresponding to Cascade goals. Of the 3461 cases where Cascade picked the correct rule first, the subject failed to pick the correct rule first in only 81 (2.3%) cases. Thus, Cascade predicted the subject's choice of rule in almost all (97.7%) cases.

In order to determine how much of this success is due to analogical search control, we split the 3461 cases according to whether Cascade's choice was determined by analogical search control. Cascade selected the correct rule first *without* the involvement of analogical search control 2702 times. In 62 of these cases, the subjects did not choose the correct rule first. Thus Cascade's default rule ordering predicted the subjects' rule choices 97.7% of the time. Cascade picked the correct rule first *with* its analogical search control mechanism 759 times, and agreeing with the subject's choice in all but 19 cases, for a success rate of 97.5%.

Initially, this seemed a disturbing result, because it appeared that analogical search control gives the model no predictive accuracy over the default rule ordering. This raises the question of whether Cascade would be better off without analogical search control. If it always used its default rule choice, would its overall prediction accuracy rise? Rather than simulate all 9 subjects with analogical search control turned off, we estimated what the fit would be. We gave Cascade all the knowledge necessary to explain and solve the examples and problems (so no impasses would be generated) and ran it on all the examples and problems. While running, it kept track of how many times it used analogical search control to choose a rule, and how many times that choice corresponded to the default rule choice. We found that analogical search control led to a choice different from the default rule approximately 12% of the time. This implies that if Cascade had been run with search control turned off, then about 89 (12% of the 740) cases where analogical search control predicted the subjects' rule choice would now become cases where its predictions fail. In addition, hand analysis of the 19 cases where analogical search control failed to predict the subject's choices indicates that only 2 cases would be successfully predicted if analogical search control were turned off. Thus, if Cascade had only its default search control, it would mispredict 106 cases that it successfully predicted with analogical search control turned on, so its accuracy would drop to 95.6% as opposed to 97.7% with analogical search control enabled. Thus, analogical search control does help.

In order to further understand why analogical search control failed to predict 19 rule choices, each was analyzed. All were generated by 2 subjects, so Cascade's analogical search control predicted the rule choices for 7 of the subjects with 100% accuracy. Moreover, it turns out that in 11 of the 19 cases, the first inference made by the subject could not be modeled by any Cascade rule. Such cases indicate an inaccurate model of the subject's knowledge, rather than an inaccurate model of the subject's search control.

In summary, our initial result appeared to suggest that analogical search control provided no closer match between Cascade's problem solving behavior and the subjects' than Cascade's normal problem solving did. However, for 7 of the 9 subjects, Cascade provides a clear improvement in matching the subjects when analogical search control is used. For the other two subjects, the failure to match appears to arise from missing prior knowledge rather than a defect in the learning mechanism. Moreover, if analogical search control is turned off, Cascade's prediction accuracy would drop.

Discussion of the fit between Cascade and individual subjects

There were two purposes in fitting Cascade to the individual subjects. The first was to find out what the subjects were doing, and the second was to find out how well Cascade could model that. We discuss these objectives together, first looking at example-studying behavior, then problem solving behavior.

The two major processes during example-studying were explanation of a line and acceptance of a line or a part of a line without explaining it. Cascade's model of self-explanation is to rederive the line via ordinary deduction. Cascade's backwards chaining control structure ensures that only inferences relevant to the top goal are made. This sufficed to model 63% of the subjects' 227 explantions (see Table 6). Cascade's model of accepting a line was simply to prune a whole subtree of an explanation by accepting a goal as achieved without trying to achieve it. This sufficed to account for 92% of the subjects' 422 cases of acceptance (see Table 5); actually, the lack of fit may be due to the goal structure implicit in Cascade's rules rather than the acceptance mechanism per se. Overall, Cascade accounts for about 75% of the subjects' behavior during example studying.[8]

The self-explanations that Cascade does not model are mostly (73%) concerned with cognitive skills that we are not interested in modeling, such as algebraic equation solving. That left only 23 explanations that Cascade really should have modeled. These fell into two groups: incorrect explanations (14 cases) and more general comments (9 cases) including abstract, partial solution plans and observations such as, "The tension is important because it transmits force between the blocks of an Atwood's machine." The first group indicates that Cascade needs more buggy rules than it currently has. In particular, it needs to model misconceptions about acceleration and motion. The second group indicated that the subjects have an ability that Cascade lacks. They can stand back from the details and abstract an overall view of either the solution (i.e., they see an abstract plan or chronological pattern in the inferences) or the system (i.e., they form a mental model of the mechanical device). Although these are certainly interesting and important types of cognition, they appeared surprisingly rarely in this study (only 9 of 227 cases, or 4%). When the Cascade research began, we expected plan recognition to be the most important kind of self-explanation. We have since learned that it occurs rarely and may have little influence on subsequent problem solving.[9] Overall, it is good news that only 23 (14%) of the 166 interesting, task-domain relevant explanations uttered by subjects require extensions to Cascade in order to model them. Even in its present form, Cascade successfully models the bulk of the subjects' self-explanations.

---

[8]The coverage figure for example studying was calculated as follows: Table 6 shows that 63% of the subjects' self-explanations are modeled by Cascade. Table 5 shows that 92% of the acceptances are modeled by Cascade. However, these tables use different units. From Table 5, we can estimate that about 40% of the subjects' behavior was acceptances, so we can use that figure to form a weighted average of the two coverages, and thus calculate that about 75% of the subjects' example studying behavior is matched by Cascade.

[9]Although it seems pointless with only 3 cases of abstract planning in the example studying protocols, we could analyze the problem solving protocols to see if these subjects' rule choices during the relevant sections of their protocol are better explained by the abstract plans they found during example studying than by the existing search control mechanisms of Cascade. Because analogical search control probably makes the same predictions about rule choices as an abstract plan, we doubt that this analysis would yield unequivocal results.

The two most common problem solving processes were backwards chaining rule-based inference and transformational analogy. We were surprised by the prevalence of transformational analogy during problem solving, although it was certainly due in part to the fact that 12 of the 21 problems in the study were isomorphic (or nearly so) to one of the three examples (i.e., there was a set of "string" problems, "incline" problems, and "pulley" problems). Although only around 6% of subjects' problem solving involved transformational analogy (see Table 7), it often had a profound affect on the direction of the subjects' search. Cascade has a simple model of transformational analogy, but it was not powerful enough to handle all the cases. Subjects sometimes find analogical mappings that Cascade cannot. They sometimes mix transformational and regular problem solving (47 cases). They sometimes prefer to use transformational analogy even when they do not have to use it (35 cases). Many of these problematic cases occur when subjects need to draw a free-body diagram and refer to the example's free-body diagram for help. They may be using well-honed skills for visual analogizing. This would explain why Cascade's transformational analogy, which is oriented towards analogical transfer of equations, is so incomplete.

On the whole, it appears that most of the example-studying and problem-solving behavior can be explained as deduction, simple acceptance of example statements, and transformational analogy. Although these three processes cover only 75% of the example studying behavior and 60-90% of the problem solving behavior, the behavior they do not cover mostly involves mathematical manipulations and other types of cognition that are outside the domain of study.

We were surprised to find that the overall control structure and local control choices were also modeled rather accurately by Cascade (see the preceding section). However, we did not stress this aspect of Cascade either during it development nor during its evaluation, so there is probably much room for improvement in both areas.

GENERAL DISCUSSION

We have completed three steps of a four-step research program. The first step was to find a computationally sufficient account for the knowledge acquisition that occured in the Chi et al. study. The major technical hurdle was finding a way to constrain search during problem solving so that impasses would occur at the right places. This was achieved by adding analogical search control, a form of symbol-level learning. There was no way to tell in advance of running Cascade whether analogical search control was sufficient. Fortunately, it was, and Cascade was able to learn all the rules that it needed to learn. Moreover, the problem of getting impasses to occur in the right places is faced by all impasse-driven machine learning systems, so this result is relevant to many machine learning systems. A minor hurdle was finding a way to transfer knowledge from the knot-is-a-body example line to problem solving. After trying several methods, we discovered a new machine learning technique, which we called analogical abduction.

The second step in the research program was to demonstrate that Cascade could explain the main findings of the Chi et al. study. As a model of the self-explanation effect, Cascade was qualitatively adequate. It could self-explain example lines as well as just accept them. It could solve problems with and without referring to examples, and its analogical references can both dive into the middle of the example to pick out a single fact (analogical search control) or read the example from the beginning searching for a useful equation (transformational analogy). In order to go beyond qualitative similarity, simulations were conducted that modeled an idealized good student and an idealized poor student. All four of the main findings from the self-explanation study were reproduced in the contrast between the two simulations. A particularly suprising result was that most of the learning occurred during problem solving even though the particular learning strategy we manipulated, self-explanation, operated only during example studying. Examination of Cascade's processing showed that the acceleration of its learning during problem solving was caused by (1) analogical search control obtaining more guidance from the experience (derivation) left behind by self-explaining the examples, and (2) problem solving having prerequisite knowledge, obtained during example studying, that allowed it to reach impasses where learning could appropriately take place.

The third step in the research program was to demonstate that Cascade can simulate real student cognition at the 5 to 10 second unit of analysis. We fit Cascade to subject protocols by forcing it to explain exactly the same lines as the subject and to do transformational analogy at exactly the same points as the subject. We also gave it initial knowledge that approximated the subject's knowledge just prior to explaining the examples. Setting these parameters, plus occasionally "nudging" the system, sufficed to fit Cascade to cover most of the subjects' behavior. Such fitting was carried out for all 9 subjects, and the resulting match between system and subject behavior was evaluated. As one might expect, given that Cascade was designed to model the subjects, almost everything it did was also done by the subjects. Tables 5 and 7 show that over 95% of Cascade's goals occured in the subjects' behavior and were achieved the same way by both simulation and subject. On the other hand, when the subjects' behavior is analyzed in terms of goals and inferences, about 75% of their example-studying behavior and between 60% and 90% of their problem-solving behavior is matched by Cascade goals and inferences. Most of the unmatched behavior concerns mathematical manipulations and other skills that Cascade was not intended to model. It was found that the main inadequacy in Cascade is its simple model of transformational analogy. Subjects were quite clever at forming useful analogies with the examples, and especially their free-body diagrams.

The fourth step in the research program is to use the fitted models of individual subjects to find out more about their learning. Unfortunately, we discovered during the current fitting that some of our assumptions about initial knowledge were incorrect, so we will have to rerun the fitting exercise with new assumptions before conducting these analyses. Nonetheless, a few speculative remarks can be made on the basis of the existing analyses.

We were surprised that there were so few clear-cut cases of impasse-driven learning in the protocols. Tables 5 and 7 show that the subjects had clear signs of impasses on only 18 occassions when Cascade did EBLC. Although analyses to be conducted later will tell us exactly why there were so few clear cases of learning, it appears that it is due to overuse of the free-body diagrams. Most of the rules learned by the idealized good student simulation are used during the initial stage of solving a problem, when a situation is analyzed and the forces and accelerations are found. The examples cover this phase by merely presenting the free-body diagram and perhaps adding a few lines of explanation for any forces that they consider unobvious. As a consequence, most subjects simply accepted the free-body diagrams without trying to explain them, and thus missed the opportunity to learn. During problem solving, the diagrams were again overused. This time they subjects tended to use transformational analogy to adapt an example's free-body diagram instead of figuring one out from their knowledge of physics. Thus, they would miss the chance to do impasse-driven learning. This problem was exacerbated by the fact that most of the 25 problems were deliberately constructed so as to have free-body diagrams that were similar to ones in the examples. Subjects who used transformational analogy for these problems tended to get them right. This meant that subjects could learn very little and yet still get high scores. For instance, one subject who did not know about normal forces nonetheless got all of the "normal force" problems right. In short, it appears that overuse of the diagrams, exacerbated by the design of the study's problem set, reduces the number of cases of impasse-driven learning.

On the other hand, it could also be that subjects had instances of impasse-driven learning, but they showed no overt signs of them. Most of the cases of overt impasse-driven learning came from just two subjects who were the most vocal of the subjects. It is likely that the other subjects had episodes of impasse-driven learning but did not report them. We had hoped to detect these silent impasses by seeing changes in the subject's behavior. That is, we had hoped to see one or more occasions where the to-be-learned rule could be applied but was not, followed by a solid string of occasions where the rule was applied. The learning event would be somewhere in the vicinity of the transition from non-usage to usage. This type of analysis succeeded in locating silent learning events in Tower of Hanoi protocols (VanLehn, 1991a) and finger counting protocols (Siegler & Jenkins, 1989; Jones & VanLehn, 1991). Unfortunately, most subjects in this study either always used a rule or always avoided it. In some sense, they had to do this. Subjects in the two earlier studies learned new alternative strategies to solve a problem that they could already solve. Thus, if they did not use one of the to-be-learned rules, they always had their old rule to use instead. This was not the case in the present study. If a rule was missing for a new kind of force, for example, then the subject's only alternative to learning a new rule was to use a transformational analogy. Once they have successfully used transformational analogy for this appearance of the new force, they would tend to use it for all other appearances. In this fashion, they would miss the opportunity to learn new rules. In short, if a subject was to learn a rule, they tended to learn it on the first occasion that it was possible to learn it. If they used an alternative to the rule, then they tended to keep using that alternative through the end of the study. Thus, we found few transitions from not using a rule to using a rule, and we have little solid evidence for silent impasses.

In short, it appears that there is less learning in the protocols than we had hoped, although this might be partly an artifact of our inability to detect silent impasses. Nonetheless, there is much to be learned by examining the instances of learning that did occur. For instance, it would be good to find out if analogical search control really did play a role in guiding the subjects to an appropriate impasse.

Cascade is based on the assumption that the self-explanation effect is due solely to a difference in example-studying habits rather than a difference in prior knowledge. Surprisingly, this assumption held up even during the fitting of individual protocols. However, we expect some challenges to arise during the next set of analyses. It may be that the subjects' policies about using transformational analogy are just as important as self-explanation in determining whether learning will take place. We suspect that effective learning requires both that the subject explain an example and that they try not to refer to it during problem solving for purposes of obtaining a free-body diagram or an equation. On the other hand, referring to the example for advice on which rule to choose (analogical search control) should be encouraged. Methodologically, the complexity of this speculative prescription shows the advantage of simulation-based analyses of behavior. A prescription based on just the Chi et. al study would be simpler and perhaps not as effective.

Cascade shows promise as a general model of cognitive skill acquisition, but it needs considerable work beyond fixing its model of transformational analogy. In order to be a more complete account of the phenomena at hand, it needs a model of analogical retrieval and of the difference between physical and mental references to the examples. We believe the existing mechanisms can also handle some well-known phenomena of skill acquisition, such as practice and transfer effects, but this needs to be demonstrated. The major limitation on the generality of Cascade 3 is its use of monotonic reasoning. With the help of Rolf Ploetzner, we are currently incorporating a version of the situation calculus which will greatly enhance the types of reasoning Cascade can model, and thus the number of task domains that it can model. We are encouraged to extend Cascade to become a more complete, more general model of learning by its similarity to other theories of cognitive skill acquisition (e.g., Anderson, 1990; Schank, 1986). It is considerably simpler than those theories and probably more thoroughly implemented and tested. We hope that its simplicity and empirical adequacy remain intact as it is extended.

Finally, these results shed some light on the possibility of using machines to acquire knowledge for expert systems from ordinary human instructional material. Any AI expert would suspect that machines would have a hard time learning from human materials because they lack common sense. It turns out that common sense was important for Cascade's learning, but it was not particularly hard to provide it. Common sense was encoded in the non-domain knowledge given to Cascade as part of its initial knowledge base. Most of the non-domain knowledge concerned geometric reasoning, common-sense physical reasoning about pushes and pulls, and most significantly, overly general rules. This knowledge was used during EBLC to form new physics-specific domain knowledge. Hence, common sense knowledge was crucial because it heavily constrained learning. On the other hand, it was not particularly hard to figure out what that knowledge should be. Whenever Cascade would reach an impasse that it could not resolve with its existing common sense knowledge, it was usually quite simple to specify that knowledge. After all, it is common sense.

REFERENCES

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

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review, 94*(2), 192–210.

Anderson, J.R. (1990) *Adaptive Control of Thought.* Hillsdale, NJ: Lawrence Erlbaum.

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

Anderson, J. R., & Thompson, R. (1989). Use of analogy in a production system architecture. In S. Vosniadou & A. Ortony (Eds.), *Similarity and analogical reasoning.* Combridge, England: Cambridge University Press.

Bielaczyc, K., & Recker, M. M. (1991). Learning to learn: The implications of strategy instruction in computer programming. In L. Birnbaum (Ed.), *The International Conference on the Learning Sciences* (pp. 39–44). Charlottesville, VA: Association for the Advancement of Computing in Education.

Bundy, A., Byrd, L., Luger, G., Mellish, C. & Palmer, M. (1979) Solving mechanics problems using meta-level inference. In B. Buchanan (Ed.), *Sixth International Joint Conference on Artificial Intelligence* (pp. 1017–1027). Los Altos, CA: Morgan Kaufmann.

Carbonell, J. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.) *Machine Learning, An AI Approach: Vol. 2.* Los Altos, CA: Morgan-Kaufman.

Charney, D., Reder, L., & Kusbit, G. (1990). Goal setting and procedure selection in acquiring computer skills: A comparison of tutorials, problem-solving, and learner exploration. *Cognitive Science, 7*, 323–342.

Chi, M. T. H., de Leeuw, N., Chiu, M., & LaVancher, C. (1991). *The use of self-explanations as a learning tool.* Manuscript submitted for publication.

Chi, M.T.H., Bassok, M., Lewis, M.W., Reimann, P. & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, 13, 145-182.

Chi, M.T. H. & VanLehn, K. (1991) The content of physics self-explanations. *Journal of the Learning Sciences*, 1(1), 69-106.

Chi. M. T. H., VanLehn, K., & Reiner, M. (1988). *How are impasses resolved while learning to solve problems.* Paper presented at the 29th meeting of the Psychonomics Society, Chicago.

Dietterich, T.G. (1986) Learning at the knowledge level. *Machine Learning*, 1, 287-316.

Ferguson-Hessler, M.G.M. & de Jong, T. (1990). Studying physics texts: Differences in study processes between good and poor solvers. *Cognition and Instruction*, 7, 41-54.

Jones, R. M. (1989) *A model of retrieval in problem solving.* Doctoral dissertation. Information and Computer Science, University of California, Irvine.

Jones, R. M., & VanLehn, K. (1991). Strategy shifts without impasses: A computational model of the sum-to-min transition. *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society* (pp. 358–363). Chicago: Lawrence Erlbaum.

Larkin, J. (1983) The role of problem representation in physics. In D. Gentner & A. Collins (Eds.) *Mental Models*, Hillsdale, NJ: Lawrence Erlbaum.

LeFevre, J., & Dixon, P. (1986). Do written instructions need examples? *Cognition and Instruction, 3*, 1–30.

Lewis, C. (1988) Why and how to learn why: Analysis-based generalization of procedures. *Cognitive Science*, 12, 211-256.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., & Gil, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence, 40*, 63–118.

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

Novak, G. S., Jr., & Araya, A. (1980). Research on expert problem solving in physics. In T. Dietterich & W. Swartout (Eds.), *Proceedings, Eighth National Conference on Artificial Intelligence* (pp. 465–470). Los Altos, CA: Morgan Kaufmann.

Pirolli, P. (1987). A model of purpose-driven analogy and skill acquisition in programming. In *Proceedings of the Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Lawrence Erlbaum.

Pirolli, P. & Bielaczyc, K. (1989). Empirical analyses of self-explanation and transfer in learning to program. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society*, Hillsdale, NJ: Lawrence Erlbaum.

Reif, F. (1987). Interpretation of scientific or mathematical concepts: Cofnitive issues and instructional implications. *Cognitive Science, 11*, 395–416.

Reimann, P. (in press). Modeling active, hypothesis-driven learning from examples. In E. De Corte, M. Linn, H. Mandl, & L. Verschaffel (Eds.), *Computer-based learning environments and problem solving.* Berlin: Springer.

Schank, R.C. (1986) *Explanation Patterns: Understanding Mechani-*

cally and Creatively, Hillsdale, NJ: Lawrence Erlbaum.

Siegler, R. S., & Jenkins, E. (1989). *How children discover new strategies*. Hillsdale, NJ. Lawrence Erlbaum.

Sweller, J., & Cooper, G. A. (1985). The use of worked examples as a substitute for problem solving in learning algebra. *Cognition and Instruction, 2*, 59–89.

Sweller, J., & Levine, M. (1982). Effects of goal specificity on means-ends analysis and learning. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 8*(5), 463–474.

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

VanLehn, K. (1991a). Rule acquisition events in the discovery of problem solving strategies. *Cognitive Science, 15*, 1–47.

VanLehn, K. (1991b). Two pseudo-students: Applications of machine learning to formative evaluation. In R. Lewis & S. Otsuki (Eds.), *Advanced research on computers in education.* New York: North-Holland.

VanLehn, K., Ball, W. & Kowalski, B. (1990). Explanation-based learning of correctness: Towards a model of the self-explanation effect. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum.

VanLehn, K., Jones, R.M. & Chi, M.T.H. (in press). A model of the self-explanation effect. *Journal of the Learning Sciences*

VanLehn, K. & Jones, R.M. (in press). Integration of analogical search control and explanation- based learning of correctness. In S. Minton & P. Langley (Eds.) *Planning, Scheduling and Learning.* Los Altos, CA: Morgan Kaufman.

Ward, M., & Sweller, J. (1990). Structuring effective worked examples. *Cognition and Instruction, 7*, 1–39.

APPENDIX

This appendix contains an example of a subject's protocol encoded at the level of Cascade-like subgoals. The protocol appears in the left column, and the encoding appears in the right. An "(R)" in the left column indicates that the subject is reading the problem aloud. The coding procedure consisted of pretending that the subject was a version of Cascade, and generating the problem-solving trace that this version of Cascade would have to generate to lead to the utterances found in the protocol. In order for the subject's actions to fit into Cascade's control structure, it was sometimes necessary to hypothesize problem-solving goals that are not verbalized.

The Cascade model contains four types of goals:

- value(X): find a value for the quantity X.
- solve(X=Y): find a solution to the equation X=Y.
- retrieve(example): find an example similar to the current problem.
- equation(X): find an equation that can be used to compute a value for X.

Each goal appears with an "S:" when the goal is posted and an "F:" when a solution to the goal has been found. In addition, the model sometimes must explicitly "backtrack" over some subgoals to account for backtracking behavior by the subject.

The coding process allowed us to fit the hypothetical Cascade model as closely as possible to the subject's behavior. In doing this, we were able to identify specific locations at which the current implementation of Cascade would fail to generate the subject's behavior. At these locations, we would have to manually "nudge" the system onto the correct reasoning path. Events of this type are marked with event numbers in the right-hand margin.

This particular protocol concerns a "poor" subject solving a combination pulley-incline problem. The subject initially retrieves the pulley example and copies the equations for tension and acceleration from that example. For the most part, the subject attempts to directly apply these equations to the current problem (a strategy that will lead to an incorrect solution). The subject computes the tension of the string and the acceleration of the free-hanging block in this manner. However, to compute the acceleration of the block on the incline, the subject assumes his result for acceleration from the copied equations is actually the projection of an acceleration vector that points down the incline. The subject apparently does not have a rule for computing the projection of an acceleration vector, so he converts it to a force vector by multiplying the acceleration by the mass of the block ($F = ma$). He then computes the projection of the force vector and converts the result back to an acceleration with $F = ma$ again.

```
Problem: q5
Subject S103                                 Hypothetical Cascade model
--------------------------------------------------------------------------------
Okay.                                       |
(R) A block of mass m1 equals 3.0 slugs     |
on a sss...smooth                           |
incline plane of angle 30 degrees is        |
connected by a cord                         |
over a small frictionless pulley to a       |
second block of mass                        |
m2 equals 2.0 slugs hanging vertically.     |
                                            |Soughts: accel(m1), accel(m2),
                                            |         tension(cord)
(R) What is the acceleration of each        |S: value(accel(m1)), value(accel(m2))
body?                                       |
Okay.                                       |
That would just...                          |
Ahhh...                                     |
Okay, the acceleration...                   |
That would be M...with...that would be      | S: retrieve(example)
like the pulley                             | F: retrieve(example): px
again...this equation. (mumbles)            | S: equation(m1), equation(m2)
This one.                                   |  S: equation(m1)
                                            |   F: equation(m1): T-m1g=m1a
                                            |   S: equation(m2)
                                            |   F: equation(m2): T-m2g=m2a
                                            |  F: equation(m1), equation(m2):
                                            |       T-m1g=m1a, T-m2g=m2a
Okay.                                       |
Ummm...                                     |
Okay.                                       |
Wait.                                       |
If...                                       |
Find the tension of the whole...            |
Okay, I want to find the tension of the     |
whole thing again,                          |
and then I can find the acceleration        |
for each one, and use                       |
that to find force for ehh...for the        |
second...well I'll use                      |
it.                                         |
Okay, if I use that to find the             + S: solve(accel(m1)=              (1)
component of the force like                 |       Fn(proj(accel(m1)))),
for the F...                                |     solve(accel(m2)=
                                            |       proj(accel(m2)))
The Y component for this m1 block on        |  S: value(Fn(proj(accel(m1)))),
the slant, and find the                     |       value(proj(accel(m2)))
whole force, then I'll find the whole       |   S: value(T) [value(tension(cord))]
acceleration for that.                      |
Okay.                                       |
So that would be T equals 2 times M...      |    S: equation(T)
Two times mass one is 3.0 times mass        |    F: equation(T): ((2m1m2)/(m1+m2))g
two is 2.0 over mass one                    |    S: solve(T=((2m1m2)/(m1+m2))g)
plus two, is 3.0 plus 2.0...time            |     S: value(m1)
slugs...                                    |     F: value(m1):3.0
                                            |     S: value(m2)
                                            |     F: value(m2):2.0
                                            |     S: value(g)
                                            +      S: value(system(problem))     (2)
What system is slugs?                       +       S: value(system(slugs))      (3)
Where is that table again?                  |
Here.                                       |
Slugs is feet and all.                      |       F: value(system(slugs)):
So that would be...                         |            british
                                            |      F: value(system(problem)):
                                            |            british
```

```
Gravity would be 32 feet per second       |   F: value(g): 32
squared.                                  |
                                          |   S: solve(T=((2*3.0*2.0)/
                                          |             (3.0+2.0))*32)
                                          |    S: value((2*3*2)/(3+2))*32)
                                          |     S: value((2*3*2)*32)
                                          |      S: value(2*3*2)
Okay, so T equals 2 times 3 is 6.         |       S: value(2*3)
                                          |       F: value(2*3): 6
Time...12 ( ) is 36 times 2 is 12.        |       S: value(6*2)
                                          |       F: value(6*2): 12
                                          |      F: value(2*3*2): 12
12...time 32 equal...divided by 5.        |      S: value(12*32)
                                          |      F: value(12*32): 384
                                          |     F: value((2*3*2)*32): 384
                                          |     S: value(384/(3+2))
                                          |      S: value(3+2)
                                          |      F: value(3+2): 5
                                          |      S: value(384/5)
Equals 76.8.                              |      F: value(384/5): 76.8
                                          |     F: value(384/(3+2)): 76.8
                                          |    F: value((2*3*2)/(3+2))*32): 76.8
                                          |   F: solve(T=((2*3.0*2.0)/
                                          |             (3.0+2.0))*32): T=76.8
                                          |  F: solve(T=((2m1m2)/(m1+m2))g):
                                          |              T=76.8
Okay.                                     | F: value(T): 76.8
And the acceleration...                   | S: value(Fn(proj(accel(m1))))
For m1 the accel...                       + Backtrack                        (4)
I'll do m2 first.                         | S: value(proj(accel(m2)))
Acceleration for m2 is the lighter        |
one, would be...                          |
Ahh...T minus m1Gm2V equals m2A...        |  S: solve(T-m2g=m2a)
76.8 minus 2 times 32                     |       [a=proj(accel(m2))]
                                          |   S: value(T)
                                          |   F: value(T): 76.8
                                          |   S: value(m2)
                                          |   F: value(m2): 2
                                          |   S: value(g)
                                          |   F: value(g): 32
equals 2A.                                |   S: solve(76.8-2*32=2*a)
2A equals 76.8 minus 64 would be 12.8.    |    S: value(76.8-2*32)
                                          |     S: value(2*32)
                                          |     F: value(2*32): 64
                                          |     S: value(76.8-64)
                                          |     F: value(76.8-64): 12.8
                                          |    S: value(76.8-2*32): 12.8
                                          |    S: solve(12.8=2*a)
                                          |     S: solve(a=12.8/2)
                                          |      S: value(12.8/2)
A would be... 6.4.                        |      F: value(12.8/2): 6.4
                                          |     F: solve(a=12.8/2): a=6.4
                                          |    F: solve(12.8=2*a): a=6.4
                                          |   F: solve(76.8-2*32=2*a): a=6.4
                                          |  F: solve(T-m2g=m2a): a=6.4
And for...                                | F: value(proj(accel(m2))): 6.4
m1...                                     |
                                          | S: value(Fn(proj(accel(m1))))
T minus m1G equals negative m1A           |  S: value(proj(accel(m1)))
                                          |   S: solve(T-m1g=-m1a)
Will be 76.8 minus 3 times 32 equals      |           [a=proj(accel(m1))]
                                          |    S: value(T)
                                          |    F: value(T): 76.8
                                          |    S: value(m1)
                                          |    F: value(m1): 3
                                          |    S: value(g)
```

```
                                              |      F: value(g): 32
negative 3A...                                |      S: solve(76.8-3*32=-3*a)
That's 76.8 minus ( )...                      |       S: value(76.8-3*32)
76.8 minus 96 equals negative 19.2.           |        S: value(3*32)
                                              |        F: value(3*32): 96
                                              |        S: value(76.8-96)
                                              |        F: value(76.8-96): -19.2
                                              |       F: value(76.8-3*32): -19.2
Negative 3A equals negative 19.2.             |       S: solve(-19.2=-3*a)
                                              |        S: solve(a=-19.2/-3)
                                              |         S: value(-19.2/-3)
                                              |         F: value(-19.2/-3): 6.4
So A equals 6.4 again.                        |        F: solve(a=-19.2/-3): a=6.4
                                              |       F: solve(-19.2=-3*a): a=6.4
                                              |      F: solve(76.8-3*32=-3*a): a=6.4
                                              |     F: solve(T-m1g=-m1a): a=6.4
                                              |    F: value(proj(accel(m1))): 6.4
It's meters per...                            |
No it isn't.                                  |
It's feet per second squared.                 |
This is feet per second squared.              |
Okay, but that's just the vertical            |
part.                                         |
Cause it's not a slant.                       |
So you can find...                            |
Force equals ma..                             +    S: solve(F=ma) [a=accel(m1)]    (5)
                                              |     S: solve(a=F/m)
                                              |      S: value(F/m)
                                              |       S: value(F)
                                              |        S: solve(F=Fy/sin 30)
                                              |         S: value(Fy/sin 30)
                                              |          S: value(Fy)
And so it would be Fy equals mAy              |           S: solve(Fy=m*ay)
                                              |                [ay=proj(accel(m1))]
Fy equals mass is 3 slug times 6.4            |            S: value(m*ay)
                                              |             S: value(m)
                                              |             F: value(m): 3
                                              |             S: value(ay)
                                              |             F: value(ay): 6.4
                                              |             S: value(3*6.4)
equals 19.2 pounds.                           |             F: value(3*6.4): 19.2
                                              |            F: value(m*ay): 19.2
                                              |           F: solve(Fy=m*ay): F=19.2
                                              |          F: value(Fy): 19.2
                                              |
Okay, then F would equal Fy over the          |
sine of 30 degrees.                           |
fif...                                        |
Equal 19.2 over the sine of 30                |          S: value(19.2/sin 30)
degrees equals 19.2 divided by                |
30 degrees sine equals 38.4 pounds.           |          F: value(19.2/sin 30): 38.4
                                              |         F: value(Fy/sin 30): 38.4
                                              |        F: solve(F=Fy/sin 30): F=38.4
                                              |       F: value(F): 38.4
Okay, then.                                   |
Acceleration...                               |
F equals ma...                                |
38.4 pounds equals...mass is 3 slugs          |       S: value(m)
times A.                                      |       F: value(m): 3
                                              |       S: value(38.4/3)
                                              |       F: value(38.4/3): 12.8
                                              |      F: value(F/m): 12.8
A equals 12.8 feet per second squared.        |     F: solve(a=F/m): a=12.8
                                              |    F: solve(F=ma): a=12.8
                                              |   F: value(Fn(proj(accel(m1)))): 12.8
                                              |  F: value(Fn(proj(accel(m1))))),
```

```
                                      |        value(proj(accel(m2))):
                                      |                12.8, 6.4
                                      | F: solve(accel(m1)=Fn(proj(accel(m1)))
                                      |        solve(accel(m2)=proj(accel(m2))
                                      |                12.8, 6.4
                                      |F: value(accel(m1)), value(accel(m2)):
                                      |                12.8, 6.4
Okay, part B.                         |
(R) What is the tension in the cord?  |S: value(tension(cord))
I found that.                         |F: value(tension(cord)): 76.8
While I was doing part A.             |
And it equals 76.8 pounds.            |
--------------------------------------------------------------------------------
```

LEARNING SCHEMAS FROM EXPLANATIONS IN PRACTICAL ELECTRONICS

David E. Kieras

Technical Communication Program
University of Michigan
Ann Arbor, Michigan 48109-2108

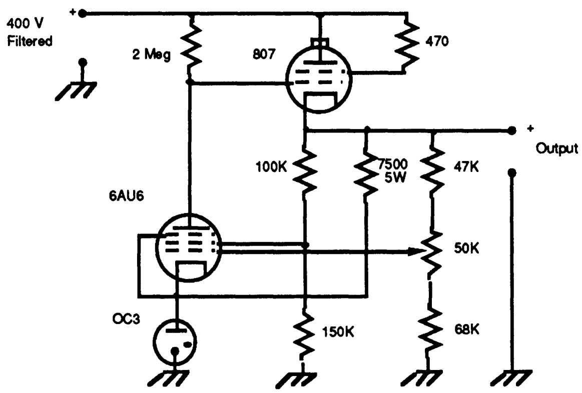
Abstract

Training materials in practical electronics appear to follow a building blocks approach in which common simple circuits are presented and then combined into more complex circuits. Each circuit is presented in the form of a circuit diagram and an explanation of how the circuit works in terms of a causal chain of events. Such materials suggest that learning electronics consists of learning schemas for the building block circuits; complex circuits can then be understood as combinations of these simpler schematic circuits. The process of learning appears to be based on extracting schemas from the explanations. This report presents human experimental results based on earlier artificial intelligence work in this project. Engineering students learned building block circuits and then learned complex circuits; the time required to understand the explanations and answer questions about the circuit behavior were compared to an AI system that learned from explanations and a model of question-answering. Generally, learning the schematic building block circuits facilitated performance, and the AI system and question-answering model could predict the amount of facilitation. However, the benefit of learning circuit schemas under these conditions was surprisingly mild.

INTRODUCTION

Explanations and Schemas in Electronics

Practical electronics textbooks, such as those used for training in the Navy (e.g., Van Valkenburgh, Nooger, & Neville, Inc., 1955), seem to be organized in terms of what can be called a *building blocks approach.* These textbooks present a series of circuit types, each of which performs a specific function, and which are then combined into more complex circuits. Each circuit is introduced with a diagram and explanatory text; the text typically explains how the circuit performs the stated function. Often the explanation takes the form of a causal chain of events that starts from some perturbation to the circuit, such as a change in the input signal, and finishes at a statement of the desired effect. Figure 1 gives an example of such a circuit and a fragment of the explanatory text from *The Radio Amateur's Handbook* (1961) .



When the load connected across the output terminals increases, the output voltage tends to decrease. This makes the voltage on the control grid of the 6AU6 less positive, causing the tube to draw less current through the 2-megohm plate resistor. As a consequence the grid voltage on the 807 series regulator becomes more positive and the voltage drop across the 807 decreases, compensating for the reduction in output voltage.

Figure 1. A sample circuit (a voltage regulator) and excerpt from the explanatory text.

These building blocks constitute *schemas* – each is a basic, frequently appearing unit; complex circuits can be analyzed into a hierarchy of these simpler circuits. The learner is supposed to learn each circuit schema by understanding its explanation, and then is expected to apply this new schema to understanding the subsequent more complex circuits. As an example of how more complex circuits can be viewed in terms of circuit schemas, Figure 2 shows a complex circuit parsed into schematic subcircuits. The behavior of this circuit as a whole can be understood by combining the behaviors of each of the schematic subcircuits.

The process of learning from such materials seems to be naturally explained in terms of how schemas are learned and applied, and how learning them can be done from explanations. Thus, this domain is a natural place to apply the concepts of schemas and explanation-based learning as they have appeared in psychology and in artificial intelligence.

The circuits studied in this work have been DC vacuum tube circuits, such as that shown in Figure 1. These are a good choice because: (1) They are well documented – this is a thoroughly mature technology with considerable instructional material having been written. (2) At this time vacuum tubes are unfamiliar to even technically sophisticated college students, so prior knowledge on the part of experimental subjects is less of a problem. (3) The DC circuits are very simple; circuits involving alternating current and multiple-state devices such as capacitors and inductors are much more complex (See Mayer, 1990).

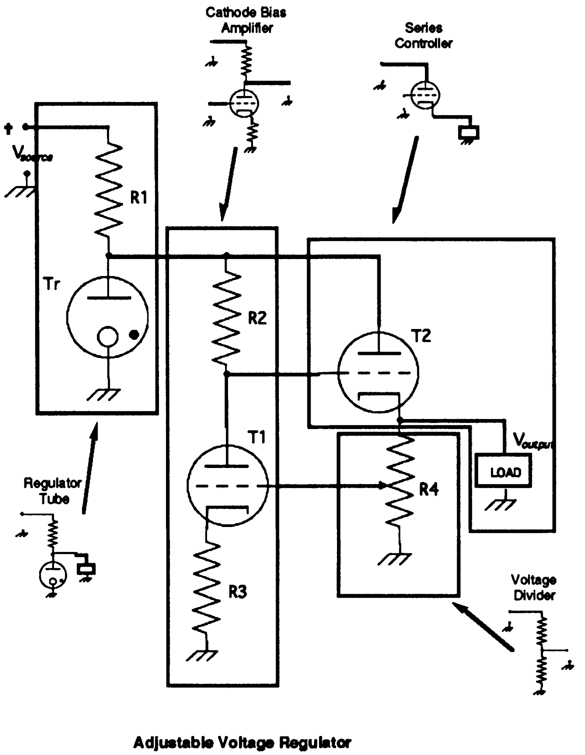


Figure 2. An analysis of a complex circuit in terms of schematic building block circuits. The basic form of each circuit schema is shown.

Judging from these training materials, research on such materials has value both for instruction and AI. The schematic building blocks approach must be instructionally important; there must be a shared belief that this is a good way to convey technical content. If we could understand how people learn from this approach, we could make better choices of the content and sequence of building blocks. There is also a possibility of automated knowledge acquisition for building AI knowledge bases. That is, considerable technical knowledge is already in textbooks which are complete enough for humans to learn technical domains from this material from explicit instruction. It is possible that large knowledge bases for AI systems could be constructed by developing mechanisms to read and understand such textbook knowledge.

The goals of the project responsible for the work in this chapter were

(1) to develop a simple AI system that learns from diagrams and text, and (2) compare it to human learning performance. The ultimate goal was to go on to explore the effects of the choice of content and sequence of explanations, but as will become more clear below, experimentation in this domain is very difficult, and some of the basic premises of this kind of learning can be called into question.

An AI System for Learning Schemas from Explanations

The AI system was developed by John Mayer as his dissertation (Mayer, 1990). Figure 3 shows the system organization and processing in Mayer's system. The AI system will not be described in detail because it is very thoroughly documented in Mayer (1990). The system is a combination of standard approaches to text comprehension, common sense reasoning, schema instantiation and construction, and explanation-based generalization. The AI system processes explanations using schemas, and then forms a schema for the new circuit described in the explanation.

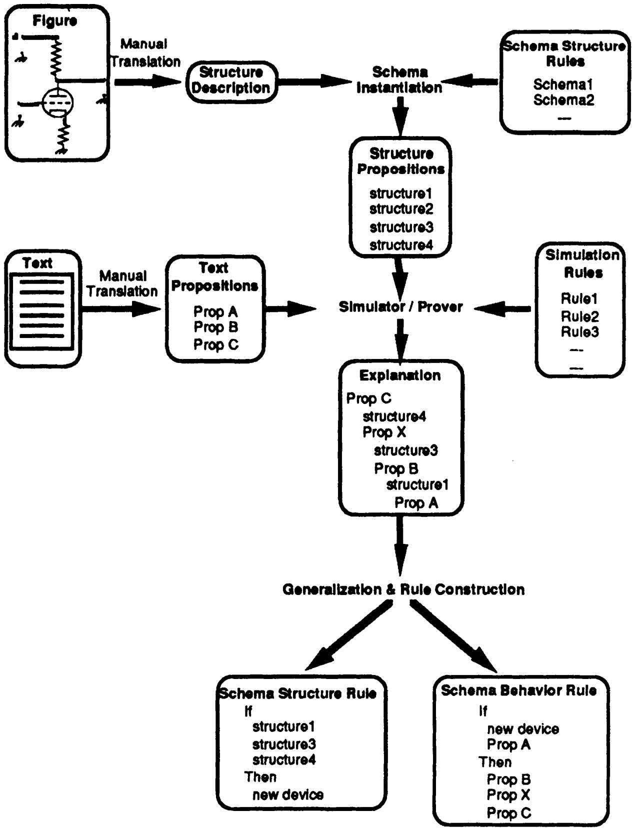


Figure 3. Organization and processing of Mayer's AI system for learning circuit schemas from diagrams and text.

The basic idea underlying the AI system is that an explanatory text constitutes an incomplete proof that the circuit performs the stated function. The explanation is a proof in the sense that it states a chain of causally-related events that concludes with the desired circuit behavior. The proof is incomplete because the explanation typically does not spell out each link in the causal chain that lies between the initial perturbation and final effect that corresponds to the circuit function; some of the intervening steps in the chain have been left out. Comprehending such a text thus consists of completing the proof, and learning from the text consists of generalizing the proof to extract a new schema for the circuit. The schema consists of two rules: one is a *structure rule* for when and how to instantiate the schema, based on the structure of the circuit; and the other is a *behavior rule* for how the schematic circuit behaves, given a triggering event. In Mayer's system, the first event described in the explanation is used as the trigger condition of the behavior rule. When this event happens, the rule is fired, and asserts all of the subsequent behaviors that appeared in the original explanation.

System processing. The process is shown in Figure 3. The system is presented with a diagram and the explanatory text. The diagram information consists of a hand-translated propositional description of the structure of the circuit. The schema instantiation process matches previously known schemas against the structure description and instantiates the appropriate schemas. The final result is a set of propositions about the circuit structure. The text information consists of hand-translated propositions about events in the circuit, listed in the order of appearance in the text. The simulator/prover component takes each text proposition and proves its truth in the circuit structure using the simulation rules. The simulation rules are either first-principle rules in the domain theory of basic electricity and electronics, or schema behavior rules from previously learned schemas. The simulator/prover does a forward simulation of the state of the circuit until it arrives at an event described by the circuit that matches the event described by the input text proposition. Then it moves on to the next input proposition, and repeats the process. When it has matched the last proposition in the text, the explanation has been completed. As shown in Figure 3, the simulator/prover may have had to infer other propositions to complete the proof, such as proposition X, which intervenes between the text propositions B and C.

The explanation is then used in a generalization process to arrive at the new structure and behavior rules. The schema instantiation rule is based on the portions of the circuit structure that were referred to in the proof. The behavior rule is formed by including the presence of the new schematic structure and the first event proposition (A) in the condition, and the assertion of all other event propositions (shown as B, X, and C) in the action.

Special properties of schema processing. It is important to note that the input and outputs of a schematic subcircuit are not distinguished, nor is the overall circuit strictly partitioned into the subcircuits. The substructure of the schematic circuit is still visible, and the simulation rule constructed for the schematic subcircuit contains all of the inferences made about the internal behavior of the schematic subcircuit. Thus, although the schema provides a short cut to the inferences about the circuit, reasoning from first principles can still go on, and behaviors internal to the schema can play a role in the reasoning.

The process of constructing the structure rule has a further important property. One function of an explanation in learning is to distinguish important features in the training instance from irrelevant ones. Thus, the only components included in a schema instantiation rule are the components that were referenced in the course of constructing the proof from the explanation. This principle had some odd effects in Mayer's system. The system recognized a cathode-biased amplifier as just the configuration consisting of R2 and T1 in Figure 4. As it happened, the original explanation of the cathode-biased amplifier schematic circuit did not involve the cathode resistor, which thus was not considered to be a mandatory component of a cathode-biased amplifier. In retrospect, this was probably not a good approach; a clearer picture of the nature of these circuits seems to result from taking the entire presented circuit as the structure of each schema. Mayer suggests that this approach would be justified by the fact that these circuits are designed to be economical, so each component that appears in the graphic accompanying the text must be necessary, regardless of whether the explanation contacts it or not.

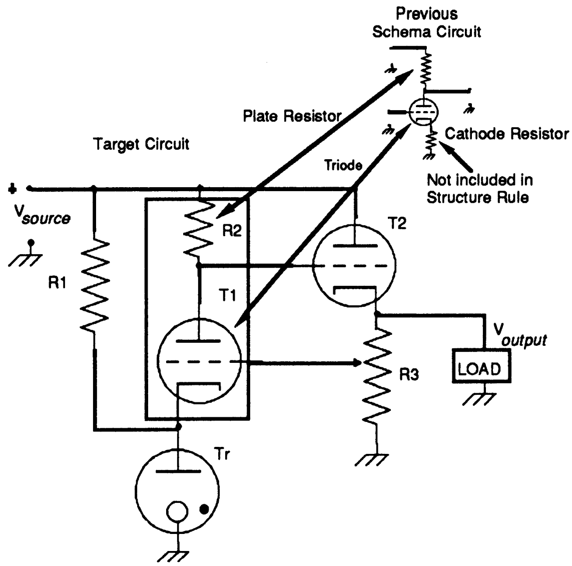


Figure 4. An example of how the AI system does not require a match of the complete building block circuit structure when instantiating a schema.

Benefits of schema availability. Mayer's system learns each circuit schema in terms of the already-acquired schemas. If relevant schemas have been previously learned, learning a new circuit is faster because the event propositions in the text can be proved sooner. The schema behavior rules will immediately add all of the schema inferences to the explanation, resulting in an earlier match to the text proposition to be proven. Thus, instead of the system having to construct the causal chain step-by-step using first-principle rules, the schemas will skip ahead to the end results. Thus the proof can be arrived at more quickly, and the simulator/prover should do less processing when applicable schemas have been previously learned.

Mayer demonstrated such an effect of schema availability for the set of materials diagramed in Figure 5. Starting with a basic voltage divider schema, the system studied and formed schemas for the building block circuits of a triode amplifier, series controller, regulator tube circuit, and cathode-biased amplifier. Then it processed explanations for a set of *target* circuits: a two-stage amplifier, a basic voltage regulator, a stabilized voltage regulator, and an additional circuit, the vacuum tube voltmeter circuit. Compared to learning the targets without the building blocks, the processing effort on the target circuits should be less if the system has already learned the building block schemas.

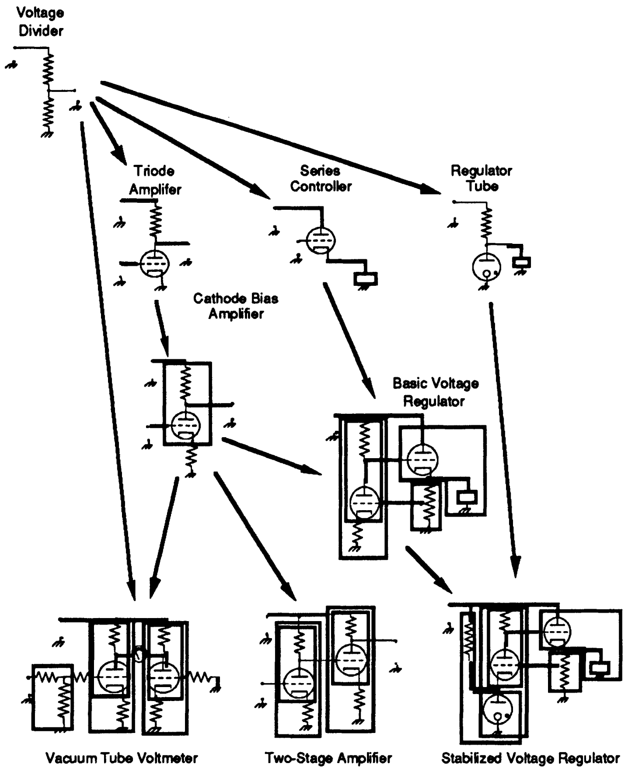


Figure 5. The schema relationships between circuits studied by Mayer and used in Experiment 1.

Mayer considered different measures of processing effort, some of which are relevant to AI technology concerns, such as CPU time. Mayer found that in terms of CPU time, the pattern matching required to instantiate schemas can overwhelm the savings from the faster processing of explanations. In addition, since even if a schema behavior rule applies, the system still makes first-principle inferences, and so can end up doing more overall processing when the schemas are available, resulting in a longer run time than when they are absent. But Mayer also considered a psychologically relevant metric, the number of *cycles* of forward simulation that had to be done while processing the explanation. Under this metric, the AI system is very similar to a production-rule cognitive model; each cycle of simulation consists of applying all of the simulation rules, both first-principle and schema rules, to deduce one set of new inferences about the circuit state. In most psychological production system models, it is assumed that the conditions of all the rules are matched in parallel, and so having additional rules in the system does not slow processing down. Thus the number of cycles performed by the AI system is a measure of processing effort which is not sensitive to underlying details of the implementation, and resembles common cognitive theoretical measures of processing time.

Figure 6 shows the number of cycles in the simulator/prover required to process the text event propositions for each of the target circuits. The Voltmeter circuit is a special case in Mayer's system. The explanation of the voltmeter takes the form of two descriptions of a steady state, rather than a description of how a change propagates through the circuit. Mayer's system therefore verifies the text propositions using its rules for reasoning about voltage relationships (see Mayer, 1990) rather than verifying using the simulation rules. Thus Mayer's system always requires zero cycles of simulation to process this circuit.

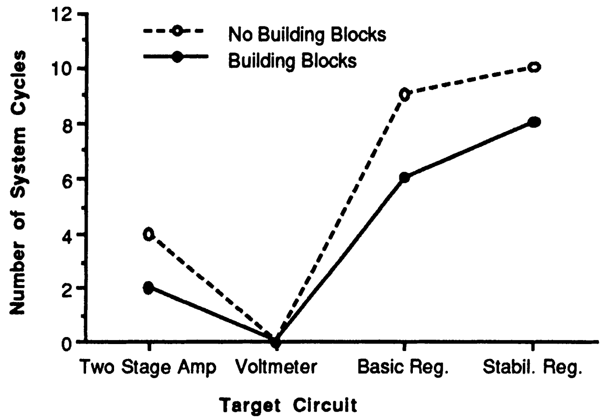


Figure 6. The number of cycles of processing needed by the simulator/prover to process each explanation.

But for the other three target circuits shown in Figure 6, the number of cycles is less when the schemas are available than when they are not. In addition, the system could learn a schema from the basic regulator circuit and apply it to the subsequent stabilized regulator circuit. Thus, the AI system can learn schemas from the explanations, and then can use these schemas to more quickly process future explanations.

Do Humans Use Schemas in Learning Electronics?

It helps the AI system to have schemas, at least measured in terms of a psychologically-relevant measure of processing effort. The psychological question is whether it helps people to have schemas. That is, can people understand the target circuits more easily if they have already learned the schematic building blocks? If so, the analysis of learning from these materials in terms of schematic building blocks would be verified, and the instructional value of the building blocks approach would be confirmed.

This question is especially relevant to the psychological theory of schemas. There has actually been very little evidence that schemas benefit acquisition processing, as shown by online measures, although this has always been claimed as an important advantage of schema knowledge (e.g., see Rumelhart, 1980; Rumelhart & Ortony, 1977). More generally, the basic claim that it is easier to learn about things one already has knowledge of is difficult to demonstrate (see Johnson & Kieras, 1983). But most of the empirical work demonstrating the use of schemas has been done in the context of recall or recognition paradigms. For example, subjects are asked to classify various stimuli, and then these classifications seem to be governed by schema or prototype representations. Or, subjects tend to make errors in memory for stories that are based on the assimilation of a story into a schema structure. But there are relatively few studies that actually demonstrate a benefit during online processing. For example, two such studies on reading comprehension time are Haberlandt, Berian, & Sandson (1980), and Graesser, Hoffman, and Clark (1980). Also, Kieras (1982) reported some results that suggest that devices seemed to evoke schemas immediately upon presentation, and subjects' descriptions of the presented device seemed to be organized in terms of schema knowledge for devices of that class. But clearly more evidence is needed that schematic knowledge has immediate processing time benefits.

Furthermore, it is not clear whether schematic knowledge can be effectively acquired and used in the time spans characteristic of classroom training of technical content. That is, within a single hour or a single day, which typically separates one lesson from the next, students are expected to form a schema for a particular type of circuit, and then are expected to apply this new knowledge to the next, more complex, circuit that they study. However, one common notion about schema knowledge is that it takes considerable exposure to develop a schema; this would accord with the general definition of a schema as being a *well-learned* familiar pattern or configuration of information. But the context of classroom study consists of brief, single, or few exposures to a concept, followed by its immediate use in a new context. The explanation-based learning work in artificial intelligence, and specifically Mayer's system, show that it is possible to acquire and make use of schemas in this single-exposure manner. But the question remains about whether this characterization is psychologically accurate. Two experiments will be described that seek to demonstrate a schema availability effect corresponding to that obtained by Mayer for the AI system.

EXPERIMENT 1

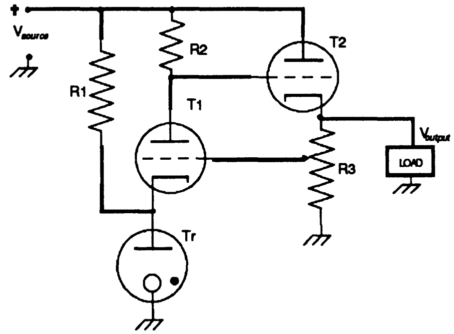
This first study was simply an attempt to determine whether providing building block information to learners would enable them to understand target circuits more readily. The experiment had three groups: the *No Building Blocks* group studied the target circuits without any prior study of the building blocks; the *Building Blocks* group studied the building block circuits before the targets; the *Descriptions* group studied the building block circuits and were given a description with the target circuits about how the schemas should be instantiated in the target circuits. The rationale for this third group was to ensure that these subjects not only know the schemas, but would also know how to apply them to the target circuits. After studying each circuit explanation, all subjects answered a set of multiple-choice questions about the circuit. The expected results were that learning and answering questions about the target circuits should be facilitated by having previously studied the building block circuits; the descriptions might produce further facilitation, depending on whether the Building Blocks subjects recognized and applied the schemas on their own.

Method

Materials and design. There were three groups. Each group studied the introductory training material. The No Building Blocks group then went directly on to study the target circuits. The Building Blocks and Descriptions groups studied the building block circuits and then went on to the target circuits. There was a deliberate confound of schema availability with the amount of practice (the number of circuits studied); this first study was simply to see if a schema availability effect would appear.

The training materials were based on actual textbook content, but were simplified in order to get a reasonable variety of circuits presented in a short amount of time. The training materials first reviewed the basic concepts of voltage, current, resistance, and voltage dividers, and then introduced the electronic components that were used in the circuits, such as resistors, variable resistors, voltage regulator tubes, and triode vacuum tubes. The building block and target circuits are shown in the Figure 5; the arrows connecting the circuits show how the circuits are assumed to be related in terms of their schema composition. The building block circuits were the regulator tube, basic triode amplifier, cathode-biased amplifier, and series controller circuits. The target circuits were the two-stage amplifier, voltmeter, basic voltage regulator, and stabilized voltage regulator.

Accompanying each circuit was a diagram which was always present during reading, and two pieces of textual information. The first piece was an introduction that gave the name and the basic function of the circuit, the schematizing description (if appropriate), and the static facts about the circuit, such as voltage relationships which were constant. The second piece contained the explanation of the behavior of the circuit; this was a series of sentences that started with a perturbation to the circuit and continued through to the final behavior corresponding to the circuit function. Figure 7 shows the diagram and introductory screen for a target circuit, and Figure 8 shows the explanation screen for the same target circuit.



Stabilized Vacuum Tube Voltage Regulator Circuit

This circuit maintains a constant output voltage, regardless of changes in either the load current or the source voltage.

R1 and Tr form a regulator tube circuit. R2 and T1 is a cathode bias amplifier with Tr serving as the cathode resistor. T2 is a series controller tube. The output of the regulator tube circuit is used as the cathode bias voltage of T1. The output of the amplifier is the input for the series controller circuit. The input of the amplifier is the voltage across the load, reduced by the voltage divider R3.

Figure 7. An example target circuit introduction.

Subjects. The subjects were engineering students without specific electronics coursework, but with background in electricity concepts. They had taken at least one physics course, and so were familiar with the basic concepts of voltage, current, resistance, electron flow and so forth, but they had not taken any courses specifically on electronic circuits. It was very difficult to selectively recruit such subjects; eventually eighteen in each of the three groups were obtained. Subjects were randomly assigned to groups.

Stabilized Voltage Regulator Circuit (continued) To Quiz

If more current begins to flow through the load, the output voltage begins to decrease. This makes the voltage on the grid of T1 more negative relative to the cathode, causing less current to flow through T1. The grid of T2 then becomes less negative, decreasing the resistance of T2, and thus keeping the output voltage the same. If the load draws less current, the output voltage rises, and the opposite effects occur. The variable resistor R3 is used to set the circuit for the desired output voltage by changing the grid voltage of T2.

If the source voltage decreases, the output voltage and the voltage on the grid of T1 will start to decrease. However, the voltage on the cathode of T1 is kept constant by the voltage regulator tube Tr. Thus, the grid of T1 will become more negative relative to the cathode, causing the grid of T2 to become less negative, and the output voltage to remain constant.

Figure 8. A sample explanation for a target circuit.

Equipment. The experiment was run on a Macintosh computer running SuperCard (a HyperCard-like program) on a two-page display. The first nineteen of the subjects were run using an ordinary Macintosh Plus computer; due to the small screen, the larger diagrams were on paper and constantly available to the subjects. The remaining subjects were run with a Macintosh IIx with a two-page display, with the circuit diagrams constantly present on the screen.

Procedure. The materials were divided into a series of segments in which subjects would study the material and then answer a set of quiz questions. They would go on to the next segment if they got all questions correct, or go back to reread the segment if not. This was intended to ensure that subjects understood the material before they went on. The initial material on basic electricity and each component consisted of several segments, and each of the building block and target circuits was a separate segment. The computer software recorded how long the subjects studied the introductory screen and the explanation screen for each target circuit, and the latency and answer to each question. The circuits were presented in a fixed order of increasing complexity, as shown in Table 1, which lists the training segments in the order presented. The experiment took 1 – 2 hours to complete.

Results

The most important results were those concerning the total time spent reading the explanations for the target circuits; this measure was most relevant to the AI system predictions, and due to problems in the experimental paradigm, was also the most reliable. See Kieras (1991) for a more complete presentation of the results.

The mean total reading time, shown in Figure 9, is the total time that subjects spent looking at the explanation screen, both the first time they read it and when rereading it after missing questions. This measure reflects both explanation difficulty and question difficulty. The main effect was marginal in the analysis of all three groups (F (351) = 2.26, p =

.11). For just the No Building Blocks and Description groups, the main effect just missed conventional significance (F (1, 34) = 4.04, p = .052). Total time tends to decrease with increasing schema availability; subjects that had the building block knowledge tended to spend less total time reading the explanation than subjects without the building blocks, and subjects given the schematizing descriptions were even faster. The stabilized regulator is studied for much less time than the basic regulator which was previously learned. The effect of schema availability is quite small for the two-stage amplifier.

Table 1
Order of presentation of materials in Experiment 1.

- Basic electricity
- Components
- Voltage Divider
- Building Blocks
  - Regulator Tube Circuit
  - Series Controller
  - Triode Amplifier
  - Cathode Bias Amplifier
- Target Circuits
  - Two-stage Amplifier
  - Vacuum-Tube Voltmeter
  - Basic Voltage Regulator
  - Stabilized Voltage Regulator

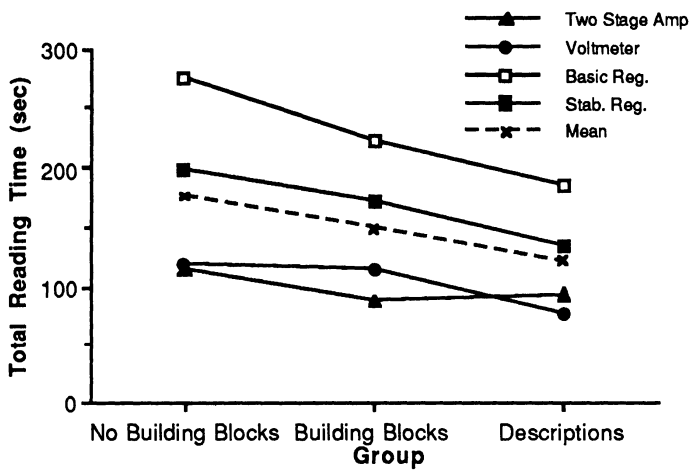


Figure 9. Mean time spent reading explanations, totaled over all rereadings, for each group and each circuit.

Comparison to the AI System

The number of simulation cycles performed by the AI system represents how much processing is performed, and so should be positively related to processing time for subjects (cf. Kieras,1984; Thibadeau, Just, and Carpenter, 1982). The number of simulation cycles was used as a predictor variable in a regression analysis with the total reading time as the predicted variable. Only the data for the No Building Blocks group and the Descriptions group (building blocks with schematizing descriptions) were used. This subset of the data corresponds most closely to the comparison in Figure 6.

Figure 10 presents the results of the regression analysis in a scatter plot showing reading time as a function of the number of system cycles. For clarity, the data points corresponding to the same circuit under the two different conditions are connected by arrows, with the tail of the arrow at the No Building Blocks condition and the the head at the Description condition. The line shown is for the regression equation :

*Reading Time (sec) = 59.5 + 8.7 *Cycles.*

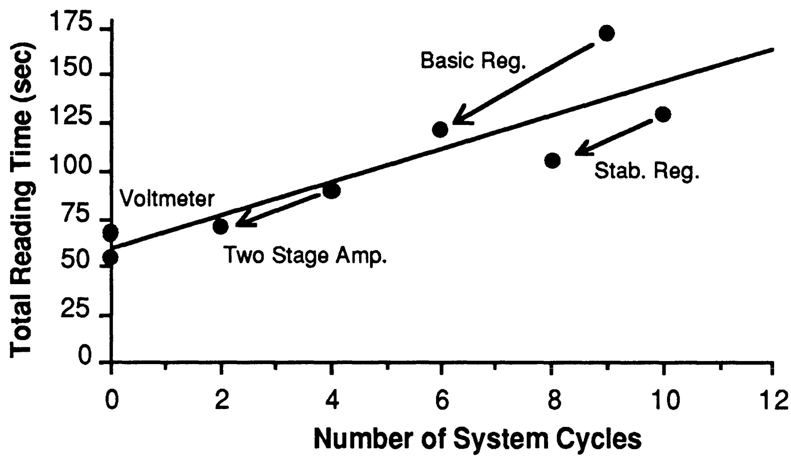


Figure 10. Scatter plot showing relation between AI system cycles and observed reading times. The arrows connect points for a circuit in the No Building Blocks condition (at tail of arrow) with the same circuit in the Descriptions condition (at head of arrow).

While there are only eight data points, 79% of the variance is accounted for, which is significant ($p < .05$.). The human total reading time and the AI system cycles depend on the amount of material processed in each explanation; typically simpler circuits take less processing than the more complex circuits, but the amount of processing also depends on the savings due to schemas from previous learning. The Description (at arrow heads) condition is normally faster than the No Building Block condition on the same circuit. Thus the AI system and human readers are clearly related in terms of the amount of processing they do on individual explanations, and in terms of the savings resulting from previous learning of schematic subcircuits.

Discussion

Problems with the experiment. The results of the experiment were problematic due to some problems in the materials and paradigm. The materials turned out to be fairly difficult for the subjects, even though they were relatively highly selected undergraduates in technical fields. The paradigm apparently allowed subjects to adopt a strategy of muddling through the experiment simply by attempting to answer the questions and if they got it wrong, going back and either rereading or just guessing again.

The experiment was designed under the assumption that the important data would be the time spent reading the explanatory material, and so the purpose of the questions was simply to encourage the subjects to read carefully. However, the subjects made many errors on the questions and so did considerable rereading of the explanations. Thus the reading times on the explanations are not very clean measurements of how difficult it was to understand the explanations. The questions were not very uniform in content or difficulty. Some of the questions could be answered simply by direct matches to the explanation text; that is, the subject could sometimes find a similar set of statements in the explanation, and answer the questions without making any deeper analysis of what was happening in the circuit. Other questions could be answered simply by reversing the statements made in the explanation, for example, by having a voltage increase rather than decrease. But a more subtle and interesting problem, discussed more below, is that the questions sometimes required reasoning which was not based on the circuit schemas overall behavior of a circuit schema, but rather required reasoning inside the schematic subcircuits.

Finally, some of the questions required the subject to remember some aspect of a component that had been presented early in the experiment and not mentioned subsequently. For example, the variable resistor was mentioned late in the questions, but was presented very early in the training. Since the experiment had a built-in confounding between whether the building blocks were present and how many circuits the subjects studied, this could have differentially affected the two groups. Many of the questions queried several intermediate states in the circuit, so that subjects had to verify the accuracy of a whole chain of events. These were very confusing, appearing to be a "word salad."

Summary of the results. The expected results were that the performance on the target circuits should be facilitated by increasing the schema availability; this result did appear, but the data is fairly noisy; they were both time and accuracy effects, and the effects were not uniform across circuits. There should be facilitation on the second regulator circuit due to schema transfer from the first regulator circuit, this appeared quite clearly. The amount of facilitation should have some correspondence to the AI system processing effort, this correspondence does seem to be present.

This first study shows strong suggestions of the expected effects, but clearly much cleaner data is needed for a definitive answer to the basic question of whether studying the schematic subcircuits improves the understanding of later more complex circuits. The materials and training used in this experiment were reasonable, though surprisingly difficult, but there were definite problems with the paradigm and the questions.

How applicable are schemas to the materials? A detailed examination of the materials and questions from the point of view of the AI system reveals a new issue, that of *schema applicability*. The questions and the explanations vary in the extent to which knowledge of the subcircuit schemas suffices to process the explanation or to answer the questions. At the level of the circuit itself, some of the target circuits parse cleanly into schematic subcircuit schemas while others do not. For example, as shown in the left panel of Figure 11, the basic regulator can be parsed into discrete subcircuit schemas, and the entire circuit can then be simplified by replacing each circuit with a "black box" for each subcircuit, as shown in the right panel of the figure. The circuit behavior is then just the composition of the behavior of the black-box subcircuits. In contrast, as shown in Figure 12, the stabilized voltage regulator cannot be parsed completely into discrete subcircuit schemas; the triode T1 does not correspond to a cathode-biased amplifier. Thus there is a fundamental problem with the materials; in some of the circuits, the schemas are not fully applicable. It would seem that the benefit of learning the schematic subcircuits would be greater if the circuits could be understood in terms of the black-box behavior of the schematic subcircuits.

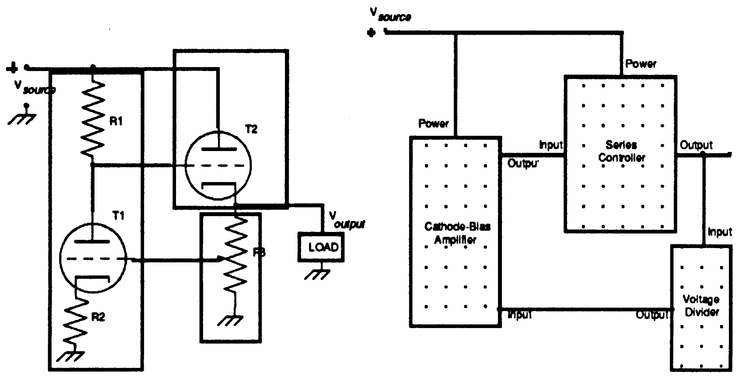


Figure 11. An example of a circuit that can be fully parsed into schema subcircuits.

The explanations also did not fully make use of schema knowledge that subjects might have. That is, the explanations often were in terms of events that would happen "inside" the schemas, instead of treating the schemas as black boxes. For example, in the explanation for the two-stage amplifier (see Figure 13), the line of reasoning is that if the input voltage increases, the cathode-to-plate resistance of T1 goes down, and the resistance of T2 goes up. But in the original explanation of the amplifier schema circuit, the change of the cathode-to-plate resistance of the triode is a subsidiary event; the black box behavior of the amplifier is that if the input voltage changes, the output voltage changes in the opposite direction and by a larger amount.

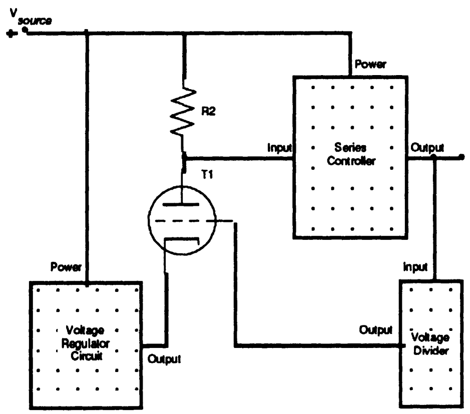


Figure 12. A circuit that cannot be fully analyzed in subcircuit schema, assuming that schemas require complete structure matches.

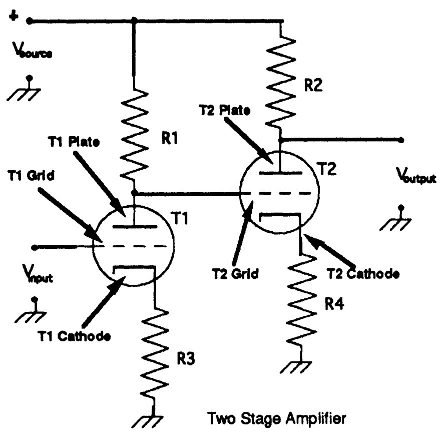


Figure 13. Reasoning based on an amplifier schema in the two-stage amplifier would refer only to voltages on the plates and grids of the triodes, not the cathode-to-plate resistance changes.

Thus, with the presented explanations, if subjects knew the subcircuit schemas, they usually could not simply shortcut their analysis of the explanation, but instead would have to verify each behavior of the circuit mentioned in the explanation, which included internal schema events. For this reason, when Mayer's AI system instantiated a schema and triggered the behavior rule, it simply added all of the internal propositions to the system's knowledge base, which could result in immediate verification of schema-internal events contained in the explanation. But what would happen if the explanations were simply in terms of the black box behavior of the schema circuit? For example, in the same two stage amplifier circuit (Figure 13), a purely black-box schema line of explanation would be that if the input voltage increases, the voltage on T1's plate goes down, and the voltage on T2's plate goes up. In this case, a reader with schema knowledge could simply verify the main behaviors predicted by the schemas, whereas a person without the schema knowledge would have to make the individual inferences required to go from the input voltage change to the final output voltage change. Thus there should be a larger benefit of schema knowledge, if the explanations could be understood directly in schema terms. In a similar way, the questions used in the experiment often involved reference to events happening "inside" the schemas. For example: if the input voltage goes up, what happens to T2's resistance? Again one would predict a larger benefit of having schemas if the questions were posed strictly in terms of the schematic behavior of the circuit. For example: if the input voltage goes up what happens to T2's grid voltage?

Can a circuit be understood in terms of black-box schemas, or do the schemas have to be unpacked into internal structure and behavior? The AI system does not black-box the schematic subcircuits, but also suffers from not doing so. On the other hand, schemas would seem to be most valuable if the schematic subcircuits can be treated as black boxes and reasoning done about a larger circuit only in terms of the external input/output behavior of the subcircuits. Thus the value of circuit schemas may depend on the extent to which the circuits, explanations, and questions involve black-boxed schemas versus reasoning about events inside the schemas.

EXPERIMENT 2

The basic problem with the first study is that the presence of the building blocks was confounded with the amount of practice (number of circuits studied) that subjects received. There was also the problem with the experimental paradigm that allowed subjects to adopt a strategy of guessing their way through the questions, and not enforcing a careful reading of the circuit explanation on the first try. In addition, some of the circuits and questions may have been too difficult for the subjects, further encouraging them to guess repeatedly. Also, as mentioned above, the circuits, explanations, and questions may not have allowed subjects to take full advantage of having schema knowledge, and so the benefit of studying the schema circuits may have been weakened.

The second study was designed to produce a clean and definitive effect of the availability of schemas. This was done by rewriting all of the materials used in Experiment 1 and adding new circuits, so that as much as possible, the schema knowledge was fully applicable.

Method

Design and materials. The experiment had two groups. The *Irrelevant Building Blocks* group studied building block circuits that were irrelevant to the later target circuits; these could not be instantiated as subcircuits in the target circuits. The *Relevant Building Blocks* group studied the same schematic subcircuits as in the first experiment, which were then instantiated in the target circuits. The target circuits were changed so that they could all be rewritten in the form of a black box parse with the building block circuits. However, there is much less variety in the circuits than in Experiment 1. Figure 14 shows the irrelevant building blocks, the relevant building blocks, and the targets, with the arrows connecting the relevant building blocks to their instantiation in the target circuits.

In addition, both groups received a description in the introductory screen, similar in format to the schematizing description in Experiment 1, thus controlling for the presence of prominent additional information. The Relevant Building Blocks group received a schematizing description as in the first study, in which the circuit is described in terms of the subcircuit schemas. The Irrelevant Building Blocks group got an irrelevant description, containing a statement of a correct technical fact or aspect of the circuit, but which had no bearing on the explanation

Figure 15 shows an example explanation that illustrates how the chain of events always referred to the input/output behavior of the circuit schemas, and an example question that illustrates the homogeneous form of the questions used in this experiment. The question presents a perturbing event, and the answers are a choice of a voltage that either increases, decreases, or stays the same.

Procedure and apparatus. The overall paradigm was very similar to that in Experiment 1. During the training portion of the experiment, subjects were required to repeat the material and questions on basic electricity, components, and the building blocks, until they answered all questions correctly. But during testing on the target circuits the subjects were allowed only one try on each question. The subjects were warned very explicitly that they would be allowed to read the target circuit explanations only once, and would not get a chance to answer a question again if they got it incorrect. The basic measures were the time to study the target circuit introduction and explanation, and the latency and answer for each question. As in Experiment 1, a Macintosh IIx with two-page display was used. The experiment required 1 -2 hours to complete.

Subjects. As in Experiment 1, the subjects were engineering undergraduate students who had studied electrical concepts in physics courses, but who had not taken any specific coursework in electronics.

Again recruiting subjects was very difficult, but fifteen were obtained in each group.

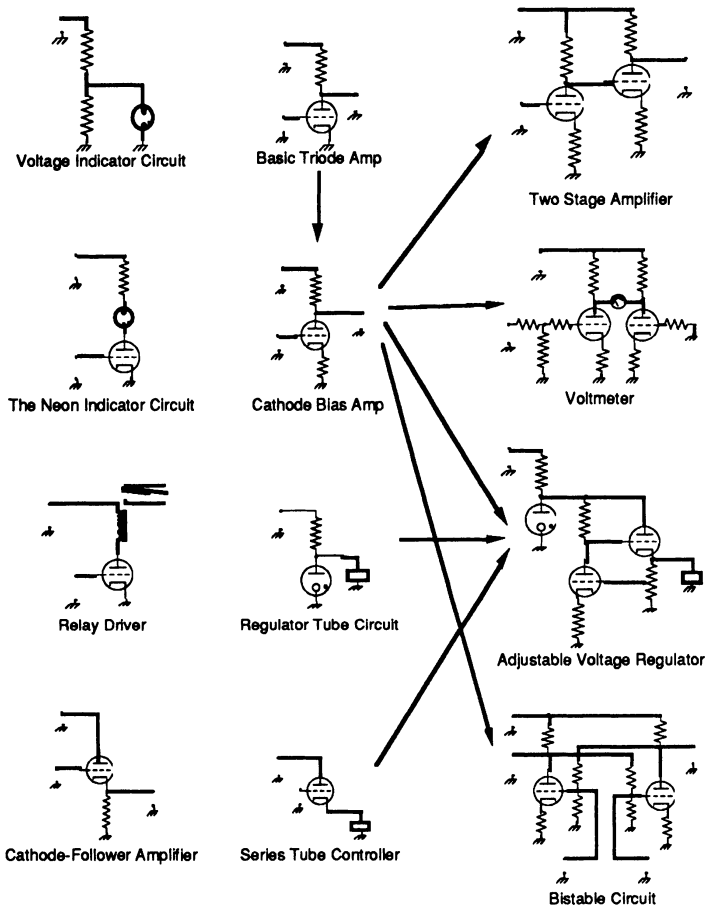
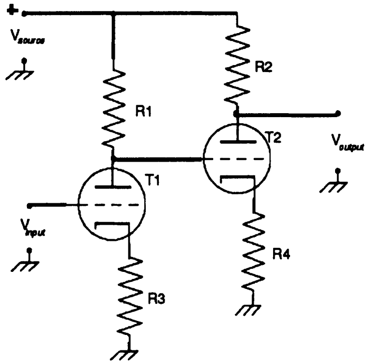


Figure 14. Circuits used in Experiment 2. The irrelevant building blocks are shown at the left; the relevant building blocks are in the center, with schema relations shown by arrows to the target circuits on the right.

Results

Like Experiment 1, the data were very noisy, and performance was poor; 40% of the subjects got two thirds or more of the questions incorrect on the target circuits. In some of the statistical analyses to be reported, either data from these poor subjects, or times for questions that were answered incorrectly, were removed from the analysis.



If the input signal, Vinput, increases, the voltage on the plate of T1 and the grid of T2 goes down, causing the output voltage, Voutput, to increase. The changes in plate voltage of T1 are much greater than the changes in Vinput, and the changes in plate voltage of T2 are even greater. Thus, the total amplification effect is much greater than for a single triode.

If V_{input} decreases, what happens to V_{output}?
• Increases
• Decreases
• Stays the same

Figure 15. A sample explanation and question from Experiment 2.

Figure 16 shows the mean time spent on the introduction screen for the four different circuits. Surprisingly, more time was spent on the introduction screen if the subjects had studied the relevant building blocks. The main effect was nonsignificant ($p > .13$), but the interaction of circuit with condition was significant (F (3, 84) = 7.32, p = .002). The effect is present on most of the circuits. Removing poor subjects from the analysis produces a significant main effect (F (1, 16) = 5.05 p = .037) and interaction (F (3.48) = 5.04, p = .004), with an overall mean of 65 sec on the irrelevant building blocks, and 86 sec on the relevant building blocks. This effect is reminiscent of the disadvantage of schemas in Mayer's system being the extra computation time required to instantiate them.

Figure 17 shows the mean time spent on the explanation for each circuit. There is an overall trend in the desired direction, in that the mean time for subjects who studied the relevant building blocks is less than those who studied the irrelevant ones. However both the main effect and interaction are nonsignificant ($p > .2$) and removing poor subjects does not improve the statistical situation. Comparing this figure with the reading time figures from the first experiment (Figure 9) shows that the overall time spent on reading the explanations is substantially less than the times spent on the explanations in the first experiment. Perhaps again, subjects were not reading carefully, or perhaps these explanations are much simpler than those in the first experiment, resulting in a ceiling effect.

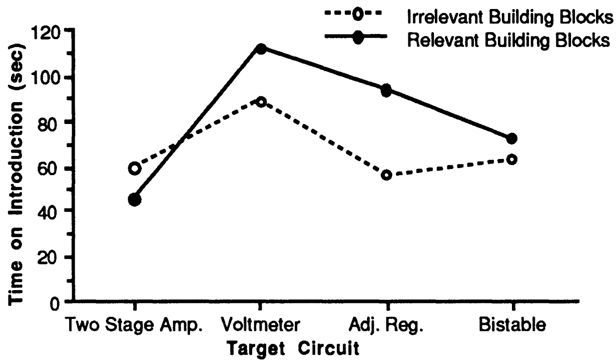


Figure 16. Mean time spent reading the introduction screen for each circuit and each group.

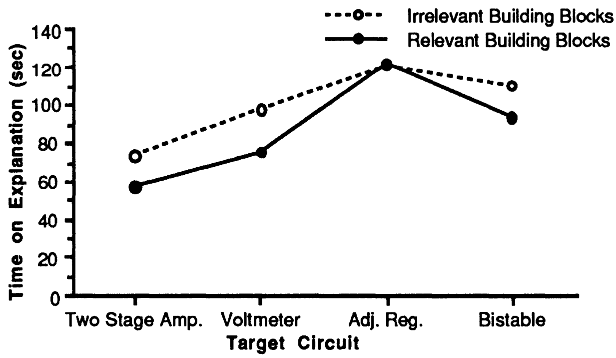


Figure 17. Mean time spent reading the explanation in each group.

The mean proportion of questions answered correctly for each circuit for the two groups showed no effect of schema availability (61% vs. 59%). Notice that the level of accuracy is fairly low; the questions had three alternatives, so the chance level of performance would be 0.33, but one of the alternatives would often be easy to eliminate (i.e., the choice *stays the same*). Thus while the average level of accuracy is greater than chance, it is not impressively so.

Figure 18 shows the latency of choosing the question answers, averaged over both correct and incorrect answers. The main effect is nonsignificant ($p > .1$), but the interaction is significant (F (3, 84) = 3.68, $p = .016$); the effect appears for all but the bistable circuit. Removing poor subjects does not change the situation statistically, and the same overall pattern appears if incorrect question times are removed from the analysis as well. If the bistable circuit is not included, then the main effect is significant (F (1, 28) = 4.84, p = .036). The question of why the bistable circuit is different is interesting – perhaps the greater graphic complexity kept subjects from seeing it in terms of schemas, regardless of the schematizing description

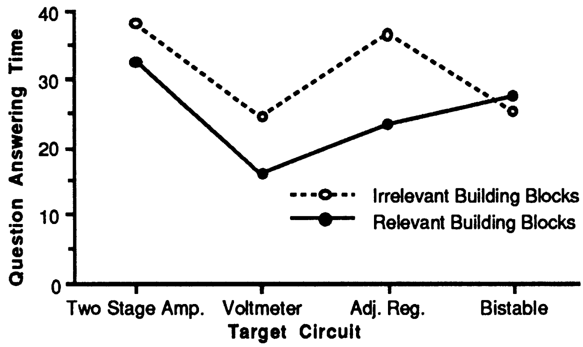


Figure 18. Mean time required to answer questions averaged over both correct and incorrect answers.

Discussion

It is clear that the methodological problems of experimentation in this domain are still unsolved. Apparently the subjects can not be depended upon to read the explanations carefully enough, or to perform well in answering the questions. Rather than an effect on the time spent processing the explanation, there is an effect on the time spent answering the questions, and this effect depends strongly on the circuit involved. Since the major effect of schema availability in this experiment is on question answering times, the issue is now whether this effect can be explained by mechanisms for using schemas during answering the questions. The next section presents a simulation model for schema use in answering questions.

A MODEL FOR SCHEMA-BASED QUESTION-ANSWERING

Overview of the Model

The model is described in more complete detail in Kieras (1991). The model is an ACT-class model, consisting of declarative knowledge represented with propositions, and procedural knowledge represented with production rules (see Anderson, 1983), and is similar to the simulation of a mental model for a simple device described in Kieras (1988). The circuit structure is represented with propositions assumed to be available constantly from the diagram, while the state of the circuit is represented with propositions in working memory. For convenience, shorthand notation is used for the propositions: no claim is being made of a specific propositional representation notation.

The production rules in the model "run" the mental model, performing the inferences and controlling the processing. The rules of most interest are those that represent the first principles in the domain theory, and those that perform schema recognition and schema-based inferences. The basic approach in the model is as follows: The question states a perturbation or change, such as to the input of the circuit. The model propagates the change through the circuit, and waits for a proposition that answers the question to appear in working memory. To simulate the Irrelevant Building Blocks condition, the rules for instantiating and making use of the subcircuit schemas are disabled; the Relevant Building Blocks condition is simulated by enabling the schema rules.

Before a comparison with the data was made, two models were developed that reflect two different overall processing strategies. The stages of model processing in both models are: (1) instantiate any schemas that might be present, (2) analyze the voltage relationships in the circuit, (3) accept the input, (4) propagate the changes until the processing is completed, (5) determine the answer. In the *terminating model,* flowcharted in Figure 19, the process of propagating the changes is terminated as soon as a proposition answering the question appears in working memory.

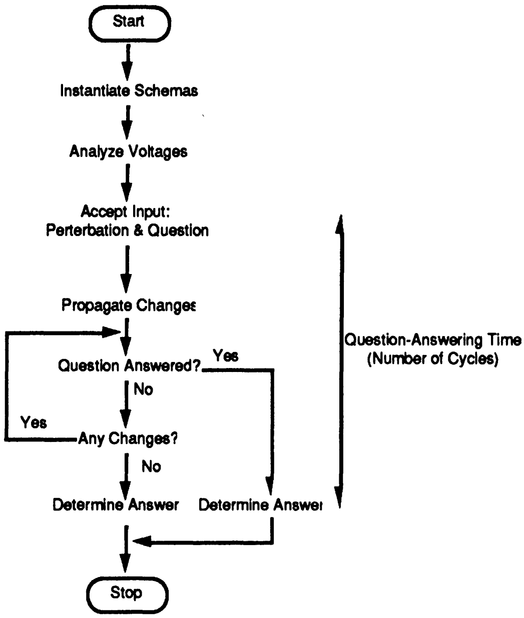


Figure 19. Flowchart of processing in the terminating model.

If no more changes can be propagated, the question is answered using whatever propositions are available. This is the typical case if the correct answer to the question is that a voltage stays the same. The predictor of the time to answer the question is the number of production-system cycles that elapses between when the input is accepted and when the answer is determined (shown in Figure 19). According to the terminating model, there should be a large benefit of having schema knowledge, because the changes will propagate more rapidly and the answer will be computed faster than if schemas are not available.

The exhaustive model, flowcharted in Figure 20, continues the propagation of the input change until quiescence (no more changes propagated), whereupon the question is answered using the available propositions about the circuit state in working memory. The predictor for the time to answer the question is again the time between when the input is accepted and when the answer is determined (shown in Figure 20). This model predicts a mild effect of schemas, because the model may well spend many cycles propagating irrelevant changes long after the answer to the question has been determined.

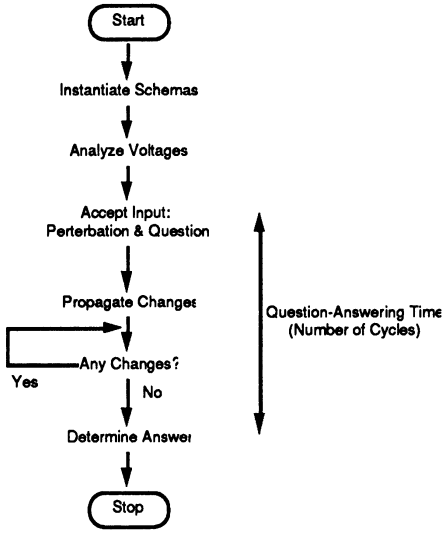


Figure 20. Flowchart of processing in the exhaustive model.

Model Details

Each individual component (which includes the input and output terminals) is described with simple ISA and HAS propositions, while a shorthand notation is used to describe basic voltage and resistance properties of the circuit or its components. The connections between the components are described by a series of CONNECTION propositions; note that each connection is one-way, so connection propositions in both directions are necessary.

Some examples of production rule will be given. Table 2 shows the production rule used to recognize the presence of a cathode-biased amplifier in a circuit. The production system used is the PPS system (see Covrigaru and Kieras 1987). In this notation, the *clauses* following the `IF` must all be present in the production system data base in order for the rule to fire, whereupon the actions listed after the `THEN` are taken. PPS has no built-in conflict resolution or refractoriness mechanism; each rule must contain condition clauses to ensure that it fires only at the right times. The condition of this sample rule is fairly elaborate; it consists mainly of a description of the components and their connection pattern that make up the structure of a cathode-biased amplifier. In the PPS notation, an item preceded by a question mark in a clause, as in `(ISA ?T TRIODE)`, represents a variable that is assigned a value when the condition is matched. If this rule finds matches in a target circuit a particular triode and two resistors which are connected as required, then it adds to working memory a proposition (in shorthand) that there is a cathode-biased amplifier based on the triode, and assigns the grid and plate of the triode as the input and output *ports* of the schematic subcircuit. It also describes various parts of the schematic subcircuit as being parts of the schema instantiation. Thus, if given the two-stage amplifier circuit shown in Figure 13, the rule will fire and deposit in working memory propositions showing the presence of a cathode-biased amplifier schema based on T1, and another based on T2. The negated clauses (using `NOT`) in the condition prevent the rule from firing more than once for each instantiation.

Table 3 shows a rule for propagating a change through an individual component, the triode vacuum tube. The clauses in the condition of this rule recognize the presence of a triode and the event in working memory that the voltage on the grid of the triode has increased. The rule adds to working memory the information that the resistance between the plate and cathode of the triode has decreased.

Table 4 shows the schema behavior rule for a cathode-biased amplifier, which would have been earlier recognized by the rule shown in Table 2. This rule is remarkably simple; if the cathode-biased amplifier has been instantiated with designated input and output ports, and the voltage at the input port has increased, then this rule simply adds to working memory the proposition that the voltage at the output port has decreased. The inference that this change is larger than the input change is not relevant in these materials, and so is not included.

Table 2
Sample schema instantiation rule.

```
(RecognizeCathodeBiasAmplifier
IF (
    (GOAL PREPROCESS CIRCUIT)
    (STRATEGY RECOGNIZE SCHEMAS)
    (ISA ?T TRIODE)
    (HAS ?T ?T-PLATE)
    (ISA ?T-PLATE PLATE)
    (ISA ?T-CATHODE CATHODE)
    (HAS ?T ?T-CATHODE)
    (ISA ?T-GRID GRID)
    (HAS ?T ?T-GRID)
    (CONNECTION ?R1-PORT2 ?T-PLATE)
    (HAS ?R1 ?R1-PORT1)
    (ISA ?R1 RESISTOR)
    (CONNECTION ?T-CATHODE ?R2-PORT1)
    (HAS ?R2 ?R2-PORT1)
    (ISA ?R2 RESISTOR)
    (HAS ?R1 ?R1-PORT2)
    (HAS ?R2 ?R2-PORT2)
    (CONNECTION ?R2-PORT2 GND)
    (CONNECTION ?HOT-PORT ?R1-PORT1)
    (ISA ?HOT-PORT VOLTAGE-SOURCE)
    (NOT (SCHEMA CATHODE-BIAS-AMPLIFIER ?T ?T-GRID ?T-PLATE))
)
THEN (
    (ADDDB (NOTE CIRCUIT PREPROCESSED))
    (ADDDB (COMMENT CATHODE-BIAS-AMPLIFIER AT ?T ?R1 ?R2))
    (ADDDB (SCHEMA CATHODE-BIAS-AMPLIFIER ?T ?T-GRID ?T-PLATE))
;?T used as label for schema instantiation
    (ADDDB (SCHEMA PORT ?T ?T-GRID))    ;input
    (ADDDB (SCHEMA PORT ?T ?T-PLATE))   ;output
    (ADDDB (SCHEMA PORT ?T ?R1-PORT1))  ;power
    (ADDDB (SCHEMA PART ?T ?R1-PORT2))
    (ADDDB (SCHEMA PART ?T ?T-CATHODE))
    (ADDDB (SCHEMA PART ?T ?R2-PORT1))
))
```

Table 3

Sample change propagation rule for a component.

```
(TriodeGridVoltageChangeIncrease
IF (
(GOAL PROPAGATE CHANGE INFER)
(ISA ?T TRIODE)
(HAS ?T ?T-PLATE)
(HAS ?T ?T-CATHODE)
(HAS ?T ?T-GRID)
(ISA ?T-PLATE PLATE)
(ISA ?T-CATHODE CATHODE)
(ISA ?T-GRID GRID)
(WM CHANGE INCREASE VOLTAGE ?T-GRID)
(NOT (WM CHANGE DECREASE RESISTANCE BETWEEN ?T-PLATE ?T-CATHODE))
(NOT (SCHEMA PORT ??? ?T-GRID))   ;apply only if ?T not schematized
)
THEN (
(ADDDB (NOTE CHANGE PROPAGATED))
(ADDDB (WM CHANGE DECREASE RESISTANCE BETWEEN ?T-PLATE ?T-CATHODE))
))
```

Table 4

Sample behavior rule for a schema.

```
(CathodeBiasAmplifierInpIncrease
IF (
(GOAL PROPAGATE CHANGE INFER)
(SCHEMA CATHODE-BIAS-AMPLIFIER ?SCHEMA ?INP ?OUT)
(WM CHANGE INCREASE VOLTAGE ?INP)
(NOT (WM CHANGE DECREASE VOLTAGE ?OUT))
)
THEN (
(ADDDB (NOTE CHANGE PROPAGATED))
(ADDDB (WM CHANGE DECREASE VOLTAGE ?OUT))
))
```

Comparison of the Model with Question-Answering Time Data

The processing time predictions of exhaustive and terminating models were compared by regression analysis to the mean times averaged over subjects taken to answer the *individual questions*. The average times included only times from the correctly answered questions. In addition, the questions whose accuracy was not above one-third (chance level) were also dropped from the comparison, giving $N = 19$. This subset of data is still quite noisy; although the questions are simple, the latencies are very long and variable as is usually the case with problem-solving latencies. Using ipsatized data in the comparisons did not result in substantially cleaner results, suggesting that most of the noise in the data is within-subject.

The predictor variable is the number of production system cycles required to answer the question starting from when the input is accepted and the propagation of the change begins, and stopping when the question answer is determined. These numbers were obtained by simply running the model on each combination of circuit and question, and calculating how many cycles were required until the answer was determined. A regression was then computed for each model using the number of cycles as the predictor variable, and the observed mean question-answering times as the predicted variable (see Kieras, 1984). Clearly the regression slope should be positive, in that more cycles should correspond to more time, and r^2 gives a measure of the goodness of fit.

An important result of the comparison is that the terminating model fails completely to account for the data. Figure 21 shows the relationship between the number of cycles required by the terminating model and the question answering time. Although this model is intuitively appealing, it clearly accounts for essentially none of the variance ($r^2 = .02$). While the utter failure of the model is discouraging, it does demonstrate rather clearly that the comparison of these models to the data is a valid exercise; a perfectly reasonable model can be disconfirmed, so one that accounts for a substantial part of the variance can be taken seriously.

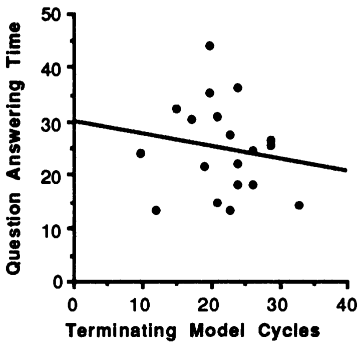


Figure 21. Scatter plot of cycles versus question-answering time for the terminating model.

Figure 22 shows the relationship between the number of exhaustive model cycles and the question answering time. The regression equation is:

$$\textit{Time (sec)} = 5.38 + 0.63 * \textit{Cycles}$$

This model accounts for a significant portion of the variance ($r^2 = .34, p < .01$). Accounting for 34% of the variance is impressive, considering that the data are quite noisy. The regression coefficient for the number of cycles is approximately 0.6 sec. There is reason to believe that production rules should take on the order of 50-100 msec per cycle to apply in the context of simple procedural tasks (see Card, Moran, and Newell, 1983, Ch.. 2; Bovair, Kieras, and Polson 1990). The fact that these rules take more time suggests that there might be some inaccuracy about how the task is represented. For example, the propagation rules in the model can immediately apply to all points in the circuit, but perhaps subjects "trace" through the circuit and allow the rules to apply to only one point in the circuit at a time. The result would be that the propagation rules would appear to fire much more slowly. Resolving this issue would require more detailed study of how people answer questions about electrical circuits.

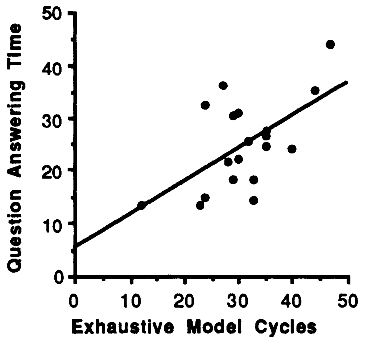


Figure 22. Scatter plot of cycles and question-answering time for the exhaustive model.

Figure 23 shows the predicted and observed times for the individual questions in each circuit, shown in the order in which they appeared in the experiment within each condition; the Irrelevant group questions appear first on the horizontal axis followed by the questions in the Relevant group. What this figure shows is that the model's predicted values track the observed values fairly well, but with some definite mispredictions. Given the noisiness of the data, it is probably not worthwhile to pursue the nature of the mispredictions in more detail. But the model does capture the effect of schema availability; notice that the times to the Irrelevant group questions tend to be longer than those for the Relevant group, and the model shows the same pattern. However, the effect of schema availability is fairly mild, both in the case of the data, and the number of cycles required by the model.

In the exhaustive model, the changes are propagated until quiescence; this strategy will tend to reduce the benefits of having the schema rules. How the rules would apply to the schematic subcircuits also suggests that the benefits would be relatively small. For example, in a simple triode amplifier circuit, the first-principle reasoning would be that if the voltage on the grid changes, then by the triode rule, the resistance of the triode changes, and by the voltage divider rule, the result is a change in the voltage on the triode plate. Thus when using first principles, going from a change in the voltage on the triode grid to a change in the voltage on the triode plate takes only two rules. The schema rule for the amplifier circuit would make the same inference in one rule instead of two. This is a relatively small change in the amount of inference required, so perhaps only mild benefits of schema knowledge should be expected for these materials and this task.

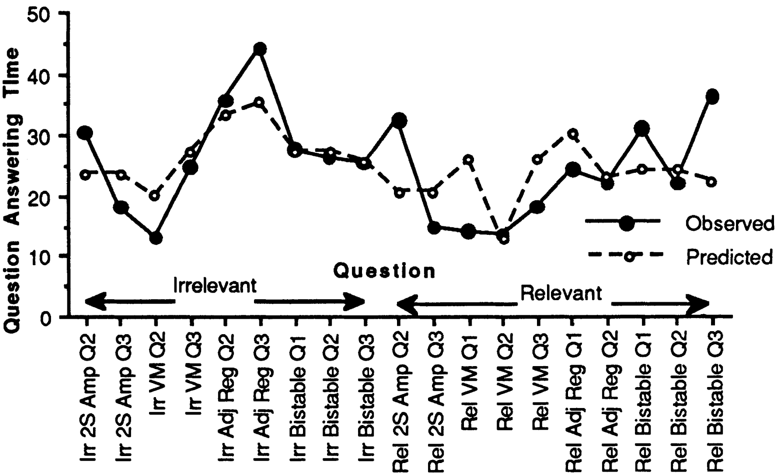


Figure 23. Predicted and observed times for each question in both conditions.

GENERAL DISCUSSION

Summary of Results

The basic question addressed by this work is whether schemas and their explanations are involved in how practical electronics material is learned. The AI system and the question-answering model shows that in principle, schemas can be learned from these explanations and then used in further learning and answering questions about this kind of material, and thereby suggest that the schematic structure of textbook material is important. The experimental work shows that learning schemas from explanations can be effective in the classroom training of technical content. In the course of one or two hours, subjects were able to learn circuit schemas, and then enjoyed some benefit from applying them to understanding explanations and answering questions about more complex circuits. The comparison of the models with the data shows that the magnitude of the benefits of schemas for human learners can be predicted to some extent by the models, which suggests that the schema mechanisms used in the AI system and in the question-answering model are plausible as psychological models.

Difficulties of Experimentation in this Domain

The experimental work reported here shows that learning circuit schemas is beneficial in later learning, but the effect is fairly small, and it is susceptible to the specific task strategies that subjects adopt in dealing with the experimental paradigm. Several problems made it difficult to get definitive data on this phenomenon. The type of experiment attempted here involves collecting problem-solving latencies, which are highly variable, under conditions that severely limit the sample size: (1) There are relatively few realistic circuits at a reasonable level of complexity. (2) It is hard to get a large number of subjects willing and able to tackle these surprisingly difficult materials. (3) There are only a small number of distinct schema-relevant questions about an individual circuit. For example, in the two-stage amplifier circuit, all of the questions that are relevant to the schema-based understanding consist simply of what happens to the output of either the first or second stage when the input changes. Thus it is not possible to ask a large number of questions about each circuit. Clearly, if subjects were asked many questions about the same circuit, their question-answering strategies would change altogether, as they simply memorized the answers to the few possible questions.

Thus, although the electronics domain appears to be a clear-cut and relatively simple domain to explore either from the AI or the cognitive modeling perspective, it seems to be a very difficult one for collecting human data on complex learning processes.

The Need for Cognitive Analysis of Large-Scale Training Materials

This research focussed on the properties of realistic training materials in a technical domain. While this research used only a very small subset of the materials, it encompassed a relatively large set of concepts which were also relatively complex. The large scale of the complete set of training materials for this domain must be appreciated. In the electronics series used here (Van Valkenburgh, Nooger, & Neville, Inc., 1955) there are about 600 pages, and a similar quantity in a prerequisite series on basic electricity. The U.S. Navy considered this to be the amount of knowledge that should be taught to trainees to qualify them for basic electronics technician jobs. However, when the learning of electricity or electronics has been studied under laboratory constraints, the researchers typically use only a fragment that corresponds to a single page, or a few pages at most, of this corpus. The result is that we have no understanding from a cognitive science point of view of how such a mass of material is structured or learned.

The building-blocks approach is one important property of these materials, but the dominant property is that most of the content is the design rationale and principles for electronic circuits — how they work, and why they are configured the way they are. For example, a key topic is that vacuum tubes and transistors must have their "bias" set by additional components to place their operating characteristics in a desired range, for example, to produce a linear response function. Considerable space is spent explaining this issue mathematically, with heavy use of graphs, and formulas are supplied and illustrated for calculating the proper component values.

What is odd about this emphasis on design rationale is that these materials are not intended to prepare students for electronics engineering and circuit design, but for electronics maintenance, in which the trainee's future task is to diagnose and correct malfunctions in the equipment. As argued elsewhere (Bond & Towne, 1979; Kieras, 1988), understanding of fundamental quantitative principles and design rationale does not seem to be important in troubleshooting tasks. In contrast, electronics troubleshooting must be learned by apprenticeship or haphazardly; there is very little published material that presents general concepts of electronics troubleshooting. But despite this misdirection, the practical electronics materials studied here contain a fairly standard presentation of the complete domain theory of practical electronics, which many thousands of people have mastered.

These materials would be an ideal place to attempt a large-scale analysis of training materials. For example, a large semantic net could be constructed to show how each concept was related to the other concepts. The kinds of pedagogical techniques used to present each concept could be listed. There are many techniques used in these materials, and it would be valuable to know whether there is any pattern to their usage. For example, specific circuits are presented with diagram graphics and textual explanations, as was studied here. Some concepts (e.g., amplification) were introduced with several pages of relation to everyday life (e.g., amplifying the size of the catch in a fish story), and with analogies (e.g., amplification as regulating the flow of water from a tank). Illustrations of the actual physical appearance of components abound, corresponding to these materials teaching about actual components and devices, rather than the idealized ones presented in non-practical treatments. Cartoons are used to show causal sequences with a kind of animation. Static cartoons and humor often seem to be used to reinforce specific concepts and excite interest. There are many mathematical arguments, presented both graphically and algebraically. Finally, much information was just presented explicitly in text. Thus, trying to determine why each concept was presented the way it was, and whether the presentation was effective in theory, would yield many insights and hypotheses about technical training.

If cognitive science is to contribute toward improving instruction in real domains, such as technical ones, it will be necessary to complement the traditional detailed analysis of fragments of the domain with analysis of the domain in the large. By analyzing the structure and content of such complete materials from a cognitive-theoretic point of view, we will be able to ensure that our future detailed research is addressing the key properties of the materials and the domain.

ACKNOWLEDGEMENT

This research was supported by the Office of Naval Research Cognitive Science Program under Contract Number N00014-88-K-0133, Contract Authority Identification Number NR 442-f002.

REFERENCES

American Radio Relay League, (1961). *Radio Amateur's Handbook*, West Hartford, Conn.

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

Bond, N. C., & Towne, D. M. (1979). Troubleshooting complex equipment in the military services: Research and prospects (Tech. Rep. No. 92). University of Southern California: Behavioral Technology Laboratories.

Bovair, S., Kieras, D. E., & Polson, P. G. (1990). The acquisition and performance of text editing skill: A cognitive complexity analysis. *Human-Computer Interaction,* **5**, 1-48.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Covrigaru, A., & Kieras, D. E. (1987). *PPS: A Parsimonious Production System* (Tech. Rep. No. 26). (TR-87/ONR-26). Ann Arbor: University of Michigan, Technical Communication Program.

Graesser, A. C., Hoffman, N. L., & Clark, L. F. (1980). Structural components of reading time. *Journal of Verbal Learning and Verbal Behavior,* **19**, 135-151.

Haberlandt, K., Berian, C., & Sandson, J. (1980). The episode schema in story processing. *Journal of Verbal Learning and Verbal Behavior,* **19**, 635-650.

Johnson, W., & Kieras, D.E. (1983). Representation-saving effects of prior knowledge in memory for simple technical prose. *Memory and Cognition,* **11**, 456-466.

Kieras, D. E. (1982). *What people know about electronic devices: A descriptive study* (Tech. Rep. No. 12, UARZ/DP/TR-82/ONR-12). Tucson: University of Arizona, Department of Psychology. (DTIC AD A122189)

Kieras, D. E. (1984). A method for comparing a simulation model to reading time data. In D. Kieras, & M. Just (Eds.), *New methods in reading comprehension research*, Hillsdale, NJ: Lawrence Erlbaum Associates.

Kieras, D. E. (1988). What mental model should be taught: Choosing instructional content for complex engineered systems. In J. Psotka, L. D. Massey, & S. A. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned* (pp. 85–111). Hillsdale, NJ: Lawrence Erlbaum Associates.

Kieras, D. E. (1991) *Human learning of schemas from explanations in practical electronics.* (Tech. Rep. No. 33, TR-91/ONR-33). Ann Arbor: University of Michigan, Technical Communication Program.

Mayer, J.H. (1990). *Explanation-based knowledge acquisition of schemas in practical electronics: A machine learning approach* . (Tech. Rep. No. 32, TR-90/ONR-32). Ann Arbor: University of Michigan, Technical Communication Program. (DTIC AD A229122)

Rumelhart, D. E. (1980). Schemata: The building blocks of cognition. In R. Spiro, B. Bruce, & W . Brewer (Eds.), *Theoretical issues in reading comprehension.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Rumelhart, D. E., & Ortony, A. (1977). The representation of knowledge in memory. In R. C. Anderson, R. J. Spiro, & E. Montague (Eds.), *Schooling and the acquisition of knowledge.* Hillsdale, NJ.

Thibadeau, R., Just, M. A., & Carpenter, P. A. (1982). A model of the time course and content of reading. *Cognitive Science*, **6**, 157-203.

Van Valkenburgh, Nooger, and Neville, Inc. (1955). *Basic Electronics.* Hayden, Rochelle Park, NJ.

Statistical and Cognitive Models of Learning through Instruction[1]

Sandra P. Marshall
Department of Psychology
San Diego State University
San Diego, CA 92182-0315

Abstract

This chapter uses statistical and cognitive models to evaluate the learning of a set of concepts about arithmetic word problems by a group of students. The statistical model provides information about how the group of students as a whole performed on an identification task involving word-problem situations and shows differences among subgroups. The cognitive model simulates the performance of each student and yields details about how learning varied from one individual to another. It is a connectionist model in which the middle layer of units is specified *a priori* for each student, according to the student's level of understanding expressed in an interview. The chapter concludes with a detailed comparison of the simulated responses with the observed student responses.

INTRODUCTION

The learning investigated here occurred as part of a study in which students received computer-based instruction about arithmetic word problems. The central topics of the instruction were five basic situations that occur with great frequency in word problems: *Change, Group, Compare, Restate,* and *Vary*. The instruction had three main segments: (1) the introduction, in which the situations were described; (2) an in-depth exploration, in which details of each situation were elaborated and presented diagrammatically; and (3) the synthesis, in which combinations of situations

[1] The research reported in this chapter was supported by Contract No. N00014-85-K-0661 and Grant No. N00014-90-J-1143 from the Office of Naval Research.

were introduced together with planning and goal-setting techniques.[2] For each of the three parts of instruction, students engaged in multiple practice exercises. The study reported in this chapter concerns only the first segment of instruction--the introduction to the situations--and the primary focus is the nature of the knowledge that individuals gained from that introductory instruction.

This chapter describes two analyses of what individuals learn from instruction. Both analyses are needed. In the first case, learning is examined in a traditional experimental paradigm, using established statistical procedures. Group features, rather than individual characteristics, receive greater emphasis in this paradigm, and conclusions drawn from the analysis describe group commonalities. In the second case, learning is examined by means of a cognitive model that simulates individual performance. In this analysis, individuals' characteristics are studied, and conclusions apply separately to each individual. As I indicate below, the information gained from each analysis is valuable in a study of learning. Neither one alone provides the complete picture.

The questions of interest in the research are *what* is the new knowledge retained in memory as a result of instruction, *when* is it retained, and *which parts* of it are later accessed and retrieved. During instruction, some new information is (presumably) acquired and added to an individual's available knowledge store. Not all possible information is taken in, and individuals vary in the type and amount of new knowledge that enter memory. It is the rare instance in which all learners learn exactly the same thing from a single instructional lesson. More often, some learners noticeably remember a great deal of the new information while others remember almost nothing.

The Instructional Domain

This section provides a short description of the five situations used in instruction. The situations are *Change, Group, Compare, Restate,* and *Vary,* and they represent uniquely almost all simple stories found in arithmetic story problems (Marshall, 1991).

The *Change* situation is characterized by a permanent alteration over time in a measurable quantity of a single, specified thing. Only the quantity associated with one thing is involved in the *Change* situation. It has a

---

2 Details about the computer-based instruction can be found in Marshall, Barthuli, Brewer, & Rose (1989), a technical report available from the author.

beginning state and an end state, with some intervention which causes a transition from beginning to end. Usually, three numbers are of importance: the amount prior to the change, the extent of the change, and the resulting amount after the change has occurred.

A *Group* situation is present if a number of small distinct sets are combined meaningfully into one large aggregate. Thus, the *Group* situation reflects class inclusion. The grouping may be explicit or implicit. If explicit, the solver is told in the problem statement which small groups are to be united. If implicit, the solver must rely on his or her prior semantic knowledge to understand the group structure. For example, in a situation involving boys and girls, the solver would typically be expected to know that boys and girls form a larger class called children. The solver also would be expected to understand that the members of the subgroups (i.e., boys or girls) retain their identity even when combined into a larger group (i.e., children). Three or more numbers are necessary in a *Group* situation: the number of members in each of the subgroups as well as the overall number in the combination.

The *Compare* situation is one in which two things are contrasted to determine which is greater or smaller. The numerical size of the difference between the values is unimportant and may not even need to be computed. The *Compare* situation relies heavily on prior knowledge that individuals have about relations. Most frequently, the *Compare* situation requires the solver to choose either the larger or smaller of two values when the operative relation is stated as a comparative adjective or adverb (e.g., faster, cheaper, shorter, more quickly). The objective is the determination of whether one's response should be the larger or the smaller of the known values. This situation most typically occurs as the final part of a multi-step item. For instance, one often sees problems in which the solver is expected to decide after several problem-solving steps which of two items offered for sale is the better buy. This final determination is a *Compare.* It requires only the recognition of which of the two items is less costly--it does not require the computation of how much less.[3] Most *Compare* items involve values for only two objects, although it is certainly possible to make comparisons among three or more.

---

[3] It should be noted that the *Compare* situation defined here differs from the semantic relation of the same name developed by Riley, Greeno, and Heller (1983).

The *Restate* situation contains a specific relationship between two different things at a given point in time. The relationship exists only for the particular time frame of the story and cannot be generalized to a broader context. There are two determining features of a *Restate* situation. First, the two things must be linked by a relational statement (e.g.,one of them is twice as great as, three more than, or one half of the size of the other).Second, the relationship must be true for both the original verbal descriptions of the two things and the numerical values associated with them. Thus, if Mary is now twice as old as Alice, then 20 years--which is Mary's age--must be twice as great as 10 years, which is Alice's age. Note that this relationship was not true one year ago nor will it necessarily be true in five years.

The *Vary* situation is characterized by a fixed relationship between two things that persists over time. The two things may be two different objects (e.g., boys and girls) such that one can describe a ratio as "for every boy who could perform x, there were 2 girls who could do the same ....", or they may be one object and a measurable attribute of it (e.g., apples and their cost) with the problem having the form "if one apple cost $.50 then five apples ....". An essential feature of the *Vary* situation is the unchanging nature of the relationship. If one of the objects is varied, the amount of the second changes systematically as a function of the known relationship. The variation may be direct or indirect.

Simple examples of these five situations are given in Table 1. During the entire course of computer instruction, each of the situations is introduced, explained, and transformed to a problem setting. Eventually, several are linked together to form multi-step problems. In the introductory lesson, each situation is described by means of an example and with the general features which define it.

Although they are very simple and readily understandable, the five situations are not intuitively known by students through previous instruction. Experiments with groups from several different student populations indicated that students (and teachers) do not typically recognize or use situational knowledge in story problems (Marshall, 1991). Those same experiments show that students of all ages are nevertheless able to learn them.

The present study was designed to investigate how that learning comes about. Because they were previously unknown to the students, the situations in story problems were, in fact, five new concepts to be learned. Thus, the study described here provides a setting for investigating how

Table 1
The Five Situations

CHANGE	To print his computer job, Jeffrey needed special paper. He loaded 300 sheets of paper into the paper bin of the laser printer and ran his job. When he was done, there were 35 sheets of paper left.
GROUP	The Psychology Department has a large faculty: 17 Professors, 9 Associate Professors, and 16 Assistant Professors.
COMPARE	The best typist in the pool can type 65 words per minute on the typewriter and 80 words per minute on the word processor.
RESTATE	In our office, the new copier produces copies 2.5 times faster than the old copier. The old copier produced 50 pages every minute.
VARY	An editor of a prestigious journal noticed that, for a particularly wordy author, there were five reference citations for every page of text. There were 35 text pages in the manuscript.

individuals learn new concepts that have obvious ties to much of their previous knowledge.

The Nature of Instruction

To model successfully the acquisition of knowledge from instruction, one must examine the nature of that instruction and the type of information contained in it. Generally, there are two ways to present new concepts to students. The instructor can introduce the name of the concept and give a prototypic example. The example contains specific details and is couched in a setting that should be well-understood by students. An alternative approach is for the instructor to provide the name of the concept and give a general description of its most important features. This information is abstract and contains basic characteristics that should apply to all possible instances of the concept. In practice, instructors typically do both. They introduce a new concept by name, give a representative case in which the concept clearly occurs, and then make a broad statement about the concept, which is intended to help the learner generalize the concept from the given example to other potential instances.

Some interesting research has been carried out to determine whether students learn differentially under different instructional conditions. Usual studies of instructional content tend to contrast one form of information with another, such that each student sees only one type. An example of this type of research is found in Sweller's (1988) comparison of problem-solving performance following rule-based or example-based instruction.

The issue I address is different: Given access to typical instruction in which both specific information (i.e., examples) and abstract information (i.e. definitions) are available, which will a student remember? Do students commit equal amounts of specific and abstract knowledge to memory? Is one type necessarily encoded first, to be followed by the other? Are there large individual differences? If so, are these differences related to performance? The following experiment provides some initial answers to these questions.

THE EXPERIMENT

Subjects

Subjects were 27 college students with relatively weak problem-solving skills. They were recruited from introductory psychology classes. On a pretest of ten multi-step arithmetic word problems, they averaged six correct answers.

Procedure

Each student worked independently on a Xerox 1186 Artificial Intelligence Workstation. All instruction and exercises were displayed on the monitor, and the student responded using a three-button optical mouse. Each student participated in five sessions, with each session comprised of computer instruction, computer exercises, and a brief interview. Students spent approximately 45-50 minutes working with the computer in each session and talked with the experimenter for about 5-10 minutes in the interviews. As stated previously, only the first session--the introduction to the five situations--is of interest here.

Data Collection

Data were collected from two sources: student answers to the first exercise presented by the computer and student responses to the interview questions. Each is described below.

Identification task. The first source of data was the computer exercise that followed the initial instructional session. The items in this task resembled those of Table 1. They were selected randomly for each student from a pool of 100 items, composed of 20 of each type. During the exercise, one item at a time was displayed, and the student responded to it by selecting the name of one situation from a menu containing all five names: *Change, Group, Compare, Restate, Vary.* The student received immediate feedback about the accuracy of the answer, and if the student responded incorrectly, the correct situation was identified.

The order of item presentation was uniquely determined for each student. Items of each situation type remained eligible for presentation until one of two criteria was obtained: Either the student had given correct responses for 2 instances or the student had responded incorrectly to 4 of them. Thus, a student responded to at least 2 items of each type and to no more than 4 of them. The minimum number of items displayed in the exercise for any student was 10, which occurred only if the student answered each of them correctly. The maximum number that could be presented was 20 items, which could happen only if a student erred in identifying the first two items of all five types. The number of items presented ranged from 10 to 18.

Interview Responses. The second source of data was information given by the students in the interviews. The interview followed immediately after the identification task described above. During the interview, each student was asked to describe the situations as fully as he or she could. The student was asked first to recall the names of the situations and then to describe each one that he or she had named. After each of the student's comments, the experimenter prompted the student to provide additional details if possible. All interviews were audiotaped and transcribed.

It is the interview data that reveal which pieces of instruction were encoded and subsequently retrieved by each student. Certainly, not all of the new knowledge acquired by an individual will be revealed in an interview. It is expected that students have more knowledge than they can access (as pointed out by Nisbett & Wilson, 1977). Nevertheless, the interview data are indicative of how the individual has organized his or her knowledge of the newly acquired concepts, and they suggest which pieces of knowledge are most salient for the individual. Following well-known studies such as Collins and Loftus (1975) or Reder and Anderson (1980), we may assume that individuals will tend to retrieve the most closely associated features and those with highest salience for the individual.

Knowledge Networks and Cognitive Maps

Data from the student interviews were used to construct knowledge networks, one for each student. Each network consists of a set of *nodes*, representing the distinct pieces of information given by the student, and *links* connecting the nodes, representing associations between the pieces of information.

The interviews were coded in the following way. First, irrelevant comments were eliminated. These were things such as "Um, let me think" or "I'm trying to remember ...." Next, distinct components or elements of description were identified. These were usually phrases but could also be single words. These became the nodes of the knowledge networks. Two nodes were connected in a network if the student linked their associated pieces of information in his or her interview response. Two research assistants and the author coded each interview with complete agreement.

In addition to the knowledge network for each student, an "ideal" network was constructed from the instructional text. As with the students' networks, nodes were created to represent each distinct piece of information. Two separate pieces of information appearing contiguously in the text were represented by two nodes with a link between them. Needless to say, this network was substantially larger than any student network. It represents all that a student could possibly encode from the instruction, and thus it serves as a template against which to measure the amount and type of information encoded by each student. The "ideal" network for all of the situational information is presented in Figure 1.

Two things should be noted about the network presented in Figure 1. First, distances between nodes and spatial orientation of the nodes have no meaning. Only the presence or absence of nodes and links is of importance. Second, in this figure, all nodes appear equally important, and the same is true for the links. Strength and activation are not shown. However, in theory each node has a measure of strength that is a function of how many times it appears in the instruction, and each link has a similar measure of activation, depending upon how frequently the two nodes are linked.

Figure 1: THE "IDEAL" NETWORK

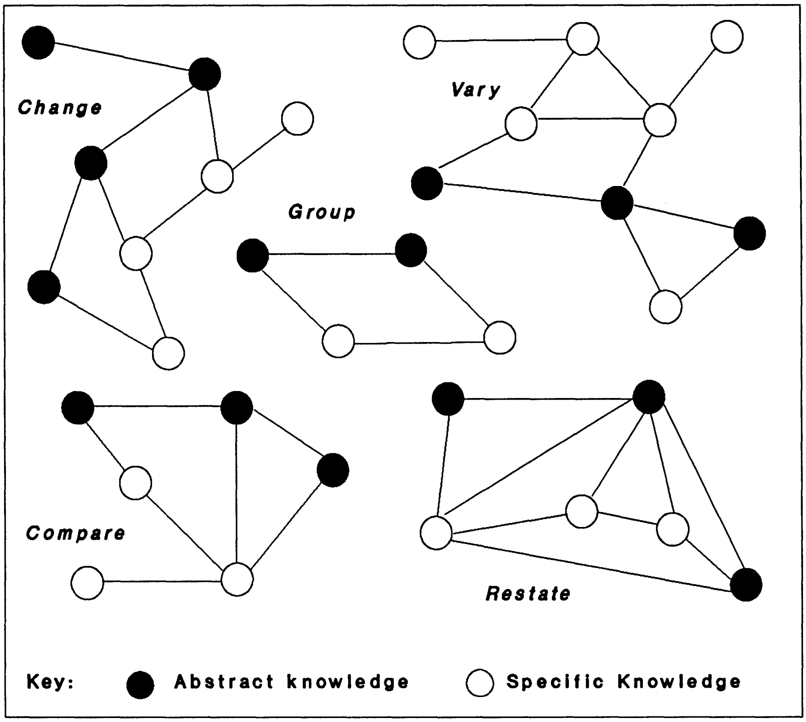


Figure 1 represents the ideal case in which all information is included in the network. Normally students do not retain all of the details, and the networks one constructs for them appear incomplete when compared with the ideal situation. Thus, we expect the student networks to be considerably sparser than that shown in Figure 1.

Several types of information may be gleaned from a student's knowledge network. First, of course, the network is an indication of how much the student remembered. The number of nodes in a network provides an estimate of this information. Second, the network shows which pieces of information are related for an individual. A measure of association can be made by counting the number of links and using that number to estimate the degree of connectivity of the entire network. Node count and degree of connectivity are standard network measures. I have discussed elsewhere how they may be used to estimate a student's knowledge of a subject area (Marshall, 1990).

In this chapter I examine two additional types of information: (a) *specificity,* which is the students' tendency to recall specific or abstract features to describe the situations and (b) *confusions,* which show the extent to which students confused different aspects of the five situations. One examines nodes to estimate the former and links to estimate the latter.

Specificity. Each node in the "ideal" network reflects one of two types of detail: specific or abstract. Specific knowledge refers to elements of information having to do with the examples presented in instruction, and it reflects the particular details of the example. Abstract knowledge refers to the general features or definition of the situation. The instruction contains approximately an equal amount of both types, as can be seen in Figure 1. The abstract nodes are represented by filled circles, and the specific ones are indicated by hollow circles.[4]

Each distinct piece of information (i.e., each node) recalled by a student was categorized as being *specific* or *abstract.* A response was considered to be *specific* knowledge if it pertained to a specific example. Typically, students giving this sort of response referred to details from the initial example used in the computer instruction. An illustration is given in the specific response of Table 2. The italicized phrases are examples of specific detail. In contrast, a response was considered to be abstract knowledge if it reflected a general definition or characterization. Table 2 also contains an illustration of an *abstract* response, and the italicized phrases indicate the abstract detail. The final example of a student response in Table 2 illustrates the case in which neither abstract nor specific detail is recalled.

Three measures of specificity were developed: the number of specific responses, the number of abstract responses, and the ratio of abstract to specific responses. These measures were used in the statistical analyses described below.

Confusions. In the networks representing situational knowledge, two types of links are possible, intra-situational and inter-situational links. Intra-situational links are judged always to be valuable. That is, if two nodes are both associated with one situation and they are connected to each other, then the retrieval of one of the nodes ought to facilitate the retrieval

---

[4] It should be noted that the instruction was not developed under the constraint that equal abstract and specific details be contained in it. The guiding principle was to explain each situation as completely as possible, using specific and/or abstract elements as needed.

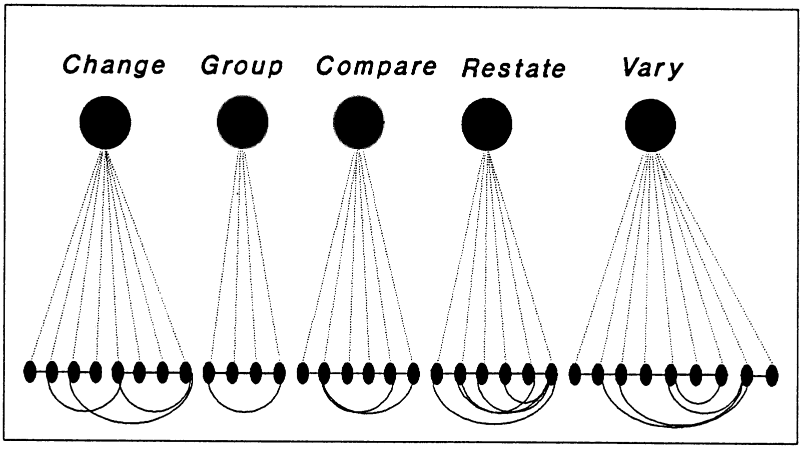
Table 2
Examples of Student Responses

ABSTRACT	Q:	What do you remember about Group?
	A:	Group is when you have different items, ***different groups of items***, that can be ***categorized into one general group.***
SPECIFIC:	Q:	What about Group?
	A:	That was when you bought *7 shirts and 4 pairs of shorts* and they grouped it into clothing. So you had *11 separate things of clothing.*
NONE:	Q:	Tell me about Change.
	A:	I pressed that review button so many times and I can't remember anything right now. Um, change was, um my mind is blank right now. I did okay on the computer. I've forgotten just about everything. I'm trying to think of an example. I know they change something and make something else.

of the other. This is the principle of spreading activation. In general, the more knowledge the individual has about a concept and the greater the number of associations connecting that knowledge, the better the individual understands it. Figure 2 shows how the "ideal" network of Figure 1 can be represented as a two-layer map. The nodes at the upper level are the five situations, and those at the lower level are the knowledge nodes developed during instruction. Connections among the nodes at the lower layer represent intra-situational links. Generally, a larger number of connections at this level indicates greater understanding on the part of the individual. It is these connections that are shown as well in the network of Figure 1.

In contrast, inter-situational links, i.e., links between different situations, may or may not be of value to the individual's learning, because they are a potential source of confusion. Such links will not always reflect confusions; situations could in principle share one or more features. In the present case, however, the instruction was carefully designed to eliminate common features among situations. This is reflected in Figure 2 by the connection from each node at the lower level to a single node at the upper

Figure 2: THE "IDEAL" MAP

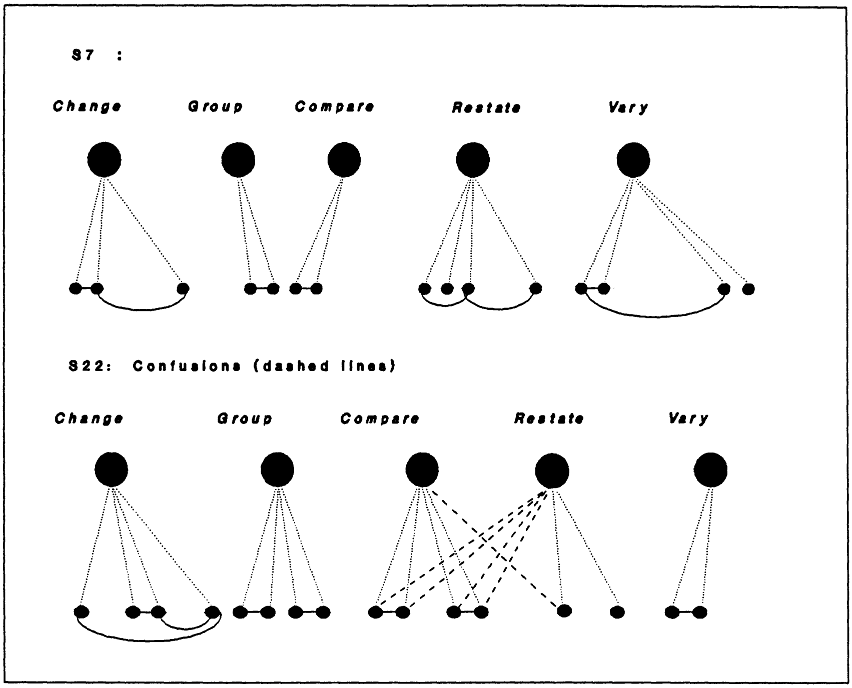


level. Given the design of instruction, there should be no inter-situational links. That is, no node at the lower level should connect to more than a single upper level node. Such linkages would be confusion links and reflect a misunderstanding about the two situations so linked.

An example of differences in students' inter-situational and intra-situational links is given in Figure 3. Two student maps are presented in this figure. Both students encoded a relatively large amount of information from the instruction, compared with other students in the experiment, but it is clear from the figure that they recalled different elements of information. Student *S7* remembered distinct pieces of information about each situation and showed no confusions. *S22*, on the other hand, expressed a number of confusions, which are represented in Figure 3 by the dashed links between the two layers of nodes. These cognitive maps are characteristics of incomplete mastery. The situational knowledge of every student can be described by such a map. Obviously, the deficits of a student are highly individual. These individual differences will be discussed further in a later section of this chapter.

In summary, the student network and its corresponding map provide information about the number of details the student remembered about a situation, the amount of connectivity, the type of knowledge (i.e., abstract or specific), and the number of confusions in the student's response. The networks and the measures described here were the bases for the statistical

Figure 3: TWO STUDENT MAPS



analyses presented below and also served as input to the simulation model, which is described in the section following the statistical analyses.

STATISTICAL ANALYSIS

Three questions are addressed by the statistical evaluation. The first is whether students remember different amounts of detail from instruction, the second is whether one can characterize the type of information encoded by a student, and the third is whether these differences are related to the students' success on the identification task. Evaluation of the student networks shows that some students were more likely to encode mostly specific details, some were more likely to encode mostly abstract information, some encoded both in about equal proportions, and some encoded almost nothing. The statistical analysis evaluates whether these tendencies are related to performance on the identification task and whether the relationship can be generalized to the entire group of students.

It is evident from the interview data that students varied greatly in the amount of information they were able to recall about the five situations. The number of different details retrieved by students extended from a low of 3 to a high of 20. The mean number of details was 13.5, with a standard deviation of 4.02.

The number of abstract and specific details recalled also varied, and the ratio of abstract to specific detail ranged from 14:3 to 6:14. Thus, the answers to both the first and the second questions are affirmative: There were clear differences in the total amount of information recalled as well as differences in the amount of abstract and specific information.

Two analyses provide insight into the importance of this difference. First, on the basis of their interview responses, students could be divided into three groups: *Abstract, Specific,* and *Both*. Students classified as *Abstract* gave predominantly definitional responses in the interview. Those classified as *Specific* used mostly example information from the computer instruction to describe the situations. Those classified as *Both* responded with approximately equal numbers of abstract and specific detail. For membership in either the *Abstract* or *Specific* group, students had to have given at least 9 different pieces of information during the interview with at least twice as many instances of one type of information as the other. Approximately equal numbers of students could be classified as *Abstract* or *Specific,* with 6 in the former and 7 in the latter. An additional 11 students were categorized as *Both*. These students gave at least 9 responses with approximately equal numbers of abstract and specific details.

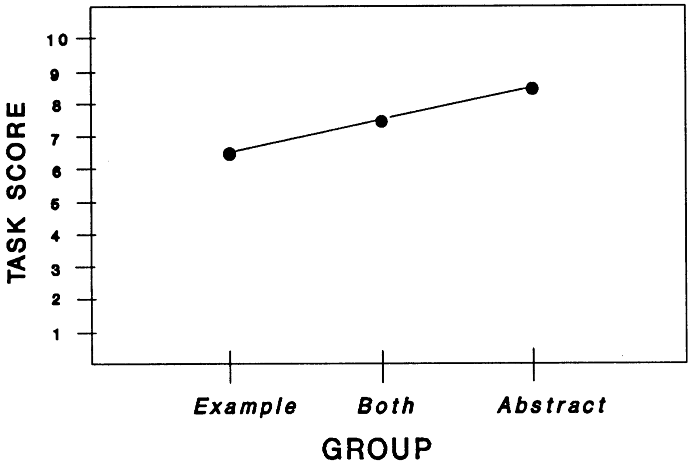
Figure 4 shows the relative performance on the identification task of the three groups described above. A one-way analysis of variance, with a dependent measure of correct responses to the identification task,[5] indicates that the groups differed significantly in their ability to recognize the situations, $F(2, 21) = 4.53$, $p < .025$.[6] As can be seen in Figure 4,

---

[5] It will be recalled that students viewed differing numbers of items on this exercise. For purposes of comparison in this analysis, only the first two exemplars of each type of situation were scored. Thus, each student received a score from 0-10.

[6] Complete data were not recorded for two students. One loss was the result of computer failure and the second was the result of a malfunction in the recording of the interview. These two students were excluded from the analyses reported here. Two other students having only 6 and 3 interview responses respectively were also excluded from this analysis.

Figure 4: GROUP PERFORMANCE



students who responded primarily with abstract characterizations of the five concepts were most successful, followed by those who used both types of information. The group relying on examples only were less successful than those using abstract only or abstract knowledge in conjunction with specific details. The performance of the abstract group was significantly higher than the performance of the example group, $t\,(21)= 3.005$, $p < .01$.

The above analysis shows that differences in student performance can be explained in terms of whether a student remembered abstract or specific information. One also expects that the absolute number of details that a student remembers--regardless of whether they are definition or example--would be a good predictor of performance. Surprisingly, this is not the case. The Pearson product moment correlation between the number of correct responses on the performance test and the total number of nodes encoded from the student's interview is .074, accounting for less than 1% of the variance.

A second and more informative way of analyzing the data is a multiple regression analysis based on the type and amount of information, the inter-situational confusions, and the interaction between the two. In this analysis, the predictors are (1) X_1, the ratio of abstract to specific detail, (2) X_2, the number of confusions mentioned explicitly by the student, and (3) X_3, a product variable of the first two predictors. The dependent measure, again, is the 10-item identification task. The resulting prediction equation was: $Y' = 6.667 + .602X_1 + .545X_2 - .617X_3$, with all coefficients reaching the conventional .05 level of significance. The model accounted for 43% of the variance and was statistically significant, $R^2 = 0.43$; $F(3, 21) = 5.38$, $p < .01$.

In general, students with higher abstract to specific ratios performed better on the identification task and made fewer confusion errors. Students with low ratios (i.e., those with more specific answers) named relatively few confusions but also responded with fewer correct answers. Students with approximately the same number of specific and abstract responses had the greatest number of stated confusions.

Thus, the statistical analyses suggest several group characteristics with respect to learning new concepts. That is, there are tendencies of response that apply over many individuals, not just a single one. These analyses are based on summaries of the cognitive maps and aggregate responses to the identification task. A more detailed investigation of individuals' responses provide additional information about the nature of learning in this study.

THE COGNITIVE MODEL

A more exacting analysis of the relationship between each student's cognitive map and his or her responses to the identification task was carried out by simulating the responses using a simple feed-lateral connectionist model. The model simulates for each student his or her response to each item of the identification task that the student actually attempted to identify.

The general model is given in Figure 5. It has three types of units: inputs, student nodes, and outputs. Inputs to the model are coded representations of the problems, and outputs are the names of the situations. As in most connectionist models, activation spreads from the input units at the lowest level to those of the intermediate level(s) through their connections. At the middle level, activation spreads laterally from the nodes directly activated by the lower level units to other nodes at the same level with which they are linked (this is represented by the two middle layers in Figure 5). Finally, the total activation coming into each unit at the output level is evaluated, and the output unit with the highest activation is the model response. Unlike many connectionist models, the units at the middle layer, their connections with other nodes at this level, and their linkages to the upper level are determined explicitly from empirical data.

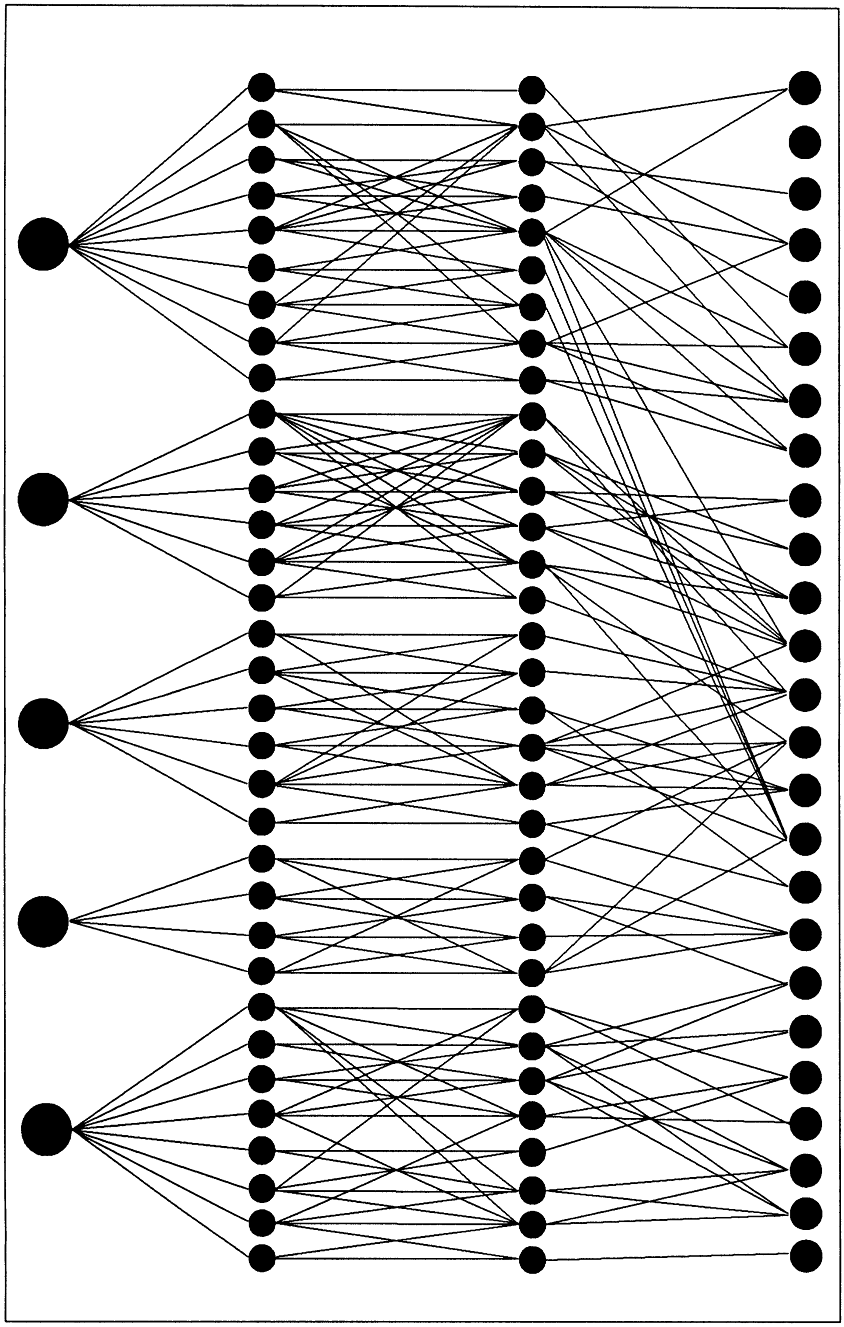


Figure 5: THE FULL MODEL

Table 3
Item Characteristics Used to Encode Story Situations

General Characteristics:
Set modification
Permanent alteration
Class inclusion (explicit or implicit)
Relation between two objects
Relation between an object and a property of that object
Fixed relation (implied)
Relative size
Size differential
Percentage
Causality
Multiple agents
Multiple objects
Unit measurement
Two identical relations

Key Phrases:
Each/every/per
As many as
Have left
Altogether/A total of
More/less
Cost
Same
If ...Then
Money

Time Features:
Specific time elements (minutes, days, weeks)
Before/after

The bottom layer of units. The inputs consist of information about the items that comprise the identification task. There are 27 possible characteristics that can be present in any item. The set of characteristics is given in Table 3. Each item is coded according to these characteristics as a 27-element vector containing 0's and 1's, with 1 indicating the presence of a characteristic and 0 its absence. Not all characteristics will be present in any single item; usually a simple situation requires only a few of them. The mean number of characteristics for the 100 items used in the identification task was 4.33. All 100 items were encoded by three raters with complete agreement.

The middle layers of units. For each student, the middle layers of the model contains a set of nodes and the connections between them. The two layers have identical sets. The nodes and links were identified from the student interviews, as described previously, and they formed the basis of the statistical analyses of the preceding section. Three trained individuals read the transcript of each individual's interview and determined which nodes were present and whether they were linked. As in the characteristics coding above, the three coders were in complete agreement.

The top layer of units. The outputs for the model are the five situation names: Change, Group, Compare, Restate, and Vary. Only one output is produced for a given input vector. The five possible outputs compete, and the one with the highest accumulated activation wins.

Connections between the bottom and middle layers. Each input element may connect directly to one or more of the nodes contained in the student's network (represented by the middle layers of nodes). Two layers are needed in this model to illustrate the feed-lateral aspect. The lower of the node layers connects to the input units. The second layer illustrates how the nodes connect with each other. Each node from the lower node set connects to itself and to any other nodes to which it is linked, as determined from Figure 2. Thus, activation spreads from the input units to the lower node layer. Each node transfers its own activation to the next layer and also spreads additional activation to any other nodes to which it is connected. This particular two-layer representation of a feedlateral network preserves the usual constraint that activation spreads upward through the model.

Some of the input elements (i.e., those units represented at the very bottom of Figure 5) may activate many nodes in the network, some may activate only a few, and some may fail to make a connection (if the student lacks critical nodes). The allowable linkages between the input and middle layers of units were determined by mapping the input characteristics to the "ideal" map of the entire instruction. Recall that the input characteristics are general features. Most of them activate multiple nodes, and these nodes are frequently associated with different situations. Thus, it is rare that one input characteristic points to a single situation. The full pattern of possible activation is shown in Figure 5. Note that this figure illustrates all characteristics as they link to all nodes and is thus a theoretical pattern. The model would never be presented with a problem containing all possible features, nor did any student have all possible nodes at the middle layers.

Once the student network receives the input, activation spreads from the nodes directly targeted by the input elements through any links they have to other nodes at this level. All of the activated nodes then transmit their total activation to the units at the upper level. The amount of activation for each situation is determined from the accumulation of activated links leading to it. The five situations compete with each other for the highest level of activation, and the one with the highest value becomes the output. Thus, the model of Figure 5 represents the input of an item, the activation of the student's semantic network, the competition among situations, and the final output as a result of total activation throughout the model.

The model depends upon the set of nodes for each student, the pattern of linkages among them, the overall association of subsets of nodes with the situation labels, and the input characteristics of the items. All except the latter are derived from the student cognitive maps described earlier.

Model Verification

As a test of the model's adequacy, a simulation was carried out in which the ideal network of Figure 2 was used as the student model. The 100 items available in the identification task were presented to the model, and its responses were compared with the correct answers. The model performed with 100% accuracy, successfully identifying the situations for all items.

Simulation Results

A simulation of each student's performance on the identification task was carried out. For each student, the response to the first item encountered in the exercise was simulated first, using that item's vector of characteristics and the student's network information. The second item followed, and then all subsequent items until the exercise terminated. Thus, the simulation covered all items presented to the student in the order in which the student saw them.

As described above, the number of items answered by students varied from 10 to 18, yielding a total of 360 item responses. A comparison of the results of the simulation of these 360 responses with the actual student responses to them is given in Table 4.

Table 4
Simulation Results

Outcome	Frequency Observed Outcomes	Frequency Adjusted Outcomes*
CSM	192	192
C_SM	64	64
C_S_M	19	13
CS_M	30	13
CM_S	55	51
Total	360	333

Key:

(1) *CSM*	Both model and student answered correctly.
(2) *C_SM*	Model and student made the same error.
(3) *C_S_M*	Model and student made different errors.
(4) *CS_M*	Student answered correctly; model erred.
(5) *CM_S*	Model answered correctly; student erred.

(C = correct response; S = student response; M = model response)

*Impossible matches excluded

Table 4 presents the observed classification of the students' responses as well as an adjusted classification against which the model was compared. All 360 items comprise the observed classification. In the adjusted classification, some items have been omitted from consideration because the model was constrained by a lack of information from the student interview. This occurred under the following condition: If a student was unable to remember the name of a situation or anything that described it in the interview, the model for that student would have no nodes at the middle layer that could link to the situation name. Thus, the model would be constrained to ignore that situation and would never generate a response pointing to it. Consequently, if a student omitted entirely a situation in the interview, all items for which the student gave that situation as a response were likewise eliminated. There were 27 of these impossible matches. As shown in Table 4, 17 of these were items which the student answered correctly, and 10 were items on which the student erred. It should be noted that these are not model failures but are interview failures.

Each application of the model to a vector of item characteristics, representing a single item, resulted in one of five outcomes, as shown in Figure 5. Outcomes *CSM* and *C_SM* are exact, successful simulations of the model. In both cases, the model generated a response that was identical to the one produced by the student. In the first, the response was correct, and in the second, it was an error. The outcome *C_S_M* is considered to be a partial success of the model. Both the student response and the model response were in error, but they were different errors. In these cases, the model accurately predicted that the student lacked critical knowledge and would err.

The remaining two outcomes, *CS_M* and *CM_S*, represent simulation failures. The most serious of these is *CS_M*, reflecting cases in which the student answered correctly but the model failed to do so. They are serious failures because they suggest that the model did not capture sufficiently the student's knowledge about the situations. It should be noted that more than half of the observed instances of *CS_M* were impossible matches, as described previously. That is, the student omitted any discussion of the situation in the interview, and the model was subsequently constrained to ignore it. As mentioned above, these instances are considered to be interview failures rather than model failures. Only the remaining 13 instances are true model failures, representing just 3.9% of all responses.

The final outcome category, *CM_S*, also represents model failure but is less critical than the failures of *CS_M*. In this category, the model made a correct response when the student did not.

Many of the *CM_S* simulation failures can be explained by considering the students' experience as they respond to the identification task. During the actual task, many students made errors on one or more situations and then apparently learned to classify these same situations correctly. This is evidenced by their patterns of responses, typically an incorrect response to a situation followed by two correct responses to the same situation, with no additional errors. What has happened in such cases is that the student's knowledge network presumably changed during the course of the task. The knowledge base that generated the early incorrect responses is not necessarily the same one that generated the later successful ones. And, it is only the latter that is reflected in the student's interview. In such instances, the model would correctly match the two correct responses, but it would also give the correct response to the first item that the student missed. The model does not learn. It simulates the state of the student at the end of the exercise, as reflected in the interview. If the student learned during the course of the exercise, we have no way of knowing what node configuration corresponded to the earlier, incorrect responses. Under the most conservative criterion of learning--an error followed by two correct responses--25% of the mismatches can be accounted for by student learning. In each case the model gave the correct response to all three items. [7]

Another 25% of the mismatches occurred when both the model and the student selected different wrong situations as the response option. In these cases, the model correctly determined that the student would not give the correct response. The model's answers may differ from the student's for a number of reasons, including guessing. These were, after all, multiple choice exercises, in which students were asked to select the correct situation from the menu of five possible ones. Students probably guessed at some of the answers, but the model does not guess.

There are other possible explanations for the model failures. On the one hand, some students may have been prone to "slip" as they made their selections using the mouse, resulting in the unintentional selection of the option residing either above or below the desired one. It is not an uncommon phenomenon, as those who use a mouse frequently can attest. Accidental errors of this sort are undetectable. Similarly, students may have used a test-taking strategy, such as avoiding the selection of one response if they used it on the immediately preceding exercise. These errors are also undetectable: The model does not take test-taking strategies into account.

If we consider the "probable learning" mismatches (i.e., those that were followed by two correct matches on the same situation) and the "different error" mismatches (i.e., those in which the model and student both erred but selected different errors) as *understandable or explainable* discrepancies, the total number of mismatches between students and the model is reduced from 77 to 51, leaving only 13 *CS_M* and 38 *CM_S* as mismatches. Thus, the model satisfactorily accounts for 85% of all student responses.

A final evaluation of the model's performance comes from examining how well individual student performance was simulated by the model. The

---

[7] Several other instances exist in which the student made multiple errors on a situation and then responded correctly to one final instance of that situation. While it is very plausible that learning also occurred in these cases, one hesitates to draw a conclusion based only on one response. Thus, these errors remain unexplained.

Table 5
Simulation of Individual Performance:
A Comparison of Model Responses with Observed Student Responses

Student	No. of Items	No. of "Impossible" Matches	Percent Exact Matches	Percent "Explained" Matches	Total Percent Matches
1	13	3	100%	0	100%
2	15	0	80%	7%	87%
3	13	3	90%	10%	100%
4	13	3	80%	10%	90%
5	16	0	75%	6%	81%
6	11	0	100%	0	100%
7	14	0	86%	7%	93%
8	13	0	92%	0	92%
9	14	5	100%	0	100%
10	15	2	85%	7%	92%
11	15	3	67%	8%	73%
12	16	0	63%	31%	94%
13	14	0	71%	0	71%
14	13	0	69%	8%	77%
15	14	3	100%	0	100%
16	13	3	90%	10%	100%
17	15	0	73%	14%	87%
18	14	0	79%	7%	86%
19	14	0	79%	7%	86%
20	18	0	67%	7%	72%
21	13	0	69%	0	69%
22	16	0	69%	12%	81%
23	16	0	56%	13%	69%
24	16	2	57%	7%	64%
25	16	0	69%	12%	81%

results for each student simulation are given in Table 5. Two measures of success are given in the table. The first is the number of exact matches, excluding the "impossible" ones. The second is the overall percentage of satisfactory matches for each individual and is given in the extreme right-hand column of the table. This percentage is based on the number of satisfactory matches, including the "probable learning" and "different error" mismatches described above but eliminating from consideration the "impossible" matches. As can be seen in Table 5, the performance of 6 of the 25 students was fit exactly by the model with 100% agreement. The model simulated the performance of an additional 12 students with accuracy between 80-99%. The model's success rate fell below 70% for only 3 students, to a low of 64%.

DISCUSSION

There are several important implications that result from this study. They are discussed below with respect to the three questions posed in the introduction: What do they learn, when do they learn it, and what can they retrieve?

What specific information does a student learn from initial instruction about a new topic?

One of the most striking findings was that students tended to encode and use specific details from the initial examples used in instruction. Almost all of the example nodes had to do with the five introductory examples, despite the fact that several other examples were given later in the instruction. (See, for example, the *Specific* response of Table 2.) This finding suggests that the very first example of a concept is highly important and should, therefore, be carefully developed. For many students, the initial examples provided the scaffolding for the semantic networks. Some of the details of those examples led to erroneous connections. As a case in point, the example for one of the situations was based on money, leading some students to expect (incorrectly) this situation to be present whenever money was in the problem. These faulty connections were very evident in their interview responses.

A general pattern of encoding was apparent from the students' responses. Several students described the situations only in terms of the examples. When prompted, they were unable to embellish their descriptions by using abstract characterizations. No instance of example information followed by abstract information was observed. In contrast, students having abstract knowledge always used it in preference to giving example details. That is, their initial responses were generalizations. When prompted for more information, they used example details to support their abstract descriptions. These findings suggest that students may first encode the example information and then build the abstract network around it. Once formed, the abstract portion of the network becomes stronger upon exposure to additional examples, whereas the example portion does not augment its activation or strength. If the abstract information is not encoded, the details of the example--which received high strength initially--remain the most salient elements of the network.

How is the information that the student encoded in memory related to the student's performance?

The statistical analyses suggest that the degree to which a student is able to use his or her abstract information is positively related to the student's success on the identification task. Those able to express mainly abstract knowledge apparently had the best understanding of the five concepts and were most easily able to identify them. Those for whom the abstract characterizations were somewhat incomplete (e.g., those who were able to give abstract description for some concepts but needed example details to describe others) performed less well but still were more successful than those who predominantly relied on example details.

The primary implication of this finding is that instruction should be developed to facilitate the linkage of abstract knowledge to easily understood example knowledge. The examples were undoubtedly salient and easily encoded. For some students, the abstract characterizations were equally easy to encode, but this was not universally true.

Does the cognitive model reflect this relationship?

The connectionist model is a useful way to examine individual performance of students as they identified these concepts. The simulation of individual performance was extremely successful. The high level of agreement between model performance and student performance suggests that the model captures most of the salient and discriminating information actually used by the students. Most important, the model demonstrates the impact of missing nodes and erroneously linked pairs of nodes. In many cases, knowledge of which nodes were missing led to accurate predictions of subjects' erroneous responses. In others, incorrect linkages among nodes also led to accurate predictions of errors. The model and its simulation provides strong support for the use of cognitive networks to represent learning of concepts.

REFERENCES

Collins, A. M. & Loftus, E. F. (1975). A spreading activation theory of semantic processing. *Psychological Review, 82,* 407-428.

Marshall, S. P. (1990). Selecting good diagnostic items. In N. Frederiksen, R. Glaser, A. Lesgold, & M. Shafto (Eds.) *Diagnostic monitoring of skill and knowledge acquisition* (pp. 433-452). Hillsdale, NJ: Lawrence Erlbaum Associates.

Marshall, S. P., Barthuli, K. E., Brewer, M. A., & Rose, F. E. (1989). *STORY PROBLEM SOLVER: A schema-based system of instruction.* Technical Report 89-01 (ONR Contract N00014-85-K-0661). San Diego: San Diego State University, Center for Research in Mathematics and Science Education.

Marshall, S. P. (1991). *Understanding the situations of arithmetic story problems: A basis for schema knowledge.* Unpublished manuscript.

Nisbett, R. E. & Wilson, T. D. Telling more than we can know: Verbal reports on mental processes. *Psychological Review, 84,* 231-259.

Riley, M. S., Greeno, J. G., & Heller, J. (1983). Development of children's problem-solving ability in arithmetic. In H. Ginsburg (Ed.), *The development of mathematical thinking* (pp. 153-196). New York: Academic Press.

Tarmizi, R. A., & Sweller, J. (1988). Guidance during mathematical problem solving. *Journal of Educational Psychology, 80* pp. 424-436.

The Interaction Between Knowledge and Practice in the Acquisition of Cognitive Skills*

Stellan Ohlsson
Learning Research and Development Center
University of Pittsburgh

Abstract

The role of prior knowledge in skill acquisition is to enable the learner to detect and to correct errors. Computational mechanisms that carry out these two functions are implemented in a simulation model which represents prior knowledge in *constraints*. The model learns symbolic skills in mathematics and science by noticing and correcting constraint violations. Results from simulation runs include quantitative predictions about the learning curve and about transfer of training. Because constraints can represent instructions as well as prior knowledge, the model also simulates one-on-one tutoring. The implications for the design of instruction include a detailed specification of the content of effective feedback messages for intelligent tutoring systems.

---

* The research reported in this chapter was supported by Grant No. N00014-89-J-1681 from the Cognitive Science Program of the Office of Naval Research. The opinions expressed are not necessarily those of the sponsoring agency, and no endorsement should be inferred. Approved for public release; distribution unlimited.

THE ROLE OF KNOWLEDGE IN LEARNING

Learning and knowledge are doubly related. On the one hand, knowledge is the outcome of learning. On the other hand, knowledge is one of the inputs into the learning process. New skills are constructed within the context provided by prior knowledge. This is no less true of technical domains such as mathematics, science, and engineering than of common sense domains such as cooking and travel planning.

Cognitive scientists from Ebbinghaus (1964/1885) to VanLehn (1982) have sought to escape the complexities of prior knowledge by studying situations in which such knowledge plays a minimal role. This simplification has payed off theoretically. Following the pioneering papers by Anzai and Simon (1979) and by Anderson, Kline, and Beasley (1979) several computational models of the acquisition of cognitive skills in the absence of prior knowledge have been proposed (e. g., Anderson, 1983; Holland et al., 1986; Langley, 1987; Ohlsson, 1987a; Rosenbloom, 1986; VanLehn, 1990). These models assume that procedural knowledge forms a closed loop: Problem solving methods generate problem solving steps which, in turn, generate the experiences from which new problem solving methods are induced. Simulation models of this kind constitute an important advance over the mathematical and verbal learning theories of the past, but the learning mechanisms proposed within this paradigm (chunking, composition, discrimination, generalization, grammar induction, subgoaling, etc.) do not explain the role of prior knowledge in learning. There is no point along the method-step-method loop at which domain knowledge can impact the learning process.

Empirical research of knowledge-based skill acquisition began with Judd's (1908) study of the skill of throwing darts at underwater targets with and without knowledge of the principle of refraction. Both he and later Katona (1940) reported dramatic effects of knowledge about underlying principles on skill acquisition. Kieras and Bovair (1984) also found such an effect, but other recent studies have found weaker effects or no effect (e. g., Gick & Holyoak, 1983; Smith & Goodman, 1984). Educational researchers frequently report that instruction in the relevant domain knowledge does not guarantee correct action (e. g., Resnick & Omanson, 1987; Reif, 1987). On the other hand, inappropriate prior knowledge--so-called misconceptions--is quite likely to interfere with successful problem solving (Confrey, 1990). The empirical results indicate that we do not yet understand how prior knowledge interacts with skill acquisition well enough to ask the right experimental questions.

Theoretical analysis of the function of prior knowledge in skill acquisition has hardly began. Ohlsson (1987b) proposed a computer model which explained how inferential knowledge about the domain enables a learner to find a more efficient strategy for a task which he or she already knows how to solve. The hypothesis behind this model was that domain knowledge allows the learner to reason about possible simplifications of his or her current strategy. The model simulated speed-up of a simple reasoning strategy, but it threw no light on the role of domain knowledge in the initial acquisition of that strategy.

The purpose of the work reported here is to explore the hypothesis that *the function of knowledge in initial skill acquisition is to enable the learner to detect and correct errors*. This hypothesis is embodied in a running simulation model which uses prior knowledge to learn cognitive skills from unguided practice. The theory predicts the negatively accelerated practice curve observed in human learning, throws some new light on the problem of transfer of training, and suggests an analysis of tutoring with some very specific implications for the design of intelligent tutoring systems.

Throughout this chapter, the terms "domain knowledge" and "prior knowledge" refer to declarative knowledge, while the terms "cognitive skill", "problem solving method", "decision rule", and "mental procedures" refer to procedural knowledge. Both common sense and philosophy have long distinguished between theory and practice, between knowing *that* and knowing *how*, but the particular formulation of this distinction used here is imported from Artificial Intelligence (Winograd, 1975).

Procedural knowledge is prescriptive and use-specific. To a first approximation, it consists of associations between goals, situations, and actions. Examples of procedural knowledge are place-value algorithms for arithmetic, methods for electronic trouble shooting, explanatory strategies in biology, and the procedure for constructing structural formulas for organic molecules. Declarative knowledge, on the other hand, is descriptive (as opposed to prescriptive) and use-independent. To a first approximation, it consists of facts and principles. Examples of declarative knowledge are the laws of the number system, the general gas law, Darwin's theory of evolution, and the theory of the co-valent bond. The function of procedural knowledge is to control action; the function of declarative knowledge is to provide generality. Intelligent behavior requires both types of knowledge (Anderson, 1976; Winograd, 1975).

If the two types of knowledge are distinct, how do they interact? In particular, if declarative knowledge is use-independent and distinct from procedures, then how does it influence action? The problem investigated in the research program summarized in this chapter is how (previously learned) declarative knowledge affects the construction of (new) procedural knowledge.

A FUNCTIONAL THEORY OF SKILL ACQUISITION

Learning happens during problem solving; to learn is to adapt to the structure of the task environment; learning is triggered by contradictions between the outcomes of problem solving steps and prior knowledge. These three principles imply a particular functional breakdown of skill acquisition.

Principle 1: Learning as Problem Solving

During practice, the learner is faced with problems which he or she does not yet know how to solve--that is why he or she is practicing. Practice is problem solving and skill acquisition is the encoding of the results of problem solving for future use. People solve unfamiliar problems with so-called weak methods, i. e., problem solving methods which are so general that they can be applied even with a minimum of information about the task environment. The weak methods people have been observed to use include analogical inference, hill climbing, forward search, means-ends analysis, and planning.

Weak methods are general but inefficient. The function of weak methods during practice is not to produce complete or correct problem solutions, but to generate task relevant behavior. Activity vis-a-vis the task provides the learner with the opportunity to discover the structure of the task environment. Cognitive skills are constructed by interpreting, storing, and indexing such discoveries so that they can be retrieved and applied later. The function of weak methods is to provide learning opportunities, not to solve problems.

Individual weak methods were formalized in the late fifties and early sixties (Feigenbaum & Feldman, 1963), but the general category of weak methods was first identified by Newell (1969, 1980). Laird (1986) has suggested that there exists a universal weak method from which all other weak methods can be derived.

The idea that learning is problem solving and that the function of weak methods is to provide learning opportunities is implicit in the concept of trial and error and thus traces its roots back to behaviorism. Although first formalized in a computational model by Anzai and Simon (1979), this idea is central to several recent models of learning (e. g., Anderson, 1986; Holland et al., 1986; Rosenbloom, 1986). In the field of machine learning, the notion that learning occurs *en route* to an answer rather than after completion of a practice problem has been emphasized by Mostow and Bhatnager (1987, 1990) in their work on adaptive search.

Principle 2: Learning as Adaptation

Weak methods are inefficient because they are general. A domain-specific cognitive skill is efficient because it reflects the structure of the relevant task environment. Skill acquisition begins with maximally general procedures (weak methods) and ends with domain-specific skills. Learning is gradual adaptation.

The process of adaptation cannot continue indefinitely. The task environment only contains so much structure and when all the structure has been absorbed, the skill cannot get any more specific or better adapted. In complex and irregular domains, expert strategies are between weak methods and algorithms in specificity. They guide behavior without fully determining it and considerable uncertainty can remain even at the highest level of expertise.

The idea that learning proceeds from the general to the specific is counterintuitive, because it is common sense that learning begins with the concrete and the specific and moves towards the general. The common sense theory has little support in systematic research. Formal analyses of induction (e. g., Angluin & Smith, 1983) have revealed that many induction problems are NP-complete and that noisy input cripples most induction algorithms. David Hume was right; induction does not work. Knowledge must be constructed in some other way. Specialization of pre-existing, general structures is one alternative. The particular version of this idea in which learning proceeds from general *methods* to task-specific *methods* was implicit in early computational models (e. g., Anzai & Simon, 1979), but was to the best of my knowledge first stated in two papers by Langley (1985) and by Anderson (1987).

The idea that learning is adaptation to the environment can be formulated in many different ways, as a comparison between Hull (1943), Piaget (1971), and Anderson (1990) demonstrates. Until recently, psychologists lacked a formal method for describing the learner's environment independently of the learner. This threatened to make the principle of adaptation circular, or at least difficult to apply. The information processing approach is a major breakthrough because it provides a formal description of task environments. Specifically, an environment is described as a *search space* (or problem space; Newell & Simon, 1972). The organism is then naturally described as a strategy for traversing that space. Adaptation has a very definite meaning within this formalization: A given strategy is adapted to a particular task environment in inverse proportion to the amount of search required by that strategy to find a path from the initial state to the goal state. A maximally adapted strategy is one which leads to the goal without extra or unnecessary steps.[1]

---

[1]In an alternative approach, Anderson (1990) describes the environment in terms of its statistical regularities. Many memory phenomena follow from the assumption

Principle 3: Learning as Conflict Resolution

Novices make many errors; that is why we call them novices. Experts do not; that is why we call them experts. The weak methods employed by novices produce errors because they are overly general, causing problem solving steps to be performed in situations in which they are not appropriate. The task-specific skills of experts do not generate errors because they constrain actions to situations in which they are appropriate. The process of adapting a general method to a particular task environment is a process of gradually eliminating errors. Error elimination consists of two subprocesses: error detection and error correction.

Error Detection. Learners can detect their errors in three ways: by observing environmental effects, by self-monitoring, and by being told by others (Reason, 1990, Chap. 6). Some task environments provide direct feedback about errors. If the unknown device exploded when the red button was pushed, pushing the red button was an error. Other task environments do not provide feedback of this sort. In such environments, learners can detect their errors by checking new conclusions against their prior knowledge. Incomplete or incorrect procedural knowledge is highly likely to generate conclusions or problem states that contradict what the learner knows is true of the domain.

As an illustration, consider the following everyday situation: You are driving to an unfamiliar location with the instruction to follow route X north and make a right-hand turn onto Y-street. You are looking for the turn and not finding it. Did you overshoot the turn or did you not go far enough? The only way to decide whether you missed your turn is to know some landmark (e. g., a bridge) which is further out on route X than the turn onto Y-street. (A thoughtful friend includes such a landmark in his or her instructions.) When you see the landmark, you know that you missed your turn. The contradiction between the prior knowl-

---

that memory is adapted to those regularities (Anderson & Schooler, 1991). Anderson (1993) applies this approach to skill acquisition as well.

edge that "Y-street is before the bridge" and the observation "here is the bridge now" allows you to recognize that you have made a mistake.

Technical skills often apply in symbolic task environments in which contradictions between outcomes of problem solving steps and prior knowledge constitute the only indicators of errors. Mathematical symbols do not complain about being inserted into false equalities, unsolvable equations, or incorrect calculations, so a good learner checks his or her calculations. Checking, say, a subtraction by adding the difference and the subtrahend requires the knowledge that the sum of the difference and the subtrahend ought to equal the minuend. Structural formulas for organic molecules do not beep when the laws of the co-valent bond are violated. Noticing an error in a structural formula requires the knowledge that each bond ought to be associated with exactly two electrons, that the total number of electrons cannot exceed the number of valence electrons for the molecule, and so on. The more knowledge, the higher the probability that the learner can detect his or her errors.

Error Correction. The detection of a contradiction between a new conclusion and prior knowledge leads to processes that aim to restore consistency by revising the relevant procedural knowledge. If the execution of action A in situation S_1 leads to a new situation S_2 which violates some principle of the domain, then the mental decision procedure that chose A in S_1 is faulty. The obvious correction is to constrain the procedure so as to avoid executing A in situations like S_1. This requires that the learner identifies the conditions that caused the error, i. e., those properties of S_1 that guaranteed that the error would occur if A were executed. Given knowledge of those conditions, the mental procedure can be revised so as to avoid similar errors in the future.

The principle that learning is error correction superficially resembles Thorndyke's Law of Effect which says that actions with negative consequences are gradually removed from the learner's behavioral repertoire (while actions with positive consequences are strengthened). However, the two principles are distinct, because a cognitive conflict is not necessarily associated with a painful or unpleasant outcome, as the examples given previously illustrate. The error correction principle is also superficially related to the hypothesis that learning is driven by impasses, i. e., situations in which existing procedural knowledge is insufficient to decide what to do next (Newell, 1990; VanLehn, 1988). However, impasses are not errors. An impasse is a situation in which there is insufficient information to make a choice, while an error is a bad choice.

The idea that cognitive change is triggered by contradictions and inconsistencies has been suggested repeatedly in the cognitive sciences. It is central to several recent cognitive models of learning. Holland et al. (1986) put prediction-based evaluation of knowledge at the center of learning: Knowledge is continuously applied in predicting events and rules that lead to wrong predictions are modified. Schank (1982, 1986) has proposed the similar idea that learning is triggered by expectation failures. In developmental psychology, Piaget (1985) designated cognitive conflict, which he called disequilibrium, as the driving force of cognitive development. Empirical investigations support this hypothesis (Murray, Ames, & Botvin, 1977). Social psychologists like Festinger (1957) have proposed that cognitive dissonance causes individuals to revise their beliefs in order to restore consistency (see Abelson et al., 1968, for an overview of cognitive consistency theory). The hypothesis that belief revision serves to maintain consistency has also been proposed by philosophers (Quine & Ullian, 1978) and by science educators (Hewson & Hewson, 1984; Posner et al., 1982).

Machine learning researchers have build systems that learn by resolving conflicts (Hall, 1988; Kocabas, 1991; Rose & Langley, 1986) and by explaining errors (Minton, 1988). The problem of what constitutes a rational response to a contradiction has been studied in logic and Artificial Intelligence under the rubric non-monotonic logic (Gardenfors, 1988; McDermott & Doyle, 1980). Finally, the idea that theory development in science is driven by contradictions between theory and data have been formulated in different ways by Duhem (1991/1914), Kuhn (1970), and Popper (1972/1935). The relevance of these philosophers for psychology is highlighted by Berkson and Wettersten's (1984) attempt to recast Popper's philosophy as a learning theory. In short, the idea of cognitive change as a response to conflict, contradiction, or inconsistency has been proposed by so many researchers independently of each other and in so many different fields that it deserves to be recognized as one of the great unifying principles of the cognitive sciences.

Summary

During practice the learner continuously monitors his or her progress by comparing the current state of the practice problem to his or her prior knowledge about the domain. A problem state that contradicts something that is known to be true of the domain indicates that an error has been made. When such a contradiction is noticed, the current problem solving method is constrained so as to avoid making similar errors in the future. As practice progresses, the general method becomes more and more constrained and better and better adapted to the task environment. Eventually it has become transformed into the correct domain-specific skill and ceases to generate errors.

According to this theory, prior knowledge impacts skill acquisition in two ways. First, knowledge allows the learner to detect his or her errors. Facts and principles of the domain generate implications that an incomplete or incorrect skill is likely to violate or contradict. The more knowledge the learner has, the higher the probability that he or she will be aware of the contradictions and conflicts generated by a faulty solution or a mistaken problem solving step.

Second, prior knowledge allows the learner to identify the conditions that caused the error. Finding the cause of an error might require complicated reasoning about the domain. The more knowledge the learner has, the higher the probability that he or she accurately identifies the cause, which in turn is a prerequisite for successful error correction.

In short, the theory put forth here claims that the function of acquiring new skills through practice consists of three main subfunctions--to generate task-relevant behavior, to identify errors, and to correct errors--each of which, in turn, can be analyzed into subfunctions. The functional analysis is summarized in Figure 1. Although the theory supports qualitative arguments and explanations, the derivation of quantitative behavioral predictions requires a working information processing system.

I. Learn to do unfamiliar task

A. Generate task-relevant actions

1. Apply forward search
    a. Retrieve possible actions
    b. Select action
    c. Execute action

B. Learn from erroneous actions

1. Detect errors
    a. Check consistency between current problem state and prior knowledge after each action

2. Correct error

a. Extract information from error
    i. Identify the conditions under which a particular action is incorrect

b. Revise current task procedure
    i. Constrain procedure so as to avoid that action under those conditions

Figure 1. The functional analysis of learning from error.

A COMPUTATIONAL MODEL

To move from a functional theory to a working model one must specify particular representations and processes that can compute the functions described in the theory. In particular, an implementation of the present theory requires (a) a performance mechanism, including a representation for procedural knowledge, (b) a representation for declarative knowledge, (c) a mechanism for detecting errors, and (d) a mechanism for correcting errors. The particular model described here is called the *Heuristic Searcher* (HS).

A Standard Performance Mechanism

Memory Architecture. HS has three memory stores. The *working memory* holds the model's knowledge state, corresponding to the learner's perception of the current state of the practice problem. The *procedural memory* holds the model's procedural knowledge, corresponding to the learner's previously acquired skills. The *long-term memory* holds the model's declarative knowledge, corresponding to the learner's prior knowledge about the domain. There is no separate goal stack. Goals are represented in working memory.

Procedural Knowledge. Procedural knowledge is represented in so-called production rules (Newell & Simon, 1972), i. e., rules of the general form

Goal, Situation --> Action,

where *Goal* is a description of what the learner believes he or she is supposed to achieve in the practice problem, e. g., "construct the structural formula for C_2H_5OH," and *Situation* is a description of a class of situations, e. g., "situations in which the carbon skeleton of the molecule has been completed but no other atoms have been connected yet." Formally speaking, both *Goal* and *Situation* are patterns, i. e., conjunctions of elementary propositions which may or may not contain (universally quantified) variables.

The action on the right-hand side of a production rule is a problem solving step that the model knows how to perform, e. g., "connect the oxygen atom to one of the carbon atoms". Actions have applicability conditions that have to be satisfied before they can be applied. For example, an oxygen atom cannot be attached to a carbon atom unless there is a carbon atom for it to be attached to. Each action is implemented as a piece of Lisp code that revises the current problem state by deleting some propositions and adding others. Syntactically, the actions are so-called Strips operators (Fikes & Nilsson, 1971). Psychologically, the actions correspond to components of the practice problem which are unproblematic for the learner.

Each production rule is a single unit of procedural knowledge, corresponding to a single problem solving heuristic. The skill required to solve problems of a particular type, e. g., to construct structural formulas in chemistry, consists of a collection of interrelated rules. All production rules are stored in the single production memory, without structural divisions between different skills.

Operating Cycle. The model solves problems by searching a problem space. The content of the working memory at the time the system is initialized is the initial state of the search space. The top goal implicitly specifies the goal state. The ensemble of operators consists of the set of actions the model has been given as input. In each cycle of operation, the *Goals* and *Situations* of the rules are matched against the working memory with a version of the RETE pattern matching algorithm developed by Forgy (1982). If a rule matches, its action is executed.

If more than one rule matches the current state, each matching rule is evoked and one new descendant of the current state is generated for each evoked rule. The entire search tree is saved in memory. Each cycle begins with the selection of which search state to install as the current state for that cycle. In some applications of HS, the selection of the current state is based on a task specific evaluation function, in which case the model performs best-first search. If the evaluation function has the right properties and, in addition, the system checks for repeated occurrences of the same state[2], then the model executes the A* algorithm (Pearl, 1984, p. 64). In the absence of any evaluation function, the state to expand next is selected randomly among the immediate descendants of the current state, in which case the model performs depth-first search. In psychological terms, the performance mechanism correspond to the hypothesis that people respond to uncertainty by thinking through alternative actions before deciding what to do next.

A Representation for Declarative Knowledge

The function of procedural knowledge is to control action. The function of declarative knowledge is not equally obvious. Philosophical discussions often assume that the function of declarative knowledge is to provide descriptions of the world ("the cat is on the mat"), predictions about future events ("the sun will rise tomorrow"), or explanations ("it is snowing, because the temperature fell"). The epistemological, logical, and semantic riddles associated with these functions have exercised thinkers in a variety of disciplines for centuries.

The HS model is based on a different view of the nature and function of declarative knowledge. Declarative knowledge is not used either to describe, predict, or explain but *to circumscribe a set of states of the world.* The unit of declarative knowledge is a *constraint.* Constraints can be interpreted descriptively, i. e., as circumscribing the set of possible states of the world. For example, the law of conservation of mass claims that the mass of the reactants in a chemical experiment is equal to the mass of the reaction products. Mass is neither created nor destroyed in a chemical reaction, so the mass of the inputs is always equal to the mass of the outputs. The point of the mass conservation law is that it circumscribes situations in which mass is conserved, which are possible, and separates them from situations in which mass is not conserved and that it rules out the latter as impossible. Figure 2 shows the constraint interpretation of the mass conservation law.

Constraints are not limited to representing abstract principles like the law of conservation. Particular facts are also constraints. For exam-

[2]This facility is computationally expensive and is usually switched off.

Example 1: A scientific principle

Idiomatic English:	Energy cannot be created or destroyed.
Constraint formulation:	If the mass of the reactants for a chemical experiment is M_1 and the mass of the products is M_2, then M_1 must be equal to M_2.
Formal representation:	(Reactants R) (Mass R M_1) (Products P) (Mass P M_2) ** (Equal M_1 M_2)

Figure 2. Encoding a scientific principle as a constraint.

ple, the fact that alcohol molecules have an OH-group corresponds to the constraint that a structural formula for an alcohol had better have an OH-group somewhere. Figure 3 shows the constraint interpretation of this fact.

Constraints can also be interpreted prescriptively, i. e., as circumscribing the set of *desired* states of the world. The ordinance that one should not drive along a one-way street in the wrong direction is a constraint. Specifically, the fact that Fifth Avenue is one-way in the westerly direction corresponds to the constraint that if you are driving on Fifth Avenue, you had better be heading west. It is not impossible to head east, it is merely undesirable. Figure 4 shows the constraint interpretation of this ordinance.

It is a mistake to try to classify individual constraints as either descriptive or prescriptive. All constraints can be interpreted in both ways, because the two interpretations determine each other. It is desirable that a chemistry experiment satisfies the constraint that the mass of the reac-

---

Example 2: A scientific fact

Idiomatic English:	Every alcohol molecule has an OH-group.
Constraint formulation:	If X is an alcohol molecule, then it must have an OH-group.
Formal representation:	(Isa X molecule) (Substance X ALCOHOL) ** (Isa Y OH-GROUP) (Part-of Y X)

Figure 3. Encoding a scientific fact as a constraint.

tants is equal to the mass of the reaction products. If this is not the case, then some error was committed in the execution of the laboratory procedure, i. e., some mass was accidentally lost or the experiment was contaminated in some way (Gensler, 1987). The constraint expressed in the mass conservation law acquires a prescriptive function because it can be interpreted descriptively; a laboratory procedure ought to conform to it precisely because it is true. The descriptive and prescriptive aspects of constraints are inseparable.

The main contribution of the HS model is a formal representation for constraints and a set of processes for using them. A constraint C is represented as an ordered pair

$$\langle C_T, C_S \rangle$$

where C_T is a *relevance criterion*, i. e., a specification of the circumstances under which the constraint applies, and C_S is a *satisfaction criterion*, i. e., a condition that has to be met for the constraint to be satis-

Example 3: An everyday fact

Idiomatic English:	Fifth Avenue is a one-way street heading west.
Constraint formulation:	If someone is driving on Fifth Avenue, then he or she ought to travel westwards.
Formal representation:	(State X DRIVING) (Location X FIFTH-AVENUE) ** (Direction X WEST)

Figure 4. Encoding an everday fact as a constraint.

fied. To continue the traffic example, if Fifth Avenue is one-way in the westerly direction, then "driving on Fifth Avenue" is the relevance criterion and "is heading west" is the satisfaction criterion. If I am not on Fifth Avenue, the direction of my travel is not constrained by this ordinance, but when I am on Fifth, then I had better be driving west rather than east. In the mass conservation example, "M_1 is the mass before the reaction and M_2 is the mass after the reaction" is the relevance criterion, while the equality "$M_1 = M_2$" is the satisfaction criterion.

The double star connective (**) that appears in Figures 2-4 is not a symbol for logical implication. Constraints are not inference rules; they do not generate conclusions. Nor are they production rules; they do not fire operators. The semantics of the double star connective is similar to the meaning of "ought to", "had better", and related phrases. The interpretation of a constraint $<C_r, C_s>$ is that whenever C_r is the case, C_s *ought to* be the case as well (or else something has gone awry). Syntactically, both C_r and C_s are patterns, i. e., conjunctions of propositions similar to the condition side of a production rule.

The HS model does not have any mechanism for acquiring or revising its declarative knowledge. The constraints are input by the user and they stay unchanged throughout a simulation run. The purpose of the constraints is to facilitate the detection and correction of errors.

A Mechanism for Error Detection

At the beginning of each operating cycle, all production rules are matched against working memory, the rules with matching condition sides are evoked, the actions of those rules are executed, and new problem states thus generated. Each new state is matched against all the available constraints. (The match is computed with the same pattern matcher which matches the production rules.) Constraints with non-matching relevance patterns do not warrant any action on the part of the system, because they are irrelevant. Constraints which have matching relevance patterns and also matching satisfaction patterns are ignored as well. The new state is consistent with the those constraints so no action is required. On the other hand, if a constraint with a matching relevance pattern has a non-matching satisfaction pattern, then the new state violates that constraint and some response or action is called for. Such a *constraint violation* signals that something is wrong with the procedure that generated the current state; an error has been committed.

Specifically, consider a rule R with goal G and a conjunction S of situation features in its left-hand side and a single action A in its right-hand side,

$$R:\ G, S \dashrightarrow A,$$

and a constraint C with relevance pattern C_r and satisfaction pattern C_s,

$$C = \langle C_r, C_s \rangle,$$

where both C_r and C_s are conjunctions of situation features. In particular, let us assume that C_r and C_s each consists of two features:

$$C_r = C_r' \& C_r''$$

and

$$C_s = C_s' \& C_s''.$$

Finally, let us assume that the effect of action A is to add the conjunction of C_r'' and C_s' to the current problem state, i. e.,

$$A = \text{Add}[C_r'' \& C_s'].$$

If a learner with rule R and constraint C encounters a problem state S_1 described by

$$S \& C_r',$$

then the left-hand side of R is satisfied because S is present, so the rule will be evoked and action A executed. The effect is that C_r'' and C_s' are added to S_1, yielding a new problem state S_2 described by

$$S \& C_r' \& C_r'' \& C_s'.$$

In this problem state, both C_r' and C_r'' are present, so C_r matches, i. e., the constraint is relevant. Although C_s' is present, C_s'' is not, so C_s is violated; hence, doing A in situation S_1 was an error.

In principle, there are two possible interpretations of the constraint violation: The fault might lie either with the procedural knowledge--the rule--or with the declarative knowledge--the constraint. Because HS was designed to model skill acquisition, as opposed to the acquisition of declarative knowledge, it assumes that the rule rather than the constraint is at fault.

A Mechanism for Error Correction

A constraint violation is a signal that the procedural knowledge that generated the current problem state is faulty and needs to be revised. HS assumes that the fault lies with the last rule to fire. The problem of how to learn from the constraint violation can be stated as follows: Given that rule R,

$$R: \; G, S \dashrightarrow A,$$

was applied to state S_1 and that it generated state S_2 and that S_2 violates constraint C, how should the rule be revised? The purpose of the revision is to avoid similar constraint violations in the future. The learning mechanism in the HS model accomplishes this by finding the cause of the constraint violation, i. e., the properties of state S_1 that were responsible for the error, and revising rule R so that it does not apply under those conditions. The learning mechanism finds the relevant properties of S_1 by regressing the violated constraint through the rule with a variant of the standard regression algorithm used in many A. I. systems (Nilsson, 1980, p. 288).

More specifically, rule R is replaced with two new rules R' and R", representing two different revisions of R. The purpose of the first revision is to constrain R so that the new rule will apply only in situations in which constraint C is guaranteed to remain irrelevant. This is accomplished by regressing the relevance pattern through the rule. Continuing the example from the previous subsection, regressing the relevance pattern $C_r = (C_r' \;\&\; C_r'')$ through the operator A = Add[C_r'' & C_s'] yields C_r' as the only output (see Nilsson, 1980, p. 288, for an explanation of the regression algorithm). The first new rule is constructed by adding the negation of the output from the regression to the original rule:

$$R': \; S \;\&\; \textit{not}\; C_r' \dashrightarrow A$$

This rule applies only in those situations in which the constraint is guaranteed to remain irrelevant if action A is executed. Psychologically, the rule corresponds to the knowledge that one should only do A when S is true but C_r' is false (e. g., "if the device needs repair and the power is not on, then open the front panel").

The purpose of the second revision is to constrain rule R so that it applies only in situations in which the constraint C is guaranteed to become both relevant and satisfied if A is executed. This is accomplished by regressing the entire constraint through the rule, instead of the relevance pattern. Regressing (C_r' & C_r'' & C_s' & C_s'') through the operator A = Add[C_r'' & C_s'] yields (C_r' & C_s'') as the output (see Nilsson, 1980, p. 288). The second new rule is constructed by adding this result to the original rule (without negating it):

$$R'': \; S \;\&\; C_r' \;\&\; C_s'' \;\text{-->}\; A$$

This rule applies only in those situations in which the constraint is guaranteed to become satisfied if A is executed. Psychologically, the rule corresponds to the knowledge that one should only do A when S, C_r', and C_s'' are all true (e. g., "if the device needs repair, the power is on, and the red light is blinking, then switch off the power").

Figure 5 provides a graphical interpretation of the learning mechanism. The set S of situations in which the original rule R applies is split into three subsets when the rule is revised. The first subset contains those situations in which the constraint is guaranteed to remain irrelevant if action A is executed. They are covered by the first new rule. The second subset contains those situations in which the constraint is guaranteed to become satisfied if A is executed. They are covered by the second new rule. The third subset contains those situations in which doing A leads to a constraint violation. They are thrown away, as it were. Neither of the two new rules apply in those situations, so the error type represented by the third subset has been eliminated.

The fact that one type of error has been eliminated does not imply that the two new rules R' and R" are correct. Although the new rules have been revised so as to be consistent with one constraint, they might

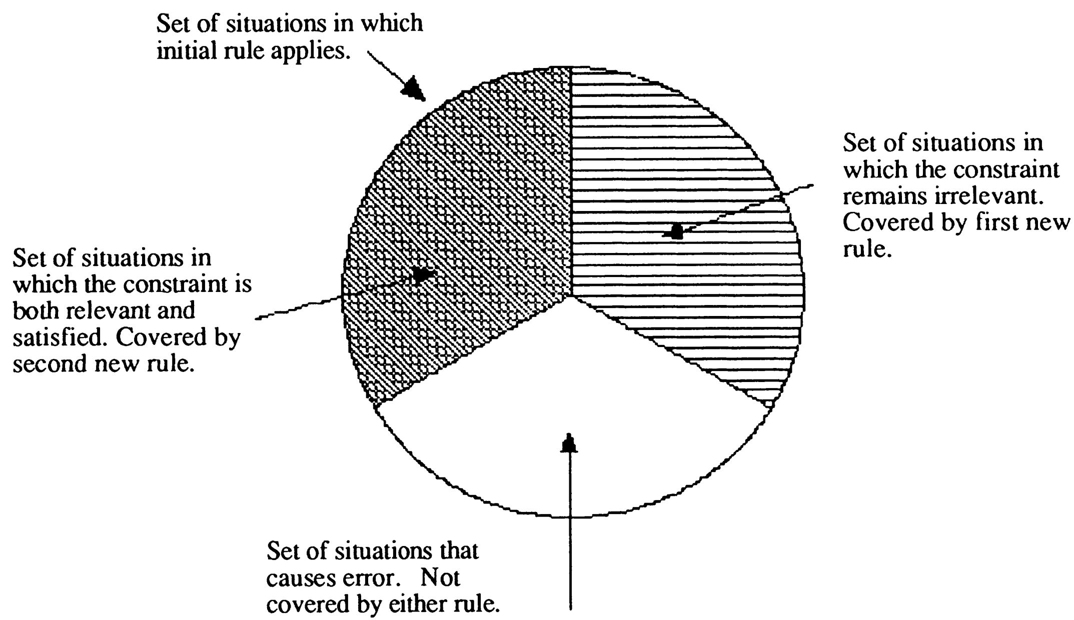


Figure 5. Graphical interpretation of the HS learning mechanism.

still violate other constraints and so have to be revised further. Repeated revisions of rules is the standard case in HS learning. Also, the fact that one rule has been revised does not imply that other rules are correct. Learning proceeds by gradual correction of the relevant rule set as a function of the errors that the model encounters during practice. A detailed analysis of the correction of an entire rule set is available in Ohlsson and Rees (1991a, Table 5).

Discussion

The HS model is based on two representational assumptions: that procedural knowledge is represented in production rules and that declarative knowledge is represented in constraints. The production system format was proposed by Newell and Simon (1972) but has been taken up by other researchers (Klahr, Langley, & Neches, 1987). The main claim of the production system hypothesis is that human action is determined by an external context, represented by the situation the learner is faced with, and an internal context, represented by the learner's goal. Procedural knowledge consists of associations between goals, situations, and actions. The individual production rule is the smallest unit of procedural knowledge; it maps a single goal/situation pair onto a particular action.

A second claim of the production system hypothesis is that the units of procedural knowledge are modular. Production rules do not access or operate upon each other. They only interact through their effects on working memory. There is strong empirical evidence for the modularity of procedural knowledge (Anderson, 1993).

The constraint format originated with the current theoretical effort (Ohlsson & Rees, 1991a) and it does not have any empirical or theoretical support other than the success of the model it is embedded in. There has been so little progress on the epistemological, logical, and semantic problems associated with the standard, propositional interpretation of declarative knowledge that any alternative conception is worth exploring.

Given the two representational assumptions, information processing mechanisms that compute the functions specified in the abstract theory (see Figure 1) can be specified. In HS, the function of generating task relevant activity is carried out by forward search, the function of detecting errors is carried out by a pattern matcher, and the function of correcting errors is carried out by a rule revision algorithm based on regression. There are alternative ways to compute each of these functions. HS could have been implemented with, for example, analogical transfer instead of heuristic search as the weak method responsible for generating task relevant behavior. Similar substitutions of alternative mechanisms are possible for each of the other functions specified in the theory. The predictions generated by running the model are consequences of both the theoretical principles that guided its design and the particular representations and processes that are implemented in it.

Compared to many other machine learning systems, HS is very simple. It combines a standard production system architecture, a well-known weak method, and an off-the-shelf regression algorithm; little else is needed. HS is implemented in Lucid Common Lisp and runs on a Sun Sparcstation 1+ with 16 megabytes of main memory. The core mechanisms have been debugged in hundreds of simulation runs in different domains over a period of four years and are very robust.

APPLICATIONS TO CLASSICAL RESEARCH PROBLEMS

A good theory should throw new light on the perennial problems of the discipline. The learning curve and transfer of training have been central problems in the theory of learning for a long time.

The Learning Curve

Background. If performance level, measured in terms of time to complete a practice problem, is plotted as a function of amount of practice, measured in terms of the number of practice problems solved, i. e., the number of trials, the result is a negatively accelerated curve. The rate of improvement is fastest at the beginning of practice and quickly slows down as mastery is approached. This type of learning curve has been observed in a large number of studies, across many different tasks, and in widely varying subject populations (Lane, 1987; Mazur & Hastie, 1978; Newell & Rosenbloom, 1981; Ohlsson, 1992c).

Armchair reasoning would lead one to expect learning to be slow in the beginning, when the learner is still groping to understand the practice task and there is little relevant knowledge or skill to build on. Later in the practice sequence, the partial knowledge built up during previous trials serves as a lever for acquiring more knowledge, with increased speed of learning as a result. However, research leaves no doubt that the opposite is the case: The rate of skill acquisition is faster the less the learner knows about the task. No theory of practice is viable unless it can explain this unexpected finding.

The hypothesis that skill acquisition is the elimination of errors provides such an explanation. According to this hypothesis, knowledge is revised when the learner becomes aware of an error. Learning is thus a sequence of *learning events*, with one error (type) being eliminated per event. The prediction of a negatively accelerated learning curve follows from this hypothesis in three easy steps:

1. The consequence of an error is floundering, i. e., unnecessary search. Performance improves when the error is corrected because the unnecessary search is eliminated. Let us assume that the amount of unnecessary search caused by an error is approximately constant across errors. Performance then improves with a constant amount *per learning event.*
2. At the outset the learner makes many errors on each practice problem precisely because he or she knows so little about the task. As mastery is approached, the number of mistakes per problem decreases because many errors have already been eliminated. There are fewer and fewer learning events *per trial* as practice progresses.
3. Constant improvement per learning event and decreasing number of learning events per trial imply a decreasing rate of improvement per trial.

This explanation does not depend on the details of particular information mechanisms. Any theory or model which claims that learning events are triggered by trouble situations--defined as cognitive conflicts, contradictions, errors, expectation failures, impasses, wrong answers or in any other way--implies this explanation, because trouble situations disappear as mastery is approached, by definition of "mastery."

The qualitative argument explains why we should expect the rate of improvement to slow down across trials, but it does not make a specific prediction about the shape of the learning curve. Newell and Rosenbloom (1981) have reviewed the evidence that the human learning curve is a member of the class of curves described by so-called power laws, i. e., by equations of the general form

$$T = A + kP^{-r} \tag{1}$$

where T is the time to complete the current practice problem, A is the asymptotic performance, P is the amount of practice in trials, and k and r are constants.

Simulating the Learning Curve. To derive the learning curve predicted by the present theory, a simulation experiment was run with the HS model. A problem solving skill from the domain of chemistry was chosen as the target for the simulation. Chemists frequently need to know the interconnections between the atoms in a molecule. The interconnections are specified in structural formulas, so-called *Lewis structures*. A Lewis structure shows which atoms in a molecule are bound to which other atoms and by which kind of bond. The task of constructing the Lewis structure for a particular molecule, specified through its molecular (sum) formula, will here be called a *Lewis problem*. Figure 6 shows the initial state and the goal state of a Lewis problem. There is usually more than one path to the goal state. Figure 7 shows one such path. The cognitive skill of solving Lewis problems is taught in the beginning of college level courses in organic chemistry (e. g., Solomons, 1988).

The HS model was given a representation for atoms, molecules, valencies, bonds between atoms, and the other entities, properties and relations that are important in the chemistry environment. The actions involved in Lewis problems are to select atoms, to connect atoms, to

Initial state:
A sum formula

$$CH_3CH_2OH$$

Goal state:
A Lewis structure

```
     H   H
     |   |
H -  C - C - O - H
     |   |
     H   H
```

Figure 6. Initial state and goal state for a Lewis problem.

make double bonds, and so on. Figure 8 summarizes the problem space for Lewis problems.

In order to attempt to solve practice problems, HS must be given an initial procedure. In this application, the model was given a set of very general initial rules that encode a procedure for how to construct Lewis structures that approximates the verbal recipes given in chemistry text-books (e. g., Solomons, 1988, pp. 10-11; Sorum & Boikess, 1981, pp. 104-107). Finally, in order to detect and correct its errors, the model must have some prior knowledge about the domain. It was given a set of constraints that encode some relevant facts about the chemistry of alcohols, ethers, and pure hydrocarbons.

Nine molecules--three alcohols, three ethers, and three hydrocarbons--were selected as practice problems. The model solved each of the nine problems, presented in random order. This corresponds to the simulation of a single subject going through a sequence of nine different

1. Connect the carbons: $\mathrm{C - C}$

2. Attach the oxygen: $\mathrm{C - C - O}$

3. Complete the OH-group: $\mathrm{C - C - O - H}$

4. Distribute the hydrogens:

$$\begin{array}{ccccccccc} & & \mathrm{H} & & \mathrm{H} & & & & \\ & & | & & | & & & & \\ \mathrm{H} & - & \mathrm{C} & - & \mathrm{C} & - & \mathrm{O} & - & \mathrm{H} \\ & & | & & | & & & & \\ & & \mathrm{H} & & \mathrm{H} & & & & \end{array}$$

5. Add electron pairs:

$$\begin{array}{ccccccccc} & & \mathrm{H} & & \mathrm{H} & & & & \\ & & | & & | & & & & \\ \mathrm{H} & - & \mathrm{C} & - & \mathrm{C} & - & \mathrm{O} & - & \mathrm{H} \\ & & | & & | & & & & \\ & & \mathrm{H} & & \mathrm{H} & & & & \end{array}$$

Figure 7. A solution path for the Lewis problem in Figure 6.

Representation
Symbols that represent atoms, electron pairs, molecules, noble gas configurations, numbers, single, double and tripple bonds, substances, types of carbon arrangements (branched structures, chains, and rings), two-dimensional spatial relations, and valencies.

Initial state
A molecular (sum) formula.

Operators
Select an atom, place the first atom, attach an atom to the molecule, identify open bonds, create multiple bonds, and add electron pairs.

Goal state
A correct Lewis structure for the given molecule. A Lewis structure must (a) connect all the atoms in the sum formula, (b) not include any other atoms than those in the sum formula, (c) have a number of valence electrons equal to the sum of the valence electrons of the atoms, and (d) give each atom a noble gas configuration.

Figure 8. A problem space for Lewis problems.

practice problems. The model was then re-initialized and run through the nine problems once again, simulating a second subject. All in all, the model worked through the nine practice problems 357 times, each time in a different random order, thus simulating a learning experiment with that number of subjects. Figure 9 summarizes the initial knowledge, the training procedure, and the outcome of the chemistry simulation.

The data from the simulation runs were aggregated by averaging the performance of all 357 simulated subjects for each trial. The average performance on each trial was plotted as a function of trial number. (This corresponds to how learning curves are constructed from psycho-

Prior procedural knowledge
The model began with a procedure that connects the heavy atoms, adding multiple bonds if needed, connects the hydrogens, and then adds the final electron pairs. This procedure generates correct Lewis structures, but requires large amounts of search.

Prior declarative knowledge
There were 16 constraints which encode knowledge about (a) properties of particular classes of molecules, e. g., that alcohols have a C-O-H group and that ethers have a C-O-C group, (b) spatial properties of the possible carbon skeletons (branched structures, chains, and rings), and (c) the distribution of hydrogens across the molecule.

Training
The model was given unsupervised practice on a mixed set of Lewis problems that included alcohols, ethers, and pure hydrocarbons.

Learning outcome
The model learned a set of rules for constructing Lewis structures for the relevant molecules with a minimal amount of search.

Figure 9. Summary of the chemistry simulation.

logical data.) Figure 10 shows the results. Performance as a function of practice approximates a straight line when plotted with logarithmic coordinates on both axes, the hallmark of a curve described by a power law. The HS model thus predicts that improvement over time follows the particular shape that has been observed in data from human learning.

The qualitative argument for why learning from error predicts a negatively accelerated learning curve is based on the simplifying assumption that there is a constant improvement per learning event. How

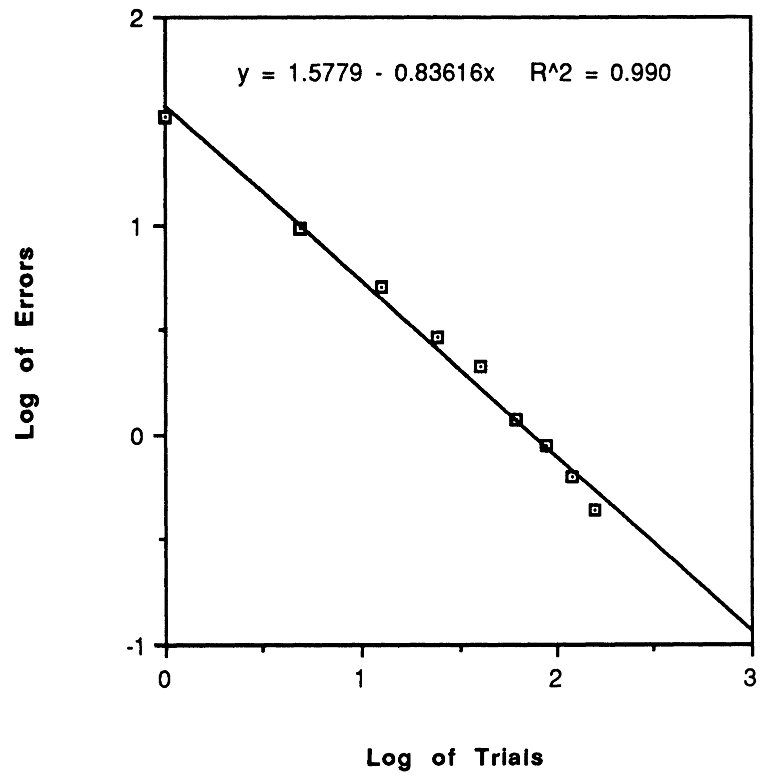


Figure 10. Performance as a function of trials.

realistic is this assumption? The assumption is true in *approximately uniform* task environments. By approximately uniform I mean that the average branching factor in a small neighborhood around a search state is equal for all states in the search space. If this is true and if performance is plotted as a function of learning events instead of as a function of trials, then the results should be a linear relationship with negative slope. Figure 11 shows the results from a simulation run in which HS was given repeated practiced on a particular Lewis problem. When performance is plotted as a function of learning events, the result approximates a negative linear relationship, indicating that the chemistry environment is, in fact, approximately uniform. An empirical test of the prediction that human learning is linear in the number of learning events (in this task environment) is possible in principle but requires a method for identifying learning events in human data.

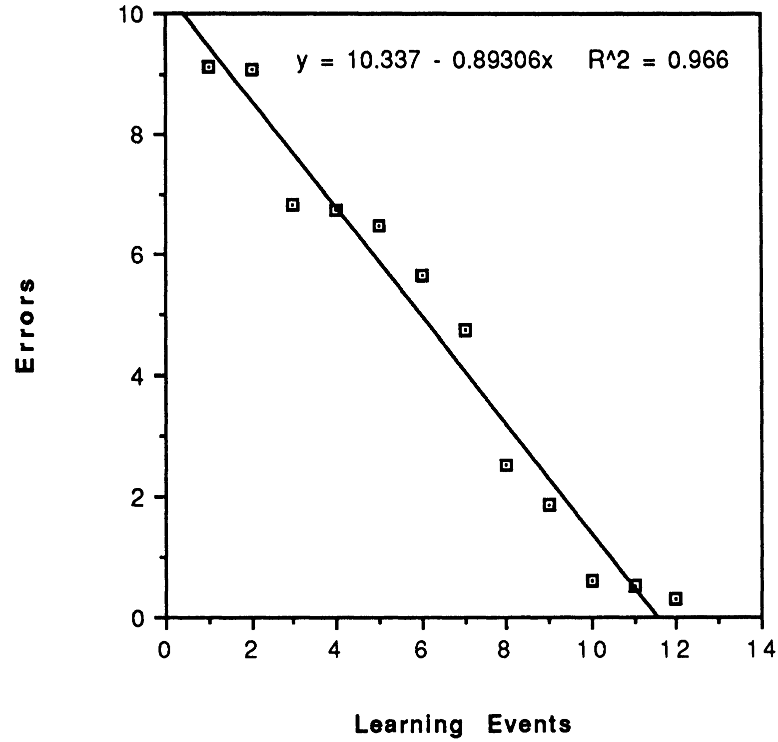


Figure 11. Performance as a function of learning events.

Transfer of Training

Background. Knowledge must be applicable in other situations than the one in which it was learned in order to be useful, but many laboratory studies have recorded little or no transfer of procedural knowledge even between isomorphic problems (Cormier & Hagman, 1987; Singley & Anderson, 1989, Chap. 1). Although many models of learning try to elucidate the mechanism of transfer (Ohlsson, 1987a; Singley & Anderson, 1989), the empirical data imply that *the main task for a transfer model is to elucidate why transfer of training does not occur.* In spite of the negative findings, psychologists keep trying to identify conditions that produce transfer, presumably because the findings strongly contradict our experience of ourselves as creatures with general and flexible competence. A second task for a theory of skill acquisition is to resolve this appearent contradiction between the laboratory findings and our intuitive self-understanding.

The production system hypothesis solves the first of these explanatory tasks. If procedural knowledge is encoded in production rules and if the rules required to solve a training task A are different from the rules required to solve a target task B, then practice on A will not affect the amount of learning required to master B, which is the typical laboratory result. Production rules are task specific, so they do not transfer.

From the point of view of common sense, the lack of transfer of training between isomorphic problems is particularly puzzling. A series of experiments with different versions of Duncker's ray problem has shown that unless subjects are explicitly reminded of the training task, transfer to an isomorphic target task is limited (Gick & Holyoak, 1987, pp. 34-37). Other experiments have verified that people behave differently on isomorphs of the Tower of Hanoi problem (Hayes & Simon, 1977) as well as on different isomorphs of the so-called selection task (see Evans, 1982, Chap. 9, for a review).

According to the production system hypothesis, these results are to be expected. Production rules for moving disks between pegs cannot also transfer globes among monsters; production rules that split up and recombine X-rays cannot also split up and recombine army platoons; production rules that decide whether envelopes have the proper postage cannot also test abstract rules; and so on. Production rules contain variables, but they quantify over arguments to predicates, not over predicates. There is no reason to expect a production rule to facilitate the construction of other rules isomorphic to itself, particularly not if the intended isomorphism is unknown to the learner.

The non-transferability of procedural knowledge leaves us with a picture of human beings as brittle systems which can only solve the very tasks that they have practiced. If this is true, then how do we survive even a single day of normal life?

The first answer is that the zero transfer prediction must be moderated by the distinction between *far transfer*, in which the target task differs completely from the training task, and *near transfer*, in which the two tasks partially overlap. In far transfer situations (which include almost all instructionally relevant situations) there is no overlap in the production rules for the two tasks and the production system hypothesis predicts zero transfer. In the near transfer case, on the other hand, there are rules in the procedure for the training task which are identical to rules in the procedure for the target task. In this case, there will be a transfer effect. Singely and Anderson (1989) claim that the number of production rules shared between two tasks is a good predictor of the amount of transfer in near transfer situations.

The second and more important answer suggested by the present theory is that generality resides in a person's declarative knowledge rather than in his or her procedural knowledge. It is our concepts and beliefs about the world that transfer from one situation to another, rather than our skills. We understand how the world works well enough so that we are able to construct the procedural knowledge required by novel circumstances and conditions. We cope by generating new procedures, not by transferring old procedures to new situations.

This explanation suggests that psychologists have been studying the wrong paradigm. Transfer studies have focussed on pairs of tasks which have similar solutions. In the typical transfer experiment, the experimenter varies the degree of similarity between the *solution* to a training task and the *solution* to a target task and expects the amount of transfer to vary accordingly. The negative findings from studies of isomorphic problems command attention because the solutions to those problems are structurally identical and so ought to yield perfect transfer.

However, the hypothesis that generality resides in declarative knowledge implies that structural similarity between solution paths is irrelevant. The important factor is whether two skills share a common conceptual rationale. If the skills required to solve two tasks A and B can both be derived from a set of beliefs or abstract principles T, then knowing T should give the ability to solve both A and B. The fact that two different procedures have the same theoretical rationale does not imply that there is any formal or structural similarity between the problem solutions generated by those procedures. For example, a chemical analysis of an unknown compound and a synthesis of a particular substance are procedurally different, but both are based on the same theory of the composition of matter.

Simulating Transfer of Training. The skill acquisition literature contains few studies of procedurally different skills which have the same declarative rationale, but developmental psychologists have found a naturally-occurring instance of this type of situation. Gelman and Gallistel (1978) have argued that children learn to count sets of objects by deriving the correct counting procedure from their intuitive understanding of its rationale. They formulated the declarative knowledge required for correct counting into a set of well-defined counting principles and presented empirical evidence that children know these principles at the time they learn how to count. Knowledge of the counting principles should give the ability to construct not only the procedure for the standard counting task, but to solve two non-standard counting tasks as well: to count objects in a particular order, so-called *ordered counting*, and to count objects in such a way that a designated object is assigned a designated number (e., g., "count the objects so that the red object becomes the fifth one"), so-called *constrained counting*. The empirical evidence confirms that children can quickly generate the correct procedures for these non-standard counting tasks (Gelman & Meck, 1983, 1986; Gelman, Meck, & Merkin, 1986).

To simulate this situation, the HS model was given a problem space for the task of counting a given set of objects. The representation included symbols for objects, numbers, for relations between objects and numbers, and so on. The actions included to select an object, to select a number, and to assign a number to an object. Figure 12 summarizes the problem space for counting.

The model was given rules which knew how to apply the operators, but which did not know how to apply them correctly. Finally, the model was given the counting principles in the form of constraints. Figure 13 summarizes the counting simulation. More detailed reports are available in Ohlsson and Rees (1991a, 1991b).

The model was trained on each of the three procedures for standard counting, ordered counting, and constrained counting. The diagonal of Table 1 shows the results as reported in Ohlsson and Rees (1991b). The effort required was measured in two ways, by the number of production system cycles and by the number of learning events, i. e., rule revisions. The model learned each procedure in approximately the same

Representation
Symbols for objects, numbers, sets of objects, and associations between numbers and either objects or sets. Both objects and numbers can have the properties of being *first*, *current*, and *point of origin*, and numbers can have the property of being the *answer*. The relations represented are *correspondancy, set membership, successor,* and *temporal contiguity*.

Initial state
A set of objects to be counted.

Operators
Associate a number with an object, associate a number with a set, select a first object, select the next object, select the first number, select the next number, and shift focus.

Goal state
A number designated as the cardinality of the given set.

Figure 12. A problem space for counting.

number of learning events. This result illustrates the generality of declarative knowledge. A single set of abstract principles gave the model the ability to construct three different procedures, each procedure being, in a sense, derived from those principles during practice. The declarative knowledge transferred from one counting task to another, even though the tasks are procedurally different.

Switching between counting tasks is an instance of near transfer. Not all rules need to be revised. Hence, the theory predicts that there will be procedural transfer as well. To illustrate this, six transfer experiments were run with the model. In each experiment, the model first practiced one of the three counting tasks until it reached mastery and then it was switched to either of the other two tasks. The results are

Prior procedural knowledge
The system began with one rule for each operator. That rule applied the operator whenever possible, i. e., in every situation in which its applicability conditions were satisfied. The result was counting-like but chaotic behavior.

Prior declarative knowledge
The were 18 constraints that encode the counting principles as identified by Gelman and Gallistel (1978): The one-to-one mapping principle, the cardinal principle, and the stable order principle.

Training
The model was given unsupervised practice on sets of 3-5 objects.

Learning outcomes
The model learned a correct, general procedure for counting any set of objects, regardless of the size of the set and the type of objects involved. It also learned correct procedures for two non-standard counting tasks, *ordered counting* and *constrained counting*. Finally, the model transfered each of the three learned counting procedures to each of the other two counting tasks.

Figure 13. Summary of the counting simulation.

shown in the off-diagonal cells of Table 1. The model solved each transfer task successfully. The amount of transfer varied depending on the exact relations between the rules for the practice task and the rules for the target task. The model also predicts asymmetric transfer between some tasks. For example, the transfer from ordered to constrained counting was 0%, while the transfer from constrained to ordered counting was 75%. These predictions are, in principle, empirically testable, although the necessary data are not available at this time.

Table 1. The computational effort required by the HS model to learn each of three counting tasks (diagonal cells) and to solve each of six different transfer tasks (off-diagonal cells).[a]

	Transfer task		
Training task	Standard counting	Ordered counting	Constrained counting
<u>Standard counting</u>			
Rule revisions	12	2	2
Prod. sys. cycles	854	110	127
<u>Ordered counting</u>			
Rule revisons	1	11	11
Prod. sys. cycles	184	262	297
<u>Constrained counting</u>			
Rule revisions	0	3	12
Prod. sys. cycles	162	154	451

[a]Data taken from Ohlsson and Rees (1991b, Tables 1 and 2).

Contrary to Singley and Anderson (1989), the present theory does not imply that the amount of transfer is predictable from the number of overlapping production rules. Instead, the variable of interest is the amount of cognitive work that has to be performed in order to adapt the rules to the target task. A single rule from the training task might cause more than one type of error in the target task and need to be revised more than once, so the number of rules that need to be revised is probably too course a predictor variable. The present analysis suggests that the number of rule revisions is a better predictor.

Unlike the number of overlapping production rules, the number of rule revisions required to master the target task cannot be calculated from a static comparison of the two procedures. It is a function of the particular learning mechanism which carries out the revisions. To verify this, the *knowledge compilation* mechanism-- a cornerstone of the ACT model described by Anderson (1983)--was implemented within the HS architecture. According to the knowledge compilation hypothesis, declarative knowledge resides in long-term memory in a format similar to inference rules. Familiar problems are solved with production rules, but unfamiliar problems are solved by interpreting (in the computer science sense) the declarative knowledge. During interpretation, new production rules are constructed which eliminates the need to re-interpret the declarative knowledge on subsequent trials. Once the rules are constructed, they are composed into larger rules which solve the relevant task more efficiently. Unlike HS, the ACT model learns from successes, not from errors.

We did not implement the ACT architecture as described in Anderson (1983). Instead, we implemented the knowledge compilation mechanism within the HS architecture. The result was a version of HS which learns through knowledge compilation instead of through constraint violations. All other aspects of the HS architecture were kept the same. I shall refer to the HS architecture as the KC model when it learns through knowledge compilation. The upshot is that we have two simulation models, HS and KC, which learn in different ways but which are otherwise identical. This provides an opportunity to compare the behavioral predictions of the two learning mechanisms.

KC was given the counting principles in the form of declarative knowledge and was then run through the same set of learning experiments and transfer experiments as HS. Because the effort measures differed by an order of magnitude (KC was on the average ten times slower than HS), they have been converted into transfer scores. There are many different ways to measure transfer (Singely & Anderson, 1989, pp. 37-41). The index used here was

$$T = \frac{E_B - E_{B/A}}{E_B} * 100 \qquad (2)$$

where E_B is the cognitive effort required to master the target task B from scratch and $E_{B/A}$ is the effort required to master B given previous mastery of training task A. The T index can be interpreted as the proportion of the effort required to learn task B that is saved by first learning task A. It is equal to zero when practice on the training task A is of no help and it is equal to 100 when practice on the training task provides mastery of B with no further training. The index is negative if practice on task A increases the amount of effort required to master B.

The effort measures for the transfer experiments with the HS and KC models were converted to transfer scores. The results are shown in Table 2. The amount of transfer predicted by the HS model varied between 8 and 100 across tasks, while the transfer predicted by the KC

Table 2. Transfer scores for the HS and KC[a] models for each of six transfer tasks in the counting domain, based on both the number of rule revisions and the number of production system cycles.

	Effort measure			
	Rule revisions		Prod. sys. cycles	
Transfer from-to	HS	KC	HS	KC
Standard-Ordered	82	100	58	100
Standard-Constrained	83	72	72	63
Ordered-Standard	92	34	78	19
Ordered-Constrained	8	34	34	29
Constrained-Standard	100	97	81	81
Constrained-Ordered	67	97	41	73

[a]KC is an acronym for *knowledge compilation.*

model varied between 19 and 100. More importantly for present purposes, the two models made different transfer predictions for one and the same task. HS predicts a score of 92 for the transfer from ordered to standard counting, while the corresponding KC prediction is 34. More important still, the differences between tasks do not always go in the same direction for the two models. HS predicts that transfer from standard to ordered counting is easier than vice versa, while KC predicts the opposite. The results in Table 2 verify the fact that the amount of near transfer between two tasks cannot be predicted from a static analysis of the procedures for those tasks, but depends upon assumptions about learning.

APPLICATION TO INSTRUCTION

A good theory should have implications for practice. The natural application domain for a learning theory is the design of instruction. The instructional implications of the present theory include an explanation of why it is possible to learn from instruction, a rationale for the most common tutoring scenario, a prescription for effective tutoring messages, and a technology for evaluating instructional designs through simulated one-on-one tutoring.

Why Instruction is Possible

Why are people able to learn from instruction? Although the origin of cognitive capacities such as language and learning is almost completely unknown, it is likely that learning evolved before language. There are no mammalian species, and probably no lower organisms, which cannot learn, so the capacity to learn was almost certainly present in the hominids when they separated from the rest of the primates 4-10 millions of years ago.

Language, on the other hand, evolved later, perhaps very much later. McCrone (1992) summarizes the fossil evidence in the following way: "The high arch in the roof of the mouth that helps with voice production is about the only telltale sign of speech that shows up on a fossil skeleton. This arch did not start to appear until *Homo erectus* arrived about 1.5 million years ago, and even then the arch was slight. Judging from fossils, modern speech came along about 100,000 years ago when the earliest examples of *Homo sapiens* were starting to walk the earth." (p. 160-161)[3] One hundred thousand years is a short time in evolutionary terms. If this estimate is correct, then special-purpose brain mechanisms for learning from verbal instruction have had little time to evolve.

These two speculative but plausible hypotheses--that learning preceeded language and that language is too recent for special-purpose brain mechanisms for instruction to have evolved--imply that our ability to learn without instruction is primary and our ability to learn from instruction secondary and parasitic upon the former. A theory of learning from instruction should therefore explain how instruction feeds into learning mechanisms that evolved for the purpose of uninstructed learning.

The theory proposed in this chapter suggests such an explanation. The two functions of detecting and correcting errors can be computed by noticing contradictions and by inferring the conditions that produced them as described previously, but they can also be computed in other ways. Instruction works, the theory suggests, because being told that one has committed an error is functionally equivalent to detecting the error oneself and because being told the cause of an error is functionally equivalent to figuring out the cause oneself. Learning from instruction is possible because instructional messages enter into the learning process in the same way, functionally speaking, as declarative knowledge retrieved from long-term memory.

A Rationale for One-on-One Tutoring

The theory proposed in this chapter implies that there are three major felicity conditions (VanLehn, 1990, p. 23) for effective instruction in cognitive skills: (a) instruction should be offered during ongoing practice, (b) instruction should alert the learner to errors, and (c) instruction should identify the conditions which caused the error. The type of in-

---

[3]See Lyons (1988, p. 153) for a different interpretation of the evidence.

struction that satisfies these three conditions is entirely familiar. In one-on-one tutoring, the teacher watches as the learner practices, points out errors, and helps the learner correct them. The present theory selects as most felicitous precisely that type of instruction which the empirical data show is most effective (Bloom, 1984).

Intelligent tutoring systems are typically designed to teach cognitive skills (Psotka, Massay, & Mutter, 1988; Sleeman & Brown, 1982). Perhaps the most successful line of intelligent tutoring systems are the so-called model tracing tutors developed by John Anderson and co-workers (Anderson et al., 1987, 1990). Skill training tutors in general and model tracing tutors in particular conform closely to the three felicity conditions: They give feedback in the context of practice, they alert the learner to errors, and they help the learner to correct the error.

The design of the model tracing tutors is said to be derived from the ACT theory of learning (Anderson et al., 1987). However, none of the six learning mechanisms described in various versions of the ACT theory--analogical transfer, discrimination, generalization, proceduralization, rule composition, and strengthening--can take a tutoring message as input and revise a faulty production rule accordingly. Analogical transfer generates task relevant activity by relating the current problem to an already solved problem; rule composition creates more efficient rules by combining existing (hopefully correct) rules; strengthening increases the probability that a (hopefully correct) rule will be retrieved. These three learning mechanisms can neither take a tutoring message as input nor revise an existing rule. Generalization and discrimination (which do not loom large in expositions of the ACT theory) revise existing rules, but cannot take a tutoring message as input. Proceduralization generates new rules on the basis of verbal input, but cannot correct existing rules. Taken literally, *the ACT theory predicts that it is impossible to learn from the teaching scenario embodied in the model tracing tutors.* Unless people can learn in other ways than those described in the ACT theory, they have no cognitive mechanisms for learning from feedback messages about errors.

To highlight the contrast between the implications of the ACT theory and the design of the model tracing tutors, consider what an intelligent tutoring system derived from the ACT theory might be like. In order to facilitate analogical transfer, such a system might keep a record of the problems the student has solved in the past and suggest possible analogies when the student hesitates. Such a system might repeat the task instructions from time to time to give the student a chance to re-proceduralize them. It might sequence practice problems in such a way that rule composition, generalization, and discrimination are facilitated. Finally, it might provide opportunities to exercise already acquired components of the target skill in order to increase their strengths. However, a tutoring system derived from the ACT theory would have no reason to alert the learner to errors and give help in correcting them.

The model tracing tutors and most other skill training systems conform to the design that follows from the theory presented in this chapter: They help the learner detect and correct errors. The instructional success of such tutors provide support for the hypothesis that error correction is the natural *modus operandi* of skill acquisition. If it were not, those tutors would not be effective but empirical evaluations show that they are (Anderson et at., 1990, pp. 30-33). In short, the present theory provides a rationale for the teaching scenario adopted by designers of intelligent tutoring systems and in turn receives empirical support from the instructional success of such systems.

Deriving the Content of Instruction from Theory

The content of feedback messages is the Achilleus heel of skill-monitoring tutoring systems. Delivering feedback messages is the major instructional action of such a system, so its instructional effectiveness depends crucially on the content of those messages. Until now there has been no theory for how to formulate feedback messages. Such messages are typically pre-formulated texts and they are written in the same way as other instructional materials: The instructional designer makes a guess about what might work based on his or her understanding of the subject matter.[4] In spite of the strong claims about the tight relation between the ACT theory and the model-tracing tutors (Anderson et al., 1987), this is true of those tutors as well. No existing tutoring system

---

[4]See Moore and Ohlsson (1992) and Reiser et al. (1991) for exceptions.

derives the content of its tutoring messages from assumptions about learning.

The learning theory proposed here implies that tutoring messages should help the student identify those properties of the current problem state which indicate that an error has been committed, so that he or she can detect his or her errors without help in the future. The general form for this type of tutoring message is "you can tell that you just made an error, because of P", where P is some conjunction of easily accessed properties of the problem state produced by the erroneous action. Unless the learner can detect his or her errors, he or she cannot learn from them.

More importantly, tutoring messages should help the learner correct his or her errors. To do so, a message must identify those properties of the immediately preceeding problem state that constitute counterindications to the problem solving step that the student took. A problem solving step A is typically correct in some situations but wrong in others. The task of the student is to figure out when, i. e., in which situations, doing A is right and when it is wrong. If doing A in situation S is incorrect, then the corresponding tutoring message should have the general form "when such-and-such conditions are the case, A is not the right thing to do". The conditions mentioned in the message should refer to the immediately preceeding problem situation, not to the situation in which the error was discovered. The student needs to learn to avoid the error, i. e., to act differently in the situation in which he or she decided to do A. The tutoring system should therefore back up and explain what makes A the wrong choice in that situation.

These prescriptions rule out some types of feedback messages which are commonly used in tutoring systems. For example, it is intuitively plausible that if a student takes step A when he or she should have taken step B, then it helps to print a message of the form "you did A but you should have done B." According to the present theory, however, this type of feedback message is likely to be ineffective, because it does not specify the conditions under which either A or B should or should not be done. Instruction should focus on the conditions of actions, not on the actions themselves. A second common type of feedback message explains what is wrong with the situation in which the error was discovered, i. e., why the error is an error. This might increase the student's understanding of the domain but it is unlikely to help him or her acquire the target skill, because it does not tell him or her how to avoid the error. A feedback message should focus on the situation in which the student decided to do A, not on the situation produced by doing A.

Simulating One-on-One Tutoring

The HS model can be interpreted as a model of learning from tutoring, with the constraints playing the role of tutoring messages. It is a matter of interpretation whether the constraints correspond to knowledge items retrieved from memory, conclusions from inference chains, or tutoring messages received through the language comprehension channel. According to the theory proposed here, these three types of knowledge elements enter into the learning process in the same way.

A runnable simulation of learning from tutoring opens up novel possibilities. We can evaluate alternative instructional designs by teaching them to the model and measuring the amount of computational work it has to expand to learn the target skill under different circumstances. If the model can reach mastery with less work under one instructional design or tutoring regime than another, then that is evidence that the former is the better design.

To explore this possibility the HS model was tutored in subtraction. The simulation experiment followed the common classroom tactic of teaching the procedure for canonical subtraction problems, i. e., problems in which each subtrahend digit is smaller than the minuend digit in the same column, and to introduce the procedure for how to handle non-canonical columns, i. e., columns in which the subtrahend digit is larger than the minuend digit, once the procedure for canonical subtraction has been mastered (Leinhardt, 1987; Leinhardt & Ohlsson, 1990). The simulation experiment followed this pedagogical tactic in that the model was first given a procedure for non-canonical subtraction and was then tutored in canonicalization.

Two different HS models of canonical subtraction were implemented. One model, called the *high-knowledge model*, was built around a representation of the place value meaning of digits. In this representation the digit 3 was represented as (3 * 10) if it appeared in the second column to the right, as (3 * 100) if it appeared in the third column, and so on. The operations by which this representation was manipulated correspond to mathematically motivated operations on numbers. The high-knowledge model was intended to simulate skill acquisition in the context of conceptual understanding of place value.

The second model, called the *low knowledge model*, was built around a representation in which a subtraction problem is a two-dimensional array of digits. In this representation, the digit 3 was represented as the digit 3 regardless of its position in the problem display. The operations by which this representation was manipulated correspond to physical operations on digits rather than conceptual or mathematical operations on numbers. The low knowledge model was intended to simulate rote learning of subtraction. Figure 14 summarizes the problem space for subtraction. The reader is referred to Ohlsson, Ernst, and Rees (1992) for a full account.

Both the high and the low knowledge models were tutored in the regrouping algorithm taught in American schools. In this method non-canonical columns are handled by incrementing the minuend of the non-canonical column and performing a corresponding decrement on the minuend in the next column to the left. Both models were also tutored in the equal addition algorithm taught in some European schools. In this method non-canonical columns are handled by incrementing the minuend in the non-canonical column and decrementing the subtrahend in the next column to the left. The simulation experiment thus followed a 2-by-2 design, with two levels of knowledge paired with two different target skills.

The procedure for tutoring the model were similar to those involved in tutoring a human student. The programmer in charge of the system watched while the model tried to solve a non-canonical problem, spotted errors, halted the model, and typed in a constraint (tutoring message) intended to correct the observed error. When the model had attained mastery, it was reinitialized and run again with all the constraints in place simultaneously, to verify that they were indeed sufficient to produce correct performance. This tutoring scenario was carried out four

Representations
Two different representations for subtraction were created. (a) The procedural or *low knowledge* representation contained symbols for written digits, perceived digits, spatial locations, scratch marks, decrements, and increments. (b) The conceptual or *high knowledge* representation contained, in addition, symbols for subtraction problems, numbers, place values, links between numbers and digits, relations between numbers, and answers.

Initial state
A subtraction problem.

Operators
Look at a digit, move the eye to another digit, write a digit, cross out a digit, assert the answer, recall number fact, create a working memory schema, and revise a working memory schema.

Goal state
A number designated as the answer to the subtraction problem.

Figure 14. A problem space for subtraction.

times, once for each combination of knowledge level and target skill. The amount of computational work required to attain mastery in each condition was recorded. Figure 15 summarizes the subtraction simulation.

Table 3 shows the number of production system cycles and the number of rule revisions required for HS to attain mastery in each of the four conditions. There are two main results. The high knowledge model required more work to attain mastery than the low knowledge model. This is true for both canonicalization procedures and for both effort

Prior procedural knowledge
The system knew at the outset how to solve a canonical subtraction problem, i. e., a problem in which the subtrahend digit is smaller than the minuend digit in every column.

Training
The model was tutored in how to handle non-canonical problems. It executed its procedure for canonical problem until it made an error. It was then halted and given a constraint that was intended to allow it to correct the error.

Learning outcome
The model learned two different procedures for non-canonical columns, namely regrouping and equal addition, with both the low knowledge and the high knowledge representations.

Figure 15. Summary of the subtraction simulation.

measures. The reason for this result is that the high knowledge model had a more elaborate representation. It requires more cognitive operations to create and update a more elaborate representation and each operation must be guided by some production rule. Hence, the high knowledge model had more to learn.

The second result is that learning the regrouping procedure required more work than learning the equal addition procedure in both the high knowledge and the low knowledge conditions. The reason for this result is that the control of the regrouping procedure becomes complicated when it is necessary to regroup the minuend recursively to handle blocking zeroes, i. e., minuend zeroes immediately to the left of a non-canonical column. The augmenting procedure is not affected by the number of blocking zeroes. The regrouping procedure also requires more complicated visual attention allocation.

Table 3. The computational effort required for the HS model to master regrouping and augmenting with either a high knowledge or a low knowledge representation.[a]

Method learned	Type of representation: High knowledge Cycles	High knowledge Revisions	Low knowledge Cycles	Low knowledge Revisions
Regrouping				
W/o blocking zeroes[b]	940	23	449	16
With blocking zeroes	1815	32	794	24
Augmenting				
W/o blocking zeroes	862	20	687	18
With blocking zeroes	862	20	687	18

[a]Data taken from Ohlsson, Ernst, and Rees (1992, Table 2).
[b]I. e., minuend zeroes to the left of a non-canonical column.

Both of these results were unexpected because they contradict the common belief among mathematics educators that regrouping is easier to learn, particularly in the high knowledge condition. This belief is based on empirical investigations carried out in the pre-Word War II era (Brownell, 1947; Brownell & Moser, 1949). A detailed discussion of these simulation results and their relation to the empirical research has been presented elsewhere (Ohlsson, 1992a).

This simulation exercise demonstrates that runnable models of learning from instruction creates new relations between learning theory and instructional design (Ohlsson, 1992a). Instead of deriving general design principles from the learning theory, as suggested by Bruner (1966) and later by Glaser (1976, 1982), we can evaluate an instructional design directly by teaching it to a simulation model. This technology has the potential to allow instructional designers to do formative evaluation without leaving their desks (Ohlsson, 1992b).

This simulation exercise also demonstrates that the application of learning theory to education requires a formal analysis of instruction. *A model of learning cannot have implications for instruction unless it contains learning mechanisms which take instruction, suitably formalized, as one of their inputs.* Computational analysis of instruction has barely begun. Some early Artificial Intelligence systems explored how a system can learn from advice and instructions (Hayes-Roth, Klahr, & Mostow, 1981; Mostow, 1983; Rychener, 1983), but the problem appears to have disappeared from the research agenda of the machine learning community. The proceduralization mechanism in the ACT model (Anderson, 1983) was a first attempt to formalize this problem in a psychological context. Although the proceduralization mechanism explains how the learner constructs new rules on the basis of task instructions, it does not explain how the learner revises existing rules on the basis of feedback messages. The Sierra and Cascade models described by VanLehn (1990) and VanLehn and Jones (this volume) learn from solved examples--a common form of instruction--but they cannot take tutoring messages as inputs. The HS model constitutes a modest first step towards a formal theory of how tutoring messages received during skill practice are translated into mental code for the target skill.

SUMMARY AND CONCLUSIONS

The theory proposed in this chapter is formulated in terms of cognitive functions instead of information processing mechanisms. The function of learning to solve an unfamiliar task is analyzed into two subfunctions: To generate learning opportunities and to construct new procedural knowledge. The latter function is in turn broken down into two subfunctions: To detect incorrect problem solving steps and to correct the procedural knowledge that generated them. To detect errors requires a comparison between the current problem state with prior knowledge. To correct an error, finally, involves identifying the conditions under which the error appears and constraining the faulty decision rule accordingly. The main claim of the theory is that this is the right functional breakdown of skill acquisition.

How does this theory explain the role of prior knowledge in skill acquisition? Domain knowledge is not needed to generate task relevant actions. Weak methods can generate behavior even in the absence of any knowledge about the task. The functions for which knowledge is needed are to detect and correct errors. Incorrect or incomplete task procedures are likely to produce situations which contradict what ought to be true in the particular domain. Domain knowledge increases the probability that the learner recognizes that he or or she has made an error. To correct the error presupposes the ability to identify the conditions under which that error will appear. This might require complicated reasoning about the domain. Prior knowledge increases the probability that the learner identifies the causes of errors correctly.

Each of the functions postulated in the theory can be computed by many different information processing mechanisms. In the particular implementation of the theory described in this chapter, task relevant behavior is generated by forward search through the problem space. Errors are detected by matching constraints against problem states with a pattern matcher. The conditions that produced a particular error are identified by regressing the match between a constraint and a state through a production rule. Errors are corrected by adding the conditions that produce them to the left-hand sides of decision rules. Other implementations of the functional theory are possible.

The simulation model generates several quantitative predictions about two classical problems in learning theory. First, it predicts that skill acquisition is negatively accelerated. More precisely, it predicts that the learning curve follows a so-called power law. Second, the theory predicts zero transfer in far transfer situations. It also predicts that the amount of transfer in near transfer situations depends upon the particular tasks involved and that transfer might be asymmetrical, i. e., that there might be either more transfer from task B to task A than from task A to task B. Finally, the model predicts that the richer the representation of the task to be learned, the more cognitive effort is needed to attain mastery.

With respect to the practical problem of designing computer-based instruction in cognitive skills, the present theory provides a rationale and an explanation for the effectiveness of one-on-one tutoring, the main teaching scenario embodied in current intelligent tutoring systems. Tutoring (by computer or by human) works, the theory of claims, because tutoring messages provide an alternative way to become aware of errors and an alternative source of information about the conditions under which the errors occur.

According to the present theory, tutoring messages can help the learner in two ways. First, to help the learner detect his or her own errors, tutoring messages should point out those properties of a problem state which indicate that an error has occurred. Second, to help the learner correct his or her errors, tutoring messages should identify those properties of a problem state which indicate that an error will occur if such and such an action is executed.

The theory proposed here is obviously incomplete. People undoubtedly learn from their errors, but they also learn from their successes. The theory needs to be extended with assumptions about how people learn from correct problem solving steps. It is not clear which of those predictions will remain constant if the model is augmented with additional learning mechanisms. The interaction between multiple learning mechanisms is a high-priority issue for computational learning theories. In past work, I combined a method for learning from error (discrimination) with two methods for learning from success (generalization and subgoaling). The resulting model learned to solve simple puzzle tasks (Ohlsson, 1987a), but it threw no light on the problem of prior knowledge.

The problem of how prior knowledge impacts learning is central for the study of skill acquisition. The outcome of practice is always a function of both the learner's prior knowledge about the domain and the new information that becomes available during practice. Any viable learning theory must describe the cognitive mechanism that interfaces those two knowledge sources. The fate of the theory proposed here will ultimately be determined by comparative evaluations with alternative computational theories of the function of prior knowledge in learning, once such alternative theories become available.

REFERENCES

Abelson, R. P., Aronson, E., McGuire, W. J., Newcomb, T. M., Rosenberg, M. J., & Tannenbaum, P. H., (Eds.), (1968). *Theories of cognitive consistency: A sourcebook.* Chicago, IL: Rand McNally.

Anderson, J. R. (1976). *Language, memory, and thought.* Hillsdale, New Jersey: Erlbaum.

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

Anderson, J. R. (1986). Knowledge compilation: The general learning mechanism. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell, (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2, pp. 289-310). Los Altos, CA: Kaufmann.

Anderson, J. R. (1987). Skill acquisition: Compilation of weak-method problem solutions. *Psychological Review, 94*, 192-210.

Anderson, J. R. (1990). *The adaptive character of thought.* Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1993). *Rules of the mind.* Hillsdale, NJ: Erlbaum. In press.

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

Anderson, J. R., Boyle, C. F., Farrell, R., & Reiser, B. J. (1987). Cognitive principles in the design of computer tutors. In P. Morris, (Ed.), *Modeling cognition.* New York, NY: Wiley.

Anderson, J. R., Kline, P. J., & Beasley, C. M., Jr. (1979). A general learning theory and its application to schema abstraction. In G. H. Bower, (Ed.), *The psychology of learning and motivation: Advances in research and theory* (Vol. 13, pp. 277-318). New York, NY: Academic Press.

Anderson, J. R., & Schooler, L. J. (1991). Reflections of the environment in memory. *Psychological Science, 2*, 396-408.

Angluin, D., & Smith, C. H. (1983). Inductive inference: Theory and methods. *Computing Surveys, 15*, 237-269.

Anzai, Y., & Simon, H. A. (1979) The theory of learning by doing. *Psychological Review, 86*, 124-140.

Berkson, W., & Wettersten, J. (1984). *Learning from error: Karl Popper's psychology of learning*. La Salle, IL: Open Court.

Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher, 13(6)*, 4-16.

Brownell, W. A. (1947). An experiment on "borrowing" in third-grade arithmetic. *Journal of Educational Research, 41*, 161-263.

Brownell, W. A., & Moser, H. E. (1949). *Meaningful vs. mechanical learning: A study in Grade III subtraction*. Durham, NC: Duke University Press.

Bruner, J. S. (1966). *Toward a theory of instruction*. Cambridge, MA: Harvard University Press.

Confrey, J. (1990). A review of the research on student conceptions in mathematics, science, and programming. In C. B. Cazden, (Ed.), *Review of Research in Education* (Vol. 1, pp. 3-56). Washington, D. C.: American Educational Research Association.

Cormier, S., & Hagman, J., (Eds.), (1987). *Transfer of learning: Contemporary research and applications*. New York, NY: Academic Press.

Duhem, P. (1991/1914). *The aim and structure of physical theory* (2nd ed.). Princeton, NJ: Princeton University Press.

Ebbinghaus, H. (1964/1885). *Memory: A contribution to experimental psychology*. New York, NY: Dover.

Evans, J. St. B. T. (1982). *The psychology of deductive reasoning*. London, UK: Routledge & Kegan Paul.

Feigenbaum, E. A., & Feldman, J. (1963). *Computers and thought*. New York, NY: McGraw-Hill.

Festinger, L. (1957). *A theory of cognitive dissonance*. Standford, CA: Stanford University Press.

Fikes, R. E., & Nilsson, N. J. (1971). STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*, 189-208.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence, 19*, 17-37.

Gardenfors, P. (1988). *Knowledge in flux: Modeling the dynamics of epistemic states*. Cambridge, MA: MIT Press.

Gelman, R., & Gallistel, C. (1978). *The child's understanding of number*. Cambridge, MA: Harvard University Press.

Gelman, R., & Meck, E. (1983). Preschoolers' counting: Principle before skill. *Cognition, 13*, 343-359.

Gelman, R., & Meck, E. (1986). The notion of principle: The case of counting. In J. H. Hiebert, (Ed.), *Conceptual and procedural knowledge: The case of mathematics* (pp. 29-57). Hillsdale, NJ: Erlbaum.

Gelman, R., Meck, E., & Merkin, S. (1986). Young children's numerical competence. *Cognitive Development, 1*, 1-29.

Gensler, W. J. (1987). Impossibilities in chemistry: Their rise, nature, and some great falls. In P. J. Davis and D. Park, (Eds.), *No way: The nature of the impossible* (pp. 73-89). New York, NY: Freeman.

Gick, M. L., & Holyoak, K. J. (1983). Schema induction and analogical transfer. *Cognitive Psychology, 15*, 1-38.

Gick, M. L., & Holyoak, K. J. (1987). The cognitive basis of knowledge transfer. In S. M. Cormier & J. D. Hagman, (Eds.), *Transfer of learning: Contemporary research and applications* (pp. 183-215). San Diego, CA: Academic Press.

Glaser, R. (1976). Components of a theory of instruction: Toward a science of design. *Review of Educational Research, 46*, 1-24.

Glaser, R. (1982). Instructional psychology: Past, present, and future. *American Psychologist, 37*, 292-305.

Hall, R. (1988). Learning by failing to explain: Using partial explanations to learn in incomplete or intractable domains. *Machine Learning, 3*, 45-77.

Hayes, J. R., & Simon, H. A. (1977). Psychological differences between problem isomorphs. In J. Castellan, D. B. Psioni, and G. Potts, (Eds.), *Cognitive theory* (Vol. 2, pp. 21-41). Hillsdale, NJ: Erlbaum.

Hayes-Roth, F., Klahr, P., & Mostow, D. (1981). Advice taking and knowledge refinement: An iterative view of skill acquisition. In J.

Anderson, (Ed.), *Cognitive skills and their acquisition* (pp. 231-253). Hillsdale, NJ: Erlbaum.

Hewson, P. W., & Hewson, M. G. A. (1984). The role of conceptual conflict in conceptual change and the design of science instruction. *Instructional Science, 13*, 1-13.

Holland, J., Holyoak, K., Nisbett, R., & Thagard, P. (1986). *Induction: The processes of inference, learning, and discovery.* Cambridge, MA: MIT Press.

Hull, C. L. (1943). *Principles of behavior.* New York, NY: Appleton-Century-Crofts.

Judd, C. H. (1908). The relation of special training and general intelligence. *Educational Review, 36*, 28-42.

Katona, G. (1940). *Organizing and memorizing: Studies in the psychology of learning and teaching.* Morningside Heights, NY: Columbia University Press.

Kieras, D. E., & Bovair, S. (1984). The role of a mental model in learning to operate a device. *Cognitive Science, 8*, 255-273.

Klahr, D., Langley, P., & Neches, R., (Eds.), (1987). *Production system models of learning and development.* Cambridge, MA: MIT Press.

Kocabas, S. (1991). Conflict resolution as discovery in particle physics. *Machine Learning, 6*, 277-309

Kuhn, T. (1970). *The structure of scientific revolutions* (2nd ed.) Chicago, IL: University of Chicago Press.

Laird, J. (1986). Universal subgoaling. In J. Laird, P. Rosenbloom, & A. Newell, *Universal subgoaling and chunking: The automatic generation and learning of goal hierarchies* (pp. 3-131). Boston, MA: Kluwer.

Lane, N. (1987). *Skill acquisition rates and patterns: Issues and training implications.* New York, NY: Springer-Verlag.

Langley, P. (1985). Learning to search: From weak methods to domain-specific heuristics. *Cognitive Science, 9*, 217-260.

Langley, P. (1987). A general theory of discrimination learning. In Klahr, D., Langley, P., & Neches, R., (Eds.), (1987). *Production system models of learning and development* (pp. 99-161). Cambridge, MA: MIT Press.

Leinhardt, G. (1987). Development of an expert explanation: An analysis of a sequence of subtraction lessons. *Cognition and Instruction, 4*, 225-282.

Leinhardt, G., & Ohlsson, S (1990). Tutorials on the structure of tutoring from teachers. *Journal of Artificial Intelligence in Education, 2*, 21-46.

Lyons, J. (1988). Origins of language. In A. Fabian, (Ed.), *Origins: The Darwin College lectures* (pp. 141-166). Cambridge, MA: Cambridge University Press.

Mazur, J., & Hastie, R. (1978). Learning as accumulation: A reexamination of the learning curve. *Psychological Bulletin, 85*, 1256-1274.

McCrone, J. (1992). *The ape that spoke: Language and the evolution of the human mind.* New York, NY: Avon Books.

McDermott, D., & Doyle, J. (1980). Non-monotonic logic I. *Artificial Intelligence, 13*, 41-72.

Minton, S. (1988). *Learning search control knowledge: An explanation-based approach.* Boston, MA: Kluwer.

Moore, J. D., & Ohlsson, S. (1992). Educating migraine patients through on-line generation of medical explanations. *Proceedings of the 14h Annual Conference of the Cognitive Science Society.* July 29 - August 1, Bloomington, Indiana, USA. In press.

Mostow, D. J. (1983). Machine transformation of advice into a heuristic search procedure. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell, (Eds.), *Machine learning: An artificial intelligence approach* (pp. 367-404). Palo Alto, CA: Tioga.

Mostow, J., & Bhatnager, N. (1987). Failsafe--a floor planner that uses EBG to learn from its failures. In *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 249-255), Milan, Italy: IJCAI.

Mostow, J., & Bhatnager, N. (1990). *Heuristic detection, explanation, and avoidance of failures as an approach to intractability in explanation-based learning of search control knowledge* (Technical Report No. 171). Rutgers, NJ: Rutgers University.

Murray, F. B., Ames, G. J, & Botvin, G. J. (1977). Acquisition of conservation through cognitive dissonance. *Journal of Educational Psychology, 69*, 519-527.

Newell, A. (1969). Heuristic programming: Ill-structured problems. In J. Aronofsky, (Ed.), *Progress in operations research* (Vol. III, pp. 360-414). New York, NY: Wiley.

Newell, A. (1980). Reasoning, problem solving and decision processes: The problem space as a fundamental category. In R. Nickerson, (Ed.), *Attention and performance VIII* (pp. 693-718). Hillsdale, NJ: Erlbaum.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Havard University Press.

Newell, A., & Rosenbloom, P. (1981). Mechanisms of skill acquisition and the law of practice. In J. Anderson, (Ed.), *Cognitive skills and their acquisition* (pp. 1-55). Hillsdale, NJ: Erlbaum.

Newell, A., & Simon, H. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Nilsson, N. J. (1980). *Principles of artificial intelligence*. Palo Alto, CA: Tioga.

Ohlsson, S. (1987a). Transfer of training in procedural learning: A matter of conjectures and refutations? In L. Bolc, (Ed.), *Computational models of learning* (pp. 55-88). Berlin, West Germany: Springer-Verlag.

Ohlsson, S. (1987b). Truth versus appropriateness: Relating declarative to procedural knowledge. In D. Klahr, P. Langley, & R. Neches, (Eds.), *Production system models of learning and development* (pp. 287-327). Cambridge, MA: MIT Press.

Ohlsson, S. (1992a). Artificial instruction. A method for relating learning theory to instructional design. In P. Winne & M. Jones, (Eds.), *Foundations and frontiers in instructional computing systems*. New York, NY: Springer-Verlag. In press.

Ohlsson, S. (1992b). The impact of cognitive theory on the practice of courseware authoring. In R. Lewis, (Ed.), *Authoring environments for computer-based courseware.* New York, NY: Springer-Verlag. In press.

Ohlsson, S. (1992c). The learning curve for writing books: Evidence from Professor Asimov. *Psychological Science*. In press.

Ohlsson, S., Ernst, A. M., & Rees, E. (1992). The cognitive complexity of doing and learning arithmetic. *Journal of Research in Mathematics Eduation*. In press.

Ohlsson, S., & Rees, E. (1991a). The function of conceptual understanding in the learning of arithmetic procedures. *Cognition and Instruction, 8*, 103-179.

Ohlsson, S., & Rees, E. (1991b). Adaptive search through constraint violation. *Journal of Experimental and Theoretical Artificial Intelligence, 3*, 33-42.

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving*. Reading. MA: Addison-Wesley.

Piaget, J. (1971). *Biology and knowledge: An essay on the relation beween organic regulations and cognitive processes*. Edingburgh, UK: Edinburgh University Press.

Piaget, J. (1985). *The equilibrium of cognitive structures: The central problem of intellectual development*. Chicago, IL: University of Chicago Press.

Popper, K. (1972/1935). *The logic of scientific discovery*. London, UK: Hutchinson.

Posner, G. J., Strike, K. A., Hewson, P. W., & Gertzog, W. A. (1982). Accomodation of a scientific conception: Toward a theory of conceptual change. *Science Education, 66*, 211-227.

Psotka, J., & Massey, L. D., & Mutter, S. A., (Eds.), (1988). *Intelligent tutoring systems: Lessons learned*. Hillsdale, NJ: Erlbaum.

Quine, W. V., & Ullian, J. S. (1978). *The web of belief* (2nd ed.). New York, NY: Random House.

Reason, J. (1990). *Human error*. Cambridge, UK: Cambridge University Press.

Reif, F. (1987). Interpretation of scientific or mathematical concepts: Cognitive issues and instructional implications. *Cognitive Science, 11*, 395-416.

Reiser, B. J., Kimberg, D. Y., Lovett, M. C., & Ranney, M. (1991). Knowledge representation and explanation in GIL, an intelligent tutor for programming. In J. H. Larkin & R. W. Chabay, (Eds.), *Computer assisted instruction and intelligent tutoring systems: Shared issues and complementary approaches* (pp.111-149). Hillsdale, NJ: Erlbaum.

Resnick, L. B., & Omanson, S. F. (1987). Learning to understand arithmetic. In R. Glaser, (Ed.), *Advances in instructional psychology* (Vol 3, pp. 41-95). Hillsdale, NJ: Erlbaum.

Rose, D., & Langley, P. (1986). Chemical discovery as belief revision. *Machine Learning, 1*, 423-452.

Rosenbloom, P. (1986). The chunking of goal hierarchies. A model of practice and stimulus-response compatibility. In J. Laird, P. Rosenbloom, & A. Newell, *Universal subgoaling and chunking: The automatic generation and learning of goal hierarchies* (pp. 135-1282). Boston, MA: Kluwer.

Rychener, M. D. (1983). The instructable production system: A retrospective analysis. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell, (Eds.), *Machine learning: An artificial intelligence approach* (pp. 429-460). Palo Alto, CA: Tioga.

Schank, R. (1982). *Dynamic memory: A theory of reminding and learning in computers and people.* Cambridge, MA: Cambridge University Press.

Schank, R. (1986). *Explanation patterns: Understanding mechanically and creatively.* Hillsdale, NJ: Erlbaum.

Singely, M., & Anderson, J. (1989). *The transfer of cognitive skill.* Cambridge, MA: Harvard University Press.

Sleeman, D., & Brown, J. S., (Eds.), (1982). *Intelligent tutoring systems.* New York, NY: Academic Press.

Smith, E. E., & Goodman, L. (1984). Understanding written instructions: The role of an explanatory schema. *Cognition and Instruction, 1*, 359-396.

Solomons, T. (1988). *Organic chemistry* (4th ed.). New York, NY: Wiley.

Sorum, C. H., & Boikess, R. S. (1981). *How to solve general chemistry problems.* Englewood Cliffs, NJ: Prentice-Hall.

VanLehn, K. (1982). Bugs are not enough: Empirical students of bugs, impasses and repairs in procedural skills. *Journal of Mathematical Behavior, 3*, 3-71.

VanLehn, K. (1988). Toward a theory of impasse-driven learning. In H. Mandl & A. Lesgold, (Eds.), *Learning issues for intelligent tutoring systems* (pp. 19-41). New York, NY: Springer Verlag.

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

Winograd, T. (1975). Frame representations and the declarative/procedural controversy. In D. Bobrow and A. Collins, (Eds.), *Representation and understanding: Studies in cognitive science* (pp. 185-210). New York, NY: Academic Press.

Correcting Imperfect Domain Theories: A Knowledge-Level Analysis[1]

Scott B. Huffman, Douglas J. Pearson, and John E. Laird
Artificial Intelligence Laboratory
The University of Michigan
1101 Beal Ave.
Ann Arbor, Michigan 48109–2122

Abstract

Explanation-Based Learning (Mitchell et al., 1986; DeJong and Mooney, 1986) has shown promise as a powerful analytical learning technique. However, EBL is severely hampered by the requirement of a complete and correct domain theory for successful learning to occur. Clearly, in non-trivial domains, developing such a domain theory is a nearly impossible task. Therefore, much research has been devoted to understanding how an imperfect domain theory can be corrected and extended during system performance. In this paper, we present a characterization of this problem, and use it to analyze past research in the area. Past characterizations of the problem (e.g, (Mitchell et al., 1986; Rajamoney and DeJong, 1987)) have viewed the types of performance errors *caused* by a faulty domain theory as primary. In contrast, we focus primarily on the types of knowledge deficiencies present in the theory, and from these derive the types of performance errors that can result. Correcting the theory can be viewed as a search through the space of possible domain theories, with a variety of knowledge sources that can be used to guide the search. We examine the types of knowledge used by

[1]Sponsored in part by NASA and the Office of Naval Research under contract NCC 2-517, and by a University of Michigan Research Partnership Fellowship.

a variety of past systems for this purpose. The hope is that this analysis will indicate the need for a "universal weak method" of domain theory correction, in which different sources of knowledge for theory correction can be freely and flexibly combined.

INTRODUCTION

Much recent research in machine learning has centered around analytical learning techniques. In particular, *Explanation-Based Learning* (EBL) (Mitchell et al., 1986; DeJong and Mooney, 1986) has emerged as an important approach to using prior domain knowledge to improve performance. In its classic conception, EBL involves using a *domain theory* to construct an *explanation* (or proof) of why a given *training example* is an instance of some *goal concept.* By analyzing the dependency structure of the explanation, the learning system can construct a generalized, *operational* rule which directly recognizes the training example as an instance of the goal concept. EBL has taken a variety of slightly different forms, such as knowledge compilation (Anderson, 1986), chunking (Laird et al., 1986), operationalization (Mostow, 1981), and schema construction (Mooney, 1990), and has proven useful in domains from recognizing simple concepts to learning reactive plans for mobile robots (Mitchell, 1990; Laird and Rosenbloom, 1990).

This technique is severely hampered, however, by the requirement of a complete and correct domain theory. Even for simple domains, a perfect domain theory is extremely difficult to construct. Therefore, to make EBL a plausible learning technique, it must be augmented by methods for correcting and extending incomplete domain theories.

There have been a number of different methods advanced to attack this problem. Researchers have focussed on examining a variety of *mechanisms* for extending and correcting domain theories. In this paper, however, we will focus primarily on the *knowledge* used to extend and correct a domain theory. Understanding the knowledge available is the first step towards the long-term goal of developing a general, domain-independent mechanism for improving domain theories, which can draw on any of the available knowledge sources in *combination.* The ability to flexibly draw upon a combination of knowledge sources should greatly enhance a system's ability to improve its domain theory. Thus, our long-term goal can be viewed as the development of a "universal weak method" (Laird and Newell, 1983) for domain theory refinement.

In this paper, we will present a framework that lays out the space of possible approaches to extending and correcting domain theories. In order to understand how various knowledge sources are used to refine domain theories, we must begin by analyzing the refinement task itself. We will examine the kinds of tasks EBL systems perform, the types of domain theory errors that can be present, and the ways in which they manifest themselves. Next, we will describe the domain theory correction process, casting it as a heuristic search through the space of possible domain theories. Knowledge that is used in the correction process can be then be viewed as controlling this search. In analyzing past work in the area (including our own) in these terms, we find that few approaches have demonstrated the ability to draw from a variety of knowledge sources in refining an imperfect domain theory.

PERFORMANCE TASKS OF EBL SYSTEMS

Two complementary tasks have been performed by EBL systems. The first type, what we will call *analysis* tasks, involve explaining or understanding some observed example. This category includes plan recognition (Mooney, 1990), in which the system tries to explain the actions of characters in a narrative, and apprenticeship learning systems (e.g., (Wilkins, 1988; Chien, 1989; VanLehn, 1987)), in which the system observes an expert carrying out some task which it must learn to perform. The second type, what we will call *generation* tasks, involve constructing (as opposed to observing) a plan to reach some goal from an initial state. This category includes typical planning tasks, such as planning a robot's actions (e.g., (Gupta, 1987; Laird et al., 1989; Bennett, 1990)), and standard concept recognition tasks (e.g., (Ourston and Mooney, 1990)), where the sequence of inferences needed to recognize an example of the concept is considered the "plan" that is generated.

These two types of tasks are similar, in that they require similar types of knowledge to be performed. Both require a knowledge of the operations performable in the domain, their applicability and their effects. For analysis tasks, the system observes a sequence of states ending at a goal, and must infer a consistent sequence of operations that would produce the sequence of states. For generation tasks, the system is given only the initial and final states, and must produce a sequence of operations leading from one to the other. Thus the analysis task is a more constrained version of the generation task. In addition, generation systems may require knowledge of how to actually carry out the operations in the external world. Many systems (e.g., (Bennett, 1990; Chien, 1989)) perform both types of problems. In the remainder of this paper we will cast our discussion from the point of view of generation tasks, but the analysis is equally applicable to analysis tasks. Where the different tasks lead to differences in the analysis, we will highlight the differences.

We can cast the performance task faced by an EBL system in generic terms using the standard AI concept of problem space search. Search in a problem space involves finding a path from an initial state to a goal state. The system has a set of *operators* which move between states in the space. The task is to find a sequence of operators which lead from the initial state to the goal state. This sequence is a *plan* in the generation case (or an execution trace if the system is executing operators before a complete plan is formed). In the analysis case, an observation must be explained. The sequence forms an explanation of the observation. Throughout the paper we will refer to the reasoning process as planning, and its result as plans.

Given this characterization of the problem, the knowledge that comprises the system's domain theory becomes clear. Simply, the system's knowledge of operators - their preconditions and effects - makes up the domain theory. In pure inference tasks, such as the cups domain (Winston et al., 1983), each inference can be viewed as an operator. Note that this simple definition of the domain theory is a result of how we have cast the problem. A system may or may not actually have its knowledge partitioned into distinct, discrete operators with explicit pre- and postconditions. Clearly, knowledge can be applicable to many different operators (for example, general "frame axioms"). However, conceptually the knowledge can be viewed as a set of operators, with distinct knowledge about each.

There may be other knowledge in the system, of course, besides what can be viewed as operator knowledge. This additional knowledge is used to *guide* search through the states of the problem space. There may be standard *search control* knowledge, which express preferences between operators, and/or reasoning strategy knowledge, such as knowledge allowing the system to perform an abstract or hierarchical search.

We do not consider such search guiding knowledge to be a part of the domain theory proper of a system. The reason is that search control is intended to affect problem solving's *efficiency*, not its *correctness*. Search control may make an intractable theory tractable, but the theory is not incorrect in either case. If a system has no search control, or incorrect search control, it will have to explore more of the search space to solve the problem, but the lack of search control will not preclude solving the problem. EBL systems that have learned search control (e.g., PRODIGY (Minton et al., 1989) and Soar (Laird et al., 1987)) did not extend or correct domain theories by doing so. However, if resource constraints can affect the correctness of the solution, then the solution may not be independent of the search process. In this case, knowledge about resource constraints must be incorporated into the domain theory.

DOMAIN THEORY PROBLEMS

Almost every domain theory is actually an approximation. This is due to the frame problem: the preconditions and effects of actions are extremely difficult to describe fully except in limited domains. For example, knowing the preconditions of actions in the real world is extremely difficult; this is known as the *qualification problem* (Ginsberg and Smith, 1987). Exceptions can be generated almost ad infinitum. Consider an operator for going through a door. The most obvious precondition is that the door is opened. What if the door is open but there are steel bars across the opening? Or, maybe a transparent glass plate? Or an unseen magnetic force field? We could go on and on. Similarly, in an environment with other complex processes or agents, it usually impossible to have a complete domain theory that will predict all the activity of the other processes and agents. This explains AI's fondness for the closed world assumption, for without it, our domain theories are always incomplete, if not incorrect.

When a system's domain theory is imperfect, errors may arise, either during planning or execution. In this section we analyze the types of domain theory imperfections that may arise, and the errors they can lead to. Note that correcting a domain theory in response to failures is different from learning search heuristics from search failures, which can be done using standard EBL methods and does not alter the domain theory (Mostow and Bhatnager, 1987; Minton et al., 1989).

Types of Domain Theory Imperfections

We have cast the problem facing an EBL system as that of finding a sequence of operators. Therefore, the domain theory consists of the system's knowledge of the preconditions and postconditions of its operators. This gives rise to the set of possible problems with the domain theory:

- Overgeneral Preconditions.

  An operator precondition is missing, or an overgeneral test is used (e.g. "fruit" instead of "banana").

- Overspecific Preconditions.

  An extra, unnecessary precondition is present, overly restricting the set of situations in which an operator is applicable.

- Incomplete Postconditions.

  The planner is unaware of some effect of an operator.

- Extraneous Postconditions.

  The planner incorrectly believes that an operator will produce some effect which it does not.

- Missing Operators.

  An entire operator is absent from the domain theory.

Note that when a postcondition is simply "incorrect" this can be viewed as a case of both an extraneous postcondition (the current effect) and a missing postcondition (the correct effect).

Observed Types of Failures

As a result of an imperfect domain theory, an EBL system might encounter a failure either during planning or during execution of the plan. These two categories are further delineated below:

1. **Planning Failures**

   (a) Incomplete Plan

   In this case, the planner is unable to form a complete sequence of steps that it believes will lead from the initial state to the goal. For analysis tasks, this corresponds to being unable to construct a complete explanation of the observed training example. Several systems attempt to construct a parse of training examples by working both bottom-up and top-down (VanLehn, 1987; Hall, 1986). A failure to complete the parse represents an incomplete plan failure.

   (b) Multiple Inconsistent Plans

   Rajamoney and DeJong (1988) examine cases in which it is known that only one plan should be formed, but multiple plans are found. In many domains, multiple paths to the same goal are possible, so forming multiple plans does not necessarily constitute an error. Even if it is known that there is only one legal path to the goal, detecting this error requires that the planner actually attempt to construct multiple plans, which is not common.

   (c) Reaching an impossible state

   During planning, the system will use its knowledge of operators to search through the states of the domain. If the domain theory is incorrect, it may be possible to reach a world state which is known to be *impossible* to achieve. For example, if during planning in a STRIPS domain the robot is found to be in two different rooms at once, this indicates a domain theory problem. Note that to detect this kind of error during planning, the system must have explicit knowledge about states that are impossible.

2. **Execution Failures**

   Execution failures occur when the system completes a plan but is unable to successfully execute it. Failure occurs when the system's expectations are not met in the external world. Note, however, that the operator being executed when the error is detected is not necessarily the one for which the domain theory is in error. The incorrect operator may have occurred anywhere before the current operator in the plan.

The failure may be detected in a number of ways:

(a) Preconditions of the next operator to be executed are not achieved.

Here an operator is slated to be executed but cannot be because one of its preconditions does not hold in the external world. This means that some earlier operator either was expected to achieve the precondition but did not, or that some earlier operator was expected to preserve the precondition but clobbered it (Carbonell and Gil, 1987).

(b) Postconditions of the operator just executed are not achieved.

The operator does not cause all of the effects that the planner predicted. Carbonell and Gil (1987) break this down further into two cases: either all postconditions are unachieved, or only some subset are unachieved. If all postconditions are unachieved, it probably indicates that the operator didn't apply at all. The operator apparently had preconditions for its application that the planner was unaware of. If only a subset of the operator's expected effects are unachieved, then the operator did apply but didn't do all it was expected to. This indicates that the unachieved effects are either incorrect or somehow conditional on the situation in an unexpected way.

(c) Failure is explicitly detected.

In some cases, the system may have rules indicating states of the world which should not be reached. This is essentially a "theory of failure." For example, Gupta's system (1987) has a rule indicating that if the robot's gripper melts, that is a failure. In classification tasks, execution corresponds to classifying an example, and incorrectly classifying an example corresponds to an explicit detection of failure.

It should be noted that not all execution failures are due to an erroneous domain theory. The domain theory is a theory of the processes that can operate on the world. To operate correctly, the domain theory depends on the agent having a correct knowledge of the *state* of the

	Planning Failures			Execution Failures		
	Incomplete	Multiple	Impossible State	Prec. Unmet	Postc. Unmet	Explicit Detection
Overgen. Preconds.		•	•		•	•
Overspec. Preconds.	•					
Missing Postconds.	•	•	•	•	•	•
Extra Postconds.	•	•	•	•	•	•
Missing Operator	•					

Figure 1: Domain theory error types and their manifestations.

world as well. This is the standard process/data distinction in computer science. Here we are considering imperfections in the agent's *process* knowledge; there might also be imperfections in the agent's *data* about the state of the world. These imperfections can result from incomplete or noisy sensing.

Mapping Knowledge Deficiencies to Failures

Each class of errors in the domain theory can produce a subset of the types of observed failures in planning or execution. The six types of failures are plotted against the five types of domain theory deficiencies in Figure 1. Many of the dots in the figure are obvious: for instance, it is clear that a missing operator can lead to incomplete plans, and that extraneous postconditions can lead to postconditions which are not met during execution.

The figure reveals some explainable patterns, with various rows containing the same set of dots. Missing operators and overspecific preconditions, for example, both lead only to incomplete plan failures. This is because both overly restrict the selection of a needed operator. This can eliminate correct plans from consideration, but it cannot cause incorrect plans to be formed.

Missing and extra postcondition errors both lead to all six types of failures. This is not surprising, since both types of errors result in the planner having an incorrect model of the world after applying the operator. Clearly, missing some necessary postcondition can preclude any plan being formed; similarly, some extraneous postcondition can wrongly clobber the preconditions of a necessary future operator. Either a missing or an extra postcondition can allow incorrect plans to be formed: missing postconditions, because the plans would have been clobbered had the postcondition been known; extra postconditions, by enabling other operators that are actually not possible in reality. These incorrect plans might be detected during planning, as a multiple plan failure, or during execution, as either unmet pre- or postconditions. Again, here the planner's model of the world does not match reality. If the system has explicit knowledge of what reality is supposed to be like, it may detect the reaching of an impossible state during planning, or an explicit failure during execution.

This leaves the overgeneral preconditions category. Overgeneral preconditions cannot preclude a plan being formed, since they simply loosen the restrictions on choosing the operator. However, they may allow incorrect plans to be formed in addition to the correct plan. These incorrect plans might lead to a state that is known to be impossible to reach, or (if executed) to a state that can be explicitly detected as an execution failure. Overgeneral preconditions can lead to the faulty operator not being applicable at execution time. However, this is detected not as a precondition failure (the system thinks the operator's preconditions are met), but as a postcondition failure, when it is detected that none of the operator's postconditions have been achieved. Overgeneral preconditions cannot lead to unmet preconditions for some future operator during execution, because if the faulty operator is able to be executed, the system's model of the world will be correct (the effects of the operator are correct); if the faulty operator is unable to be executed, this will be detected before any further operators are attempted.

Finally, we can briefly examine the columns of the table. The impossible state and explicit detection columns are identical, because in both cases the domain theory error leads to a world state that is explicitly known to be a failure. The dots in the multiple plans column correspond to the union of the dots in all three execution failure columns. An inconsistent plan may be detected during planning, if multiple plans are considered, or it may be detected when executed, by leading to any of the three types of execution failures.

KNOWLEDGE SOURCES FOR CORRECTING DOMAIN THEORIES

The previous section described the ways in which various types of domain theory errors might manifest themselves during planning or execution. But once an error is detected, what is to be done about it?

There is a tradeoff between simply fixing the *error*, and fixing the domain theory that led to the error. In some cases, fixing the domain theory may not be cost effective. For example, consider an execution error in which a slippery block slips out of a robot arm's gripper. If the block only slips one in a thousand times, it is probably enough for the system to simply grab the block again when it falls. However, if the block slips ninety percent of the time, it is probably better to fix the underlying domain theory (for instance, to squeeze harder for suspected slippery objects). This tradeoff has not been examined in detail, and we will not discuss it further here.

Let us assume, then, that in response to an error, the system will attempt to improve its domain theory. The problem can be viewed as a search through the space of domain theories. The goal is to alter the current domain theory, usually in the smallest amount possible, so that the new domain theory will not commit the error.

This search can be formalized as follows. Let us call the original domain theory T. T is a set of operators. We will denote the preconditions of an operator *op* as $Pre(op)$, and the effects of *op* as $Post(op)$.

We assume that the pre- and postconditions of operators may be generalized or specialized in known ways. For a pre- or postcondition p, $Gen(p)$ returns the set of nearest generalizations of p, and $Spec(p)$ returns the set of nearest specializations.

As an example of how *Gen* and *Spec* might work, consider the case in which all possible predicates in the domain are organized into a generalization hierarchy. The hierarchy specifies a set of generalizations and specializations for each predicate. For example, the condition `isa(?X,banana)` might have the (nearest) generalization `isa(?X,fruit)`. If such a hierarchy is present, it is easy to define $Gen(p)$ and $Spec(p)$: each simply traces either up or down the generalization hierarchy, for each of the predicates appearing in p. However, note that errors in the generalization hierarchy could preclude correcting the domain theory error.

The search for a correct domain theory is a heuristic search through the space of possible domain theories, starting at the current theory. The states of the space are full-fledged domain theories, and the operators alter a domain theory to produce a new one. These *theory revision operators* either alter an existing operator of the domain theory or create a new one:

- **ReplacePre**(op, p) - Replaces the preconditions of operator op with preconditions p.
- **ReplacePost**(op, p) - Replaces the postconditions of operator op with postconditions p.
- **CreateNewOp** - Creates a new, empty operator and adds it to the domain theory. The new operator has no preconditions or effects.

At each step in the search for a new domain theory, we may either generalize or specialize a precondition or effect of an operator, or add a new, empty operator. Thus, the set of possible operators that could be applied to alter a domain theory at each step is:

$$\begin{array}{ll} \forall op \in T[\forall p \in Gen(Pre(op))\mathbf{ReplacePre}(op,p)] & \cup \\ \forall op \in T[\forall p \in Spec(Pre(op))\mathbf{ReplacePre}(op,p)] & \cup \\ \forall op \in T[\forall p \in Gen(Post(op))\mathbf{ReplacePost}(op,p)] & \cup \\ \forall op \in T[\forall p \in Spec(Post(op))\mathbf{ReplacePost}(op,p)] & \cup \\ \mathbf{CreateNewOp} & \end{array}$$

Clearly, this is an infinite search space, including all possible domain theories that can be constructed out of the domain's predicates. To make the problem tractable requires strong biases and knowledge to control the search. In the rest of this section, we attempt to classify the types of biases and knowledge sources that are used.

Nearly all research on correcting domain theories has biased the search in domain theory space by using some type of hill-climbing. That is, only a single "node" of the search tree - a single variant domain theory - is stored and considered for further modification. In addition, most assume even stronger biases; for example, it is typical to assume that the domain theory errs in only a single operator (or is missing only a single operator). For example, if an execution error occurs, it might be assumed that the error was caused by the most recent operator being executed. This type of assumption greatly reduces the search space. Some systems make assumptions about which types of conditions to consider altering. For example, Gil (1991a,1991b) use heuristics such as considering only properties of objects being directly manipulated. Finally, it is often assumed that a single operator will only be altered in a single pre- or postcondition, thus reducing the depth of the search to a constant.

The search being done to alter a domain theory must solve a form of the credit assignment problem. Credit assignment is the problem of determining which of a possible set of causes is responsible for some observed effect. Successfully correcting the domain theory involves determining which knowledge (or lack of knowledge) was responsible for the failure, and altering the theory accordingly. In general, credit assignment is a very difficult problem. It is not unique to correcting domain theories, but shows up any time an effect must be attributed to one or more of a set of possible causes. This is a ubiquitous problem in machine learning. It is faced, for example, by concept learning programs, which must determine which combinations of a set of features determine category membership. It is also faced by reinforcement learning systems (e.g., (Sutton, 1990; Holland, 1986; Rumelhart and McClelland, 1986)) that learn which actions will achieve which effects in the world, by receiving positive feedback from the environment when some goal is met.

There are three basic sources of knowledge that a system can use to guide the search for a corrected domain theory:

1. **Internal Knowledge.** The system may use knowledge that it already has. For example, the system might have a complete (but intractable) deductive theory of the domain, or it might have knowledge of past cases of failures which can be modified to account for the current case.

2. **Teacher.** The system may receive knowledge from outside itself, from a teacher. This knowledge might take the form of direct advice (for example, indicating which knowledge in the domain theory is faulty), or perhaps carefully selected examples that will guide the system to the proper revision of the theory.

3. **External World.** The system might make use of observations of the external world; that is, additional examples. These examples

might be randomly chosen (such as those given to a typical concept learner). Alternatively, the examples might be *generated* by the system, in order to distinguish between alternative theory revisions.

These sources of knowledge can overlap and interact. For example, if a teacher gives the system a key example, the example consists of knowledge from the external world that has been wisely selected by the teacher.

Next, we examine each source of knowledge in more detail.

Internal Knowledge

By internal knowledge, we mean knowledge that the system already contains, which provides guidance in the search through the space of theory revisions. We have already mentioned one general form of internal knowledge that limits the search for a corrected theory - namely, the assumptions and biases that the system uses in performing the search. Many systems contain heuristic biases that limit the range of theory revisions considered. For instance, Gil (1991a,1991b) discusses heuristics such as assuming immediate feedback for operator effects. This allows only the most recently executed operator to be considered for revision upon execution failure, greatly reducing the credit assignment problem.

Another type of internal knowledge, which is available to all systems, is the original domain theory itself. Although the domain theory is imperfect, it is generally still useful in narrowing the set of possible alterations needed to fix the theory. For example, in analysis tasks, many systems have dealt with the problem of not being able to fully explain an example (corresponding to our "incomplete plan" category). Often these systems will form a maximal *partial explanation* using its incomplete domain theory. The parts of the example which are left unexplained can then be focussed on, to generate a set of hypotheses for the lack of knowledge within the domain theory (Pazzani, 1988; VanLehn, 1987; Hall, 1986). Empirical techniques are often used to discriminate these hypotheses (discussed further below).

In addition to the domain theory used to perform the task, the system may have other knowledge which it can use to analyze and understand its failure. Below, we investigate additional types of internal knowledge the system may have.

Complete Underlying Domain Theory. In some domains, it is possible to write a domain theory which is complete, but intractable. For example, it is easy to write down the rules of chess completely and correctly, but clearly intractable to use those rules to reason completely and correctly (and thus play perfect chess). In such cases, one way to reason tractably on the task is to use an *approximate* domain theory. For example, a novice player's domain theory for chess might contain the rule, "If you can capture the opponent's queen, always do so."

The approximate domain theory, used by the EBL system in trying to perform its task, will be imperfect, and errors will occur when using it. In such cases, it may be possible to use the intractable domain theory to analyze and correct the error in the approximate theory.

If the theory was originally intractable, why might it be tractable to use it once an error arises? The reason is that explaining a specific error is much more constrained than planning to avoid any *possible* error. Consider the situation in chess. It is much easier to explain why your queen was taken in a particular game, after a particular sequence of moves, than it is to *plan* a move that will truly minimize the possibility of losing your queen later on. In the first case, the full sequence of moves has been chosen; it simply needs to be analyzed. In the second case, we must consider every possible move that might be taken from this point in the game on.

In terms of search through the space of theory revisions, using a complete underlying domain theory to explain a failure corresponds to reasoning to pinpoint exactly the correct theory revision operator(s) to use in correcting the domain theory. Since the underlying theory is complete and correct, it is able to completely solve the credit assignment problem.

Gupta (1987) describes a scheme in which an complete underlying domain theory is used to explain an execution-time error. In Gupta's system, the domain theory is approximated by not considering certain goal interactions. Once a plan is produced, errors are detected by explicit failure rules, which monitor a plan's simulated execution and fire when failures occur. For example, a failure occurs if the robot's gripper melts when trying to grasp a part. An explanation of the conditions leading to the failure is then produced, using the underlying complete domain theory. This explanation pinpoints the preconditions that should be added to the operators in the approximate domain theory that is used for planning.

Other researchers have forwarded similar schemes, approximating a complete domain theory in various ways. Chien (1989) approximates a complete domain theory by initially reasoning only with the *direct effects* of operators, and assuming persistence for all other facts. When plans fail (or unexpectedly succeed) during execution, the inferred effects of operators are reasoned with to explain the failure. Here again, the intractable theory is made tractable when it is used to explain a particular interaction between operators, as opposed to planning for every possible contingency. Tadepalli (1989) approximates a domain theory for chess by considering only a limited subset of possible moves from each board position (those moves the system has learned in observing past games). Bennett (1987) deals with intractability in mathematical reasoning by approximating mathematical formulas. Doyle (1986) presents a system given a set of domain theories at multiple abstraction levels, which reasons with the most abstract first, and falls back on more and more concrete theories as failures arise. FAILSAFE-2 (Bhatnager and Mostow, 1990) learns overgeneral search heuristics during planning by assuming the most recent operator applied is to blame for search failures, but later specializes these heuristics when all search paths are eliminated. Bennett's (1990) GRASPER system approximates how far a gripper should be opened to pick up an object, and upon failure, uses a theory of tuning continuous approximations to alter the opening width. Ellman (1988) describes a methodology for choosing good simplifying assumptions for certain types of intractable theories. Flann (1990) approximates a domain theory by assuming a limited number of objects will be present.

Other work on approximating a domain theory to make it tractable to reason with has appeared under a different guise, namely abstraction planning. ABSTRIPS (Sacerdoti, 1974) is the classic example, showing that search could be greatly reduced by abstracting out preconditions of operators during planning. More recently, Unruh and Rosenbloom (1989) and Knoblock (1990,1991) have demonstrated methods that automatically abstract preconditions from operators, producing abstract versions of problem spaces to improve efficiency.

Other internal knowledge. Other types of internal knowledge besides a complete domain theory may be used in correcting imperfect domain theories as well. Some examples include using *analogy* to relate the error to knowledge in related domains; using *past cases* in which similar errors have been encountered and corrected; and using the original domain theory *abductively* to determine possible causes of an error. Not as much research has been done using these knowledge sources as using a complete but intractable domain theory, however.

Work in using analogy to repair domain theories has thus far been limited to making simple analogies to other operators within the faulty theory. An unknown operator or an operator being corrected can be compared with known operators that have similar preconditions or effects. Differences indicate revisions to the operator being corrected that may prove useful. Thus in analogy, the system makes use of previous knowledge of operators to guide the search for theory revisions.

Gil (1991a,1991b) compares an operator being revised with other operators which are structurally similar, identifying differences between them. This process is used to filter the set of possible revisions to the operator. The remaining revisions are chosen between via experimentation (discussed further below). Genest *et al.* (1990) try to understand an unknown goal concept (`borrow(Person, amount)`) by explaining examples of it as if they were examples of a known goal concept (here, `withdraw(Person,amount)`). These explanations are incomplete, and are completed using an abductive process (see below). The explanation structure produced using the known concept can then be analogized as similar to the structure of the explanation for the unknown concept, allowing the unknown concept to be learned. Kodratoff and Tecuci (1987) present a system which proposes new domain theory rules by analogizing the conditions of a newly learned rule to similar substructures within a semantic network of domain predicates.

Case-based approaches rely on storing complete past cases, and altering these cases to apply to new situations. CHEF (Hammond, 1986) uses a complete causal theory to explain and repair execution-time errors, but then stores the repaired plan in its case library, indexed under the type of error repaired. When a similar error is predicted in the future, the system can recall the past case and alter it to apply to the new situation. Redmond (1989) discusses using a case-based approach to explain unexpected situations.

Abduction involves reasoning from effects to causes. Logically, it can be viewed as using implication rules "backwards". For example, suppose we know $LightswitchPosition(off) \supset Darkness$. Then, if the room becomes dark, we might use this rule backwards, positing that the reason for the darkness is that the lights were turned off. If there are multiple conditions that imply darkness, we may have to use additional knowledge to choose between them. Thus abduction can be viewed as a way of limiting the possible theory revisions being considered. OCCAM (Pazzani, 1989; Pazzani et al., 1987) uses an abductive method to fill in missing knowledge in its domain theory. When some aspect of an observation cannot be explained, OCCAM looks for a domain theory rule that contains that aspect as its result. This rule's conditions are then generalized so that the rule covers the current case. This is a dangerous step, however, because overgeneralizations can easily result. OCCAM does not deal with the situation where multiple rules have the unexplained aspect as their result, and additional knowledge must be used to choose between them. Genest *et al.* (1990) also use abduction to complete partial explanations, but do not extend the domain theory using the abductive conclusions.

Determinations (Davies and Russell, 1987) are a type of internal knowledge that provide capabilities similar to abduction. A determination makes explicit the functional dependencies between attributes within the domain. It is similar to an implication, but not as strong: if A,B, and C *determine* D, this means that for given values of A, B, and C, D will always have the same value. If A, B, and C *imply* D, then given values of A, B, and C, we know what the value of D *is*, not simply that it is fixed. Determinations are thus "incomplete," but in a very limited way. Given examples, we can directly turn a determination into a set of implications. This kind of domain theory completion is demonstrated by Mahadevan (1989) . Widmer (1989) uses partial determinations, which guide the construction of "plausible" explanations that may then be confirmed or disconfirmed by a teacher. (Widmer's system allows additional types of domain theory incompleteness as well, and incorporates inductive learning techniques). A related method of constructing plausible explanations from incomplete knowledge is described by Oblinger and DeJong (1991) .

Teacher

In addition to internal knowledge, knowledge from outside the system itself may be called upon in correcting the domain theory. By knowledge from a teacher, we mean knowledge given by an informed source, meant to direct the system to the right correction of its domain theory. This is as opposed to knowledge from the external world which is either selected randomly, or selected by the system (experimentation).

A number of systems use a teacher as a *passive* source of knowledge. In these systems, the teacher simply verifies that a domain theory revision is or is not correct. The system uses some other knowledge or bias to generate hypotheses about how to fix the domain theory, and then these are validated or invalidated by the teacher. Examples of systems using this kind of passive teaching include (Kodratoff and Tecuci, 1987; Widmer, 1989).

A more interesting use of knowledge from a teacher are cases where the teacher can actively guide the domain theory repair process. Two main categories of teaching have been used. First, the teacher may provide examples to the system. These could be complete plans for the system to "observe," or unsolved problems for the system to solve. The examples are chosen to help the system repair its domain theory by causing it to concentrate on the right aspects of the situation. The second category of teaching is where the teacher provides advice to the system directly.

Teacher provided examples. Examples provided by a teacher can direct the system to faulty knowledge in its domain theory, and can be used to guide the correction process. In some systems, assumptions about the structure of the examples provides a strong bias that allows the system to solve the credit assignment and revision problems. One such system is VanLehn's SIERRA (VanLehn, 1987). SIERRA learns to perform multi-column subtraction by explaining examples of worked out problems given by a teacher. SIERRA receives its examples organized into groups called *lessons*. The key assumption is that the examples in each lesson will differ from those seen in the past by a single disjunct (or operator in our terminology); this is the "one-disjunct-per-lesson" constraint. This constraint limits the search in the space of theory revisions to considering revisions in a single operator. SIERRA learns by completing partial explanations of examples. The one-disjunct constraint greatly reduces SIERRA's search space when completing the explanations, which could be completed in a number of different ways. Since every example within a lesson is known to be incompletely explained due to the lack of the same disjunct, the "holes" in their explanations can be intersected to hone in on the missing disjunct.

Hall (1986) relies on a teacher to supply paired examples of structures that are equivalent. The system attempts to explain why the two structures are equivalent using rules that can transform one into the other. When a complete explanation cannot be found, the system creates a new transformation rule equating the two parts of the examples which were unable to be explained. Roy and Mostow (1988) use a similar approach.

Other systems make use of teacher-provided problems that are less structured. For example, in Tadepalli's (1989) system, the teacher appears to provide examples simply to "break" the domain theory - examples the teacher knows the system will not perform on correctly. Presumably such examples would eventually come up if examples were randomly selected. Here the teacher simply speeds up that process. Likewise, Redmond's (1989) system takes worked-out examples from a teacher, but these examples are not necessarily structured in any particular way.

Advice. Learning from advice (or "learning by being told") is a rather broad category. Advice could conceivably be given to help at any stage of a problem. In the context of recovering from an error caused by an incorrect domain theory, advice could specify how to recover, and/or how to correct the domain theory itself. This could be more or less direct; for instance, the advice could specify exactly which operator was incorrect, or could only indicate which features of the example caused the error.

There has not been much work on correcting imperfect domain theories by taking advice. Mostow's FOO (1981,1983) was a key early system. Mostow's focus was on making advice *operational* - that is, transforming it into a form the system could use directly. FOO took advice for the card game Hearts, such as "Avoid taking points." The operationalization process involved transforming the advice using a large number of both domain-independent and domain-specific transformation rules. However, which transformation to apply had to be manually selected. The result of operationalizing a piece of advice was a set of heuristics for the game.

FOO's initial domain theory of hearts contained the basic rules of the game. In a sense, the learning FOO did can be viewed as using advice to learn a tractable, approximate theory from an intractable one. However, advice was not used to correct imperfections in the tractable domain theory (the one used to actually play the game).

Martin and Firby (1991) have begun developing a system that learns to correct execution-time failures by being told. The approach is a combination of Martin's DMAP parser (Martin, 1989) and Firby's RAPs system (Firby, 1987). Upon an execution failure, the system takes advice in natural language indicating how to correctly perform the task. Comprehending the language input is aided by specific expectations which are set up by the failure. The advice allows the system to select an operator to complete performance of the task. The system learns to perform the task correctly next time. However, the issue of how to properly generalize the applicability of the advice is finessed. The system does not attempt to identify the relevant aspects of the state that led to the failure; rather, it seems to assume that the entire state is relevant. This may result in overgeneralizations in some cases.

Another example of using information given by a teacher to correct an imperfect domain theory is the Robo-Soar system (Laird and Rosenbloom, 1990; Laird et al., 1990). Robo-Soar's domain theory for picking up blocks is incorrect, because it does not realize that for certain pyramid-shaped blocks, the orientation of the gripper must match the orientation of the block (otherwise the gripper fingers slide off the sides of the pyramid, and the block cannot be picked up). When an execution failure occurs (the block isn't picked up), Robo-Soar prepares to search to discover which operator was at fault and what conditions of the state caused the error. Advice from the user guides this search. The guidance is extremely direct - the user indicates exactly which feature (here, block orientation) caused the failure, and which operator should be used to correct it (here, `rotate-gripper`). A similar approach is used to extend domain knowledge, in the case where the `rotate-gripper` operator is entirely missing from the original domain theory. Here, the advisor indicates the pre- and postconditions of the missing operator, and its implementation as an external command.

We are working on extending this work in two directions. The first is allowing more flexible types of instruction and advice, given in natural language. This again raises the issue of operationalization: how can the advice be translated into a form the system can use and learn from in a general way? Secondly, we are working on experimentation techniques, that will allow the system to learn when advice is not available. Instead of performing all the experiments that would be needed to exactly identify the domain theory, the system will use inductive techniques in an attempt to generalize the information gleaned from each experiment.

External World

In addition to internal knowledge or knowledge provided by a teacher, the external world - the execution environment - may be viewed as a knowledge source for correcting imperfect domain theories. This knowledge source has the property of being reliable and necessarily "correct". The system is faced with the problem of finding a domain theory which agrees with this knowledge source.

Specific observations of the external world can be used to constrain the search space of possible domain theories. These observations might be randomly selected; for instance, the system might be given a set of randomly chosen examples in a concept formation task. Alternatively, the observations might be selected by an informed source. A teacher may be able to give the system examples that will highlight the faulty knowledge in the domain theory, and its proper repair (as discussed above). Without a teacher's input, the system may be able to generate *experiments* with the intent of eliminating theories from the space or of confirming the predictions made by a particular theory. Correcting a domain theory using knowledge gleaned from experimentation is an *active* use of the external world; using randomly selected observations is a *passive* use.

Active use of the External World. Consider the case where a domain theory for a STRIPS-like domain has the operator **Go-thru-door(door)**, lacking the precondition **Open(door)**. Upon execution of some plan, the robot executes **Go-thru-door** but finds its postconditions unmet. Even if it is possible to localize the error to the preconditions of the **Go-thru-door** operator and make use of some internal knowledge to prune the set of possibilities, the system may still be left with a number of hypothetical corrections to the domain theory.

For example, the system might hypothesize that either `Open(door)` or `-Inroom(box,room)` (there are no boxes in the room the robot starts from) could be added as a new precondition. At this point, the system might use the external world as a knowledge source by experimenting to select the correct theory update from the set of candidates. In this case, the robot could be moved through another door which happens to be open (and succeed), or might remove all the boxes from the room it is in, and attempt **Go-thru-door** again (and fail, if the door were still closed). By carefully selecting experiments to perform, the system is able to directly prune hypotheses for domain theory correction from consideration.

Since the number of domain theory alteration operators which may be applied to the incorrect domain theory is quite large, and experimentation is fairly expensive, typically experimentation systems employ various biases and heuristics to limit the set of theory changes to be differentiated with experiments. The idea is to limit the set of hypotheses as much as possible before performing experiments. Rajamoney and DeJong (1988) , for example, appear to use a form of abduction to complete an incomplete explanation, leading to a set of multiple inconsistent explanations. Experimentation is used to choose among the various abductive assumptions and correct the domain theory. Gil (1991a,1991b) employs a number of heuristics, such as locality of action, analogy to similar operators, and generalization of previous experiences where an operator was successful to restrict the set of possible alterations to the preconditions of the operator. In addition, immediate feedback from actions is assumed, restricting the domain theory error to the most recently executed operator. Carbonell and Gil (1987) use heuristics based on the type of failure observed to guide the theory correction process. For example, if an operator was applied but none of its postconditions were met, it is likely that the preconditions of the operator are overgeneral (as opposed to, say, that all of its postconditions are wrong). Previous cases where the operator was successful and internal searches for other operators which may have clobbered the failing operator are used to limit the number and type of experiments to be run by the system.

Once the set of hypotheses has been sufficiently pruned, a system can formulate experiments to select between them. A single experiment may eliminate multiple hypotheses from consideration; thus, if experimentation is expensive, it may be worthwhile to do some analysis to determine the minimal number of experiments which are guaranteed to fully distinguish the hypothesis. This is a "minimal set-covering" problem, similar to that done to find a minimal set of test vectors for a digital integrated circuit.

Passive Observation of the External World. In some domains, it is not possible to actively control the set of examples the system receives. Rather, the major source of corrective knowledge here is a set of randomly chosen examples. The systems which rely on this type of knowledge typically make use of inductive techniques (such as version spaces (Mitchell, 1982) or information-theory approaches (Quinlan, 1986)) to determine a set of changes to the domain theory that will be consistent with all (or most) examples. Since the discriminating knowledge that can be gleaned from each example is less concentrated than for experimentation approaches, more examples are required for these systems to correct a faulty domain theory.

Pure inductive systems, such as ID3(Quinlan, 1986), can be viewed as learning an entire domain theory from scratch. This is the degenerate case of an "incomplete domain theory." In this paper, the focus is on correcting an already existing domain theory that has proven faulty. Closely related are techniques in which an existing domain theory is used to bias an inductive learner, allowing induction to converge more quickly on correct concept definitions (Flann and Dietterich, 1989; Towell et al., 1990; Bergadano and Giordana, 1988; Danyluk, 1987).

Consider our example of the previous section. In the absence of additional examples, the system might initially add *both* preconditions - `Open(door)` and `-Inroom(box,room)` - to the **Go-thru-door** operator. However, these preconditions may be marked as tentative. Later, when the system is able to go through an open door from a room with boxes in it, it can generalize the preconditions of **Go-thru-door** by removing `-Inroom(box,room)`.

Ali (1989) presents a system that uses a similar technique to learn a new operator. In this system, an overspecific tentative rule is induced to cover an unexplained concept. This rule is produced by conjoining all "related" predicates in the partial proof, where related predicates are those that share arguments. When future failures occur, tentative rules formed from the partial explanation of the new example are intersected with the original tentative rule, generalizing it. Eventually the rule stabilizes. This technique thus learns a "missing operator" - a new rule that the system didn't have before. The approach is similar to VanLehn's technique of intersecting partial explanations, but here there are no assumptions about the form of the examples. Thus the rules formed are sometimes faulty.

OCCAM (Pazzani, 1988) also uses an empirical approach to learn missing domain theory rules. Part of the knowledge OCCAM uses to build explanations is a very general set of rules describing causal relationships. These rules derive an abstract "explanation pattern" that is then verified by more specific domain knowledge. When an example cannot be fully explained, OCCAM uses the explanation pattern to pinpoint which step in the causal chain was not verified, and induces a new domain theory rule to verify the causal link. On subsequent examples, the new rule is generalized by an intersection technique, as in Ali and VanLehn's work. Thus the causal knowledge (internal knowledge) constrains the domain theory corrections that are considered, and induction over examples selects the correct one from amongst them. OCCAM goes a step further by maintaining strengths for empirically induced rules, and eliminating rules that prove incorrect on future examples. This protects the system from overgeneralizations; it might be viewed as a composite "specialization" operator.

Danyluk (1989) discusses a related approach in which a similarity-based learner is used to induce missing domain theory rules. Here, the partial explanation is used to bias the priorities of features for the similarity-based learner, providing a "context" for induction. Widmer (1989) allows similarity-based "explanations" as part of the full explanation of an example. Some feature of the example that cannot be explained either deductively or via determinations (see the discussion of Other Internal Knowledge) can be explained by searching for similarities between the example and previous examples which share the feature. Similarly, Wilkins' ODYSSEUS (1988) proposes relationships between predicates which are needed to complete incomplete explanations, and then uses induction over examples to verify the proposed relationship and learn a specific rule relating the predicates. In all these systems, as in the experimentation systems, we see some internal knowledge or heuristics being used to initially constrain the set of domain theory revisions that is considered, and then external world knowledge (examples) being used to make (or verify) the final decision.

One of the more comprehensive attempts to use inductive methods to correct imperfect theories in a classification task is presented by Ourston and Mooney (1990) . Their system, EITHER, identifies two types of errors: overgeneral preconditions, which lead to incorrect classifications (an explicit detection of failure), and overspecific preconditions (or missing rules), which prevent an example from being classified (an incomplete plan). If examples cannot be classified, EITHER constructs partial proofs of the examples, and then determines a minimal set of assumptions (new rules) needed to complete the proofs (using a greedy covering algorithm). If examples are incorrectly classified, EITHER uses its covering algorithm to find a minimal set of rules deemed responsible for the error. However, simply adding or removing rules may introduce new domain theory errors. If this is the case, then EITHER invokes an inductive learner (ID3) to *specialize* either the new rules being proposed or the old rules which caused incorrect classification.

ANALYSIS SUMMARY

We have presented classes of domain theory error types, the types of performance failures that can result, and knowledge sources for correcting errors. By crossing these categorizations with one another, we can examine which types of errors have been addressed using which types of knowledge for theory correction. Since this is a knowledge-level analysis, we are interested in examining which types of knowledge have been applied to which types of theory errors. For each knowledge source and error type, there may be a number of *mechanisms* that can be used in correcting the domain theory; we do not consider that aspect here.

We have chosen fifteen representative systems to classify (two or three for each type of corrective knowledge source). Figure 2 plots the space of error types versus corrective knowledge sources; Figure 3 plots out performance failure types versus knowledge sources. Into each square in these grids, we have placed references to those systems that have addressed applying a particular type of knowledge to correct a particular error type, or errors detected in a particular way.

One key thing to notice about these tables is the presence of empty squares, and the clumping of research within other squares in each column. Research using each type of knowledge source has tended to focus

	Internal Knowledge		Teacher		External World	
	Complete DT	Other	Examples	Advice	Random	Experiment
Overgen. Preconds.	Gupta Chien Bennett	Gil (heur. biases)		Robo-Soar Martin &Firby	Ourston &Mooney	Carbonell&Gil Rajamoney &DeJong Gil
Overspec. Preconds.		OCCAM (abduction) Widmer (determinations)			Ourston &Mooney Ali Widmer OCCAM	
Missing Postconds.	Chien (Gupta)					Carbonell&Gil Rajamoney &DeJong
Extra Postconds.						Carbonell&Gil Rajamoney &DeJong
Missing Operator		OCCAM (causal schemas) Wilkins (confirmation th.) Widmer (determinations) Genest+ (analogy)	Hall VanLehn (Wilkins)	Robo-Soar	OCCAM Wilkins Widmer Ali Ourston &Mooney	

Figure 2: Error type versus corrective knowledge source

	Internal Knowledge		Teacher		External World	
	Complete DT	Other	Examples	Advice	Random	Experiment
Incomplete Plan		OCCAM (causal schemas) Wilkins (confirmation th.) Widmer (determinations) Genest+ (analogy)	Hall VanLehn (Wilkins)	Robo-Soar	OCCAM Wilkins Widmer Ali Ourston &Mooney	
Multiple Plans						Rajamoney &DeJong
Impossible State						
Unmet Preconds.	(Chien)					Carbonell &Gil
Unmet Postconds.	Bennett (Chien)			Robo-Soar Martin &Firby		Carbonell &Gil Gil
Explicit Detection	Gupta (Chien)				Ourston &Mooney	

Figure 3: Performance failure type versus corrective knowledge source

on correcting only a subset of the types of errors (correspondingly, errors detected because of only a subset of the performance failure types). This may be because particular sources of knowledge are most amenable to correcting particular types of errors. However, there does not appear to be an a-priori reason why each knowledge source might not be applied to correcting all of the different kinds of errors.

A second thing to notice is that not many systems appear in multiple columns. That is, most systems have used a single knowledge source for domain theory correction, or have combined sources only in fairly simple ways (although there are exceptions). We feel an important area for future research is combining the different types of knowledge available for correction. Different types of knowledge might be available at different times and in different situations. A truly flexible system would be able to employ any and all of the various types of knowledge successfully, whenever such knowledge is available.

RELATIONSHIP TO OTHER FRAMEWORKS

Mitchell *et al.* (1986) made the first attempt to describe the basic ways in which a domain theory may be faulty. In their classification there are three categories. First, the theory may be *incomplete*, meaning that no explanation can be constructed for some examples. Second, the theory may be *inconsistent*, in which case conflicting statements can be proved from the theory. Third, the theory may be *intractable*, meaning that it is computationally prohibitive to construct explanations using it.

However, we have shown that a single type of underlying domain theory error may lead to more than one of these categories. For example, a domain theory containing an operator with missing postconditions may result in incompleteness (no plans) or inconsistency (multiple plans). In addition, we have seen that an intractable domain theory is typically not employed for reasoning. If there is a complete but intractable theory available it cannot be employed directly; instead, it is a knowledge source which may be employed to correct approximations and assumptions made in the actual domain theory used for reasoning. Therefore an intractable theory typically gives rise to either an incomplete or inconsistent theory.

Rajamoney and DeJong (1987) further subdivided the classes that

Mitchell *et al.* had defined. They distinguish incompleteness due to a lack of relevant knowledge, as opposed to incompleteness due to a lack of sufficient detail. A lack of relevant knowledge gives rise to incomplete plans; a lack of sufficient detail can lead the system to make assumptions, leading to multiple plans. However, in our view, making assumptions is part of the process of recovering from domain theory imperfections - a way of "generating" possible fixes. The assumptions can then be tested to determine which are correct, perhaps by using experimentation to distinguish between the multiple plans that arise from them. This is the method used by Rajamoney and DeJong (1988). Similarly, inconsistency problems are subdivided into inconsistency due to incorrect knowledge and inconsistency due to missing knowledge that would have defeated the inconsistent deductions.

Mitchell *et al.*'s classification focuses on the *performance* of the domain theory: does it lead to incomplete explanations? Does it lead to inconsistent proofs? Rajamoney and DeJong's classification moves closer to the classification of domain theory problems we have proposed here, in that there is more of a focus on the *knowledge* within the domain theory. However, performance is still the primary division, but subdivisions are attempted based on the type of knowledge deficiency giving rise to the performance failure.

In developing our classification, we have instead used the *type of knowledge deficiency* within the domain theory as the primary distinction. The goal was to cleanly separate the type of domain theory imperfection from the way in which it manifests itself in performance. Starting from the types of knowledge deficiency, we were able to derive the types of performance failures that each deficiency can lead to. This was found to be a one-to-many mapping: the various knowledge deficiencies can give rise to multiple types of performance failures. This one-to-many mapping has caused a confounding of error type (knowledge) and error detection (performance) in previous classifications.

Our analysis has revealed that although a number of useful mechanisms for correcting domain theories have been developed, none have yet demonstrated the ability to flexibly make use of any of the possible knowledge sources that might be available to guide the theory revision process. Theory revision is a complex process, taking place in a potentially huge search space. Any knowledge that is available should be utilized to guide the search. Therefore, our future goal is the development of a "universal weak method" of theory revision that is able to flexibly combine whatever knowledge is available to correct a faulty domain theory.

Acknowledgments
Thanks to Greg Weber, Craig Miller, and the other members of the Michigan Soar group for useful comments and discussion of earlier drafts of this chapter.

References

Ali, K. M. (1989). Augmenting domain theory for explanation-based learning. In *Proceedings of the International Workshop on Machine Learning*, pages 40–42.

Anderson, J. R. (1986). Knowledge compilation: The general learning mechanism. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An artificial intelligence approach, Volume II*, pages 289–310. Morgan Kaufmann.

Bennett, S. W. (1987). Approximation in mathematical domains. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 239–241.

Bennett, S. W. (1990). Reducing real-world failures of approximate explanation-based rules. In *Proceedings of the International Conference on Machine Learning*, pages 226–234.

Bergadano, F. and Giordana, A. (1988). A knowledge intensive approach to concept induction. In *Proceedings of the International Conference on Machine Learning*, pages 305–317.

Bhatnager, N. and Mostow, J. (1990). Adaptive search by explanation-based learning of heuristic censors. In *Proceedings of the National Conference on Artificial Intelligence*, pages 895–901.

Carbonell, J. G. and Gil, Y. (1987). Learning by experimentation. In *Proceedings of the International Workshop on Machine Learning*, pages 256–265.

Chien, S. A. (1989). Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 590–595.

Danyluk, A. P. (1987). The use of explanations in similarity-based learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 274–276.

Danyluk, A. P. (1989). Finding new rules for incomplete theories: Explicit biases for induction with contextual information. In *Proceedings of the International Workshop on Machine Learning*, pages 34–36.

Davies, T. R. and Russell, S. J. (1987). A logical approach to reasoning by analogy. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 264–270.

DeJong, G. F. and Mooney, R. J. (1986). Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176.

Doyle, R. J. (1986). Constructing and refining causal explanations from an inconsistent domain theory. In *Proceedings of the National Conference on Artificial Intelligence*, pages 538–544.

Ellman, T. (1988). Approximate theory formation: An explanation-based approach. In *Proceedings of the National Conference on Artificial Intelligence*, pages 570–574.

Firby, R. J. (1987). An investigation into reactive planning in complex domains. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 202–206.

Flann, N. S. (1990). Applying abstraction and simplification to learn in intractable domains. In *Proceedings of the International Machine Learning Conference*, pages 277–285.

Flann, N. S. and Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning*, 4:187–226.

Genest, J., Matwin, S., and Plante, B. (1990). Explanation-based learning with incomplete theories: A three-step approach. In *Proceedings of the International Conference on Machine Learning*, pages 286–294.

Gil, Y. (1991a). Acquiring domain knowledge for planning by experimentation. Thesis Proposal, Carnegie Mellon University, School of Computer Science.

Gil, Y. (1991b). A domain-independent framework for effective experimentation in planning. In *Proceedings of the International Machine Learning Workshop*, pages 13–17.

Ginsberg, M. L. and Smith, D. E. (1987). Reasoning about action I: A possible worlds approach. In Brown, F. M., editor, *The Frame Problem in Artificial Intelligence: Proceedings of the 1987 Workshop*, pages 233–258. Morgan Kaufmann.

Gupta, A. (1987). Explanation-based failure recovery. In *Proceedings of the National Conference on Artificial Intelligence*, pages 606–610.

Hall, R. J. (1986). Learning by failing to explain. In *Proceedings of the National Conference on Artificial Intelligence*, pages 568–572.

Hammond, K. J. (1986). Learning to anticipate and avoid planning problems through the explanation of failures. In *Proceedings of the National Conference on Artificial Intelligence*, pages 556–560.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An artificial intelligence approach, Volume II.* Morgan Kaufmann.

Knoblock, C. A. (1990). Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 923–928.

Knoblock, C. A. (1991). Search reduction in hierarchical problem solving. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 686–691.

Kodratoff, Y. and Tecuci, G. (1987). DISCIPLE-1: Interactive apprentice system in weak theory fields. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pages 271–273.

Laird, J. E., Hucka, M., Yager, E. S., and Tuck, C. M. (1990). Correcting and extending domain knowledge using outside guidance. In *Proceedings of the International Machine Learning Conference*, pages 235–241.

Laird, J. E. and Newell, A. (1983). A universal weak method: Summary of results. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 771–773.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Maching Learning*, 1(1):11–46.

Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64.

Laird, J. E. and Rosenbloom, P. S. (1990). Integrating execution, planning, and learning in Soar for external environments. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1022–1029, Boston, Mass. AAAI Press.

Laird, J. E., Yager, E. S., Tuck, C. M., and Hucka, M. (1989). Learning in tele-autonomous systems using Soar. In *Proceedings of the NASA Conference on Space Telerobotics.*

Mahadevan, S. (1989). Using determinations in EBL: A solution to the incomplete theory problem. In *Proceedings of the International Workshop on Machine Learning*, pages 320–325.

Martin, C. E. (1989). Pragmatic interpretation and ambiguity. In *Proceedings of the Annual Conference of the Cognitive Science Society*, pages 474–481.

Martin, C. E. and Firby, R. J. (1991). Generating natural language expectations from a reactive execution system. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 811–815.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., and Gil, Y. (1989). Explanation-based learning: A problem-solving perspective. *Artificial Intelligence*, 40:63–118.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, 18:203–226.

Mitchell, T. M. (1990). Becoming increasingly reactive. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 1051–1058, Boston, Mass. AAAI Press.

Mitchell, T. M., Keller, R. M., and Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning*, 1.

Mooney, R. J. (1990). Learning plan schemata from observation: Explanation-based learning for plan recognition. *Cognitive Science*, 14:483–509.

Mostow, D. J. (1981). *Mechanical transformation of task heurisitics into operational procedures*. PhD thesis, Carnegie-Mellon University, Department of Computer Science.

Mostow, D. J. (1983). Learning by being told: Machine transformation of advice into a heuristic search procedure. In Michalski, R. S., Carbonell, J. G., and Mitchell, T. M., editors, *Machine Learning: An artificial intelligence approach*. Morgan Kaufmann.

Mostow, J. and Bhatnager, N. (1987). Failsafe - a floor planner that uses EBG to learn from its failures. In *Proceedings of IJCAI-87*, pages 249–255.

Oblinger, D. and DeJong, G. (1991). An alternative to deduction. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 837–841.

Ourston, D. and Mooney, R. J. (1990). Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the National Conference on Artificial Intelligence*, pages 815–820.

Pazzani, M. (1989). Detecting and correcting errors of omission after explanation-based learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 713–718.

Pazzani, M., Dyer, M., and Flowers, M. (1987). Using prior learning to facilitate the learning of new causal theories. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 277–279.

Pazzani, M. J. (1988). Integrated learning with incorrect and incomplete theories. In *Proceedings of the International Machine Learning Conference*, pages 291–297.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.

Rajamoney, S. and DeJong, G. (1987). The classification, detection and handling of imperfect theory problems. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 205–207.

Rajamoney, S. A. and DeJong, G. F. (1988). Active explanation reduction: An approach to the multiple explanations problem. In *Proceedings of the International Machine Learning Conference*, pages 242–255.

Redmond, M. (1989). Combining case-based reasoning, explanation-based learning and learning from instruction. In *Proceedings of the International Workshop on Machine Learning*, pages 20–22.

Roy, S. and Mostow, J. (1988). Parsing to learn fine grained rules. In *Proceedings of the National Conference on Artificial Intelligence*, pages 547–551.

Rumelhart, D. E. and McClelland, J. L., editors (1986). *Parallel Distributed Processing.* MIT Press, Cambridge, MA.

Sacerdoti, E. D. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135.

Sutton, R. S. (1990). Integrated architectures for learning, planning and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224.

Tadepalli, P. (1989). Lazy explanation-based learning: A solution to the intractable theory problem. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 694–700.

Towell, G. G., Shavlik, J. W., and Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the National Conference on Artificial Intelligence*, pages 861–866.

Unruh, A. and Rosenbloom, P. S. (1989). Abstraction in problem solving and learning. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence*, 31(1):1–40.

Widmer, G. (1989). A tight integration of deductive and inductive learning. In *Proceedings of the International Workshop on Machine Learning*, pages 11–13.

Wilkins, D. C. (1988). Knowledge base refinement using apprenticeship learning techniques. In *Proceedings of the National Conference on Artificial Intelligence*, pages 646–651.

Winston, P. H., Binford, T. O., Katz, B., and Lowry, M. (1983). Learning physical descriptions from functional descriptions, examples, and precedents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 433–439.

A Cognitive Science Approach to Case-based Planning.*

Kristian J. Hammond
Department of Computer Science
The University of Chicago
1100 East 58th Street
Chicago, IL 60637

Colleen M. Seifert
Department of Psychology
University of Michigan
330 Packard Road
Ann Arbor, MI 48104

Abstract

In this chapter, we will review our current work in case-based approaches to planning. Our research examines the organization and structure of episodic memory as determined by its functional role in guiding reasoning. The theoretical thrust is based upon the intuition that a large part of the reasoning process is guided by the contents and organization of events in memory. The major goal of this work is the development of functional explanations of the representational organization and conceptual vocabularies that support reasoning tasks. Our current work can be characterized in terms of three major tasks:

- Uncovering *vocabularies for describing causal similarities* between episodes in memory.
- Examining the use of features in the *access and retrieval of cases* from memory.
- Understanding the role of memory organization in the support of *opportunistic planning and problem solving.*

*This work was supported, in part, by the Office of Naval Research under contract N0014-85-K-010 to the University of Chicago and contract N00014-91-J-1128 to the University of Michigan.

A COGNITIVE SCIENCE APPROACH

There is often a tension within Psychology and Artificial Intelligence collaborations. From a psychological perspective, the scientific mandate is to examine and describe behaviors, and *then* work to develop a functional rationale for them. From an AI point of view, the goal is to develop functional theories of behavior and *then* try to explain how the theory relates to human activity. The underlying goals of the two areas of research, descriptive power for psychology and functionality for AI, are often in conflict.

Another issue is the nature of the relationship between AI systems and human behavior. In work such as Newell and Simon's GPS (Ernst and Newell, 1969), Feigenbaum's EPAM (Feigenbaum, 1990), or Hayes-Roth and Hayes-Roth's opportunistic planning (Hayes-Roth and Hayes-Roth, 1979a), the AI systems can be construed as mimicking human behavior at the I/O level. As a result, the quality of such systems has been defined as how well their performance actually matches that behavior. However, these matches imply very little about the nature of the systems that support the behaviors. Due to the differences between the environments in which humans and machines exist, and the background knowledge to which they have access, it is difficult to evaluate this match since there is no way to control for these differences. A further, more extreme, argument could be made that such behavioral matches between man and machine are in fact evidence *against* a match at the algorithmic level, because we can assume that there is not a match at the level of background knowledge and reasoning environment.

One possible fix to this problem would be to normalize background knowledge by building simulations in domains requiring little or no knowledge. EPAM (Feigenbaum, 1990), a model of verbal learning, successfully used this strategy, but the fact that EPAM was able to mimic human behavior in this task revealed little about the task itself or the way humans tend to learn. The study involved behavior that is so far removed from normal information processing that what is observed is the disintigration of a system at its edges rather than its normal functions.

The reverse of this problem arises within the confines of AI work on "discovery" (Langly, 1981; Langley *et al.*, 1983), where the goal is to recreate a line of reasoning that led to the formation of a natural law. In these systems, the environment (in terms of ongoing input and background knowledge) is much more impoverished than that of any comparable human task; therefore, any system that actually does replicate the results of human reasoning may be doing so using methods that have little to do with that reasoning. Because they are functioning in different environments, the fact that the final results look similar actually argues that the processes must be different. They will be dissimilar for the same reasons that a water plant in near the Great Lakes will be unlike one at the edge of the Gobi dessert: they must produce the same product, but under different circumstances. Therefore, that fact that I/O behavior matches human performance may reveal very little about whether the mechanisms involved in the AI model are the same as the ones producing the behavior in human cognition.

Rather than limiting comparisons to I/O behavior, we argue for the importance of testing the fundamental *principles* involved in the machine model. The question is not simply "can this system match human performance", but instead, "can this system's process model fit the requirements for a process model in humans?" We propose that the most meaningful method of comparison for human and machine learning systems is to test not performance, but assumptions: Can the assumptions of the model be shown to hold for human cognition?

In our own work, we have focused on testing specific claims of our machine learning models with human subjects. This involves identifying specific claims made by the model and testing whether or not these principles hold as well for human subjects; if so, the class of models with those assumptions can be ruled *in* as possible explanations for human processing. A rigorous study of the central claims of any machine learning model will verify the generalizations of the theory that hold across other tasks and learning situations, and that distinguish the approach from other competing theories.

By examining the assumptions of an AI model, we can determine whether it is plausible as a psychological model while avoiding the temptation of adjusting specific versions in order to exactly match human behavior in a particular task. Testing the assumptions means greatly improving the generality of the comparison, so that the results obtained can be applied to a number of tasks rather than the exact situation modeled in the program. It is a challenging task to design experiments that get at the important assumptions of an AI model as well as produce psychological results that are of interest for their substantive contribution.

In the following sections, we present an extended example of an AI model and some experiments designed to examine a central claim of the paradigm it represents. The model itself is case-based planning (CBP). A specific hypothesis from this model—that features used in indexing and retrieving past examples will be differentially utilized based on their predictive ability—is examined in a series of studies on human subjects. In the sections that follow, we describe a specific hypothesis from CBP and the experiments conducted to examine this claim. Next, we provide an overview of our current projects in CBP. It is our hope that our collaboration will serve as an exemplar for the integration of psychological experiments with computational models.

AN EXAMPLE OF INTEGRATED RESEARCH: CASE-BASED PLANNING

In the following sections, we will describe current research on a computational model of cognition and the psychological research that directly relates to and informs it. Both research efforts focus on the area of case-based planning. Systems within this paradigm have the following basic features:

- New plans are built from old plans.
- Projection is replaced with anticipation, which is based on experience rather than simulation.
- Plans are selected on the basis of the goals that they satisfy, the problems that they avoid, and the features in the world that have been associated with them in the past.
- Knowledge of the effects of individual operators is not used in planning itself. It is used instead to explain failed plans in order to repair them and discover the features that can be used later to anticipate and thus avoid the problems in later planning.

- Planning problems due to faulty modification of existing plans are repaired, and the repairs are added to the set of changes that have to be made when adding the goals to other existing plans.

The thrust of the case-based approach is to include "lessons learned" into a plan so that past errors can be pointed out and avoided. One way in which this is done is to *anticipate and avoid* problems due to plan interaction. To do this, case-based planners keep track of the features in their domains that are predictive of particular problems so those problems can be predicted in future situations. They also save plans in memory, indexed by the goals that they satisfy and the problems that they avoid. As a result, the prediction of a problem allows a case-based planner to find the plans in memory that avoid it. This is done, however, without the cost of complete projection, because the planner is accessing its memory of *actual* problems rather than attempting to project through all *possible* ones.[1].

The nature of the actual *content* of features used in the organization of memory is an important principle within this approach. In particular, CPB depends upon the type and source of features used in order to access maximally useful prior cases from episodic memory. A processor would most benefit from retrieving related past case simutaneous with the current planning situation. This would require indexing prior cases based on the abstract features that will be available to the processor at the time of desired retrieval. For example, in planning situations, this point would occur when one knows the current planning conditions and constraints and is considering what action to take. Retrieving an appropriate case from memory at that point could provide a potential solution, or a warning about an avoidable past planning failure.

Therefore, one of the paradigmatic assumptions of case-based planning is that *memory access must be provided through the features that tend to predict potential failures in planning.* This claim serves as a testbed for comparing human cognitive abilities to the requirements of a CBP model – can case access be shown to differentially favor features related to predicting failures? If human memory is indeed functionally organized

[1]For a more detailed description of case-based planning, see (Hammond, 1990a). For a discussion of learning from failure see (Hammond, 1986).

as claimed in the CBP approach, cases sharing features predictive of future planning failures should better access each other than cases sharing other, nonpredictive features.

For example, consider this story:

> A chemist was trying to create a new compound designed to allow preservation of dairy products stored at room temperature. The chemist was so confident that his experiments would succeed that he went ahead and ordered several truckloads of fresh dairy products to be delivered to demonstrate the utility of the new compound.

This story contains features with *predictive functionality*; that is, the features present in the story allow the retrieval of past information, and the prediction of a possible planning failure, *before* the complete structural analog is present (namely, *counting your chickens before they're hatched.*) Frequently, planning decisions must be made before all matching information is available. Therefore, the *partial* feature set apparent before the decision point must be sufficient to retrieve potential solutions and possible pitfalls. This *predictive features* hypothesis states that access to prior related exemplars is possible with partial feature sets, and that access based on predictive features will be superior to other features for case retrieval.

In order to examine this issue, five experiments were conducted to investigate the retrieval of prior cases based on new, incomplete feature cues. We developed a single session reminding paradigm (based on Gentner and Landers, (1985)), where subjects study a set of base stories and later given a set of cue stories and asked to write down any base stories that come to mind. The stories were based on thematic abstraction units (TAUs) (Dyer, 1982) selected because of their relevance in planning. TAUs are based on abstract interactions of goals and plans as reflected in cultural adages—especially those involving expectation and planning failures—and are likely to be familiar to subjects. TAUs may contain planning information about potntial problems that can occur and how to avoid or solve them, and serve to organize storage of similar individual episodes in memory.

If functionality is an organizing principle for structures like TAUs in memory, then features that predict failure should form a privileged set that leads to access at a higher rate than that attained using other sets of features. Optimally, one should access a relevant theme after receiving information related to the planning decision, but before deciding and taking action. Remindings based on other features sets would not provide the opportunity to avoid a planning failure. Thus, under a predictive features hypothesis, one would not expect privileged access based on stories which provide only the outcome of an incident. In the experiments, cue stories were used that contained only the features predictive of a planning failure, or only the features stating the failure and its aftermath.

The results showed that, while both predictive features and outcome features resulted in reliable retrieval of base stories, the set that included features predictive of a planning decision were better cues than decisions and outcomes. The predictive features provide better selectivity: More mismatches occurred in response to theme-outcome stories, as compared to predict-theme stories. This indicates that the elements in the predict-theme stories distinguished the themes more clearly, and thus subjects tended either to find the right story or to have no answer. The theme-outcomes, however, tended to evoke a wider range of responses, indicating that the features available in them were shared by other potentially retrievable episodes. Such outcome cues may not have sufficient specificity, and so could lead to spurious matches as well as accurate ones.

We examined whether predictive features would provide better access than other features. Predictive feature sets included only the thematic features apparent before the planning decision was made; by contrast, outcome features included the thematic elements containing the actual planning decision made and its outcome. The main findings were, first, that both predict and outcome cues were matched at higher-than-chance level based on structural features alone. This replicates earlier findings about structural remindings, but extends the phenomenon to find reliable access to matching exemplars based on only a subset of the structural similarities in a true analog. In terms of number of matches, there was no difference between the two cues types. However, the predict-theme cues produced significantly fewer mismatch intrusions than did outcome features. Therefore, the stories containing predictive features led to more *reliable* access to matching stories in memory.

In followup experiments, we determined that subjects rated the sets of predictive cues and outcome cues as equally similar to the base stories overall and thematically; thus, these results are not due to any differences in length of cue or amount of information in the two types of cues. We also asked subjects to match pre-decision stories and post-decision stories to the base stories directly rather than from memory. The results suggest that the predictive features more *distinctively* characterize the theme, and therefore are more useful in recognizing the relevent past episode. Thus, the memory retrieval differences observed must be due to the utility of the cues in planning.

These results confirm the predictive functionality assumption in our case-based models ROENTGEN (Berger and Hammond, 1991) and RUNNER (Hammond, 1990c), and serve as a strong verification of this approach. In human memory, indices related to when and how to make a particular decision are more useful than equally related information about the decision. Much of the previous work on memory retrieval of cases and analogies has focused on the features that determine similarity; however, these results suggest distinctiveness, as well as similarity, is very important for recognizing *when* to apply prior knowledge. From a functional perspective, the most useful incomplete feature sets contain elements that predict potential problems and allow access to relevant cases *before* one must make a planning decision, thereby improving the planner's ability to predict and avoid planning failures.

These experiments confirm a central claim of the case-based reasoning model: that memory access to prior cases can be facilitated by cues containing particular predictive features. Evidence for this functional perspective lends support to case-based models as potential models of human reasoning and memory. By expanding and testing the types of features useful in analogical retrieval, this work has application and interest for issues in psychology beyond serving as a direct test of a particular machine learning model.

OVERVIEW OF RESEARCH

This form of collaboration has resulted in a unique research relationship between the computational and empirical research teams. The work on the functional models of memory use done by Hammond and his students is actualized in three working case-based systems for planning and design. The experimental work by Seifert and her students has demonstrated a powerful relationship between reasoning and memory. The resulting collaboration has recently given rise to new directions in case-based reasoning.

Our computer modeling and empirical studies address three major topics:

- Uncovering *vocabularies for describing causal similarities* between episodes in memory.
- Examining the use of features in the *access and retrieval of cases* from memory.
- Understanding the role of memory organization in the support of *opportunistic planning and problem solving.*

Our joint research is driven by the assumption that there is a functionality inherent in the processing of cognitive agents. For example, we are interested in a memory model that reflects experimental data, but also explains that data in terms of functional justifications (Anderson, 1990). This constraint guides us in analyzing and observing human performance, and underlies arguments for the structure of the computational models.

Computer Modeling

The Artificial Intelligence group at Chicago has been exploring the use of episodic memory in planning and problem solving. We have been examining how an episodic memory of successes, failures, and useful optimizations can be applied to all aspects of reasoning: analysis, plan construction, plan recognition, and plan execution. Work to date has shown that our initial planning model greatly facilitates execution-time monitoring and adaptation of plans. Most recently, this work on planning and execution has given rise to a more general theory of the memory-based generation and control of action in autonomous agents.

Several findings arise from the current projects. We have succeeded in moving the standard case-based planning model over to three new, more technical domains: vehicle scheduling (Mitchell *et al.*, 1989), radiation treatment planning (Berger and Hammond, 1991), and common-sense planning (Hammond *et al.*, 1990). We have developed a model of opportunistic memory that supports the flexible production and execution of plans (Hammond, 1989b). We have proposed a method of domain stabilization, called *enforcement*, that plays a role parallel to that of learning in that it allows a system to build a correspondence between the external world and the internal representation (Hammond, 1991). We have continued our work in identifying planning vocabularies to facilitate the functional organization of memory. In particular, we have been extending our initial work on the organization of failure and repair knowledge into the areas of competitive and cooperative planning (Goldweic and Hammond, 1991; Hammond *et al.*, 1991). Finally, we have proposed a memory-based model of planning, learning, and, activity, called *agency*, in which *all* processing is driven by the recognition of elements in a dynamic episodic memory (Hammond *et al.*, 1990).

These research foci are embodied in three main projects, all in the demonstration or production-level stage. All of these projects are case-based in nature; that is, they all depend on libraries of example or experiences to drive their reasoning. As such, each is designed as a learning system that develops special purpose structures to deal with the details of their domains.

These projects are:

- RUNNER: A reactive planner in the domain of "errand running" that plans from episodic information rather than a rule base of operators.
- ROENTGEN: A radiation treatment planner that plans for new problems by finding and modifying prototypes that are tested and debugged in a simulated world.
- POLYA: A case-based problem-solver in the domain of high- school geometry that takes its input from diagrams as well as symbolic descriptions. POLYA currently deals with only one simple example.

In the sections that follow, we will outline each of these four projects, describe their domains, the scientific issues that each confronts, and the progress that has been made.

RUNNER: RUNNER is a project aimed at a case-based approach to common-sense planning. Based on the TRUCKER scheduling project, the RUNNER project is aimed at modeling the full spectrum of activity associated with an agent—goal generation, plan activation and modification, action execution, and resolution of plan and goal conflict—and not just the more traditional aspect of plan generation alone.

The scientific goals of the RUNNER project are straightforward. In RUNNER we are studying the coordination of plan production and action. In the the area of production, we have been investigating the reuse of plans and a knowledge based approach to plan adaptation. In the area of action and its coordination with planning, we have used RUNNER as a vehicle to examine execution-time plan modification, opportunistic goal satisfaction, environmentally driven execution of existing plans and the interleaving of planning, action and monitoring.

The achievements of the RUNNER project include:

- The development of a model of execution-time plan modification that makes use of episodic memory to store a planner's goals and recognize opportunities to satisfy them.
- The expansion of the current theory of learning from execution-time failure to include learning from opportunity.
- The development of a plan *executive* that allows for execution-time reactivity as a by-product of plan monitoring.
- The introduction of a theory of learning through enforcement in which a planner actively organizes its environment so as to add to predictability.
- The development of a model of activity that is fully integrated with the production of plans and the monitoring of the environment.

This project has included the development of a model of planning and execution that places both processes under a single, general architecture. This model of *agency* is based on three precedents: Schank's structural model of memory organization (Schank, 1982), our own work in opportunistic memory and dependency directed repair (Hammond, 1990b), and the work of Martin and Riesbeck on Direct Memory Access Parsing (1990).

ROENTGEN: The basic architecture of our earlier work on CHEF is now being applied in the ROENTGEN project to the more demanding domain of planning radiation therapy for cancer. As with CHEF(Hammond, 1989a), ROENTGEN plans from a memory of actual cases—past successes and failures in treatment planning—to develop plans for new cases. As new problems are presented to it, ROENTGEN retrieves similar, successful cases from memory and then uses them as suggestions to help formulate its first attempt at a plan. It performs standard modifications and then uses a radiation dose calculator to compute the dose distribution over an entire body cross subsection. The results of this calculation are then used to discover problems with the plan such as over-radiation of a sensitive structure or hot spots in the body. If it finds a failure, ROENTGEN uses a description of the failure to select and apply a plan repair rule. The repaired plan is then passed back to the dose calculator and the cycle is repeated.

The ROENTGEN project has produced a set of results including:

- The transfer of the basic model of case-based planning embodied in CHEF over to a more technical and scientifically grounded domain.
- The construction of a full ROENTGEN prototype system that is able to retrieve, modify, test, and repair plans in the domain of radiation treatment therapy.
- The use of a dynamic indexing scheme that allows for an easy translation from quantitative to qualitative indices.
- The addition of ideas from qualitative physics to our model of learning from failure.

- The development of an **Apprentice–Assistant–Advisor** approach to the life-cycle of intelligent tools.

This last effort, the **Apprentice–Assistant–Advisor** life-cycle, is aimed at centering the development of an intelligent tool to aid rather than replace existing human experts. Our vision of this life-cycle is one in which the "intelligent" aspects of the system involve building and maintaining libraries of plans, modifications, known problems, and resulting repairs.

POLYA: POLYA is a project in memory-based problem-solving. Its domain is geometry, and its task is the construction of proofs of angle and line congruency. The input to POLYA includes not only the standard symbolic representations of the problem but also a graphic representation of the problem that the system itself must parse and understand.

Because of the nature of the input, work on POLYA includes an interesting mix of goals. On one hand, it is a straightforward attempt to bring case-based techniques to bear on a traditional AI problem-solving domain with the associated issues of indexing, retrieval and solution adaptation. On the other, it is the first serious attempt that we know of to model the entire range of problem-solving subtasks—including drawing information from images and text, selection and application of problem-solving strategies, and the production of external notes—within the confines of a single architecture. POLYA is also a variant of the basic RUNNER architecture and also reflects our interest in the modeling of agency.

Psychological Studies

The experimental studies in our joint work have addressed issues central to the models of case-based reasoning described above. Three projects: representational vocabulary for planning, indexing and access to cases, and opportunistic problem solving are summarized in this section. Complete descriptions are available in Seifert and Patalano (1991), Hammond, Seifert, and Gray (1991), and Johnson and Seifert (1991).

Vocabulary: The memory organization of plans.

Human planning involves strategies that deal with the interaction of plans for multiple goals. (Hayes-Roth and Hayes-Roth, 1979b; Miller *et* *al.*, 1960; Byrne, 1977). Previous work on representing goal interactions (Schank and Abelson, 1977; Wilensky, 1983; Schank, 1982) introduced the notion of relating episodes on the basis of similarities in the pattern of goals and plans they contain. A conceptual vocabulary is needed to describe the interactions between goals and plans as they are represented in memory.

Our theory of planning vocabulary (Hammond, 1989a; Hammond *et al.*, 1991) specifies the structures used to describe planning conflicts in terms of abstract causal characterizations of planning problems. In addition, we identified strategies for resolving specific conflicts along with particular plans to implement the strategies. The vocabulary includes specific strategies that tell not just *where* a particular causal chain can be effectively altered, but also *how* to alter it. For example, here are some of the strategies one might apply in a specific situation where one is planning against a problematic precondition:

- Get an alternate agent to run the plan.
- Run the plan as quickly as possible
- Use an alternate plan that does not require that precondition.

Finally, the model allows the determination of whether a particular plan can apply in the current domain situation, based on the features that the strategy alters or uses in constructing a plan. Thus, the only features used to find one of the strategies in memory are those which have some causal relevance to the way in which that strategy is implemented within a particular domain. The resulting indices and vocabulary information informs the planner about how to apply specific planning strategies, and in what circumstances the individual strategies are relevant.

This model constitutes a normative theory of what one optimally could and should learn about planning through investigation of causal relationships. To determine whether it constitutes a cognitive model of planning, human subjects were asked to provide common-sense planning solutions to a set of problems. The planning problems used were exemplars based on the model of vocabulary for indexing complex multiple goals and plans. Each of six instances represented a single goal interaction conflict, framed in terms of a planning problem with familiar surface content (such as celebrating a sick friend's birthday, jogging after dark, and picking up an exam while avoiding one's professor). Subjects were asked to solve the problems using commonsense knowledge of problem-solving in the real world. Their solutions were then compared to the set predicted by the vocabulary model.

Since few novel intrusion occurred, the results suggest that the plans generated fit the model very well. However, the model failed to predict several important findings. First, subjects frequently abandoned the goal in the face of the conflict, an option not selected by the model as long as one of the repair strategies was possible. Second, subjects strongly preferred one of the three proposed sets of strategies but the model does not predict such a preference in strategy type. However, the uneven pattern of subjects' strategy use across examples suggests that subjects were indeed using the features which the model predicts will be useful in indexing of repairs. Further studies will examine how these vocabulary features are used by subjects in planning in novel domains.

Thus, the representational scheme in the model provides a way to organize and access plans and past episodes relevant to current planning problems. This makes it possible to store plans designed for specific planning interactions such that it can be accessed, when the interaction arises again with a different set of specific goals. The data collected so far support the validity of this representational scheme in human cognition.

Indexing and access to cases.

A central issue in research on analogical reasoning is how to gain access to a relevant analog in memory at the appropriate time (Gick and Holyoak, 1980; Gentner, 1983; Pirolli and Anderson, 1985; Gentner and Landers, 1985; Holyoak, 1985; Seifert *et al.*, 1985; Anderson, 1986; Ross, 1987; Ross, 1989). The main findings have examined the features involved in access to analogies in terms of either superficial or structural relationships to the intended analogical meaning. Superficial features have been found to result in better access than structural features in two different tasks; in one, there was low overlap of superficial features in the retrieval set (only one superficially related story existed in memory) (Gentner and Landers, 1985); in the other, novices learning in a new domain (statistics) tended to rely on surface features in accessing prior examples (Ross, 1987). However, in these and other studies, more abstract, relational features also reliably produced access to prior structurally-related examples (Gentner and Landers, 1985; Ross, 1987; Seifert *et al.*, 1985).

Much of the existing work in memory retrieval has seemed to indicate that human memory is organized around fairly surface level descriptions of the world rather than more causally relevant features. As a result, one goal of our empirical studies was to resolve the apparent conflict between strong prior evidence for lack of analogical transfer with the strong argument for the functionality of that retrieval ability within any memory system. Prior work on types of features in retrieval of exemplars argued that an advantage exists for surface features vs. structural or inferred features (Gentner and Landers, 1985; Gentner and Ratterman, 1987). An alternative explanation for these results is that surface features seem to dominate memory retrieval whenever the task allows shallow processing of the information, which would make surface features available sooner than inferred features. We designed an experiment that manipulated the level of processing subjects were able to utilize on the test cases. In one condition, the Gentner and Landers (1985) and Gentner and Ratterman (1987) manipulation was replicated; in another, the word order of the same test stories were randomized, with the resulting scrambled stories presumably allowing only shallow processing of meaning (words such as "squirrel" are easily identified, but connections between words—structural features—could no longer be understood from the scrambled story). In the scrambled condition, access was presumably provided to surface features but not to structural features.

Our results replicated most of the accessibility effects described in Gentner and Landers (1985) and Ratterman and Gentner (1987). The general pattern of results in the scrambled cue story condition does not differ from the pattern of results in the intact story conditions. We obtained the identical pattern of significant differences between pairs of means for both groups, and the interaction between match type and word order was not significant. This supports the conclusion that the advantage of surface features in retrieval may be due to greater access to surface than structural features, rather than a preference or advantage to surface features given equal availability of both feature types.

Abstract features are particularly important in situations where the superficial features in the domain are not as helpful in indicating causal relationships. For example, access based on structural features alone will be necessary when learning in a new domain where past experiences don't share surface features with new problems; when the superficial features are insufficient to distinguish among many related examples; and when the needed information is stored in terms of the surface characteristics of another domain so that cross-contextual reminding is required. Thus, the use of abstract features, supported in prior experiments ((Seifert *et al.*, 1986), (Rattermann and Gentner, 1987)), can occur independently of competition from more available surface features. Resolving this controversy is an important step in supporting models of case-based reasoning, where indexing and features are central to the approach.

Opportunistic Memory: How to index pending goals.

There is some suggestion from psychological experiments that human cognition is able to take advantage of improved circumstances in order to reattempt previously failed task goals. (Hayes-Roth and Hayes-Roth, 1979b). Why might memory be designed to note and take advantage of task status in retrieving past problems? From our modeling work in RUNNER, it is clear that keeping track of the status of problem-solving attempts and being able to use that information is very useful in the later solution of the interrupted problems. If a goal is not satisfied, information about that failure can be used to preserve and encode the problem in a way that might facilitate its later retrieval. Consequently, failed problems may be more likely to be recalled, and to be pursued for a second time. In order to do so, however, incomplete problems must be stored and retrieved from memory. What, if any, are the differences in the way in which we encode and remember completed versus interrupted problems?

We began to address this issue by examining the results of a classic experiment by Zeigarnik (1927), which claimed that interrupted problems are more frequently retrieved from memory than completed ones. Zeigarnik and other researchers accounted for this effect in terms of social, motivational, and personality factors (see (Prentice, 1944)), suggesting that when a subject sets out to perform the operations required by one of the tasks, the subject develops a "quasi-need" for the completion of the task. By re-examining the Zeigarnik effect in terms of modern theories of problem representations, goals, and context effects, we set out to explain the circumstances under which access to pending cases will occur, and how this effect may function within a broader cognitive architecture.

In the experiments, we explored factors including the nature of the interruption, the processing time spent on problems, and the relative set size of the incomplete problems. In the first study, we attempted to match Zeigarnik's methods (1927) as closely as possible; using word problems, we manipulated task interruption versus completion on each problem. Our results showed that free recall memory for completed tasks is better than memory for interrupted tasks; however, this is not surprising given that subjects spent substantially more time, both when correct and incorrect, on the completed problems than on the incomplete problems. In a second experiment, we attempted to replicate this result, and added a condition where we attempted to highlight subjects' attention to the blocked problems. The highlighted version used a game board, where subjects were led to believe they would have to return to incomplete problems in order to complete the experiment. However, the highlighted condition did not improve memorability of the unsolved problems. From these first two experiments, which failed to replicate the Zeigarnik effect, we conclude that the effect is not simply a consequence of interruption.

In a third experiment, we tested incomplete problems resulting from impasses in solution attempts, rather than simple interruption. In this procedure, subjects work through a large set of problems, determining which condition a given problem falls into by either completing or getting stuck on the problem. The results of this experiment showed that heightened memorability for unanswered problems occurs when such problems are in a blocked state, rather than merely in a state of interruption. In a fourth experiment, we removed the possible confound of longer time spent on incomplete problems by allowing a specified amount of time to work on each problem. The results still demonstrated better memory for blocked problems, allowing the conclusion that the Zeigarnik effect does not depend on time differences.

However, this successful replication also included set size differences in favor of incomplete problems. We hypothesized that under blocked rather than simply interrupted conditions, a heightened memorability exists for problems in the category of smaller set size. We created such a situation in a fifth experiment by including more difficult problems, and allowed subjects less time to work on each problem, resulting in a completion rate of about one half. When set sizes were approximately equal, the free recall results showed no difference in memory for answered compared to unanswered problems.

One possible explanation for set size effects is expectation failure. In these experiments, initial experience with the set may create expectations of success or failure, so that when unexpected outcomes occur, the problem becomes more memorable. For example, if one can answer all but a few items in a crossword puzzle, the unsolved clues become quite salient; similarly, if most of the cues are unsolved, then the successful answers appear more salient. Thus, the enhanced memorability of the Zeigarnik effect may be part of a larger phenomenon – expectation failures – that can be used to mark significant events that deviate from predictions. This serves to highlight memories that may be important and to enhance availability for later processing.

Thus, access to incomplete problems may be facilitated under certain circumstances; namely, when processing time, the nature of interruption, or expectation failures make incomplete problems salient. Under these conditions, the status of completion can serve as a useful index to past problem situations. Within the program environment, the results suggest ways to enhance the availability of unfinished tasks in memory.

CONCLUSION

In our work in case-based planning, our objective is to discover more about the structure and use of episodic memory in reasoning processes. Specifically, we are interested in a theory of how episodic memory can be effectively used to deal with new planning and problem-solving situations over a wide range of activities. The work described in this chapter is aimed at extending our understanding of case-based reasoning across three dimensions: vocabularies for encoding of cases in memory, features for access and retrieval of episodes, the role of memory organization in problem-solving and opportunism. All involve the examination of fundamental principles of case-based reasoning; in particular, representational vocabularies, indexing and access to past cases, and opportunistic retrieval of past episodes. These empirical projects are closely tied to specific implementational models (such as RUNNER and ROENTGEN) that help to constrain the hypotheses examined in the experimental studies.

Our current work involves developing a theory of *agency*, the active pursuit of goals in the face of a changing environment. As our work continues, we are focusing on the following issues:

- Identify the circumstances under which previous cases can be exploited to learn how to deal with new situations.
- Discover more about how cases are stored in memory and how they are evoked by new related situations.
- Examine how learning occurs during planning, as a result of explaining and repairing failures.
- Uncover vocabularies for describing abstract similarities between episodes in memory.
- Catalog features used in the access and retrieval of episodes from memory.
- Understand the role of memory organization in the support of *opportunistic planning.*
- Demonstrate the use of cases in human learning.

Our two parallel research initiatives in Artificial Intelligence and Psychology function by proposing computer models that provide functional motivation for proposed experiments, while the data from those experiments provide a solid set of constraints for the computer modeling projects. Our overall goal is an integrated, interdisciplinary theory of case-based memory and planning that is empirically demonstrated by the experimentation and functionally justified by the computer models. We hope that the interdisciplinary approach will provide results that will converge on a unified theory of the use of memory in planning, problem solving, explanation, and decision making.

ACKNOWLEDGMENTS

We would like to thank Mary Bellmar for her help in editing this chapter.

References

Anderson, J. R. (1986). Knowledge compilation: The general learning mechanism. In R. S. Michalski, . T. M. M., J. G. Carbonell, editor, *Machine learning: An artificial intelligence approach*, volume 2, pages 289–310. Kaufman, Los Altos, CA.

Anderson, J. (1990). *The adaptive character of thought.* Lawrence Erlbaum, Hillsdale, NJ.

Berger, J. & Hammond, K. (1991). Roentgen: A memory-based approach to radiation therapy treatment design. In Bareiss, R., editor, *Proceedings of the 1991 DARPA Workshop on Case-Based Reasoning*, Los Altos, CA. Morgan Kaufmann Associates.

Byrne, R. (1977). Planning meals: Problem solving on a real data base. *Cognition*, 5.

Dyer, M. (1982). In-depth understanding: A computer model of integrated processing for narrative comprehension. Technical Report 219, Yale University Department of Computer Science.

Ernst, G. & Newell, A. (1969). *GPS: a Case Study in Generality and Problem Solving.* ACM Monograph Series, Academic Press, New York.

Feigenbaum, E. (1990). The simulation of verbal learning behavior. In *Readings in Machine Learning.* Morgan Kaufmann Publishers.

Gentner, D. & Landers, R. (1985). Analogical reminding: A good match is hard to find. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, Tuscon, AZ.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Psychology*, 7:155–170.

Gick, M. L. & Holyoak, K. J. (1980). Analogical problem solving. *Cognitive Psychology*, 12:306–355.

Goldweic, P. & Hammond, K. (1991). Multi-agent interaction: Dynamics of cooperation. Working notes of the 1991 AAAI Fall Symposium Series.

Hammond, K. J., Converse, T., & Martin, C. (1990). Integrating planning and acting in a case-based framework. In *The Proceedings of the 1990 National Conference of Artificial Intelligence.*

Hammond, K. J., Seifert, C., & Gray, K. (1991). Intelligent encoding of cases in analogical transfer: A hard match is good to find. *The Journal of the Learning Sciences.*

Hammond, K. (1986). Learning to anticipate and avoid planning problems through the explanation of failures. In *The Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA. AAAI. 15 pages.

Hammond, K. (1989a). *Case-Based Planning: Viewing Planning as a Memory Task*, volume 1 of *Perspectives in Artificial Intelligence*. Academic Press, San Diego, CA.

Hammond, K. (1989b). Opportunistic memory. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. IJCAI.

Hammond, K. J. (1990a). Case-based planning: A framework for planning from experience. *The Journal of Cognitive Science*. Ablex Publishing, Norwood, NJ.

Hammond, K. J. (1990b). Explaining and repairing plans that fail. *Artificial Intelligence Journal*, 45(2).

Hammond, K. J. (1990c). Intelligent encoding of cases. In *Working notes for the 1990 AAAI Spring Symposium Series: Case-based Reasoning*.

Hammond, K. J. (1991). Learning and enforcement: Stabilizing environments to facilitate activity. In *aaai91*.

Hayes-Roth, B. & Hayes-Roth, F. (1979a). A cognitive model of planning. *Cognitive Science*, 3(4):275–310.

Hayes-Roth, B. & Hayes-Roth, F. (1979b). A cognitive model of planning. *Cognitive Science*, 3(4).

Holyoak, K. (1985). The pragmatics of analogical transfer. In Bower, G., editor, *The Psychology of Learning and Motivation*. Academic Press, Inc., Orlando, FL.

Johnson, H. K. & Seifert, C. M. (1991). Predictive features in analogical access. *Journal of Memory and Language*.

Langley, P., Bradshaw, G., & Simon, H. (1983). Redicovering chemistry with the bacon system. In Michalski, R., Carbonell, J., & Mitchell, T., editors, *Machine learning: An Artificial Intelligence Approach*. Morgan Kaufman, Los Altos, CA. 24 Pages.

Langly, P. (1981). Data-driven discovery of physical laws. *Cognitive Science*, 5:31–54. 23 Pages.

Martin, C. E. (1990). *Direct Memory Access Parsing*. PhD thesis, Yale University.

Miller, G. A., Galanter, E., & Pribram, K. H. (1960). *Plans and the structure of behavior*. Holt, Rinehart and Winston.

Mitchell, M., Hammond, K. J., & Converse, T. (1989). Planning in an open world: A pluralistic approach. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society.*

Pirolli, P. L. & Anderson, J. R. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology*, 39:240–272.

Prentice, W. (1944). The interruption of tasks. *PsychologicalReview*, 51.

Rattermann, M. & Gentner, D. (1987). Analogy and similarity: Determinants of accessibility and inferential soundness. In *Proceedings of the 9th Annual Conference of the Cognitive Science Society*, Seattle, WA. Cognitive Science Society.

Ross, B. H. (1987). This is like that: The use of earlier problems and the separation of similarity effects. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13(4):629–639.

Ross, B. H. (1989). Remindings in learning and instruction. In *Similarity and analogical reasoning.* Cambridge University Press, Cambridge.

Schank, R. C. & Abelson, R. (1977). *Scripts, Plans, Goals and Understanding.* Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Schank, R. C. (1982). *Dynamic memory: A theory of learning in computers and people.* Cambridge University Press.

Seifert, C. M. & Patalano, A. J. (1991). Memory for interrupted tasks: The zeigarnik effect revisited. In *Proceedings of the Thirteenth Annual Cognitive Science Society.* Lawrence Erlbaum Associates.

Seifert, C., McKoon, G., Abelson, R., & Ratcliff, R. (1985). Memory connections between thematically similar episodes. *Journal of Experimental Psychology: Learning, Memory and Cognition.*

Seifert, C. M., McKoon, G., Abelson, R. P., & Ratcliff, R. (1986). Memory connections between thematically similar episodes. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 12:220–231.

Wilensky, R. (1983). *Planning and Understanding: A Computational Approach to Human Reasoning.* Addison-Wesley Publishing Company, Reading, Massachusetts, Reading, Mass.

Zeigarnik, B. (1927). Das behalten erledigter und unerledigter handlungen. *Psychologische Forschungen*, 9:1 – 85.

Bias in Planning and Explanation-Based Learning[1]

Paul S. Rosenbloom, Soowon Lee
Information Sciences Institute and
Computer Science Department
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292

Amy Unruh
Knowledge Systems Laboratory
Computer Science Department
Stanford University
701 Welch Rd. (Bldg. C)
Stanford, CA 94305

Abstract

Biases enable systems to make decisions in realms where all legitimate sources of knowledge have been exhausted. This article investigates the application of biases to the problem of planning, and how this can indirectly induce effective biases in a learning process that is based on a planner's experiences. Experimental results from six biased planners, plus several more complex multi-method planners, indicate complex tradeoffs among planner completeness, planning efficiency, and plan length. Learning also varies in complex ways among these planners, with one notable result being the ease with which some planners learn rules that can generalize from one object to many; a phenomenon known in machine learning as *generalization to N.*

INTRODUCTION

Bias, as originally defined in the context of inductive concept learning from preclassified training instances, is "any basis for choosing one gen-

1. We would like to thank Mike Barley, Pat Langley, and Steve Minton for their helpful feedback on this work. This work was sponsored by the Defense Advanced Research Projects Agency (DOD) and the Office of Naval Research under contract number N00014-89-K-0155. This chapter is a reformatted reprint of a version of Rosenbloom, P. S., Lee, S. & Unruh, A. (1992). Bias in planning and explanation-based learning. In S. Minton (Ed.) *Machine Learning Methods for Planning and Scheduling*. San Mateo, CA: Morgan Kaufmann. In Press.

eralization over another, other than strict consistency with the observed training instances (Mitchell, 1980)." It has proven to be a particularly useful notion in this context because it isolates and highlights a crucial aspect of induction algorithms: the knowledge and processes that determine how the algorithms go beyond the training instances; that is, which inductive leaps they make. For example, it makes it clear that explanation-based learning (EBL) can be viewed as inductive concept learning, where the domain theory and operationality criterion provide a particularly strong bias on the induction process. In general, the idea in induction is to start with the notion of an *unbiased hypothesis space* that consists of every possible generalization of the observed training instances. The *unbiased version space* is then the portion of this space that is consistent with the observed training instances. The bias determines which element — if any — of the unbiased version space is returned as the output of the induction algorithm.[2]

As just described, bias affects the output of the induction process, but not the efficiency with which it proceeds. This is because bias is implicitly used solely as part of the test for a generate-and-test method — first the elements of the unbiased hypothesis space are generated, and then tested to see if they meet the criteria of the bias. However, if the bias can be incorporated directly into the generator (Bennett & Dietterich, 1986), then it can also have a significant impact on the efficiency of the induction process by reducing the number of candidates that are generated. In this way, bias can lead to effective control of search. However, despite this close relationship between search control and bias — as we shall see later, search control can also lead to bias — the two notions are not isomorphic. Bias determines which answer is given, while search control determines the efficiency with which that answer is found.

The principal thesis of this chapter is that the notion of a bias — suitably generalized — can also be usefully applied to planning. Several of the potential benefits are straightforward mappings from the inductive concept learning case: (1) it can help to organize and understand many of the concepts in planning — such as linearity and protection — by focusing on their effects on selection from the hypothesis space (that is,

2. In search terms, the set of possible generalizations provides a problem (or solution) space, consistency with the observed training instances provides a goal test, and the bias determines which of the states that satisfy the goal is actually reached.

the *plan space*); and (2) it can reduce computational effort by reducing the number of hypotheses (plans) that must be examined if a good bias is selected.

A third potential benefit, and one that is not derived from the standard usage of bias in inductive concept learning, is that planning biases can indirectly induce an effective bias on learning. The basic issue here is how to make the rules learned from planning episodes more utile than they would be otherwise. Without such modifications, the rules may actually hurt performance rather than help it (Minton, 1990; Tambe, Newell, & Rosenbloom, 1990). In fact, much of the research in plan learning over the past several years can be construed as investigating the direct application of biases to improve the utility of learned rules; for example, ULS (Chase et al., 1989) uses statistical information to abstract learned rules by dropping conditions that have a high conditional probability of being true given that the preceding conditions are also true. The main problem with this type of approach is that it uses the biasing knowledge only for post hoc revision of the learned rules, and not to assist the planner in doing its job. Thus an opportunity is missed to reduce planning effort, and thus to reduce learning time, since learning time is closely linked to the time required by the planner for it to reach situations where a rule can be learned. The alternative approach is to bias the planner — for example, by having it create plans that ignore preconditions of little statistical relevance — and then to learn from these altered planning episodes. Recent evidence shows that, at least for some forms of abstraction, this approach can reduce planning time (and thus reduce learning time), and increase the generality and utility of the rules learned (Unruh & Rosenbloom, 1989; Knoblock, Minton, & Etzioni, 1991).

In the following three sections of this chapter we lay out in more detail the application of bias to planning, describe how (biased) planning can be implemented within Soar (Laird, Newell, & Rosenbloom, 1987; Rosenbloom, Laird, & Newell, in press), and provide results from some of the biases implemented so far. This is followed with an investigation of one approach to the flexible use of multiple biases within a single planner, by constructing a set of multi-method planners out of sequences of increasingly less biased planners. Such multi-method planners can alter the trade-offs among planning efficiency, plan length, and planner completeness. They also provide an opportunity to investigate how to learn about which planners — and thus which biases — to use for particular problems. The remainder of the chapter examines whether biased planning can lead to useful biases on learning; in particular, on explanation-based learning (EBL) of plans. This proceeds via a case study in generalization to N (Boström, 1990; Cohen, 1988; Shavlik, 1989; Subramanian & Feldman, 1990).

BIAS IN PLANNING

Figure 1 displays the analogy between inductive concept learning and planning that underlies the transfer of the notion of bias to planning. In both cases the output of the process is to be some element of the unbiased hypothesis space that is consistent with the process's input. Where the two cases differ is in the definitions of "unbiased hypothesis space" and "input". In concept learning, the unbiased hypothesis space is the power set of the possible instances, and the input is a set of preclassified training instances. In planning, the unbiased hypothesis space is the power sequence — that is, the set of all sequences — of the possible operators,[3] and the input is the combination of an initial state and a goal. In either case — despite these differences — in the absence of a bias, any element of the hypothesis space consistent with the input (that is, any element of the version space) is as good as any other. Thus, in both cases, it is the bias that breaks this deadlock and determines which such element becomes the output of the process.

Biases can be either absolute or relative. An absolute bias completely removes regions of the unbiased hypothesis space, creating an incomplete *biased hypothesis space*. For example, in concept learning, a generalization language provides an absolute bias by eliminating any element of the unbiased hypothesis space not expressible in the language. Because absolute biases introduce incompleteness into the process, devising a "good" bias is critical: if it is too weak it has no effect, but if it is too strong it can eliminate the desired output. A relative bias defines a partial order

3. The specification here assumes that the plan space contains only totally-ordered sequences of operators, but it does not rule out a search strategy that incrementally specifies an element of the plan space by refining a partially-ordered plan structure.

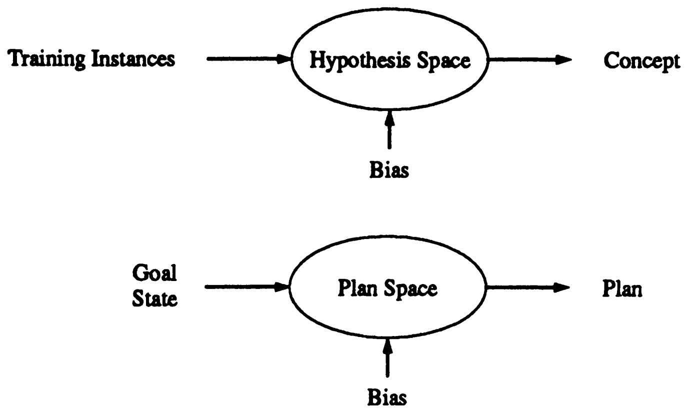


Figure 1. Analogy between concept learning and planning.

on the elements of the hypothesis space. Returning to the concept learning case, a preference for simpler hypotheses provides a relative bias. Relative biases do not generate incompleteness, but rules learned from relative biases tend to be more complex than those learned from absolute biases, so that the utility of learned rules may be decreased. In the case of planning, both absolute and relative biases have been used, though not generally under these labels.

On the absolute side, two common planning biases are *linearity* and *protection.* A linearity bias removes from the hypothesis space all plans in which operators in service of different unachieved goal conjuncts occur in succession; that is, once an operator for one unachieved goal conjunct is in the plan, operators for other conjuncts can occur only after the first goal conjunct has been achieved. For example, given the initial state and the goal conjuncts in Figure 2(a), plans such as the one in Figure 2(b) would be eliminated, while plans such as the one in Figure 2(c) would remain. A protection bias eliminates all plans in which an operator undoes a goal conjunct established by an earlier operator in the sequence. For example, given the initial state and the goal conjuncts in Figure 3(a), plans such as the one in Figure 3(b) would be eliminated since the operator (move A Table) undoes the goal conjunct (On A B) which is established by the earlier operator (move A B), while plans such

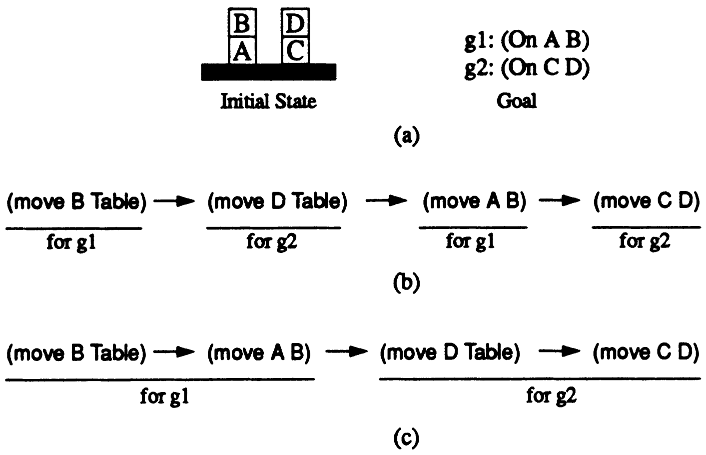


Figure 2. Example of the effects of a *linearity* bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

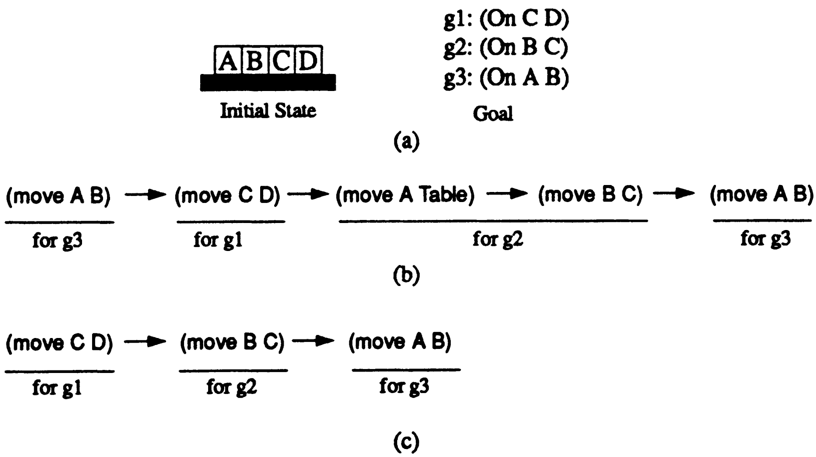


Figure 3. Example of the effects of a *protection* bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

as the one in Figure 3(c) would remain.

Two less common, but nonetheless interesting, absolute biases are *directness* and *nonrecursiveness.* A directness bias eliminates all plans in which there is at least one operator that does not directly achieve a goal conjunct included in the problem definition. For example, given the goal conjuncts and operators in Figure 4(a), plans such as the one in Figure 4(b) would be eliminated since the operator (move B A) does not directly achieve any of the goal conjuncts in the problem definition, while plans such as the one in Figure 4(c) would remain. A nonrecursiveness bias eliminates all plans that require a derivation embodying recursive subgoals. For example, given the goal conjuncts and operators in Figure 5(a), plans such as the one in Figure 5(b) would be eliminated because it requires a derivation embodying a recursive subgoal — operator (move B Table) is chosen in service of conjunct (Clear C), but in making it applicable, a recursive Clear conjunct (Clear B) is generated (resulting in the selection of (move A D) as the first operator). On the other hand, plans such as the one in Figure 5(c) would remain.

Because these biases are absolute, they all engender incompleteness in the planner; that is, they reduce the number of plans that the planner can possibly generate for particular problems. This incompleteness can be used to speed up the planner. However, it only really helps if the bias is an appropriate one; otherwise, the effort expended in searching the biased space is wasted. Thus, in order to show that using one of these absolute biases is reasonable, some form of appropriate justification is needed. The most common form of justification is an *independence assumption.* Linearity and protection both depend on some form of independence assumption. For linearity, one assumes that while solving one goal conjunct, operators not in service of that conjunct need not be considered. For protection, one assumes that while solving one goal conjunct, operators that interact negatively with previous goal conjuncts need not be considered. Other justifications include *progress* and *boundedness.* A progress assumption — that it is always possible to move forward, and never required to move backward — underlies all greedy biases, of which protection is one. Boundedness assumptions limit the total effort that it is reasonable to expend in solving a problem. Nonrecursiveness and directness are both justified by boundedness assumptions, though based on different bounds.

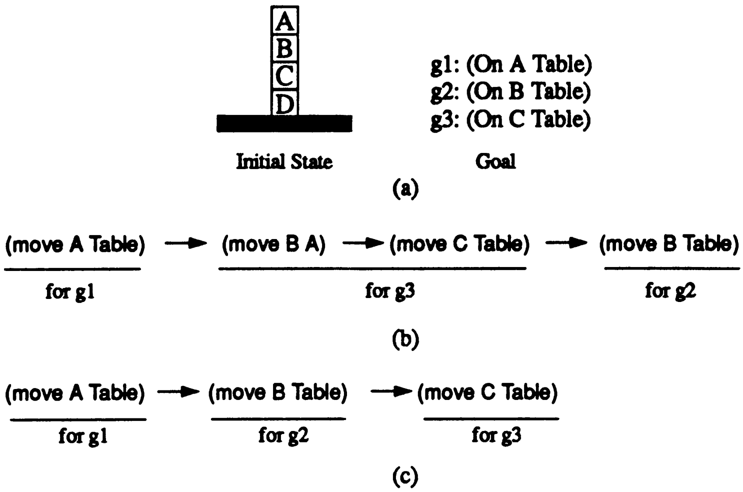


Figure 4. Example of the effects of a ***directness*** bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

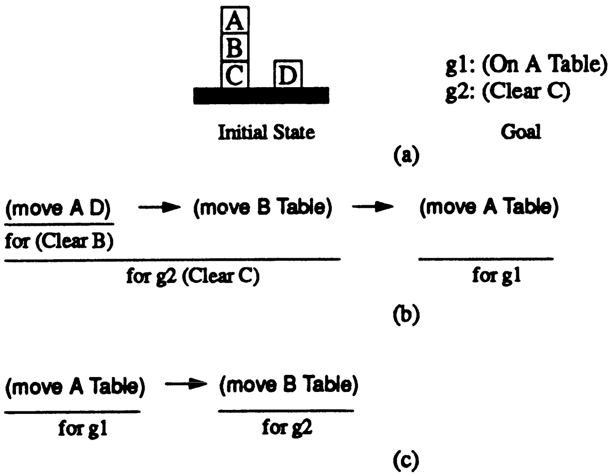


Figure 5. Example of the effects of a ***nonrecursiveness*** bias on the plan space: (a) initial state and goal conjuncts, (b) plan eliminated, (c) plan remaining.

Each of the absolute biases can also be made into a corresponding relative bias by just preferring plans that meet the bias to plans that do not. For example, a relative protection bias would prefer protected plans to unprotected ones, but still fall back on unprotected ones if necessary. In addition, there are a number of biases which are most naturally cast directly in relative terms. Three common relative biases are *abstraction*, *earliness*, and *shortness*. An abstraction bias prefers plans that can be generated by locally filling in the gaps in an abstract plan; an earliness bias prefers plans discovered earlier in the search; and a shortness bias prefers shorter plans. The primary justification for those biases is *efficiency*; specifically, planning efficiency for abstraction and earliness and execution efficiency for shortness.

The dichotomy between absolute and relative biases parallels the comparable dichotomy in the use of control knowledge. For example, Gratch and DeJong (1990) distinguish between *structural* and *ordering* modifications to control strategies, which amounts to a distinction between the addition of absolute and relative control knowledge. The two distinctions are also closely coupled, as imposition of a bias of one type can engender control of that same type, and vice versa.

LEARNING AND PLANNING IN SOAR

Our investigations of biased planning, and its influence on learning, have been performed in the context of Soar, an architecture that integrates basic capabilities for problem-solving, use of knowledge, learning, and perceptual-motor behavior (Laird, Newell, & Rosenbloom, 1987; Rosenbloom et al., 1991). Soar has not traditionally been seen as a planning architecture, partly because it does not create structures that resemble traditional plans, and partly because its problem-solving approach does not closely resemble the traditional planning methods. However, appearances can be deceiving. In this section we first very briefly review familiar territory, summarizing how learning works in Soar, and then examine planning in Soar from the perspectives of both plans and planning methods.

Learning in Soar

Soar learns via a chunking process that creates new rules that can recreate the results of subgoals in relevantly similar future situations (Laird, Rosenbloom, & Newell, 1986). For each independent result of each subgoal it creates a rule that has an action side based on the result, and a condition side based on a dependency analysis of the subgoal processing that led to the result. In effect, chunking is much like explanation-based learning (Rosenbloom & Laird, 1986).

The Soar representation of plans

This is not the place to attempt resolution of the philosophical questions over what is and is not a plan. However, enough of a working definition is needed to allow the identification of what structures in Soar act as plans. Thus, for the purpose of this identification, we will assume the following generic definition of a plan for a problem (that is, a state and a goal):

> A *plan* for a problem is a structure that represents the sequence of actions to be taken for that problem.

This definition captures a number of the important aspects of what it means for a structure to be a plan: that a plan is a representation of actions, that the actions in the plan have not yet taken place, and that a plan is for the solution of some problem (or class of problems). The definition is also neutral on a number of issues for which there is no present need to take a stance: the way the plan is encoded (whether declaratively or procedurally, with what syntax, and with what degree of expressibility), by whom the plan was created (the agent that is to execute it or some other other agent), and to whom the plan is representational (to the agent or to an external analyst). Other aspects ignored by this definition which may ultimately be of importance are: whether the structure is operational for control — for example, whether the plan can lead to action in a bounded amount of time or whether something like exponential theorem proving is required to derive indirect action consequences — and whether there is a deliberate act of creation or selection of the plan for the problem.

With this definition of a plan in hand, it is now possible to identify the two predominant components that serve as plans in Soar: (1) sets of instantiated preferences in preference memory[4] serve as instantiated plans for active goals; and (2) sets of variabilized control rules in production memory serve as generalized plans for classes of potential goals. To illustrate this, Figure 6(a) contains a generalized plan for the class of block-stacking problems shown in Figure 6(b). This single-rule generalized plan gets instantiated once for each successive triple of blocks in a desired stack, avoiding the mistake of putting the top block on the second block until it is itself already in place on the third block. Figure 6(c) shows the sequence of steps for a four-block-stacking problem. For each step it shows the problem state, the conjuncts that have not yet been achieved, the operators that have been proposed, and the portion of the instantiated plan — that is, the set of *worst* preferences — that applies at that step. Figure 6(d) shows the actual operator sequence this plan generates.

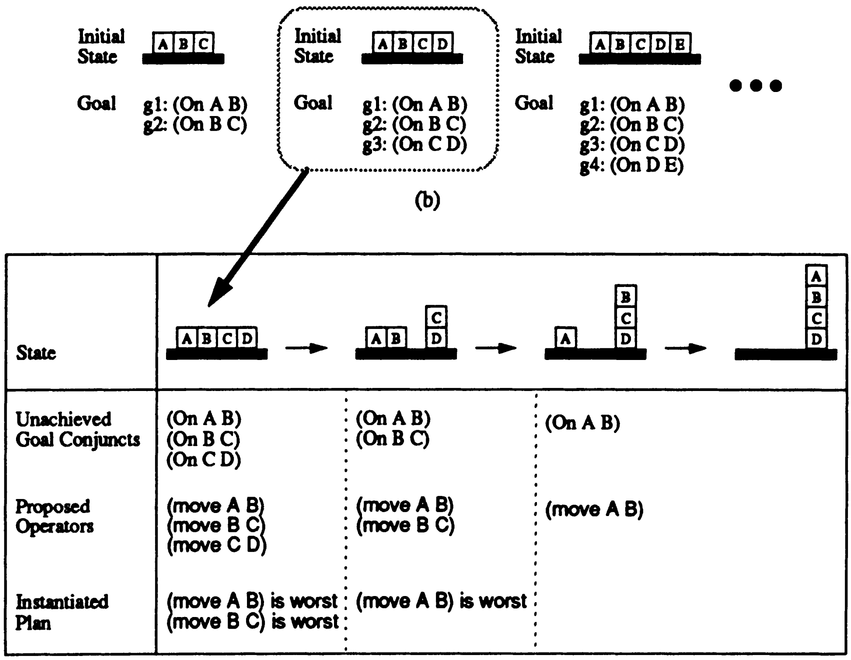
The reason that Soar does not appear to have plans is that they are rarely represented as unitary entities. The generalized plan in Figure 6(a) consists simply of the set — in this case a singleton set — of control rules, out of what would be the entire set of rules in memory, that are relevant to the class of problems in Figure 6(b). For other classes of problems, some of these same rules may be relevant, while others may not be. Likewise, the instantiated plan in Figure 6(c) cuts an unnatural swath through Soar's preference memory. First, it ignores the preferences that might simultaneously be in preference memory for other goals. Second, it contains preferences for an entire sequence of decisions, whereas preference memory focuses on preferences for the currently active decisions; that is, it would only contain the preference subset for one step at a time. The instantiated plan is thus assembled dynamically, and through time, rather than existing as a static unitary structure that can easily be read off by an external observer.

The generalized plan representation combines aspects of three existing formalisms — linear operator sequences, partially-ordered operator sequences, and stimulus-response rules — but it also goes beyond them in

4. In contrast to previous versions, in Soar5 — the current major release — transiently retrieved preferences are maintained in a separate preference memory, rather than in working memory (Laird et al., 1990).

Goal protection can hold
Want a stack of at least three blocks
Neither of the top two blocks (out of the three) are in position
Both of the top two blocks (out of the three) are clear
An operator is proposed to put the top one on the second one
-->
The operator is worst

(a)



(c)

(move C D) → (move B C) → (move A B)

(d)

Figure 6. Goals, plans, and operator sequences: (a) a generalized plan, (b) the class of problems for (a), (c) the sequence of steps for a four-block-stacking problem, (d) the sequence of operators.

several ways. The preference language, in common with linear operator sequences and stimulus-response rules, has an imperative construct (*best*) that allows relatively direct specification of the next action to perform; however, it also goes beyond this to allow, in common with partially-ordered operator sequences, specification of partial order — using binary preferences such as *worse* and *better* — as well as beyond this to operator avoidance (*worst* and *reject*). The use of control rules, in common with stimulus-response rules, provides a fine-grained conditionality and context sensitivity that allows it to easily encode such control structures as conditionals and loops. In addition, the variabilization of the control rules allows a single plan fragment to be instantiated for multiple related decisions.

Planning Methods in Soar

At the problem-solving level, Soar is based on the idea of multiple problem spaces (Newel et al., 1991) — that is, on multiple sets of operators and states, their selection, and the application of operators to states to yield new states — and their interaction through goal-subgoal interfaces. Early work on Soar demonstrated how this organization, in conjunction with small amounts of additional knowledge — structured as *method increments* — could yield a wide range of standard problem-solving methods. Although this included means-ends analysis (MEA) — the primordial planning method — most of the exhibited methods, such as depth-first search and hill-climbing, did not resemble classical planning methods. Thus Soar was conventionally viewed as a search system rather than as a planning system. However, recent work on a Soar-based framework for planning has demonstrated how versions of such standard planning methods as linear, nonlinear,[5] and abstraction (hierarchical) planning can be derived by adding method increments that include core means-ends knowledge about what operators to suggest for consideration, and varying knowledge about how to respond to impasses resulting from precondition failures (Rosenbloom, Lee, & Unruh, 1990).

---

5. The term "nonlinear" has several different meanings in the world of plans. The specific sense intended here is that operators generated in service of different goal conjuncts can be interleaved.

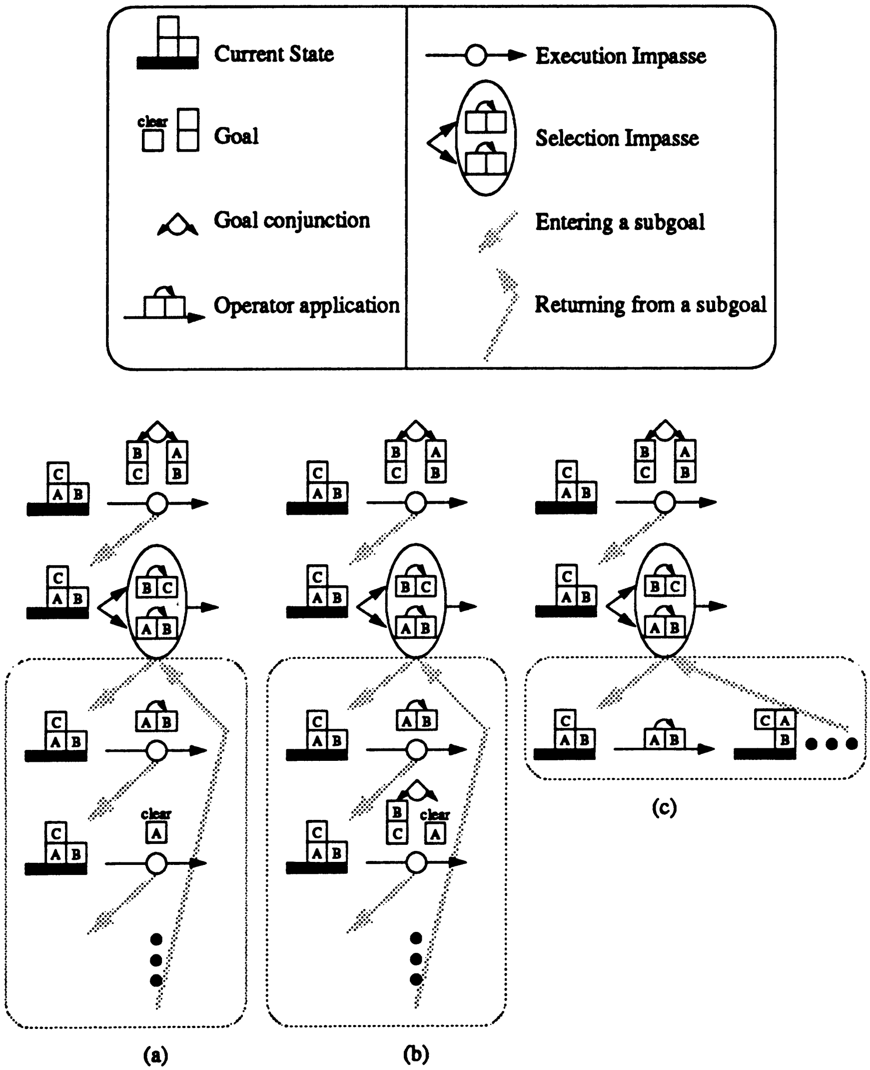


Figure 7. Planning in the blocks world using (a) linear; (b) nonlinear; and (c) abstraction (hierarchical) biases.

Figure 7 provides initial traces of how particular versions of these three forms of planning proceed, in their current Soar implementation, for Sussman's anomaly (in the blocks world).[6] They all start with a high-level operator that is to achieve the entire conjunctive goal — (On B C) and (On A B) — directly from the initial state, and reach an execution impasse if there is no information about how to do this. In response to this impasse, a subgoal is created where means-ends analysis is used to generate the set of candidate operators — (move B C) and (move A B) — that may be able to achieve any of the goal conjuncts. A selection impasse then occurs unless there is information about how to pick among them (or unless only one operator is generated). In this selection impasse, the alternatives are evaluated by simulating their consequences. The simulation begins by selecting one of the alternatives to evaluate — here it is (move A B). Its preconditions are tested and if it is known to be applicable, it is executed. If it is not known to be applicable, what happens next depends on whether or not there is abstraction. With abstraction, the operator is executed anyway and problem solving just continues. In Figure 7(c), for example, operator (move A B) is executed even though block A is not clear. Without abstraction, as in Figure 7(a) and (b), an execution impasse occurs again. In response to this impasse, a new set of goal conjuncts is generated from the operator's unmet preconditions.

The difference between linear and nonlinear planning, at least for these versions, is in how the focus of operator generation shifts from the original set to a set including these new ones. Linear planning follows a stack discipline, where attention shifts completely to these new conjuncts — (Clear A) in this example — stays with them until they are achieved, and then pops back to the original conjunct that led to the impasse. Once the original conjunct is achieved, processing shifts to one of its siblings (if there are any). Nonlinear planning instead shifts to an expanded set of conjuncts that includes the new set plus the original set minus the conjunct that led to the impasse, yielding (Clear A) and (On B C) in this example. At any point in time, an operator can be selected for any of these conjuncts, enabling operator sequences to be interleaved as nec-

6. For comparison purpose, we are showing abstraction in the blocks world. Although we have not actually implemented it in that domain, it has been implemented in several similar domains.

essary (similar to the *casual-commitment* approach to nonlinear planning (Veloso, 1989)). For both planning methods, once the new focus has been determined, planning continues recursively by using means-ends analysis to generate candidate operators for the new set of goal conjuncts.

Although we have so far been referring to these methods as "planning methods", because they are versions of classical methods used in the creation of plans, nothing has yet been said about how they in fact yield plans — that is, sets of either instantiated preferences or generalized control rules. Plans — actually *plan fragments* — are generated whenever operator preferences are created in working memory. This can happen simply by the instantiation of a generalized plan fragment — that is, by the execution of a control rule — or by the returning of a result from an operator-selection subgoal. For example, in Figure 7(a) a best preference is returned from the selection subgoal if the result of evaluating (move A B) is success, whereas a worst preference is returned if the result is failure. These preferences act directly as fragments of a plan for the currently active goals. In addition, whenever a preference is returned as a result of a subgoal, it triggers Soar's chunking process, which creates and stores a control rule that acts as a generalized plan fragment for classes of problems.

Most plan fragments that are not created simply by instantiating generalized plans are generated from projection (that is, lookahead) episodes in subgoals. In projection, one or more domain operators are tried out in simulation to see which ones lead to success or failure. Success engenders *best* preferences and failure engenders *worst* preferences. Thus projection plays an integral role in determining which plans are created. In its turn, what is projected, and what is considered to be success or failure, is determined by the planning method. These relationships are summarized by the following two influence paths.

planning method $\succ$ projection $\succ$ instantiated plan

planning method $\succ$ projection $\succ$ learning $\succ$ generalized plan

Within this framework, planning biases are implemented by altering the planning method, which then — through the influence paths above — determines which plans are created. For example, a protection bias is implemented by altering the planning method to terminate lookahead with failure any time a projected path leads to a protection violation. In comparison to the same planner without this bias, the protection planner will lead to the creation of worst preferences (and negative control rules) which will avoid paths that violate protection. If relative biases are used, it should also be possible to learn control rules that generate binary preferences (such as *better* and *worse*) encoding partial-order information, but this has not yet been investigated (at least in the context of bias and planning).

IMPLEMENTED PLANNING BIASES

The planning biases that we have concentrated on recently are linearity, protection, directness, and abstraction. The first three have all been implemented as options within a single planning system — and will be the focus here — while the latter has been implemented separately (Unruh & Rosenbloom, 1989). The implemented system that combines linearity, protection, and directness is constructed as a nonlinear planner that can optionally employ any of several different bias values along two independent dimensions — *goal flexibility* and *goal protection*. The nonlinear planner is described in Figure 7(b). It uses means-ends analysis on the entire set of goal conjuncts to decide which operators to consider for selection, performs search to decide among the set of considered operators, and generates new goal conjuncts whenever one or more preconditions of the selected operator are not achieved.

The goal-flexibility dimension is shown in Figure 8. It ranges over the planner's degree of flexibility in the pursuit of subgoals for precondition failures, and subsumes the directness and linearity biases. The most restricted point along this spectrum disallows all pursuit of subgoals for precondition failures (Figure 8(a)), yielding a single-level subgoal hierarchy. That is, if an operator has unsatisfied preconditions when the attempt is made to incorporate it into a plan, that plan — actually, any plan incorporating that operator in that role — is rejected. This implements a directness bias because means-ends analysis ensures that operators are only considered if they achieve a goal conjunct,[7] and the

7. Means-ends analysis in general may not guarantee this, but the restricted form of MEA typically used by planning systems — where it is based on the unification

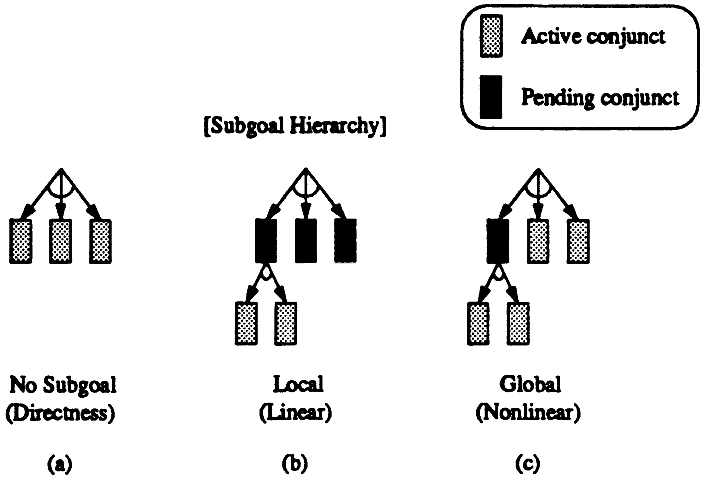


Figure 8. The dimension of goal-flexibility bias.

only goal conjuncts that are allowed are the ones in the initial problem specification, so no operators are allowed in the plan except for those that directly achieve goal conjuncts in the initial problem specification.

The second point along the flexibility dimension allows the local use of subgoals (Figure 8(b)). Here, precondition failures lead to generation of new goal conjuncts, but only a single local set of conjuncts are attended to at any point in time. Initially the local set consists of the conjuncts in the problem specification. However, whenever a selected operator has one or more unmet preconditions, the previous local set is pushed on a stack, and the operator's unmet preconditions become the new local set. When the operator's conditions are satisfied, the stack is popped to return to the previous set. This local focus of attention has two main consequences for the planner. First, it reduces the branching factor of the planner's search — with respect to the nonlinear planner — by restricting the set of operators that the planner can consider at any point in time to just those that may achieve the local conjuncts. Second, it enforces linearity on the resulting plans by restricting the placement of

---

of operator actions with goal conjuncts — does guarantee it.

an operator to within the context of the local conjuncts from which it arose.

The third point along the flexibility dimension allows the global use of subgoals; that is, new goal conjuncts are generated for unmet preconditions, and operators are simultaneously considered for all unsatisfied conjuncts (Figure 8(c)). This is the least restricted version, and enables nonlinear planning by allowing operators for multiple goal conjuncts to be interleaved.

The two points implemented along the goal-protection dimension correspond to full goal protection — that is, no achieved goal conjunct in the plan can be violated — and no goal protection. The main consequence of imposing full goal protection is that the search tree is reduced in size because paths that violate goal protection are cut off before full plans are created.

Figure 9 characterizes a 3×2 set of planning methods derived from these bias dimensions. The most biased planner (P1) is at the top-left corner of the figure. This is a direct goal-protection planner. Although quite restrictive, it is sufficient to solve the block-stacking problem shown in that cell of the figure. The least biased planner (P6) is in the bottom-right corner of the figure. It is a nonlinear planner without goal protection, and is the only planner in the figure capable of generating an optimal solution to the blocks world problem shown in that cell.[8] Between these two extremes, moving up or to the left yields more bias, while moving down or to the right yields less bias. In each of these intermediate cells, the problem shown is one that is just hard enough to require that planner; that is, the problem can be solved optimally by the planner represented by that cell, but not by either the planner to its left or the planner above it.

Tables 1(a-c) show the same 3×2 matrix, but each cell now contains experimental results bearing on the trade-offs between efficiency and completeness for these six planners. These data come from running each planner on three replications of the same set of fifty blocks-world problems, for a total of 150 trials each. Problems are repeated to help average

8. In this domain both P5 and P6 are complete planners in that they can potentially solve every problem (though P5 may not be able to generate an optimal solution). However, in domains with irreversible operators, P6 is the only complete planner.

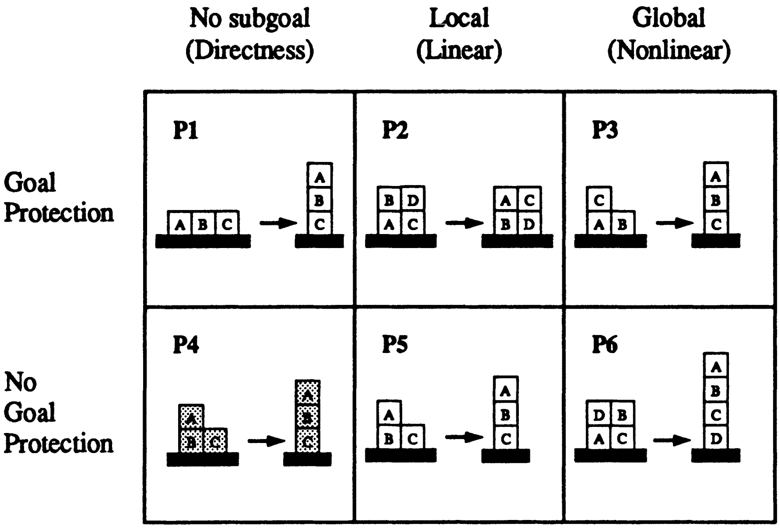


Figure 9. The planning methods generated by the bias dimensions (the bottom-left corner represents an extended blocks world problem where a block that is not clear can be moved, dropping all the blocks above it onto the table).

out the variations caused by nondeterminism in the planners — whenever they reach a decision at which they are indifferent among the set of alternatives, one alternative is picked at random. The first ten problems out of the fifty all involve two blocks and two goal conjuncts. For each ten subsequent problems the maximum number of blocks and goal conjuncts were each increased by one (the last ten problems thus have a maximum of six each). For each problem, an initial state was randomly generated containing between two and the (current) maximum number of blocks. Likewise a set of goal conjuncts was randomly generated that numbered between two and the number of blocks in the initial state. Learning was turned on for each problem, but only within-trial transfer was allowed; that is, rules learned during one problem were not used for other problems. This provides a generalized form of dependency-directed backtracking, but does not get into the issues of across-problem interactions.

Table 1(a) shows the number of problems solvable in principle — that is, if sufficient time is provided — by that cell's planner, plus a label for the problem set that this implicitly defines. Not surprisingly, this shows a monotonic trend between planner bias and scope, from a low of 28 problems for the most restricted planner to a high of 50 problems for the least restricted planner. Tables 1(b) and (c) show the average number of decisions and the average plan length, which should positively correlate, respectively, with planning time and execution time. This data arises from applying each of the six planners to those of the four problem sets defined in Table 1(a) that they can in principle solve. The four problem sets are associated with the four rows within each cell of the tables. The averages in each cell only include the data from the trials that were solved within an a priori limit of 300 decisions. Since 99% of the solvable problems were actually solved within this limit, this includes nearly all of the trials.

The timing results in Table 1(b) show that planning effort is a monotonically decreasing function of the amount of bias along these dimensions. For example, for problem set S1, effort ranged from a low of 16.5 decisions for the no-subgoal (direct) methods to a high of 36.6 decisions for global flexibility (nonlinear planning) without protection. If this trend holds more broadly across other domains, the resulting trade off between efficiency and completeness — efficiency decreases as the bias is relaxed, while completeness increases — implies that context-sensitive bias selection will be critical for finding solutions quickly across broad ranges of problem difficulty.

Plan length in Table 1(c) shows a similar monotonic trend, though there is one reversal when going from linear to nonlinear planning (both without goal protection). The most likely cause of this reversal is that the linear planner's bias is weak enough to allow solutions to be found for all blocks world problems, but strong enough to eliminate optimal solutions for some of the problems; for example, when the shortest plan requires operators in service of different goal conjuncts to be considered before the current goal conjunct is achieved.

	No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
Goal Protection	28 (S1)	45 (S2)	46 (S3)
No Goal Protection	28 (S1)	50 (S4)	50 (S4)

(a) Number of problems solvable in principle.

		No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
	S1	16.5	17.2	17.5
Goal	S2	-	28.9	36.4
Protection	S3	-	-	36.4
	S4	-	-	-
	S1	16.5	23.2	36.6
No	S2	-	38.8	50.7
Goal	S3	-	43.4	54.5
Protection	S4	-	45.1	58.0

(b) Average number of decisions per problem solved.

		No subgoal (Directness)	Local (Linear)	Global (Nonlinear)
	S1	1.7	1.8	1.8
Goal	S2	-	2.8	3.0
Protection	S3	-	-	3.0
	S4	-	-	-
	S1	1.7	2.3	2.5
No	S2	-	3.7	3.6
Goal	S3	-	4.1	4.0
Protection	S4	-	4.3	4.1

(c) Average plan length per problem solved.

Table 1. Results from the six planners on three independent repetitions of fifty randomly generated blocks world problems.

MULTI-METHOD PLANNERS

The ideal planner would be able to solve each problem with a minimum of excess work (at both planning and execution time). However, each of the planners examined in the previous section is either incomplete or performs a significant amount of excess work for some of the problems. Given this, an alternative way to approach the ideal is to construct a multi-method planner that uses, for each problem, the least costly planner that is sufficient for it. Toward this end we have created a set of multi-method planners, each consisting of a sequence of primitive planners. For each multi-method planner, planning starts with the most restricted planner, and falls back on failure to successively more relaxed planners until one is found that is sufficient for the problem. The general approach is similar to how multiple biases have been used in inductive concept formation (Rendell, 1986; Russell & Grosof, 1987; Utgoff, 1986), how preservable constraints are relaxed on failure in FAILSAFE-2 (Bhatnagar & Mostow, 1990), and how rejection (absolute) biases are weakened in (M. Barley, personal communication, 1991); however, there are a number of differences in the details, such as which biases are used, how it is decided to weaken the biases, how much the biases are weakened at one time, etc.

Multi-method planners are implemented via a bias space containing operators that set the bias. Without knowledge about which biases work for a problem, a lookahead search is performed in which the biases are tried out in the sequence specified — in essence the system is projecting on the bias as well as on the domain operators. As soon as a bias is found that works for the problem, the lookahead search is terminated, and that bias is applied in actually solving the problem. Figure 10 shows a trace of multi-method planning for a sequence of three planners: (1) the most restricted planner, P1 (direct goal-protection); (2) an intermediate planner, P3 (nonlinear goal-protection); and (3) the least restricted planner, P6 (nonlinear no-protection). The planner starts by evaluating the planning methods. In the case of Sussman's anomaly, as shown in the example, the evaluation of the direct goal-protection method returns failure because the goal conjunct (On A B) cannot be achieved without generating a new subgoal. Once a method fails, the next most relaxed method — in this example it is the nonlinear goal-protection method —

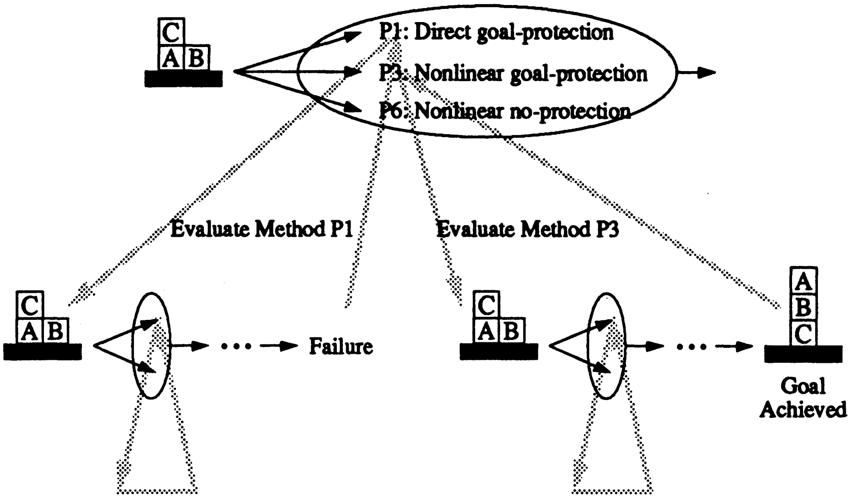


Figure 10. A trace of multi-method planning.

is tried, and so on, until a solution is found.

Table 2 compares the performance of the two single-method planners that are complete for this domain — the linear no-protection planner (P5) and the nonlinear no-protection planner (P6) — versus four multi-method planners. Since all of the multi-method planners in this table contain a complete single-method planner, they are also complete. For each of these six complete planners, the table shows the sequence of primitive planners out of which it is composed, the average number of problems that it actually solved within 300 decisions (averaged over the same set of 150 trials as in the previous section), the average number of decisions for the solved problems, and the average plan length for the solved problems. To aid in comparing the methods, the parenthetical numbers in the last two columns provide the same data for the 43 problems that all six methods solved on all three repetitions. The results on these common problems reveal a monotonic trend whereby adding more planners marginally increases planning time, in exchange for reductions in plan length. In general the linear planners were quicker than the nonlinear planners, while there was no strong pattern for plan lengths.

Planning type		Average number of problems solved	Average number of decisions	Average plan length
Single-method Planning	P5	48.7	45.1 (32.3)	4.3 (3.3)
	P6	48.7	58.0 (44.6)	4.1 (3.2)
Multi-method Planning	P1 - P5	48.3	44.3 (37.9)	3.5 (2.8)
	P1 - P6	48.7	58.5 (46.4)	3.8 (3.0)
	P1 - P2 - P5	46.7	43.3 (40.7)	3.0 (2.8)
	P1 - P3 - P6	47.7	53.3 (46.1)	3.1 (2.7)

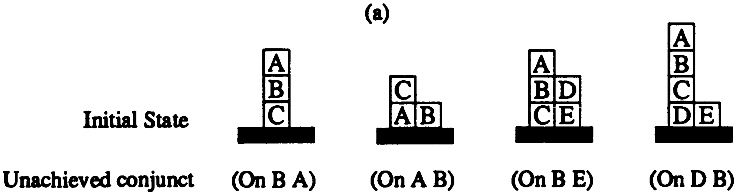
Table 2. Single-method versus multi-method planning.

Ideally the multi-method planners would not only have produced shorter plans, but would also have taken less time to do so. The idea is that problems solvable by more restricted planners should be solved more quickly, while problems requiring less restricted planners should not waste too much extra time trying out the insufficient early planners. The intuition behind this is based on iterative deepening (Korf, 1985). In iterative deepening, a sequence of depth-first searches are performed, each to a greater depth than the previous one. If a solution is found at a shallow depth, the cost of searching to a greater depth is saved. If a solution is not found at a particular depth, a deeper search is performed. The cost of doing the shallower searches is then wasted, but since the deeper search costs at least B times the cost of the shallower search — where B is the branching factor of the search tree — this cost can be relatively quite small. Thus, if the proportion of problems solvable at shallow depths is large enough, and the ratio of costs for successive levels is large enough, there should be a net gain. However, the results in Table 2 show that, at least for these methods and problems, these assumptions are not met. The planning-time balance is instead in favor of the single-method approaches.

One way to further ameliorate the effects of wasting effort on insufficient planners is to use learning, in particular of two sorts. The first sort of learning is about which planners to use for which classes of problems. To the extent that this can be done, the effort wasted in trying inade-

Problem-space is "select-method"
One conjunct is unachieved
Want a stack of at least two blocks
The upper block is not in position
The upper block is not clear
An operator is proposed to use "directness & protection" method
-->
The operator is worst

(a)



(b)

Figure 11. Example of learning which planners to use for which classes of problems: (a) a learned rule to avoid the direct goal-protection planner, (b) a class of problems in which this rule is applicable.

quate methods can be avoided. In our Soar-based implementation, bias selection is structured just as would be any other selection, so this sort of learning can happen automatically by chunking. From an experiment with such learning, Figure 11 shows a rule learned to avoid using the most restricted method — that is, direct goal-protection — under specific circumstances where there is only one goal conjunct but (at least) two blocks must be moved to achieve it. This rule was learned during the first problem and used in three later problems to avoid even trying this method.

The second sort of learning is within-planner learning that can transfer across planners (possibly for the same problem). If a projection is performed within one planner, and the results of the projection depend only on aspects of the planner that are shared by a second planner, then it should not be necessary to repeat that projection when the second planner is tried. For example, the rule in Figure 6(a) is learned from a plan violating goal protection in the direct goal-protection planner and transfers to the nonlinear goal-protection planner, where it prevents the planner from reprojecting along paths that violate goal protection.

Though we have examined instances of both of these forms of learning in the context of multi-method planning, no systematic study has yet been made of their effectiveness or of whether issues of overgeneralization and/or undergeneralization will prove troublesome. Future work should include rerunning the experiments summarized in Table 2 with both of these forms of learning enabled.

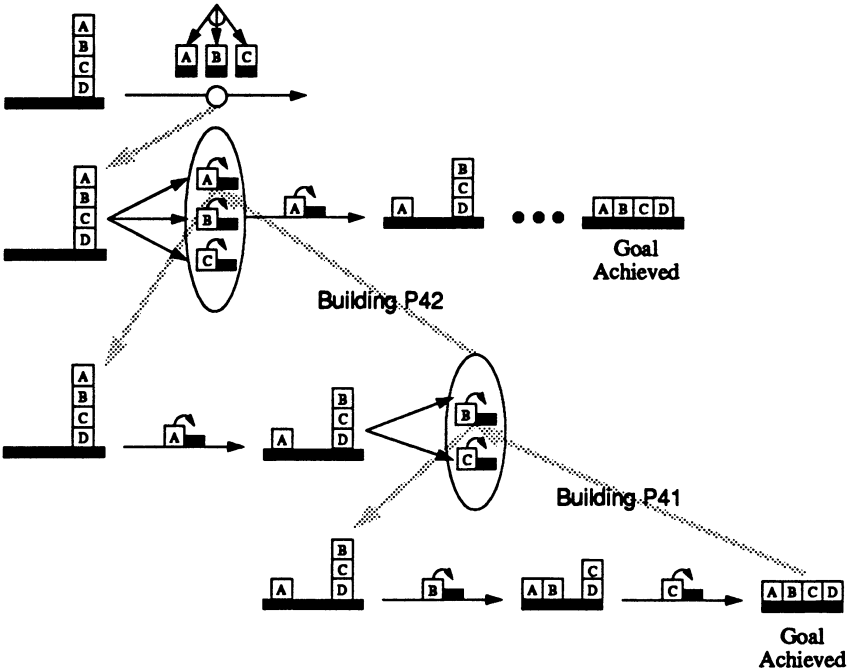
Another potential way to improve the performance of multi-method planners is to reduce the granularity at which the individual planning methods are selected and used. If there are a significant number of problems where most of the subgoals are solvable by a very cheap method (such as directness) while the remainder of the problem requires a more complex method (such as linear planning), then making an independent bias selection each time the planner recurs on a set of subgoals may allow focused reductions in planning time and plan length. There would be increased overhead because of the extra decisions, but that may be more than compensated for by the use of simpler methods. This is all quite speculative for now, but does provide an interesting future area for investigation.

BIAS IN EXPLANATION-BASED LEARNING

As defined in the introduction, the bias in concept learning is anything other than strict consistency with the observed training instances that influences which generalization is chosen. Thus, when an explanation-based method is used for concept learning, its bias includes the entire set of inputs exclusive of the training example — that is, the domain theory, the goal concept, and the operationality criterion (Mitchell, Keller, & Kedar-Cabelli, 1986) — plus any other factors that influence which explanation is used and which definition/rule is extracted from the explanation. Each of these factors has been varied in at least one recent research effort in service of increasing the utility of the rules acquired by EBL: restrictions on domain theory expressiveness (Tambe, Newell, & Rosenbloom, 1990);[9] variations in target (goal) concepts for acquiring control knowledge (Minton et al., 1989); variations in the operationality criterion (Braverman & Russell, 1988; Letovsky, 1990; Segre, 1987); explanation selection based on criteria such as coverage or (non)recursiveness (Cohen, 1990; Etzioni, 1990); and postprocessing via a range of deductive and inductive transformations (Chase et al., 1989; Cohen, 1990; Flann & Dietterich, 1989; Minton, 1988; Shavlik, 1989).

In the framework of this chapter, explanation-based learning of plans is performed over the planner's projection process: the elements to be explained are the preferences generated during projection, and the explanations are the traces of the projections that led to the preferences. Thus, if the planner's bias is reflected in an altered planning method, which in turn yields an altered projector, then the planner's bias can indirectly induce a bias in the resulting EBL process. Directness provides a simple example of this. Figure 12(a) shows a path projected without directness, by the nonlinear planner, for a simple four-block-unstacking problem. This projection proceeds through multiple selection impasses until the problem is successfully solved. As shown in Figure 12(b), this results in a pair of positive control rules, one for each correct decision on the solution path. These rules are relatively specialized, because each must encapsulate the entire explanation for why a particular operator will eventually lead to success. In larger problems these explanations get even larger, and the rules end up being even more specialized. Figure 13(a) shows a path projected with directness, for the same block-unstacking problem. In contrast to the previous case, this projection is terminated with failure as soon as the non-applicable operator (move B Table) is selected. The explanation for this failure is quite short — based as it is on the explicit assumption that directness can hold and on the failure of the first selected operator to be applicable — yielding the negative control rule in Figure 13(b). As it turns out, this single rule is general enough to handle the entire problem, by removing from consideration all operators that attempt to move unclear blocks onto the table. The bias in this case has thus yielded faster planning and learning

9. The use of "low belief" or "overgeneral" domain theories for knowledge level learning is another form of domain theory variation that provides a qualitative improvement in utility — from symbol level learning to knowledge level learning (Flann & Dietterich, 1989; Rosenbloom & Aasman, 1990).



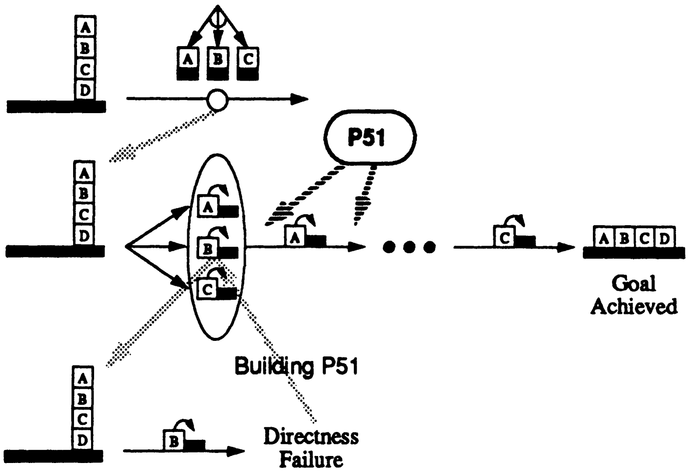
(a)

P41: Want two blocks on the table
One block is on top of the other
The top block is clear
An operator is proposed to put the top one on the table
-->
The operator is best

P42: Want three blocks on the table
These blocks are stacked one on top of another
The top block is clear
An operator is proposed to put the top one on the table
-->
The operator is best

(b)

Figure 12. Four block unstacking with nonlinear planning: (a) a projected path, (b) learned rules.



(a)

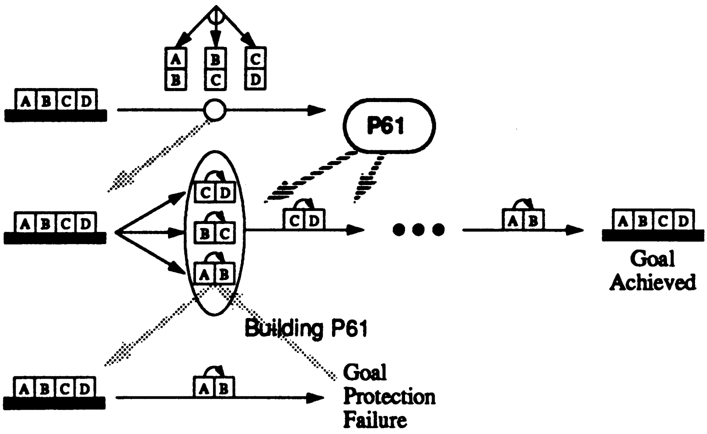
P51: Directness can hold
Want a block on the table
That block is not on the table
That block is not clear
An operator is proposed to put that one on the table
-->
The operator is worst

(b)

Figure 13. Four block unstacking with directness: (a) a projected path, (b) a learned rule.

— because of shorter projections and explanations — and has resulted in the acquisition of fewer, more general rules.

Implicit in this example is one approach to producing *generalization to N* (Boström, 1990; Cohen, 1988; Shavlik, 1989; Subramanian & Feldman, 1990), where a plan learned for a problem of a particular size can transfer to solve problems with the same structure but of arbitrary size. Without directness, the control rules are specific to particular numbers of blocks, and thus can only be used to directly solve terminal subregions



(a)

P61: Goal protection can hold
Want a stack of at least three blocks
Neither of the top two blocks (out of the three) are in position
Both of the top two blocks (out of the three) are clear
An operator is proposed to put the top one on the second one
-->
The operator is worst

(b)

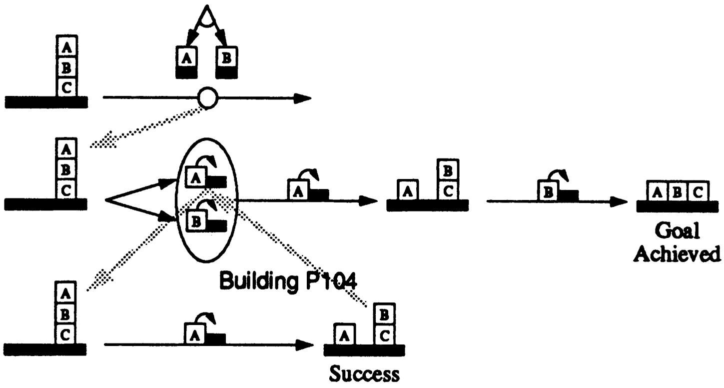
Figure 14. Four block stacking with protection: (a) a projected path, (b) a learned rule.

of larger problems. However, with directness, a single rule is learned that removes from consideration at each decision all operators that move unclear blocks to the table, *no matter how many unclear blocks there are.* This idea can be applied to other problems and biases as well. Figure 14(a), for example, shows a path projected with protection for a four-block-stacking problem. As with the directness bias in block unstacking, a protection bias leads here to learning a single negative rule (Figure 14(b)) that can be applied to stacking problems of arbitrary size.

A third type of bias that can also induce generalization to N is *complete* protection. Complete protection is a variant on goal protection that provides a very strong bias by not only protecting established goals, but also protecting established operator sequences. That is, it disallows any backtracking on operator selection, thus letting projection be terminated with success whenever an operator is selected, rather than waiting until the entire problem has been solved. As with the directness example, projection is terminated here after the first operator is selected (Figure 15(a)). However, in this case it is terminated with success as soon as the top block is moved to the table. The explanation for this success depends only on the explicit assumption of complete protection and on the fact that the operator was successfully applied, so a relatively general, positive control rule is learned (Figure 15(b)). Although this is a positive rule, it also turns out to produce generalization to N, but now by always specifying that the one clear block that is not already on the table — if it were already on the table, there would be no active goal conjunct for it — should be moved to the table. The resulting rule can transfer to any number of iterations, as shown in Figure 15(c).

The key to producing generalization to N with these biases is that they enable learning from non-iterative paths — in this way it is similar to Etzioni's (1990) work on restricting EBL to learn from only non-recursive paths. In the directness and protection cases, the success paths are iterative, but (negative) rules can instead be learned from non-iterative failure paths. In the complete-protection case, learning occurs from a fragment of the success path that corresponds to just a single cycle of iteration. In both cases, the resulting rules can transfer to any number of iterations.

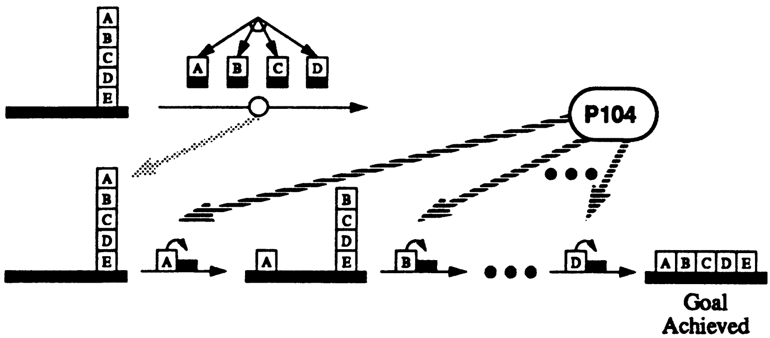
An even closer relationship to Etzioni's work could potentially be achieved by adding a nonrecursiveness bias to the planner. If this were added as an absolute bias, it would terminate projection with failure along any path that recurred. This would restrict learning to nonrecursive portions of projections, but would also go seriously beyond this to eliminate recursive plans from the space. Etzioni dealt with this by distinguishing those projections used for learning from those used for planning, and then only using the bias during learning. An alternative approach is to weaken nonrecursiveness by making it a relative bias. If all of the nonrecursive paths are projected without yielding enough preferences to generate an unambiguous plan, then it should still be possible to go back and project along the recursive paths. The result should



(a)

P104: Complete protection holds
Want a block on the table
That block is not on the table
That block is clear
Operator is proposed to put that one on the table
-->
The operator is best

(b)



(c)

Figure 15. Four block unstacking with complete protection: (a) a projected path, (b) a learned rule, (c) transfer of the learned rule.

be an overall preference for nonrecursive plans, and for recursive plans learned from nonrecursive projections, over recursive plans learned from recursive projections. Investigating such a relative nonrecursiveness bias in the Soar-based planner is an interesting possibility for future work.

Another example of engendering a useful bias on EBL by biasing the planning can be found in the work on learning from abstract planning (Unruh & Rosenbloom, 1989; Knoblock, Minton, & Etzioni, 1991). In this work, projections are performed with abstracted operator definitions rather than with the full operator definitions. The resulting abstract projections tend to be shorter and simpler than would be the comparable projections with unabstracted operators; so when rules are learned from these projections, they also tend to be shorter and simpler, and thus more general.

Given the evidence that planning biases can induce interesting and useful biases in EBL, and that in so doing the biases can assist both planning and learning, it is an interesting question to ask whether any other approaches to biasing EBL — such as post-hoc rule modification — are needed. Although it is premature to answer this question at this point, it is worth noting that the ultimate answer will depend on the scope of learning biases that can be generated in this fashion, and whether achievement of these learning biases requires distorting the planner to such an extent that it cannot properly achieve its task.

CONCLUSION

In this chapter we have taken seriously the notion of bias as a means of characterizing variations among planners. We hypothesized that this would yield three benefits: (1) help organize and understand many of the concepts in planning; (2) reduce the computational requirements of planning; and (3) induce effective biases in learning. On the first benefit, though it has not yet led to a complete theory or taxonomy of planning methods, it has led to the development of several orthogonal bias dimensions which provide fragments of organization over the space of methods. Six planners — defined by the cross-product of two bias dimensions — have been implemented in Soar as variations on a single core planner. Further work is required to identify the remaining biases that underlie effective planning methods, and to build a unified planner that can optionally use arbitrary subsets of them.

On the second benefit, initial experiments with the six planners suggest (mostly) monotonic trade-offs between completeness and efficiency as the bias dimensions are traversed from least to most restrictive. In an attempt to move off of this trade-off curve, a set of multi-method planners were constructed from sequences of increasingly less biased planners. When the correct planner is not known a priori, a search is performed starting at the most restricted planner until a sufficient one is found. An experiment comparing these multi-method planners with the two complete single-method planners in the blocks world provided mixed results: reduced plan length no longer needed to be sacrificed for completeness, but reductions in plan length were accompanied by increases in planning time. Two learning strategies were presented as potential ways of further reducing the planning time required by the multi-method planners: acquisition of control rules that transfer among planners, enabling searches with more restricted planners to assist search with less restricted ones; and acquisition of rules that help select appropriate planners. These possibilities need to be investigated further, in conjunction with a deeper understanding of the entire set of trade-offs and experimentation in more realistic task domains. In addition, the idea of allowing a new method to be selected whenever the planner recurs on a new set of subgoals also needs to be examined.

On the third benefit, the effects of changes in planning bias on bias in explanation-based learning were investigated with a case study in generalization to N. Depending on the exact biases used, control rules were learned that either: (1) did not provide generalization to N (for nonlinear planning), provided it by eliminating all but the correct iterative option (for goal protection and directness), or provided it directly by specifying the correct iterative option (for complete protection). Although this is encouraging, considerable future work is still needed in evaluating the impact of the full span of planning biases on learning, and evaluating whether this provides a sufficient set of biases on learning.

REFERENCES

Bennett, J. S., & Dietterich, T. G. (1986). *The test incorporation hypothesis and the weak methods* (Technical Report 86-30-4). Department of Computer Science, Oregon State University.

Bhatnagar, N., & Mostow, J. (1990). Adaptive search by explanation-based learning of heuristic censors. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 895-901). Boston, MA: MIT Press.

Boström, H. (1990). Generalizing the order of goals as an approach to generalizing numbers. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 260-267). Austin, TX: Morgan Kaufmann.

Braverman, M. S., & Russell, S. J. (1988). Boundaries of operationality. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 221-234). Ann Arbor, MI: Morgan Kaufmann.

Chase, M. P., Zweben, M., Piazza, R. L., Burger, J. D., Maglio, P. P., & Hirsh, H. (1989). Approximating learned search control knowledge. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 218-220). Ithaca, NY: Morgan Kaufmann.

Cohen, W. W. (1988). Generalizing number and learning from multiple examples in explanation based learning. *Proceedings of the Fifth International Conference on Machine Learning* (pp. 256-269). Ann Arbor, MI: Morgan Kaufmann.

Cohen, W. W. (1990). Learning approximate control rules of high utility. *Proceedings of the Seventh International Conference on Machine Learning* (pp. 29-33). Austin, TX: Morgan Kaufmann.

Etzioni, O. (1990). Why Prodigy/EBL works. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 916-922). Boston, MA: MIT Press.

Flann, N. S., & Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187-226.

Gratch, J. M., & DeJong, G. F. (1990). A framework for evaluating search control strategies. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control* (pp. 337-347). San

Diego, CA: Morgan Kaufmann.

Knoblock, C. A., Minton, S., & Etzioni, O. (1991). Integrating abstraction and explanation-based learning in PRODIGY. *Proceedings of the Ninth National Conference on Artificial Intelligence* (pp. 541-546). Anaheim, CA: MIT Press.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence, 27*, 97-109.

Laird, J. E., Congdon, C. B., Altmann, E., & Swedlow, K. R. (1990). *Soar user's manual: version 5.2* (Technical Report CMU-CS-90-179). School of Computer Science, Carnegie Mellon University.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*, 1-64.

Letovsky, S. (1990). Operationality criteria for recursive predicates. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 936-941). Boston, MA: MIT Press.

Minton, S. (1988). *Learning search control knowledge: An explanation-based approach.* Boston, MA: Kluwer Academic Publishers.

Minton, S. (1990). Quantitative results concerning the utility of explanation based learning. *Artificial Intelligence, 42*, 363-391.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., Etzioni, O., & Gill, Y. (1989). Explanation-based learning: A problem solving perspective. *Artificial Intelligence, 40*, 63-118.

Mitchell, T. M. (1980). *The need for biases in learning generalizations* (Technical Report CBM-TR-117). Department of Computer Science, Rutgers University.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation based generalization: A unifying view. *Machine Learning, 1*, 47-80.

Newell, A., Yost, G. R., Laird, J. E., Rosenbloom, P. S., & Altmann, E. (1991). Formulating the problem space computational model. In R. F. Rashid (Ed.) *Carnegie Mellon computer science: A 25 year commemorative.* New York, NY: ACM Press.

Rendell, L. (1986). A general framework for induction and a study of selective induction. *Machine Learning, 1*, 177-226.

Rosenbloom, P. S., & Aasman, J. (1990). Knowledge level and inductive uses of chunking (EBL). *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 821-827). Boston, MA: MIT Press.

Rosenbloom, P. S., & Laird, J. E. (1986). Mapping explanation-based generalization onto Soar. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 561-567). Philadelphia, PA: MIT Press.

Rosenbloom, P. S., Laird, J. E., & Newell, A. (in press). *The Soar papers: Research on integrated intelligence.* Cambridge, MA: MIT Press.

Rosenbloom, P. S., Laird, J. E., Newell, A., & McCarl, R. (1991). A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence, 47*, 289-325.

Rosenbloom, P. S., Lee, S., & Unruh, A. (1990). Responding to impasses in memory-driven behavior: A framework for planning. *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling, and Control* (pp. 181-191). San Diego, CA: Morgan Kaufmann.

Russell, S. J., & Grosof, B. N. (1987). A declarative approach to bias in concept learning. *Proceedings of the Sixth National Conference on Artificial Intelligence* (pp. 505-510). Seattle, WA: MIT Press.

Segre, A. M. (1987). On the operationality/generality trade-off in explanation based learning. *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 242-248). Milan, Italy: Morgan Kaufmann.

Shavlik, J. W. (1989). Acquiring recursive concepts with explanation-based learning. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 688-693). Detroit, MI: Morgan Kaufmann.

Subramanian, D., & Feldman, R. (1990). The utility of EBL in recursive domain theories. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 942-949). Boston, MA: MIT Press.

Tambe, M., Newell, A., & Rosenbloom, P. S. (1990). The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning, 5*, 299-348.

Unruh, A., & Rosenbloom, P. S. (1989). Abstraction in problem solving and learning. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (pp. 681-687). Detroit, MI: Morgan Kaufmann.

Utgoff, P. E. (1986). Shift of bias for inductive concept learning. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.) *Machine Learning: An Artificial Intelligence Approach, Vol. II.* Los Altos, CA: Morgan Kaufmann.

Veloso, M. (1989). *Nonlinear problem solving using intelligent casual-commitment* (Technical Report CMU-CS-89-210). School of Computer Science, Carnegie Mellon University.

Knowledge Acquisition and Natural Language Processing*

Robert Wilensky
Division of Computer Science
University of California, Berkeley 94720

Knowledge acquisition and natural language processing are two fields of Artificial Intelligence that have much to offer each other. Natural language requires such large amounts of knowledge that it will probably be necessary to automate the acquisition process for this field to achieve its goals. Machine learning has focused on incremental improvements of performance; but the acquisition of knowledge is probably more of a key bottleneck for building intelligent systems. Huge volumes of knowledge are available now, in machine readable form, if only we could understand how to use it. Natural language processing technology holds the key to this storehouse.

Several quite different approaches to knowledge acquisition involving natural language are being developed. Using natural language interaction to acquire domain facts and linguistic facts has been shown to be possible, but tedious. The potential for knowledge acquisition by reading is very large, but realizing it is still a long-term research goal. Some newer techniques for acquiring vocabulary from a machine readable dictionary seem to have definite, if limited, usefulness. Using analogical reasoning to infer extended word senses from context seems both practically promising and of much theoretical interest. Finally, the possibility of using heuristic natural language techniques for extracting information from text corpora is beginning to be explored. This approach holds the possibility of extracting a large amount of ordinary knowledge in the immediate future.

---

*This research was sponsored by the Office of Naval Research, under contract N00014-89-J-3205, and by the Defense Advanced Research Projects Agency (DoD), monitored by Space and Naval Warfare Systems Command under Contract N00039-88-C-0292.

INTRODUCTION

Knowledge acquisition and natural language processing are two fields of Artificial Intelligence that are the focus of much recent activity. While the two disciplines have an entirely different character, they have begun to come into contact, to their mutual benefit. It is the focus of this paper to examine some of the recent work at this intersection, and discuss the prospects it may offer us.

There are several reasons why it is natural for such seemingly disparate areas to have grown into contact. Knowledge-based systems derive their power from the knowledge they possess. Since knowledge is needed in large quantities for many intelligent tasks, a key problem is acquiring this knowledge. Humans acquire their knowledge in many ways, but one particularly significant class of means involves the use of natural language. We might ask someone for some information we seek; we may look for information in a book, article, or manual; or we might attend a lecture. It is natural, therefore, to see if such a useful means of knowledge acquisition for humans can be exploited by knowledge-based systems.

Natural language processing itself is a knowledge-intensive task. Natural language users presume a great deal of shared general knowledge, along with knowledge of the domain of discourse. Indeed, the hardest problems in building natural language systems are not so much problems of language per se, but problems about how to use our non-linguistic knowledge to make sense out of what is said. For example, the utterance "I saw her duck" would have rather different interpretations, depending upon whether it was preceded by the question "Did Mary get out of the way?" or "Have any of Mary's aquatic fowl been this way?" While our knowledge of the grammar and lexicon of the language can inform us what the *possible* interpretations of the utterance may be, only our knowledge of the world can tell us what the *reasonable* interpretation is.

Thus, the need for general knowledge acquisition is acute for natural language processing systems as well as for other kinds of knowledge-based systems. But natural language processing systems require specific knowledge about language as well. The most significant body of knowledge about language pertains to the lexicon. The lexicon comprises thousands of words, most of which have numerous senses and grammatical properties of which a natural language processing system requires mastery. Linguistic knowledge is of a rather specialized sort, so it is natural to largely separate our discussion into two parts, the use of natural language processing for acquisition of knowledge about the world or about a domain, and the acquisition of knowledge about natural language itself.

INTERACTIVE KNOWLEDGE ACQUISITION TECHNIQUES

We begin, however, with an approach that is applicable to both types of knowledge. This is to allow a knowledgable user to supply each fact by interacting with the system in natural language. An example of such a mechanism is one we developed as a component of our UNIX Consultant (UC) system (Wilensky et al. 1984, Wilensky et al. 1988). UC is an experimental program that converses in natural language with naive users to help answer their questions about the UNIX operating system. A knowledge acquisition system, called UCTeacher, lets a knowledgable UNIX user inform the system of additional facts via an extended discourse. In the following examples, the prompt 'UCT>' distinguishes an interaction with UCTeacher from one with UC proper:

(1)

```
UCT> There is a new command.
What is its name?
UCT> lpr
What is it used for?
UCT> It is used to print files.
Where does the output go?
UCT> the lineprinter
You mean that typing 'lpr filename' will print the
file filename on the lineprinter.
UCT> Yes.
```

(2)

```
UCT> There is a new command.
What is its name?
UCT> sort
What is it used for?
UCT> It is used to sort files.
You mean that typing 'sort filename' will sort the
file filename.
UCT> Yes.
```

(3)

```
UCT> Use the rm command to delete files.
You mean that typing 'rm filename' will delete the
file named filename.
UCT> Yes.
```

The output of each interaction is a set of statements in a knowledge representation language used to update UC's knowledge base about UNIX. Thus, subsequently, UC can use the acquired information to give a correct response to a user's query:

```
# How can I print a file?
Use lpr.
For example, to print the file named foo, type
'lpr foo'.
```

UCTeacher has a number of interesting properties, and a UCTeacher-style system has a niche in the knowledge acquisition phylum. However, it has some significant limitations. For large amounts of domain knowledge, the manner of entry is too tedious; for lexical knowledge and for world knowledge, the required knowledge is not only voluminous, but much of it may only be known implicitly by the user. Thus, however useful such approaches may be, more automated methods are highly desirable.

One more automatic approach to acquisition of domain knowledge is to learn new facts by reading. In the case of the UNIX domain, information can be extracted by reading the on-line manuals. We succeeded in developing a text understander, called MANDI, that can perform this task to some degree. The problem with this approach is that reading manuals involves most of the very difficult problems associated with natural language understanding, although some of these are mitigated by the restricted domain. For example, consider the following sentence from the UNIX manual page for ''uptime'':

> Uptime prints the current time.

Two interesting inferences are required to understand the content of this simple sentence. One is the interpretive inference that running the command ''uptime'' causes the current time to be printed. MANDI determines that this is the case by a combination of knowledge about English (that sentences with a non-agentive subject but agentive verbs can have an interpretation in which subject undergoing some process is the cause of the event described) and knowledge about UNIX (that the process a UNIX command is likely to undergo is that of being run).

The second inference is that ''the current time'' refers to the time at which the command is run, rather than, say, the time of reading the manual. This inference is based on an extension of a general algorithm for text inference we developed previously (Norvig 1987) so that it can exploit knowledge about typical user goals. That is, the preferred reading is preferred because knowing what time it is at any given time is known to be a reasonable goal of UNIX users, while knowing what time a user read the manual page is not.

MANDI differs from other text understanders in that it takes greater advantage of the structure of the texts. Manual pages, and expository texts in general, make use of rhetorical devices to present information. We call these ''text point constructions'', as they relate in a regular format certain kinds of information typically of interest to a user. For example, the entry for ''mv'' (the UNIX renaming command) states unequivocally that ''mv moves (changes the name of) file1 to file2''. Later on, we are informed that a quite different interpretation applies if one of the arguments is a directory. This presentation corresponds to a text point construction in which information of the prototypical case is presented as if it were universal, with exceptions following.

In addition to taking advantage of the special discourse structure apparent in manual pages, other facts make this problem different from the general natural language understanding problem. To a significant degree, the circumscription of the domain to UNIX lessens the problem of general world knowledge, and eases the burden of lexical disambiguation. However, these mitigating circumstances do not resolve all intrinsic difficulties. Problems of reference still remain; also, the nature of discourse structure is still poorly understood.

Another difficulty with this approach is that, even for humans, reading a fact is not the same as learning the encoded information. This is especially true for something complex, like operating systems. For example, consider the following sentence from the BSD4.2 UNIX manual page for the command "ps":

> All output formats include ... the process id PID, control terminal of the process TT, cpu time used by the process TIME ... , the state STAT of the process, and an indication of the COMMAND which is running.

True to the tradition of manual writing, this text is impenetrable to most readers. However, consider interpreting the text having run the command:

```
% ps
  PID TT STAT  TIME COMMAND
  557 co IW    0:00 /bin/sh /usr/sww/X11/bin/startx
  559 co S    12:46 X :0
  .
  .
  .
```

It is now possible to interpret the enigmatic capitalized text present in the manual as referring to headings produced by running the command.

The moral is consistent with much of our other experience: Many stated truths cannot be appreciated without practice. Combining reading with practice in a machine learning system is currently a completely unexplored area, one we think offers considerable promise.

Thus, the approach underlying MANDI is fundamental and long term. It may ultimately provide a means to acquire very large quantities of knowledge, but to do so involves addressing difficult basic issues. Natural language is still, after all, an "AI-complete" problem. Below we discuss other approaches that may have greater payoff in the near term.

INFERRING WORD SENSES FROM CONTEXT

We have described a method of obtaining facts about word meanings by interacting with a knowledgable user. Another approach is to infer word meanings from context. For example, Selfridge (1982) models acquisition of language, including lexical knowledge, in children, and Granger (1977) attempts to hypothesize word meanings from the use of a word in context. Such systems attempt to infer word meanings from the semantic constraints imposed on words by the utterances in which they are found, and from the world knowledge about the situations to which the words are being applied.

This approach does not appear very promising, however. The necessary information is simply not present. If a word-inferring system hears that "A car struck an abutment", for example, and is not familiar with the final word, it might infer that an abutment is a physical thing, but it cannot infer much more. Such superficial information is generally available from other sources, such as machine readable dictionaries, as we will discuss further below. Having more instances of the use of a word can help overcome such "single instance generalization" type of difficulties. But difficult problems remain to be solved to exploit such information. For example, we would need to know that two uses involve the same sense in order to attempt a generalization process.

One way to ameliorate this situation is to find additional information about word meanings in the form of knowledge about other, related word meanings. Such an approach is embodied in MIDAS (Metaphor Interpretation, Denotation, and Acquisition System, Martin (1988)), a system that learns metaphorical word senses extensions. While we might think of metaphor as something used predominantly in poetic language, linguists such as Reddy (1979) and Lakoff and Johnson (1980) point out that ordinary senses of words have what we might call a metaphoric motivation. For example, when we think of a word like "give", the sense involving transfer of possession probably comes to mind. This is the *central* sense of this term. But "give" has many other uses. For example, when we say "Jan gave Pat a cold", we mean that Jan infected Pat with a cold from which Jan was suffering. If only "give" worked this way, we might dismiss the oddity as an isolated idiom. But people can also "have colds", "get the flu", "catch pneumonia", and so forth.

In MIDAS, "conventional metaphors" are postulated to capture generalizations that can be made about such utterances. Conventional metaphors comprise representations of mappings from one domain (the source) to another (the target). Words whose senses are metaphoric in nature are represented as being *motivated* by particular metaphors. For example, the sense of "have" that is the same as that of "being infected with" is represented as motivated by a metaphor we might call "being-infected-is-possessing". This metaphor is represented as an object in the representation system connecting the central, possession sense of "have" with the concept "being infect with". Similarly, "give" has a sense that is related to one of "infect"; this sense would be represented as motivated by a metaphor we might call "infecting-is-giving". Moreover, this metaphor is represented as extending the first metaphor. I.e., (central) giving results in (central) having, just as infecting results in being infected, and the metaphor that allows being infected to be expressed as having also allows infecting to be expressed as giving.

For production and understanding, these word senses are treated just like any other, with no appeal to metaphoric motivation presumed. However, suppose a new use is encountered by the system, say, "John got the flu from Mary." Suppose further, for the sake of this example, that the system has previously encountered only expressions like "give a cold" and "have a cold", so that its representation of the metaphors underlying these expressions is restricted to colds rather than diseases, and that the use of "get" in this domain is new to the system. Then both the available sense of "get", along with the metaphors involving diseases and possession, are brought to bear to hypothesize the sense that might be in play. In MIDAS, this hypothesis is generated by two kinds of lexical extension processes, *core extension* and *similarity extension*. Understanding "get a cold" given an appropriate prior metaphoric understanding of "give a cold" involves core extension, as the core metaphor "a cold is a possession" is extended to the "getting" concept (i.e., a new sense for "get" is hypothesized which metaphorically extends the central sense); understanding ''get the flu'' given an understanding of ''get a cold'' involves similarity extension, as the generalization about a role in the metaphoric structure must be extended from colds to diseases in general. Understanding ''get the flu'' given an understanding of ''give a cold'' involves both kinds of extension.

Upon encountering an unfamiliar word sense, MIDAS examines known senses of the word and of other words in an attempt to hypothesize a sense applicable to the given usage. In the following example, MIDAS, running in conjunction with UC, encounters the phrase ''kill a process''. Here the knowledge base does not have a sense of kill directly applicable to this utterance. Therefore, MIDAS tries to find a related sense and use it to hypothesize a new sense applicable to the current usage. In this case, MIDAS knows the sense underlying ''kill a conversation'', and is able to extend that one to apply to the input. The interpretation resulting from this application is then used as the basis for UC's response:

```
#  How can I kill a process?

No valid interpretations.
Attempting to extend existing metaphor.
Searching for related known metaphors.
Metaphors found:
     Kill-Conversation Kill-Delete-Line
     Kill-Sports-Defeat

Selecting metaphor Kill-Conversation to extend from.
Attempting a similarity extension inference.
Extending similar metaphor Kill-Conversation
    with target concept Terminate-Conversation.
Abstracting Terminate-Conversation
    to ancestor concept

Creating new metaphor:
  Mapping source concept Killing
      to target concept Terminate-Computer-Process
  Mapping source role killer
      to target role c-proc-termer.
```

```
  Mapping source role kill-victim
      to target role c-proc-termed.

Calling UC:

You can kill a computer process by typing ^c to the
shell.
```

In this example, the utterance cannot be interpreted because no known sense of "kill" is applicable to processes. Thus, MIDAS is called in an attempted to see if a sense extension is possible. MIDAS first retrieves a number of metaphors related to the input; of these, "Kill-Conversation" is chosen as most applicable. A simple similarity extension is attempted, resulting in a proposed "Terminate-Computer-Process" metaphor for interpretation of the input. The interpretation thus provided is passed along to UC, which can answer this question. Meanwhile, the new, metaphoric sense is incorporated into UC's knowledge base, which allows UC's language generator to use the same terminology in encoding the answer.

The following are representative examples of the kinds of generalizations MIDAS is capable of making:

Given Known Metaphor	System Can Learn
enter LISP	Exist LISP, Enter mail
kill a conversation	kill a process
kill a process	process died
open a file	close a file
give a cold	have a cold, get a cold, give the flu, give an idea

Is this approach something specific to metaphor, or is there a more general lesson to be learned? In my opinion, MIDAS works not so much because it is dealing with metaphor per se, but because it is exploiting a kind of *lexical subregularity*. By subregularity, I mean any phenomenon that is systematic but not predictable. To qualify as a useful lexical subregularity, there must be a regularity that holds between a number of different examples; at the same time, the regularity does not predictably hold wherever it might. The systematic metaphoric structuring alluded to above comprises one important class of subregularities, since they usually apply to many lexical items, but not to all items to which they might. For example, while we have the various extension of possession verbs to mean infecting, as discussed above, we do not hear expressions like "receive a cold", which might equally well be predicted.

The question arises as to just what kinds of subregularities there might be, in addition to metaphoric ones, and how they might be exploited in general. Norvig and Lakoff (1987) offer six types of links between polysemous (i.e., closely related) word senses in what they call *lexical network theory*. Links in a lexical network map senses to one another, starting from some central sense and encoding some minimal variation. However, we have uncovered what seems to be many, many more links than these, or at least, many subcases which are necessary to distinguish.

Rather than give a list of the several dozen subregularities we have collected, let us examine one or two. One such subregularity states that, for some noun whose central meaning is a functional object, there is another sense of that word that occurs without determination, and means the primary activity associated with the central sense of the term. For example, the word "bed" has as a central sense a functional object used for sleeping. However, the word can also be used in utterances like the following: "go to bed", "before bed", and "in bed", (but not, say, "during bed"). In these cases, the noun is determinerless, and means being in bed for the purpose of sleeping for a significant period of time (i.e., to retire).

Other examples include "jail", "conference", "school" and virtually all the scheduled meal terms of English, e.g., "lunch", "tea", "dinner". For example, it would be misleading to say that I was "in jail" yesterday if I were visiting a relative, but acceptable to say that I was "in *the* jail" under such a circumstance. Note further than I can "send my children to school", but not "to school down the street", while "to the school down the street" is acceptable. (Indeed, non-referential use of the noun presumably motivates its determinerless nature.) Also, which words conform to this subregularity is apparently a function of dialect. British English allows "to hospital" and "at university", while American English does not. The dialect difference underscores the point that this is truly a subregularity: concepts that might be expressed this way are not necessarily expressed this way.

There are many subregularities relating nouns and verbs. For example, one extends a functional nominal to a verb denoting a specialized use of that nominal. Examples of this are "tree" as in "The dog treed the cat" and "knife" as in "The murderer knifed his victim". Note that the activity is rather specific, and that the way in which it is specified is pragmatically determined. Thus, the subregularity tells us that "treeing" involves a tree, but only our world knowledge can suggest to us that it involves trapping; similarly, the subregularity can tell us that "knifing" involves the use of a knife, but cannot tell us that it means stabbing a person, and not, say, just cutting.

As a final example, consider what I call "frame complementation". For example, the central sense of "gate" is a door in a fence; but in the related senses having to do with a sports area or university campus, the term refers to the space itself through which one might enter or exit an area. "hole" follows a similar extension to produce the empty space, as in "doughnut hole".

These examples are not, in themselves, particularly important generalizations about English. Rather, they are facts of limited scope. But there appear to be many such facts, and each of them may be useful for learning related cases.

Now we come to the issue of how to exploit these subregularities. The general problem is to hypothesize, given a set of prior word senses, the best (or at least, a reasonable) new word sense for a word in a given context. To address it, we propose a framework for a new kind of analogical reasoning that combines some of the benefits of symbolic and connectionist learning models. The model is as follows:

(1) We store both individual facts, plus the subregularities abstracted from these facts. Along with each individual fact we maintain a "confidence parameter", a measure of our certainty that the stored data is correct.

(2) When an item is encountered such that we can retrieve an individual fact applicable to the known item and of an acceptable certainty level, then

the stored fact is simply used.

(3) When an item is encountered for which applicable knowledge does not exist or has an unacceptable certainty level, we attempt to retrieve applicable subregularities. Some of these regularities (presumably, the single subregularities deemed "most relevant") are applied to produce a hypothesized response. If there is no relevant subregularity, then "similar" previous instances, and similar subregularities, are retrieved, and used analogically to produce a response. The result is used if it is more certain than any stored applicable fact; otherwise the stored fact is used.

(4) If the result of (3) is deemed successful, then the result is stored. Also, if the form was the result of the application of a subregularity, it is stored as such. If it were the result of analogy from another instance or subregularity, a generalization is created and stored to serve as a subsequent subregularity.

Like individual instance-based analogical reasoning, this algorithm schema produces symbolic representations that can be used flexibly. However, it relieves the tension MacWhinney et al. (1989) describe between rote, rules and analogy, and which they turned to connectionist architectures to relieve.

Thus far, no knowledge acquisition system has been built based on this general notion of subregularity. However, the initial success of systems like MIDAS and subsequent work on subregularities causes us to view this approach as an intriguing prospect.

ACQUIRING WORD SENSES FROM MACHINE READABLE DICTIONARIES

An entirely different approach to the acquisition of lexical information is to extract it from dictionary entries. Many researchers are currently studying machine-readable dictionaries as a source for lexical information (see, for example, Alshawi (1987), Boguraev (1987), Guthrie (1990), Jensen (1987), Nakamura and Nagao (1987), Wilks (1988)). This approach has a number of advantages and disadvantages. On the positive side, dictionary processing provides computationally useful information about tens of thousands of words. There are several important limitations to this approach, however. Dictionaries contain only some of the desired information. In particular, information about valence (i.e., about the kinds of linguistic structures individual lexical items expect) is partial at best; word senses are often arbitrarily partitioned – some senses seem to be omitted, and some seemingly arbitrary distinctions introduced; many subtle distinctions are not acknowledged. Perhaps more significantly, the information that has thus far been successfully extracted in this manner has been rather impoverished, generally restricted to very coarse taxonomic information. Even then, human intervention is typically required, and the techniques used are specific to particular dictionary formats.

Recently, there have been some attempts to extract more substantive syntactic and semantic information as possible from dictionary definitions. In particular, Jaramillo and Hearst (1991) are attempting to apply a more general natural language processing approach to the interpretation of the definitions. This work is based on COBUILD, a learner's dictionary, whose definitions are generally stated simply. For the purpose of lexical acquisition, COBUILD's most salient feature is its definition style. Most dictionaries tend to write definitions which are directly substitutable for their headwords. In contrast, COBUILD's definitions are written in complete sentences. Therefore, it possible to apply simplified versions of general natural language processing techniques to its definitions.

In general, COBUILD definitions consist of two parts. The first part of the definition, often a conditional clause, gives a "template" for this usage of the word, indicating what kinds of contexts this sense is likely to participate in. The main clause of the sentence forms the actual definition of the word, usually referring to the contextual elements mentioned in the "template" clause.

Preliminary surveys indicate that COBUILD provides complement and cooccurrent preferences (i.e., information about syntactic and semantic features of typical complements or cooccurrents of a word used in a particular sense) more often than other dictionaries do. Complement and cooccurrent preferences are expressed directly in the "template" in the first part of the definition. These preferences can offer cues for disambiguation, as discussed below.

Another apparent advantage of COBUILD over other dictionaries is that its definitions tend to be less terse and more informative. It often gives explicit definitions for word senses that are usually left implicit in examples in other dictionaries. COBUILD prefers to explicitly point out specific senses of a polysemous word, rather than to leave them implicit in a more general definition. Finally, since COBUILD's definitions are written in ordinary English, we do not need to build a set of patterns specific to the dictionary; we can use standard parsers and grammars in interpreting the definitions. Therefore, the mechanisms used can be transferred to other tasks, and vice versa.

Narcisco Jaramillo has developed a system that extracts such information from the COBUILD dictionary. Below is an example of this system acquiring a sense of the word "rocket". The input is the sentence comprising the dictionary entry; the output is a number of knowledge representation language statements representing the knowledge extracted from the utterance.

One of COBUILD's definitions for "rocket" is as follows:

> rocket 1.1: [N COUNT] A rocket is a missile that contains an explosive and that drives itself through the air by sending out burning gas [...]

The following output is produced by the definition interpreter:

```
;;; -*- Mode:Common-Lisp; Base:10 -*-

(rocket-word AIO CountNoun)
(rocket-expresses167 AIO NounExpresses)
(rocket-word -word Rocket-Expresses167)
(Rocket166 -concept Rocket-Expresses167)

(Every Thing (Thing AIO Rocket166) =>
  (Thing AIO Missile1))

(Every Thing (Thing AIO Rocket166) =>
  (A Explosive1169 (Explosive1169 AIO Explosive1) =>
     (Explosive1169 -explosive1 Thing))
```

```
  (A PhysContain168 (PhysContain168 AIO PhysContain) =>
     (PhysContain168 -physcontain-explosive1 Thing)
     (Thing -container PhysContain168)
     (Explosive1169 -contained PhysContain168)))

(Every Thing (Thing AIO Rocket166) =>
  (A Drive1170 (Drive1170 AIO Drive1) =>
     (Drive1170 -drive1-self Thing)
     (Thing -driver Drive1170)
     (Thing -drivee Drive1170)
     (A MoveHasDirThrough4171
       (MoveHasDirThrough4171 AIO MoveHasDirThrough4) =>
        (MoveHasDirThrough4171 -MoveHasDirThrough4-air Thing)
        (Drive1170 -move MoveHasDirThrough4171)
        (Theair -throughsubstance MoveHasDirThrough4171))
     (A Gas1174 (Gas1174 AIO Gas1) =>
        (Gas1174 -gas1 Thing))
     (A BurningState175 (BurningState175 AIO BurningState) =>
        (BurningState175 -gas1-burningstate Thing)
        (Gas1174 -burner BurningState175))
     (A SendOut173 (SendOut173 AIO SendOut) =>
        (SendOut173 -sendout-gas1 Thing)
        (Thing -sender SendOut173)
        (Gas1174 -sendee SendOut173))
     (A Enable172 (Enable172 AIO Enable) =>
        (Enable172 -enable-sendout Thing)
        (Drive1170 -enabled Enable172)
        (SendOut173 -enabler Enable172))))
```

What is involved in getting such a method to scale up to successfully interpreting a substantial percentage of the definitions of the COBUILD dictionary, acquiring both syntactic and semantic information about individual words? The following subproblems seem particularly acute:

(1) Developing a suitable ontology

An ontology, or basic hierarchical knowledge base, is needed to provide a basis for making simple inferences about causation, time, beliefs, and so forth. Furthermore, interpretations of dictionary definitions need to be integrated into a framework of existing knowledge in order for them to be useful. Also, having an existing knowledge base can aid in disambiguation, as discussed further below.

No one yet has a substantial ontology, although several efforts are underway. While extracting a simple genus hierarchy from the dictionary is possible (cf. Wilks et al. 1988), such a hierarchy is not adequate to handle all of the inferences needed for disambiguation. Investigating ways in which this particular task may be automated from dictionary sources, as well as other means of automatic construction, is a topic of current concern. Below we will discuss one rather new approach involving text corpora that may prove to be of some assistance.

(2) Bootstrapping a lexicon of entries for basic vocabulary items

By its very nature, the dictionary cannot stand on its own. Although it gives definitions for every word it contains, it must assume that the reader already understands some set of words. In general, dictionary definitions assume that the reader has a fairly complete ontology, and understands the vocabulary that expresses the basic concepts in that ontology. Although the dictionary defines the words in that basic vocabulary, those definitions do not express the rich body of knowledge needed to understand such basic concepts. Thus, in addition to an existing ontology, we need to assume that the meanings of some word senses are hand-coded beforehand in terms of the given basic ontology. The best we can hope for in terms of automation here is a method to determine which words need to be included in this hand-coded basic vocabulary.

Amsler (1981) notes that clusters of circular definitions in the dictionary are a good candidate for forming part of such a basic vocabulary. These clusters consist of words which are defined in terms of each other. For example, one such cluster of words in Webster's Pocket Dictionary includes "class," "group," "type," "kind," "set," "division," "category," "individual," "grouping," "part," and "section." It would be difficult to define any one of these words without using any of the other words in the cluster, to make it "bottom out." Thus, it seems possible to define these words directly in terms of a given ontology, since these are precisely the kinds of words that represent basic ontological concepts.

However, there may be basic words that do not happen to participate in circular clusters. Thus, in order to discover a complete set of words that would cover all definitions, other methods must be investigated. One promising approach is to take all of the words used in all the definitions in the dictionary, then take all the words used in the definitions of that first set of words, until we reach the point where words in the basic clusters will continue to cycle in and out of the set, but the set will not become much smaller. The resulting set should be candidates for direct definition.

(3) Developing dictionary-specific disambiguation methods

Given a set of basic vocabulary, an ontology, and a few other resources, such as a good grammar of the language, the approach we have described thus far is essentially that applicable to interpreting any natural language utterance. It departs from such a general approach in two ways. First, if the system encounters an unknown word in a definition, it recursively attempts to interpret the definitions of that word's senses. Since we have presumably eliminated circularities by including a basic vocabulary, this process is guaranteed to bottom out eventually.

Secondly, we would like to take what advantage we can of knowledge about the dictionary and definitions to overcome otherwise difficult natural language processing problems. For example, it seems that some difficult general natural language problems, such as pronominal reference, may not be as bad for dictionaries as for general text. Indeed, the presence of pronouns is sometimes helpful. For example, consider COBUILD's definition of a sense of the word "priority":

> Something that **takes** *or* **has priority** over someone or something else is regarded or treated as more important than them.

Here, we can infer by standard pronoun resolution methods that the referent of "them" in the definition is the object of "over" in the template. In other words, the object of "over" in this use of "priority" is the thing that is less important. Whether other, more difficult analyses of pronouns is required remains to be seen.

Probably the most important problem during dictionary interpretation is that of lexical disambiguation; we need to determine which sense of each word in the definition is intended. COBUILD provides us with information about syntactic and semantic information about typical complements of word senses, usually in the template of each sense's definition. When we encounter a polysemous word used in conjunction with a particular complement, we can determine how well that complement fits the description given in the definitions of various senses of that word, then pick the sense with which that complement is most compatible syntactically and semantically.

One problem is that none of the words are completely disambiguated at the beginning of the process; thus, when trying to decide whether a sense of one particular word is appropriate, we do not know which senses of the words around it are intended, thus making it difficult to check complement preference constraints. It has been suggested that a constraint-filtering-and-search technique can be applied to possible combinations of word senses. And perhaps, for coarse-grain distinctions, recent corpora-based methods we will now describe will be applicable.

EXTRACTING INFORMATION FROM TEXT CORPORA

Let us return now to the issue of acquiring domain knowledge and world knowledge. Above we discussed the possibility of learning by reading. We pointed out that while this approach is a promising way of acquiring significant amounts of information, it requires addressing fundamental problems. The question then arises as to whether there is some other way to exploit the information available in text without having to wait for all these difficult problems to be resolved.

One possibility that has become the focus of some recent investigations is to use heuristic methods on text corpora to address problems whose algorithmic solution is fundamentally difficult. Previously, techniques for exploiting large quantities of text have had an entirely different character than other natural language techniques. Typically, while operating more robustly than more theoretically defensible techniques, they perform only a very superficial analysis; they do not guarantee correctness for a small domain of discourse, but rather, promise a high degree of accuracy for a very large domain.

The problem with such approaches is that they have generally remained too superficial to help with more interesting natural language or other AI tasks. Recently, however, there have been a number of promising developments. A number of systems have been constructed that can extract specific kinds of information from text. For example, the SCISOR system (Jacobs and Rau, 1990) extracts information about corporate takeovers from a newswire source; CONSTRUE (Hayes and Weinstein, 1990) classifies Reuters financial news stories into one of more of 674 predefined categories. These systems are interesting in that they represent real applications that deal with large quantities of text in flexible ways.

Both systems are rather limited, though. CONSTRUE does only a very superficial analysis, and a great deal of effort is required for it to be able to classify into a particular category. SCISOR uses somewhat more sophisticated techniques. However, it also is designed with special purpose acquisition goals in mind.

A Corpus-based Disambiguation Heuristic

We have been experimenting with some approaches that might offer greater generality. Before describing such an approach to acquisition, let us discuss some related, but more developed work on disambiguation. Here the issue is to determine the intended sense of a word in a given usage. This problem is of general interest to all natural language processing applications, as alluded to in the discussion above. It can probably only be solved completely by rather complex processes that involve reasoning with large quantities of world knowledge. Applications in which no error is tolerable will have to await the development of such a technology. However, for applications with less stringent requirements, heuristic approaches are feasible.

Hearst (1991) has developed an accurate, relatively inexpensive corpus-based algorithm that seems to address a significant portion of this problem, namely, the disambiguation of noun homonyms. The algorithm, called CatchWord, operates without complex knowledge bases, feature representations, or inference mechanisms. Rather, CatchWord checks some readily identifiable contextual features surrounding a noun against those of previously recorded instances and chooses the sense for which the most evidence is found.

For CatchWord to operate, a training cycle is required for each noun. During this cycle, a set of sentences containing the target word is analyzed for a suite of orthographic, syntactic, and lexical properties. The sentences are hand-labelled for the intended word sense of the target word, and statistical frequencies are compile relating each sense to each of the suite of features. After the training period, new untagged sentences can be analyzed for the same features, and a resemblance to the statistical profile of the previous cases used to determine the intended word sense of the word in the given sentence. Moreover, after an initial period of training, CatchWord can improve its results by automatically disambiguating new sentences containing the target word, and using their feature profile as an additional source of evidence.

The evidence used is a combination of grammatical and lexical context. For example, one piece of evidence examined is whether the target word occurs within a prepositional phrase headed with a particular preposition. The sense of the word ''bank'' pertaining to rivers tends to occur more as the complement of ''on'', for example, while the sense pertaining to financial institutions is more common as the complement of ''in''. An important quality of these criteria is that they are easily detected, and do not require recourse to complicated semantic analysis or presume a large knowledge base of facts. The properties used for disambiguation are summarized in Figure 1.

After training the system on several words by hand (usually with about 50 examples of each sense of a word), the system was tested on arbitrary sentences for Grolier's *Academic American Encyclopedia*. Between 89-94% of the test cases were correctly disambiguated, depending upon the particular lexical item. Afterwards, we allowed the system to train itself in unsupervised mode by automatically labeling additional sentences with its current statistics, and then using this data as additional information. Generally, performance improved, reaching 100% accuracy in some cases. In general, the CatchWord algorithm's performance is comparable or superior to any other published approach.

There are several ways the CatchWord algorithm might be improved and extended. For example, CatchWord's evidence metric is rather simplistic. It is biased toward senses for which it has seen more examples. Also, the current approach applies only to a binary sense distinction, and needs to be

Recorded for Target Noun

target is capitalized
target's modifier is capitalized
target is modified
target is not modified
target modifies another item
target within a prep phrase is headed by one of *in, on, of, out*
target within a prep phrase other than those listed above
a prep phrase adjacent to target is headed one of *in, on, of, out*

Recorded for each Lexical Item

item modifies target
item is modified by target
item is a head in a construct adjacent to target
item is a modifier in a construct adjacent to target
item is a verb in an adjacent construct

Figure 1: **Properties used by CatchWord**

generalized to any number of senses. Thus far, we have focussed exclusively on noun homonyms. We need to determine whether the technique will generalize to other parts of speech, and possible, to other kinds of ambiguity, such as constituent structure.

More interestingly, the time required for hand-labeling training sentences is prohibitive. One possible way to obtain this information cheaply might be to examine bilingual aligned corpora, i.e., paired sentences of translations. Since homonyms typically are not the same across languages, a bilingual dictionary might serve to determine which sense is present in a given sentence. Such an approach might provide the necessary information to bootstrap the CatchWord algorithm.

Corpus-based Knowledge Acquisition

Now let return to the theme of knowledge acquisition and ask the question of to what extent is it possible to heuristically address problems like knowledge acquisition. We have been investigating an approach to knowledge acquisition that resembles the approach to disambiguation just sketched. In particular, the idea is to scan text for patterns that are indicative of certain semantic relations. The patterns themselves involve rather superficial features, such as a particular word in a particular grammatical context, so that the presence of a pattern is relatively easy to detect. Such an approach would be able to extract simple information from an arbitrary domain without substantial manual effort. The semantic relations to be extracted this way would generally be rather restricted, for example, to hyponyms (i.e., set-superset relations), class membership, and whole-part relations. However, automatically acquiring instances of even this restricted set of relations would be a large step toward automating some tasks, such as the acquisition of a suitable ontology for dictionary reading, as described previously.

As an example of this approach, note that, from a text fragment such as

developing countries, such as India, ...

one can infer that India is an instance of the category of developing countries. To spot this information, one need only look for an occurrence of the phrase ''such as'', and isolate the noun phrases it connects. By extending the set of such patterns, other kinds of knowledge can potentially be extracted.

We have been experimenting with such techniques using Grolier's *Academic American Encyclopedia*, Medline (a corpus of medial abstracts) and the U. S. Constitution as corpora. For example, here are some facts extracted from Grolier's using this technique:

Ganda amadinda is a kind of *key xylophone*

Bambara ndang is a kind of *bow lute*

Gelidium is a kind of *red algae*

p, v, f are *labial consonants*

pie crusts are *bakery products*

synthetic sapphire is a kind of *hard material*

bombs, artillery shells, rockets are kinds of *projectiles*

meats, flour, meal, cereal, grains, cheeses are kinds of *food*

electron is a kind of *subatomic object*

Luba, Lunda, Lala, Bisa, Lamba are *central Bantu matrilineal peoples*

There are several problems inherent in developing a knowledge acquisition algorithm based on this approach, however. First, when such an algorithm can infer that a relation exists between noun phrases, it does not necessarily know what categories the relation holds between. In one of the examples above, the algorithm readily inferred that a relation holds involving the category denoted by the phrase 'developing countries''. But determining what this category is is often difficult. For example, from the sentence

> The newest type of aquarium, the oceanarium, often maintains large tanks or holding areas directly on the ocean, stocked with some of the largest marine forms, such as dolphins or sharks.

we can extract the fact that ''dolphins'' are types of ''largest marine forms.'' The problem is that the algorithm that can find this relation is not capable of determining what ''largest marine forms'' denotes. But doing so is necessary for obtaining useful information. Similarly, grammatical ambiguities may preclude determination of which objects the relation is between. For example, in a phrase like

> the problems of developing countries, such as poverty and disease,...

it is clear that poverty is an instance of a problem of developing countries, not an instance of a developing country. Thus, the algorithm must be able to make the same determination if it is to produce a meaningful result.

Finally, the information that is extractable may not make much sense out of context. For example, consider the following sentence fragment (from the

U.S. Constitution):

> then the Vice-President shall act as President, as in case of the death or other constitutional disability of the President.

Here one can infer that death is a constitutional disability, but this is not a very useful fact unless further interpreted within the context of the rest of the text.

One class of problems seems amenable to an extension of our approach to disambiguation. For example, it seems possible to find statistical profiles to help resolve syntactic ambiguities in order to determine what noun groups a relation holds between. (We have conducting some preliminary experiments in this regard. We found, for example, that in the case of the preposition "of", most of the problematic instances fall into a small number of classes for which a statistic decision procedure is plausible.) Similarly, at least part of the interpretation process can be addressed by using the homonym disambiguation algorithm on the nouns in the noun groups.

Another class of problems can be addressed by developing interpretation heuristics specific to the task. For example, it should be possible to hypothesize more specific interpretations for very general lexical items, so that we can interpret "marine forms" as referring to marine life forms. Similarly, we should be able to extract some information by eliminating so-called "natural" modifiers from a noun phrase. Thus, something that is a kind of "largest marine form" is a kind of "marine form" (whereas "imitation cheese" is not a kind of "cheese"). Removing such items, especially comparatives, will usually result in more reliable, context-independent facts.

In general, the richer the ontology we have available, the easier it will be to make an interpretation. In the worst case, though, we may not be able to interpret some noun phrases very deeply. In this case, there are still various tasks for which the more superficial information may be useful. For example, knowing that a "broken bone" is an injury might be useful even without knowing exactly what a "broken bone" is.

We have also developed a procedure to automatically find patterns for particular relations. This is accomplished by gathering, for a particular relation, a list of terms for which it holds, finding places in the corpus where these expressions occur syntactically near one another, recording the environment, and then finding the commonalities among these environments. For example, running this procedure using just the pair of terms with a couple of pairs and the "set-member" relation, the algorithm proposed several "such as" patterns, an "including" pattern, an "and other" pattern, and a pattern involving "especially." The latter was new to us. Thus, automatic pattern extraction seems rather promising. We plan to investigate automatic pattern extraction as a means to increase the kind of information we can extract in this fashion.

Obviously, this approach is still in its infancy. Despite some intriguing initial results, we cannot be sure the technique will not crash on the rocky shoals of world knowledge. That is, for the technique to be useful, we must demonstrate that a significant amount of knowledge can be extracted this way without presuming much in the way of world knowledge or reasoning. An endeavor to make such a demonstration is currently under way.

CONCLUSION

Natural language processing and machine learning are two young technologies that need each other. Natural language requires large amounts of knowledge of various kinds; it is hard to imagine that field can achieve its goals without substantially automating the acquisition process. On the other hand, much work on machine learning has focused on incremental improvements based on performance. But the acquisition of knowledge is probably more of a key bottleneck for building intelligent systems. Huge volumes of knowledge are available now, in machine readable form, if only we could understand how to use it. Natural language processing technology holds the key to this storehouse.

We have described some approaches aimed at acquiring knowledge about language, and at using language to acquire knowledge. While much additional work is needed to realize the full potential of this interaction, significant applications of the more well-understood techniques are currently being employed. We suspect that many new ideas at the interaction of these disciplines to emerge shortly: If the individual fields are young, a product of their marriage is in its infancy.

REFERENCES

Alshawi, H. (1987). Processing Dictionary Definitions with Phrasal Pattern Hierarchies. *Computational Linguistics* 13:3-4, pp. 195–202.

Amsler, R. (1981). A Taxonomy for English Nouns and Verbs. In *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics*, pp. 133--138.

Boguraev, B., T. Briscoe, J. Carroll, D. Carter, and C. Grover. (1987). The Derivation of a Grammatically Indexed Lexicon from LDOCE. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pp. 193–200.

Granger, R. H. FOUL-UP: A Program that figures out the meanings of words from context. (1977). In the *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. Cambridge, MA.

Guthrie, L., Slator, B. M., Wilks, Y., & Bruce, R. (1990). Is there content in empty heads? In the *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki, Finland.

Hayes, P. J. and Weinstein, S. P. (1990). CONSTRUE/TIS: A System for Content-Based Indexing of a Database of News Stories. In the *Proceedings of the Second Annual Conference on Innovative Applications of Artificial Intelligence*, AAAI, Georgetown University, Washington, D. C.

Hearst, Marti. (1991). Toward Noun Homonym Disambiguation Using Local Context in Large Text Corpora. In the *Proceedings of the 7th Annual Conference of the University of Waterloo Centre for the New OED and Text Research.*.

Jacobs, P. and Rau, L. (1990). SCISOR: A system for extracting information from financial news. In the *Communications of the ACM*, 33(11).

Jaramillo, N. and Hearst, M. (1991). Acquiring the Semantics of Simple Phrasal Patterns Using COBUILD. In the working notes of the AAAI Spring Symposium on Machine Learning of Natural Language and Ontology, Stanford, C.

Jensen, K., and J.-L. Binot. (1987). Disambiguating Prepositional Phrase Attachments by Using On-Line Dictionary Definitions. *Computational Linguistics* 13:3, pp. 251–260.

Lakoff, G. and Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press.

MacWhinney, B., Leinbach, J., Taraban, R. & McDonald, J. (1989). Language Learning: Cues or Rules? In the *Journal of Memory and Language*, 28, pp. 255-277.

Martin, James. (1988). A Computational Theory of Metaphor. Berkeley Computer Science Technical Report no. UCB/CSD 88/465.

Nakamura, J.-I., and M. Nagao. (1988). Extraction of Semantic Information from an Ordinary English Dictionary and its Evaluation. In *Proceedings of the Twelfth International Conference on Computational Linguistics*, Budapest, pp.

459–464.

Norvig, Peter. (1987). A Unified Theory of Inference for Text Understanding. Ph. D. Thesis, University of California, Berkeley Technical Report No. UCB/CSD 87/339.

Norvig, Peter and Lakoff, George. (1987). Taking: A Study in Lexical Network Theory. In the *Proceedings of the Thirteenth Annual Meeting of the Berkeley Linguistics Society*. Berkeley, CA. February.

Reddy, Michael. (1979). The Conduit Metaphor. In A. Ortony (ed.), *Metaphor and Thought*, pp. 284-324. Cambridge University Press, Cambridge.

Selfridge, M. (1982). Computer Modeling of Comprehension Development. In W. G. Lehnert & M. H. Ringle, *Strategies for Natural Language Processing*. Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Wilensky, R., Arens, Y. and Chin, D. (1984). UC: An intelligence natural language interface to UNIX. *Communications of the ACM*, Vol. 27, no. 6.

Wilensky, R., Mayfield, J., Chin, D., Luria, M., Martin, J. and Wu, D. (1988). The Berkeley UNIX Consultant Project. *Computational Linguistics* 14-4.

Wilks, Y. A., Fass, D. C., Guo, C. M., McDonald, J. E., Plate, T., & Slator, B. M. (1988). A Tractable Machine Dictionary as a Resource for Computational Semantics. In the *Proceedings of the 12th International Conference on Computational Linguistics (COLING-88), pp. 750-755. Budapest, Hungary.*

INDEX