THE JOURNAL OF THE LEARNING SCIENCES, 2(1), 1-59

A Model of the Self-Explanation Effect

Kurt VanLehn, Randolph M. Jones,
and Michelene T. H. Chi
Learning Research and Development Center
University of Pittsburgh

Several investigators have taken protocols of students learning sophisticated skills, such as physics problem solving and LISP coding, by studying examples and solving problems. These investigations uncovered the *self-explanation effect:* Students who explain examples to themselves learn better, make more accurate self-assessments of their understanding, and use analogies more economically while solving problems. We describe a computer model, Cascade, that accounts for these findings. Explaining an example causes Cascade to acquire both domain knowledge and derivational knowledge. Derivational knowledge is used analogically to control search during problem solving. Domain knowledge is acquired when the current domain knowledge is incomplete and causes an impasse. If the impasse can be resolved by applying an overly general rule, then a specialization of the rule becomes a new domain rule. Computational experiments indicate that Cascade's learning mechanisms are jointly sufficient to reproduce the self-explanation effect, but neither alone can reproduce it.

If you teach college courses, you have probably been visited by students who are bright, work hard, and yet get low grades on examinations. They want to know what they are doing wrong. If you suggest that they study harder, they ask you, "How?" Feeling slightly sheepish, you roll out the litany of "good study habits" that your teachers and parents told you: Study in a well-lit place that is free from distractions, review the chapter for main ideas both before and after reading it, take good notes and review them, etc. The students reply, with perhaps some irritation, that they already do that. They want to know how to study in such a way that they can extract the information from the textbook and homework problems that you, the

Requests for reprints should be sent to Kurt VanLehn, Learning Research and Development Center, University of Pittsburgh, 3939 O'Hara Street, Pittsburgh, PA 15260.

teacher, expect them to extract. They know it is possible because their friends apparently have no trouble extracting the requisite information. They want to know how they can study as effectively as their friends.

Cognitive science does not yet have a complete answer for these students, but it has made steady progress toward understanding effective studying processes. Much research has involved subject areas that involve extensive problem solving, such as science, mathematics, engineering, and computer science. In these task domains, studying worked examples appears to play a key role in effective learning. Several studies have shown that students attend more to examples than other forms of instruction both in controlled experiments (LeFevre & Dixon, 1986) and in natural settings (Anderson, Farrell, & Saurers, 1984; Chi, Bassock, Lewis, Reimann, & Glaser, 1989; Pirolli & Anderson, 1985; VanLehn, 1986). When students solve problems, they often refer to examples (Anderson et al., 1984; Chi et al., 1989; Pirolli & Anderson, 1985), but how much they learn from such analogical problem solving appears to depend on how well they understand the examples (Pirolli & Anderson, 1985), which probably depends on how they studied the examples. Some researchers (Pirolli, 1991; Pirolli & Bielaczyc, 1989; Reed, Dempster, & Ettinger, 1985; Sweller & Cooper, 1985; Ward & Sweller, 1990) compared the learning of students who were given worked example problems with the learning of students who were given the same problems and had to solve them themselves. It was often found that examples were similar to problems in that the same factors predicted transfer but were different from problems in that less training time was needed to achieve the same level of performance. In several studies (Charney, Reder, & Kusbit, 1990; Reed et al., 1985; Ward & Sweller, 1990) examples that varied in the amount of explanation accompanying their solutions were compared. Less explanation often led to more learning.

Although experiments comparing instructional materials have shed some light on studying processes, several researchers have used a more direct paradigm for understanding which studying processes are most effective. They compared the behaviors of effective and ineffective learners as they studied the same material. The students who learned more appeared to study the examples by explaining them to themselves (Chi et al., 1989; Fergusson-Hessler & de Jong, 1990; Pirolli & Bielaczyc, 1989). For instance, when Chi and VanLehn (1991) analyzed the protocols of effective and ineffective learners as they studied examples, they found that the good learners made more comments about the conditions under which specific actions were advisable, the relationships between actions and goals, the consequences of actions, and the meanings of mathematical expressions. This finding does not completely determine the good learners' studying process, but it strongly suggests that they were somehow explaining the example to themselves by filling in the details that the example left out and

by highlighting the relationships between general pieces of domain knowledge and the specific actions taken in solving the example. This process (or processes) was named *self-explanation* by Chi et al. (1989). Bielaczyc and Recker (1991) showed that students can be taught how to self-explain and that, when they do, they learn more effectively.

The main goal of the present research was to specify precisely the processes of self-explanation and to understand why they enhance learning. Protocols from the Chi et al. (1989) study were reanalyzed, and several learning processes were uncovered. They were modeled in a machine learning system, called Cascade. Cascade is able to simulate all the Chi et al. findings, which suggests that it is a fairly complete model of the studying processes used by both effective and ineffective learners.

In order to understand which learning processes were responsible for the self-explanation effect, Cascade was run several times with various combinations of its learning processes turned off. We were surprised to find that a learning process that acquires search control knowledge is necessary for successful learning by the other processes, which acquire domain rules and principles.

In Cascade, the only difference between effective and ineffective learners is their strategies for studying examples. The good learner chooses to rederive the example's solution, whereas the poor learner simply accepts the solution without trying to check it or regenerate it. As expected, this strategy difference causes the effective learner to learn more rules while studying the example than the poor learner. However, we were surprised to find that it also causes the good learner to learn more rules than the poor learner *while solving problems* even though the problem-solving strategies are the same for both good and poor learners. Thus, studying examples properly raises the learning *rate* on subsequent problem solving. This is consistent with Pirolli and Anderson's (1985) observation that the way students study examples seems to influence how much they learn while solving problems.

A second goal of the present research was to extend current theories of cognitive skill acquisition. Most theories of skill acquisition propose two classes of learning mechanisms, which we call knowledge acquisition methods and knowledge compilation mechanisms. Knowledge acquisition methods are responsible for acquiring an initial version of the skill from whatever instructional material is available. Knowledge compilation mechanisms are responsible for the slow changes in performance that accompany practice. Most theorists (e.g., Anderson, 1983; Holland, Holyoak, Nisbett, & Thagard, 1986; Newell, 1990) propose that there are only a few knowledge compilation mechanisms, such as chunking, proceduralization, and strengthening, and that they are part of the human cognitive architecture. That is, they are present in all individuals beyond a certain young age

and are probably biologically determined. In contrast, knowledge acquisition methods, such as studying a text effectively or using examples to help guide one's problem solving, are believed to be cognitive skills themselves. They are not part of the cognitive architecture. That is, they are learned and not innate, although they may be highly automatized if they have been practiced enough, so subjects may not be aware of their habits for acquiring knowledge. Different training situations evoke different methods, and different individuals may use different methods even in the same training situation. It is impossible to precisely specify all knowledge acquisition methods, because novel training methods may call forth novel knowledge acquisition methods and thus add new members to the class. Sometimes it is not even possible to distinguish one method from another, because variations among the methods as they are adapted to different situations make them blend into one another.

Currently, there are much better theories of knowledge compilation mechanisms than of knowledge acquisition methods. For instance, ACT* (Anderson, 1983) has two knowledge compilation mechanisms—proceduralization and strengthening—that model the power law of practice, several kinds of transfer, the decreasing reliance on training materials during the second stage, and other practice effects. Soar's (Newell, 1990) chunking has also been thoroughly explored. However, ACT* and Soar are intended to be models of the human cognitive architecture, so according to theory they should contain only knowledge compilation mechanisms. Knowledge acquisition methods should be learned. Perhaps because of this theoretical position, less attention has been paid to simulating human knowledge acquisition methods. On the other hand, because machine learning has invented hundreds of knowledge acquisition methods, there is no lack of hypotheses about what people could be doing to acquire knowledge. The problem is that we know very little about what they actually do to acquire knowledge. For instance, we all know that one can skim an example or one can study it intensely and try to understand it deeply. What difference does that make, and what exactly is involved in understanding an example deeply? Would one learn just as much by skimming the example on its first presentation and studying it intensely only if necessary for solving a problem encountered later? Despite all the wonderful methods of machine learning and the well-wrought mechanisms of knowledge compilation, little is known about human knowledge acquisition methods. Advancing the field's understanding of this pedagogically crucial area is the problem addressed by the present research.

Two common criteria for evaluating computational models of skill acquisition are computational sufficiency and empirical adequacy. A model is computationally sufficient if it can produce the observed changes in knowledge using only the kinds of information available to the human

student. Computational sufficiency is harmed if, for instance, the model's programmer intervenes in order to guide the model back onto the right path when it gets lost. Empirical adequacy is assessed by comparing the model's behavior with some kind of human behavior. In this article we compare Cascade's behavior with several findings, collectively called the *self-explanation effect*. Because these findings in themselves are not constraining enough to completely determine the knowledge acquisition methods students are using, we also report informal analyses of the protocols that motivated Cascade's design. Ultimately, we would like to simulate the protocols on a line-by-line basis, as was done by Newell and Simon (1972), Ohlsson (1990), VanLehn (1991a), and a few others. Such a simulation is in progress, and we hope to report the results at a later time.

In addition to computational sufficiency and empirical adequacy, we believe a good model should be supported by competitive argumentation (VanLehn, 1990; VanLehn, Brown, & Greeno, 1984). The major hypotheses that define the model should be made explicit, plausible alternatives to each should be articulated, and the alternatives should be shown to be empirically inadequate or computationally insufficient. Although this article does not present a complete competitive argument for Cascade, it does explicate the major hypotheses and provide some empirical evidence for them.

In the next section, the self-explanation effect is described. A description of the Cascade system follows. The computational experiments that simulate the self-explanation effect are presented next. Thereafter follows a long, optional section wherein each of the major hypotheses embedded in Cascade is presented and motivated with protocol data. We conclude with a discussion of what was discovered by implementing and testing Cascade, a comparison of Cascade with two other models of the self-explanation effect, and a discussion of Cascade's weaknesses and plans for its development.

THE SELF-EXPLANATION EFFECT

The task domain used by Chi et al. (1989) was Newtonian particle dynamics, the first topic in a typical first-year college physics course. Figure 1 shows a typical problem and its solution. Solving such problems generally involves finding relevant forces and drawing them on a free-body diagram, projecting the forces onto axes, applying Newton's Second Law ($F = ma$; the net force acting on a body equals its mass times its acceleration), and solving systems of algebraic equations.

Although the Chi et al. (1989) study had a complex format, the basic activities of the subjects were as follows:

Problem: Figure a shows an object of weight W hung by strings. Consider the knot at the junction of the three strings to be "the body." The body remains at rest under the action of the three forces shown in figure b. Suppose we are given the magnitude of one of these forces. How can we find the magnitude of the other forces?

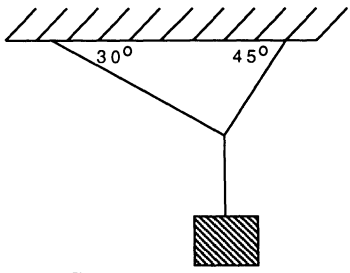

Figure a

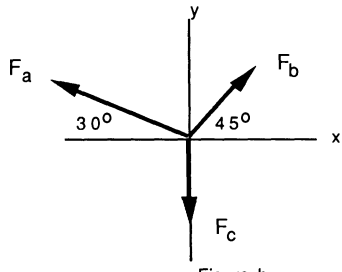

Figure b

Solution:

Fa, Fb and Fc are all the forces acting on the body. Since the body is unaccelerated,

Fa+Fb+Fc=0.

Choosing the x- and y-axes as shown, we can write this vector equation as three scalar equations:

Fax+Fbx = 0

Fay+Fby+Fcy = 0

using eqn 5-2. The third scalar equation for the z-axis is simply

Faz = Fbz = Fcz = 0.

That is, the vectors all lie in the x-y plane, so that they have no z-components. From the figure we see that

Fax = -Fa cos 30° = -0.866Fa,

Fay = Fa sin 30° = 0.500Fa,

and

Fbx = Fb cos 45° = 0.707Fb,

Fby = Fb sin 45° = 0.707Fb.

Also,

Fcy = -Fc = -W,

because the string C merely serves to transmit the force on one end to the junction at its other end. Substituting these results into our original equations, we obtain

-0.866Fa + 0.707Fb = 0

0.500Fa + 0.707Fb - W = 0

If we are given the magnitude of any one of these three forces, we can solve these equations for the other two. For example, if W=100N, we obtain Fa=73.3N and Fb=89.6N.

FIGURE 1 A physics example.

1. Take pretests.
2. Study each of the first three chapters of the textbook (Halliday & Resnick, 1981) until a criterion test on the chapter is passed. This phase was intended to give subjects the prerequisite knowledge needed for learning classical mechanics.
3. Study the first part of the textbook's chapter on classical mechanics. This part introduced the concepts of force, mass, and gravitational acceleration. It gave the history and experimental evidence for Newton's laws. It ended with a five-step procedure for solving

mechanics problems. Henceforth, this part of the chapter is called the *text* to distinguish it from the remainder of the chapter, which consisted of worked examples and exercise problems.

4. Take a test on the declarative knowledge of the chapter. For instance, one question asked students to state Newton's laws in their own words. Students who failed this test were sent back to Step 3 of the study.
5. Study the textbook's worked examples while talking aloud. Figure 1 is an example from the textbook. The protocols collected during this phase were transcribed and classified to determine, among other things, how many self-explanations were given by each subject.
6. Solve quantitative problems while talking aloud. Subjects were allowed to refer to the textbook, and they often did. They referred mostly to the examples and not the text. None referred to the chapter's five-step procedure, which is consistent with the findings mentioned earlier that students prefer worked examples over other forms of instruction.
7. Take posttests.

On the basis of the scores on quantitative problem solving (Phase 6), Chi et al. divided their 8 subjects into two groups. The 4 students with the highest scores were called the *Good solvers;* the others were called *Poor solvers.*

Two similar studies have been performed. Pirolli and Bielaczyc (1989) used a similar design, with LISP coding as the task domain. Fergusson-Hessler and de Jong (1990) had subjects give protocols as they studied a manual on applications of principles of electricity and magnetism to the Aston mass spectrometer. In both studies, subjects were classified as Good and Poor solvers on the basis of their test scores.

The first main result of the Chi et al. (1989) study was derived by classifying each of the subjects' comments during example studying as either self-explanations or other kinds of comments (e.g., paraphrases, mathematical manipulations, and metacomments). Good solvers uttered a significantly larger number of self-explanations (15.5 per example) than did the Poor solvers (2.7 per example). Pirolli and Bielaczyc (1989) corroborated this finding when they found that their Good solvers made significantly more domain-related explanations than did their Poor solvers while studying examples. Fergusson-Hessler and de Jong (1990) found that in the categories most representative of deep processing, Good solvers had more than twice as many episodes as the Poor solvers (45% vs. 18%), whereas the Poor solvers had almost twice as many episodes of superficial study processes as Good solvers (19% vs. 10%). In short, there is evidence from all three studies that students who learn more utter more self-explanations while studying examples.

While studying examples, students often say whether they understood

what they have just read. Chi et al. (1989) classified such self-monitoring statements as either positive (e.g., "Okay, that makes sense") or negative (e.g., "Wait. How did they get that?"). They found that the Good solvers were more accurate in their self-monitoring statements in that 53% of their statements were positive and 46% were negative. The Poor students were significantly less accurate, saying 85% of the time that they understood when, on the basis of their problem-solving performance, they obviously had not. Thus, students who learn more make more accurate self-monitoring statements during example studying. Fergusson-Hessler and de Jong (1990) made a similar finding. The statement "Everything is clear" occurred three times more often in the protocols of Poor solvers than of Good solvers.

When Chi et al. (1989) analyzed protocols from the subjects' quantitative problem solving, they found that both Good and Poor solvers referred to examples on most problems (75% for Good solvers; 83% for Poor solvers). However, they found that the Good solvers referred to the examples less often per problem (2.7 times) and more briefly (reading on average only 1.6 lines per reference) than did the Poor solvers, who referred to the examples more frequently (6.7 times per problem) and tended to start at the beginning of the example and read many lines (on average, 13.0 lines per reference). Thus, students who learn more refer less frequently and more specifically to examples during analogical problem solving.

To summarize, the Good solvers differed from the Poor solvers in four major ways:

1. Good solvers uttered more self-explanations during example studying.
2. Their self-monitoring statements during example studying were more accurate.
3. They made fewer references to examples during problem solving.
4. Their references to examples during problem solving were more targeted.

These four findings constitute the self-explanation effect.

THE CASCADE MODEL

Cascade has two basic abilities. It can explain examples and it can solve problems. These two processes are discussed separately in the following sections.

Explaining Examples

An example consists of a problem and a solution. Figure 1 shows an example studied by subjects in the experiments. The solution consists of a

list of lines. The lines clearly follow from one another, but the reasoning connecting them is not completely specified. For instance, the example says, "Fa, Fb and Fc are all the forces acting on the body," but it does not say why those are the only forces acting on the body or how they were derived. When Cascade is simulating a Good solver, it tries to derive each line. When it is simulating a Poor solver, it does not try to derive lines. This is the key difference between Good and Poor solvers, according to the Cascade model.

Deriving a line is a two-stage process. The first stage is to match the line to equations stored in memory. The example line "Fax = −Fa cos 30°" in Figure 1 is represented as follows:

```
projection(force(knot,string_A), axis(knot,x,0)) =
  −1 * magnitude(force(knot,string_A)) * apply(cos,30)
```

The variables, Fax and Fa, have been replaced by their meanings. Because the comments of Good and Poor solvers indicated that both groups of subjects figured out the meanings of the variables, Cascade does not model this process, because it would be the same for both the Good and Poor solver simulations.

Equations are represented as Prolog rules. The conclusion (to the left of the :− symbol) is an equation, and the antecedent (following the :− symbol) contains conditions that must hold for the equation to be applicable. Capitalized symbols are Prolog variables. For instance, the following rule matches line "Fax = −Fa cos 30°":

```
constraint(projection(V,A) =
    sign(proj(V,A)) * magnitude(V) * trigfn(proj(V,A))) :−
  instance(V,vector),
  instance(A,axis),
  origin(A,O),
  vertex(V,O).
```

This rule says that if V is a vector, A is an axis, and the origin of the axis is the vertex of the vector, then the projection of V onto A is the magnitude of the vector multiplied by a sign and a trigonometric function that depends on the geometric relationship between the vector and the axis.

Matching the line to the equation pairs four quantities from the equation with four values from the line (see Table 1). The second stage in the derivation is to prove that each of the quantities has the value with which it is paired. In the case of Quantities 1 and 3, this is trivial, because they are equal. The other two quantity-value assertions are proved by backward

TABLE 1
Quantities (First Line) Paired With Values
(Second Line) by Matching a Rule to a Line

1.	projection(force(knot,string_A),axis(knot,x,0))
	projection(force(knot,string_A),axis(knot,x,0))
2.	sign(proj(force(knot,string_A),axis(knot,x,0)))
	−1
3.	magnitude(force(knot,string_A))
	magnitude(force(knot,string_A))
4.	trigfn(proj(force(knot,string_A), axis(knot,x,0)))
	apply(cos,30)

chaining through equations. For instance, to prove the fourth assertion, Cascade uses the following conditioned equation:

```
constraint(trigfn(proj(V,A)) =
    apply(name(trigfn(proj(V,A))),angle(trigfn(proj(V,A))))) :-
  instance(V,vector),
  instance(A,axis),
  origin(A,O),
  vertex(V,O).
```

The sought quantity, trigfn, matches the quantity on the left side of the equation. Its value, apply(cos,30), is matched to the right side of the equation, which sets up two pairings (see Table 2). Cascade recurses to prove each of these quantity–value assertions. The recursion terminates when the value of a sought quantity is provided in the problem statement. For some reasoning, such as the geometric reasoning required for proving the second quantity–value assertion of Table 2, we care only about the outcome and not the process, so this kind of reasoning is represented with tables and ad hoc Prolog code. This kind of reasoning also "terminates" the recursive reasoning.

As a side effect of deriving a line, Cascade stores the derivation in memory for later use during analogical problem solving. For each

TABLE 2
Quantities (First Line) Paired With Values
(Second Line) by Applying a Rule

1.	name(trigfn(proj(force(knot,string_A), axis(knot,x,0))))
	cos
2.	angle(trigfn(proj(force(knot,string_A),axis(knot,x,0))))
	30

quantity-value assertion it proves, Cascade stores a triple consisting of the name of the example, the quantity, and the rule used to derive the quantity's value. Storing all the derivational triples is an idealization. Even the best students cannot recall every step in their derivation of a line. However, if they need to know which rule was used to derive a certain goal in order to carry out some kind of analogy, they can probably recover that fact by examining the line and perhaps even rederiving it. Thus, the information recorded in Cascade's triples is available to Good solvers, albeit sometimes not directly from memory. In the Poor solver simulation, lines are not derived so no derivational triples are stored.

If Cascade cannot prove a quantity-value assertion, it first tries to backtrack and find another way to derive the line. When Good solvers reach an impasse, they often do exactly that, as well as checking to see if they made any careless mistakes (slips). Because Cascade does not make slips, it checks only for alternative solution paths. If Cascade is missing some relevant domain knowledge, then all alternative paths will also fail. It returns to the original failure impasse and tries to ferret out the missing knowledge. The main method for constructing missing knowledge is to try to resolve the impasse by using commonsense physics (e.g., that blocks sliding down inclined planes do not jump into the air or fall through the plane's surface) or overly general rules (e.g., parts have the same property values as the whole, so the pressure in a part of a container is equal to the pressure in the whole container). For instance, one subject could not figure out how to prove one of the aforementioned quantity-value assertions, that the sign of the projection was negative (Pair 2 in Table 1). First the subject tried looking for an appropriate explanation in a table of trigonometric identities. This failed. She then looked up the value of cos(30) in a table of cosines. This also failed. (Notice that the subject tried multiple methods for acquiring the missing knowledge, which is just what Cascade does, too.) The following exchange then took place:

> S: Hmmm, negative cosine 30, why would they say ahhh, ummm. . . . The, ohh, okay, maybe it's just because the A component is; the X component of force A is negative. So they just. . . . Well okay I'll, I'll try that for a while. Let's see if that works, 'cause that makes sense.
> E: What makes sense?
> S: The reason the negative is there is because the X component is in the negative direction on the x-axis.

The subject produced the correct rule for determining projection signs, but it is not clear from this protocol how she did so. We believe that she noticed that the vector was nearest the negative portion of the x-axis, and applied an overly general rule that says that signs are often carried from one property

to another during mathematical operations.[1] Cascade's representation for such a rule is as follows:

constraint(sign(P(X,Y)) = sign(Q(X,Y))) :– true.

Matching this equation to the sought quantity yields the following substitutions:

P = proj
X = force(knot,string_A)
Y = axis(knot,x,0)

The variable Q is still unbound, so Cascade's subgoal is to prove that −1 is the value of the quantity

proj(force(knot,string_A), axis (knot,x,0))

In English, the goal is to find a property of the projection whose value is a negative sign. Finding that the nearest half-axis is negative in this situation achieves the subgoal, achieves the goal, and thus resolves the impasse.

Whenever an overly general rule resolves an impasse, Cascade creates a specialization of it by instantiating the rule then substituting variables for problem-specific constants, such as physical objects and numbers. In this case, it creates the following rule:

constraint(sign(proj(force(K,S), axis(K,x,R))) =
  sign(nearest_half_axis(force(K,S), axis(K,x,R)))) :– true.

The variables K, S, and R have been substituted for knot, string_A, and 0, respectively. The new rule says that the sign of the projection of a force onto an x-axis is the same as the sign of the half-axis that is nearest that force.

This rule is added to the domain knowledge tentatively. If Cascade later

[1]Projection of vectors is reviewed in Chapter 2 of the textbook, so it is possible that this subject was recalling this rule rather than constructing it. However, four other subjects also had problems with projections onto negative axes, so the textbook's review seems to have been ineffective. Moreover, this particular subject had more problems with mathematics than others. Given these prior probabilities and the fact that the subject spent several minutes looking in vain for this rule before producing it, we believe that the subject was not recalling the rule but was actually constructing it. By the way, in the simulation runs discussed later, all relevant knowledge that was covered in the textbook was included in the rules given to Cascade before it started explaining the examples. Because this rule was included as prior knowledge, this particular impasse and learning described here did not occur. When we fit Cascade's behavior to individual subjects, the model of this subject will not be given this rule as prior knowledge, and Cascade will have to learn it.

succeeds in deriving the line, then the rule is made a permanent part of the domain knowledge base; otherwise, it is deleted. This is another idealization. Subjects' comments make it clear they do not always recall a newly invented rule, and even if they do, they remain suspicious of it until it has been used several times (cf. VanLehn, 1991a). In a later version of Cascade, levels of belief may be added to rules to represent this growth in confidence in self-invented rules.

This particular method for resolving impasses and learning new rules is called *explanation-based learning of correctness* (VanLehn, Ball, & Kowalski, 1990).

Cascade has a second method for learning new rules by resolving impasses. When neither domain knowledge nor overly general rules can prove that a certain quantity has a certain value, Cascade gives up and just assumes that the example is right in assigning that value to that quantity. It also builds a rule that sanctions analogies to this specific assumption. For instance, a line in the example of Figure 1 reads, "Consider the knot at the junction of the three strings to be 'the body.' " Upon reading this, one subject said, "Why this should be the body? I thought W was the body. OK, let's see later." None of the 8 subjects was able to explain why the knot was chosen as a body. Indeed, we do not think that there is a proper explanation for this choice, even with overly general rules. Experts probably have many highly specific rules that tell them the right choices for common problems. For unfamiliar problems, experts make a tentative choice, plan a solution, and change their choice if a solution cannot be planned (Larkin, 1983). Probably the best that a learner can do in the knot situation is to form a rule that says, "In problems like this one, choose the knot as the body." This appears to be what the subjects did, because when they later tried to solve problems that also had three strings converging on a knot, they all referred back to the three-strings example and then chose the knot as the body.

Cascade simulates this behavior with a type of abduction (Pople, 1973) that produces rules that cause analogies. Normal abduction produces "P(a)" when given "Q(a)" and "P(a) implies Q(a)." That is, if assuming P(a) explains why Q(a) holds, then we assume P(a). Cascade's abduction is similar, except it produces a generalization of "P(a)" in the form of a rule which says, "if X is analogous to a, then P(X)." For instance, when Cascade cannot prove that the only body of problem sx is knot_sx (i.e., that bodies (sx) = [knot_sx]), it builds the following rule:

```
constraint(bodies(Problem) = V) :-
  current_situation(CurrentSit),
  analogical_retrieval(CurrentSit,situation_sx),
  analogical_mapping(CurrentSit,situation_sx,Map),
  apply_map(Problem,sx,Map),
  apply_map(V,[knot_sx],Map).
```

This rule says that if an analogy can be built between the current situation and situation_sx, and the current problem is analogous to sx, then the current problem's bodies are analogous to [knot_sx]. This rule can be used to find bodies for other problems by finding objects corresponding to knot_sx. This method of learning new rules at impasses is called *analogy abduction,* because it abduces analogy-producing rules.

Analogy abduction and explanation-based learning of correctness are just two of the many methods that subjects use to resolve impasses and acquire new rules. In a case discussed earlier, a subject looked up information in trigonometric tables. Had she succeeded, she probably would have built a new rule. In another case, a subject tested his memory of an algebraic operation by generating a test problem and solving it. Cascade currently has just two knowledge acquisition methods because these two seem to be the most popular in this particular instructional situation.

To summarize, two major kinds of learning occur when Cascade derives example lines: (a) The derivation itself is stored in the form of triples that pair sought quantities with the rules used to derive their values. (b) New rules are created when an impasse is resolved via explanation-based learning of correctness, analogy abduction, or other yet-to-be modeled methods.

Solving Problems

Overall, problem solving is similar to example explaining. Explanation-based learning of correctness can occur during both, and derivations are recorded in memory for both. A minor difference is that analogy abduction applies only during example solving. It is based on assuming that a specified quantity has a specified value, but the value part of the pair is only available during example explaining and not during problem solving.

Solving a problem is most similar to the second stage in deriving an example line. In that stage, the goal is to prove quantity–value assertions, and this is done by backward chaining through equations. In problem solving, the goal is to find a value for a quantity, and this is also done by backward chaining through equations. For instance, if the goal is to find the value of

```
projection(force(knot2,string_1),axis(knot2,x,0)),
```

then Cascade can use the same equation as before:

```
constraint(projection(V,A) =
    sign(proj(V,A)) * magnitude(V) * trigfn(proj(V,A))) :-
  instance(V,vector),
  instance(A,axis),
```

origin(A,O),
vertex(V,O).

It matches the left side of the equation to the sought quantity and sets the quantities on the right side as subgoals. When it has found values for each of the subgoals, it multiplies them together and returns the result as the value of the projection.

Usually there are several rules whose equations match the currently sought quantity. Cascade must choose one. This is a search control decision; if Cascade's choice fails to yield a solution to the problem, it can back up and try a different rule. Nonetheless, it should try the rule most likely to succeed first. Cascade's main heuristic for making such choices is to select the rule that was used to find a similar quantity in an analogous example. To implement this heuristic, Cascade first retrieves an example whose diagram is similar to the current problem's diagram. (This type of analogical retrieval is somewhat idiosyncratic to the materials used in the Chi et al. (1989) study, so Cascade models it with a table look-up rather than an actual visual indexing process.) It then forms an analogical mapping that pairs objects in the problem with objects in the example. Using this mapping, it converts its sought quantity, say,

projection(force(knot2,string_1),axis(knot2,x,0)),

into a quantity that uses the example's objects, say,

projection(force(knot,string_A),axis(knot,x,0)).

Cascade looks this quantity up in the triples that encode the example's derivations and determines what rule was used to find this quantity's value. This is the rule that it will try first to achieve the problem's goal. This process is called *analogical search control.*

Usually, there are many times in the course of solving a problem when multiple rules match the sought quantity and choices must be made. Cascade does analogical retrieval and mapping only for the first one. It stores the map and reuses it for all the others. This is consistent with subjects' behavior. If they have not committed the example to memory, then they flip through textbook pages until they find the right example and reread it to refresh their memory for its derivation. This usually occurs only on the first analogical reference during an attempt to solve a problem and not during other analogical references during the solving of that problem. Even if they have committed the example to memory, when they start a new problem they seem to spend a little bit longer on their first analogical reference to the example than they do on the others. Perhaps they are forming a mapping, just as Cascade does on its first encounter with a new

example-problem pair. As mentioned earlier, the current version of Cascade is an idealization, in that it assumes that the example and its complete derivation are always held in memory.

As Chi et al. (1989) noted, the analogical references of the Poor solvers were qualitatively different from those of the Good solvers. The Poor solvers often started at the beginning of an example and read the whole thing, whereas the Good solvers started in the middle and read only a line or two. The latter behavior is consistent with analogical search control, because most of its references occur after the initial mapping is made, so the major purpose in rereading the example is to refresh one's memory of a specific line whose derivation is likely to contain the sought quantity. However, analogical search control is not consistent with constantly rereading the solutions from top to bottom, so the Poor solvers' behavior seems to have been a different kind of analogy. Their comments made it clear that they were hunting for a line that contained a quantity similar to the currently sought quantity. They did not seem to care how the line was derived. For instance, as long as it contained a tension and they were seeking a tension, then they were happy to use it just as if it were a valid equation from the domain. We implemented this kind of analogy in Cascade and named it *transformational analogy,* after a type of analogy studied by Carbonell (1983, 1986) that also ignores derivations. Cascade uses transformational analogy whenever an impasse cannot be resolved by explanation-based learning of correctness.

In principle, we could have transformational analogy learn new (probably incorrect) rules, just as Carbonell did. However, the subjects often commented that they hated hunting randomly for equations, so we doubt that they would believe that they had discovered a new rule of physics even if they did resolve an impasse using transformational analogy. Thus, Cascade does not create new rules when it uses transformational analogy.

Summary

Table 3 presents the main loop of the Cascade interpreter. On the first pass, Cascade uses only domain knowledge and not the overly general rules of explanation-based learning of correctness. If it fails to find a solution path, it makes a second pass using the overly general rules as well as the domain rules. If this fails, then a third pass is made and impasses are settled by analogy abduction. The "create" statements (e.g., Step 1g) indicate storage of new information in long-term memory. All such additions to the knowledge base are undone if backtracking goes through them. Thus, only the information created along the solution path survives.

The current version of Cascade does not adequately model the difference between retrieving information from memory and retrieving it from the

TABLE 3
The Main Loop of Cascade's Rule Interpreter

In problem P, to find a value V for quantity G or to show that V is the value of quantity G, try these methods in order until one succeeds:

1. Analogical search control.
   Do the following five steps in order, failing if any one fails and the failure can't be handled:
   a. Retrieve an example E that is similar to P.
      If retrieval fails, then
      flip pages looking for an example with a diagram that is similar to P's diagram.
   b. Retrieve a mapping between E and P.
      If retrieval fails, then
        reread problem statements of E and P, and
        create a mapping.
   c. Using the mapping, substitute terms in G to form a target goal T.
   d. Retrieve a triple (E T R), where R is bound by retrieval to a rule.
      If retrieval fails, then
        reread lines of E's solution to stimulate recall.
        If rereading lines stimulates only partial recall, then
          redo the derivation of the line that stimulated partial recall, and
          retrieve a triple from the new derivation.
        If rereading lines fails to stimulate recall, then
          redo the whole derivation, and
          retrieve a triple from the new derivation.
   e. Show that R's conditions are met.
   f. Apply R's equation to G and V.
   g. Create a triple (P G R).
   h. Return whatever Step f returned.

2. Regular rule selection and application.
   Do the following steps in order, failing if any one fails and the failure cannot be handled:
   a. Retrieve a domain rule (or any rule if this is not pass 1)
      whose equation contains a quantity unifying with G and
      whose condition is met by the current situation. Call the rule R.
   b. Plant a backup point so that a different rule can be
      retrieved if R leads to failure.
   c. Apply R to G and V.
   d. If R is an overly general rule, then
        create a specific version of the rule by instantiating R
          and substituting variables for problem-specific constants.
        Call this new rule R and
        mark it as a domain rule of P's task domain.
   e. Create a triple (P G R).
   f. Return whatever Step c returned.

(continued)

TABLE 3 *(Continued)*

3. Transformational analogy.
   If a problem is being solved, then
   do the following steps in order, failing if any one fails and the failure cannot be handled:
   a. Retrieve an example (as in Step 1a).
   b. Retrieve a map (as in Step 1b).
   c. Create a target goal T via mapping G (as in Step 1c).
   d. Retrieve a line of the example that contains T.
      If retrieval fails, then reread each line to see if it contains T.
   e. Substitute terms in the line via the map to put it in terms of P.
   f. Apply the line's equation to G.
   g. Return whatever Step f returned.

4. Analogy abduction.
   If this is the third pass, and
   an example is being explained and a value V for G is known, then
   do the following steps in order, failing if any one fails and the failure cannot be handled:
   a. Create an analogy rule R (see text), and
   b. Mark it as a domain rule of P's task domain, and
   c. Create a triple (P G R).
   d. Return success.

5. Impasse: No rules apply to G.
   If there are backup points, then resume one,
   else if this is Pass 1, then start over with Pass 2,
   else if this is Pass 2 and an example is being explained, then start over with Pass 3,
   else fail utterly. This problem/example cannot be solved/explained.

To apply an equation E to a quantity G when the value is unknown:
1. Let S be all quantities in E except G.
2. Recurse to find the values of each quantity in S.
3. Substitute values for quantities in E.
4. Solve E for G.
5. Return the result as G's value.

To apply an equation E to a quantity G when the value V is given:
1. Solve E for G, obtaining expression X.
2. Match X to V, obtaining a set S of quantity-value pairs.
3. Recurse to show that each quantity in S has the value with which it is paired.
4. Return success.

external world (e.g., the worksheet with the problem written on it). It is not difficult to add such a distinction, and Table 3 shows the main loop as if the distinction were already embedded in Cascade. The added code, shown in italics, specifies strategies for retrieving information from the external world whenever a memory retrieval fails. For instance, Step 1a claims that subjects attempt to retrieve an analogous example from memory and, if that

fails, to flip through pages of the textbook looking for an example whose diagram matches the problem's diagram. At this point in Cascade's development, we do not want to defend any particular memory model. One has been included here in order to clarify the relationship between Cascade's activities and the subjects' activities.

MODELING THE SELF-EXPLANATION EFFECT WITH CASCADE

A simple hypothesis for explaining the difference between Good and Poor solvers is that Good solvers chose to explain more example lines than did Poor solvers. This in turn caused more learning and hence better performance and all the other differences between the Good and Poor solvers. This hypothesis is nearly vacuous unless one specifies exactly how explaining examples causes learning. Cascade is such a specification. In this section we report a test of the conjoined hypotheses (a) that Cascade models learning in the Chi et al. (1989) study and (b) that the root cause of the self-explanation effect is that Good solvers explained more example lines than did Poor solvers.

Several simulation runs were made, varying the number of example lines explained and turning on and off various learning mechanisms. All these simulations began with the same initial knowledge state. Before the runs are described, the initial knowledge state is described along with the method used to determine it.

Initial Knowledge

Cascade represents knowledge in many ways. Much of the knowledge is provided initially rather than learned. The algebraic knowledge that is built into the interpreter is, of course, provided initially. Commonsense knowledge about classes of objects is provided initially as Prolog code. Initial knowledge of physics was encoded as 29 rules (conditioned equations) that were derived as follows. First, an extensive task analysis and simulation were conducted with the aid of Bernadette Kowalski and William Ball. Starting with the task analyses of Bundy, Byrd, Luger, Mellish, and Palmer (1979), Larkin (1981, 1983), and Novak and Araya (1980), a set of rules and a representation of physics problems were developed that were simple and yet sufficient for solving all but 2 of the 25 problems in the Chi et al. (1989) study (solving the 2 problems would have required a type of mathematical reasoning that we did not bother to implement). During this time, extensive informal analyses of the Chi et al. protocols were conducted in an effort to align the proposed knowledge representations with the subjects' comments. The resulting target knowledge base contained 62 physics rules. Next, two

people who were not involved in developing the target knowledge base were asked to judge each rule and determine whether it was mentioned anywhere in the text (i.e., the textbook before the examples). There was 95% agreement between the judges. Disagreements were settled by a third judge. Of the 62 rules in the target knowledge base, only 29 (47%) were judged to be present in the text. These rules were given to Cascade as its initial knowledge of physics for the runs that simulated the self-explanation effect.

The remaining Cascade knowledge consists of 44 rules used by explanation-based learning of correctness (EBLC). There are three kinds, which will be described in turn (see Table 4).

Eleven rules are overgeneralizations of common patterns of scientific inference. For instance, Rule 1 says that the property of a whole often has the same value as the property of a part. Cascade used this rule to learn a new domain rule that says that the pressure in a whole container is equal to the pressure in one of its parts.

There are 28 rules that encode knowledge about commonsense physics (not shown in Table 4). Most of these rules describe commonsense forces (i.e., pushes and pulls). Some of these rules, for instance, state that a compressed spring pushes and a stretched spring pulls.

There are six overly general rules that link commonsense physics to proper physics. EBLC uses these rules to relabel commonsense quantities as proper physics quantities. Because only a specialized version of the relabeling rule is kept, Cascade converts commonsense quantities to formal physics quantities one at a time. It does not learn the sweeping (and incorrect) generalization that all commonsense quantities are also formal physics quantities. In the process of relabeling commonsense concepts, EBLC also gives them a more mathematical formulation. For instance, when a commonsense force is turned into a formal physics force, EBLC gives it explicit vectorial properties, namely, magnitude and direction. This approach to learning scientific concepts seems plausible, considering that many scientific concepts, such as force or acceleration, are modifications of lay concepts.[2]

[2]There is a whole literature on the development of scientific concepts (for a recent discussion, see Chi, in press). Some theorists (e.g., Carey, 1985) believe that acquisition of scientific concepts like force require a complete restructuring of the subject's belief systems, similar to the Kuhnian paradigm shifts that supposedly accompanied the historical development of such concepts. Other theorists (e.g., di Sessa, 1988) believe that acquisition of scientific concepts results from gradual modification of naive concepts. Cascade shows that the gradual-acquisition account is computationally sufficient, at least for physics concepts such as force and acceleration, provided that the learner has already distinguished between formal and naive physics and has erected an equation-based representation for formal physics. According to Chi, seeing forces, accelerations, etc. as formal quantities rather than substances possessed by objects is the crucial step that lays the foundation on which gradual acquisition mechanisms, such as Cascade's, can build.

TABLE 4
Rules Used by Explanation-based Learning of Correctness

Overly general rules about properties and values

1. If P is a part of W, then the value of a property for W is the value of that property for P.
2. If P is a part of W, then the value of a property for P is the value of that property for W.
3. If P1 and P2 are parts of W, then the value of a property for W is the value of a property for P1 and P2.
4. If P1 and P2 are parts of W, then the value of a property for P1 and P2 is the value of a property for W.
5. If P is a part of W, then the value of a property for W is the perpendicular to the value of that property for P.
6. If P is a part of W, then the value of a property for W is the opposite of the value of that property for P.
7. If a structural predicate relates object A to object B, then there is a force from B on A.
8. If a structural predicate relates object B to object A, then there is a force from B on A.
9. If the value for a property P1 of object X is equal to the value for that property of object Y, and property P2 can be derived from P1, then the value for P2 of object X is equal to the value for that property of object Y.
10. If the value for a property P1 of object X is equal to the value for that property of object Y, and property P2 is derivable from P1, then the value for P2 of object Y is equal to the value for that property of object X.
11. A property P of an object is the magnitude of a force of type P from the object on a body.

Overly general rules that link common sense with physics

39. If F is a commonsense force, then it is a physics force.
40. If a commonsense force F has a property P, then the analogous physics force has the same property.
41. If A is a commonsense acceleration, then it is a physics acceleration.
42. If a commonsense acceleration A has a property P, then the analogous physics acceleration has the same property.
43. If X is a set of commonsense axes, then it is a set of physics axes.
44. If F is a force from S on B, then the sense of the force depends on whether it "pushes" or "pulls" on B.

The Simulation Runs

Run 1 was intended to simulate a very good student who explains every line of every example. Cascade first explained the three 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. Table 5 lists the rules acquired. The number of times each was used appears in square brackets. All these rules were acquired by EBLC

TABLE 5
Rules Learned by Cascade During Run 1

1. If X slides on Y, then there is a normal force from Y on X. [9]
2. If there is a normal force from Y on X, then the sense of the force is the relative position of X with respect to Y. [9]
3. If there is a normal force from Y on X, then the incline of the force is perpendicular to the incline of Y. [9]
4. The axes can be chosen from any two perpendicular vectors in the free-body diagram. [2]
5. If X slides down Y, then the sense of the acceleration of X is down. [9]
6. If X slides down Y, then the incline of the acceleration of X is the incline of Y. [5]
7. If the magnitude of the displacement of X is equal to the magnitude of the displacement of Y, then the magnitude of the acceleration of X is equal to the magnitude of the acceleration of Y. [6]
8. If X floats in Y, then there is a buoyant force on X due to Y. [1]
9. If there is a buoyant force on X due to Y, then the incline of the force is 90°. [1]
10. If there is a buoyant force on X due to Y, then the sense of the force is up.
11. The "tension of X" means the magnitude of the tension force on something due to X. [1]
12. If Y is a pusher and tied to X, then there is a compression force on X due to Y. [1]
13. If there is a compression force on X due to Y, then the incline of the force is the incline of Y. [1]
14. If there is a compression force on X due to Y, then the sense of the force is the relative position of X with respect to Y. [1]
15. If X is an object and Y supports X, then there is a pressure force on X due to Y. [1]
16. If there is a pressure force on X due to Y, then the incline of the force is the incline of Y. [1]
17. If there is a pressure force on X due to Y, then the sense of the force is the relative position of X with respect to Y. [1]
18. If Y is a piece of X, then the pressure of X is equal to the pressure of Y. [1]
19. The "pressure of X" means the magnitude of the pressure force on something due to X. [1]
20. If an object X moves through the air Y, then there is a friction force on X due to Y. [1]
21. If there is a friction force on X due to Y, then the incline of the force is the incline of the velocity of X. [2]
22. If there is a friction force on X due to Y, then the sense of the force is opposite to the sense of the velocity of X. [2]
23. If the current situation is analogous to situation_sx, and the current problem is analogous to sx, then the bodies of the current problem are analogous to [knot_sx]. [3]

except Rule 23, which was acquired by analogy abduction. 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. However, some of the rules dealt with situations that occurred only once in this problem set, so they were never used after their acquisition.

Run 2 was intended to simulate a very poor student who explains none of the example lines. To simulate a student who merely reads an example without explaining it, the lines from the three examples were made available for transformational analogy but were not explained. Thus, there was no opportunity for EBLC and analogy abduction to learn new rules and there were no derivations left behind to act as search control for later problem solving. Cascade was given the same 23 problems it was given in Run 1. It correctly solved 9 problems. As it solved these problems, it acquired 3 correct rules via EBLC. On 6 problems, Cascade found an incorrect solution, during which time no rules were acquired. On the remaining 8 problems, either Cascade failed to find a solution or its search went on for so long that it was cut off after 20 min. Although EBLC was used extensively, 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, rules acquired during failed solution attempts were deleted.

Run 3 was intended to separate the benefits of EBLC from the benefits of analogy. Cascade studied the examples as in Run 1, learning the same 8 rules as in Run 1. During problem solving, both analogical search control and transformational analogy were disabled. As expected, it answered only 19 of 23 problems correctly. A large interaction was found with EBLC. When analogy was not used during problem solving, EBLC learned 10 rules, only 6 of which were correct. Moreover, 3 of the 6 rules were the same 3 that it had learned on Run 2. Thus, of the 15 rules learned during problem solving on Run 1, 3 could be learned without benefit of the rules learned during example studying, 3 others required the example studying rules but could be learned without analogy, and the remaining 9 required both analogy and the example-studying rules. This finding makes sense. Analogical search control and, to a less extent, transformational analogy influence the exact location of impasses, which in turn determine the rules learned by EBLC. Their influence is strong enough that analogy is necessary for EBLC to learn 9 of the 15 rules (60%) acquired during Run 1's problem solving.

In order to determine whether this effect was due to transformational analogy or analogical search control, a fourth run was conducted that was similar to Run 3 except that only analogical search control was disabled. Cascade still used transformational analogy. This allowed it to get 2 more problems correct, raising its score to 21 of 23 problems. More important, EBLC acquired the same six correct rules as in Run 3. The fact that no further correct rules were acquired implies that it was analogical search control and not transformational analogy that helped EBLC during Run 1. Thus, it appears that analogical search control (or some other kind of search control) is necessary during problem solving if EBLC is to learn successfully.

Table 6 summarizes the results of the four runs. The processes turned on during each run are listed beside the run's name.

TABLE 6
Results of the Simulation Run

Run 1: Self-explanation, analogical search control, transformational analogy	
8	Rules learned during example studying
15	Rules learned during problem solving
23	Total rules learned
23	Problems solved correctly
Run 2: Analogical search control, transformational analogy	
0	Rules learned during example studying
3	Rules learned during problem solving (all correct)
3	Total rules learned
9	Problems solved correctly
Run 3: Self-explanation	
8	Rules learned during example studying (same rules as Run 1)
10	Rules learned during problem solving
3	Same as rules learned during Run 2
3	Other correct rules
4	Incorrect rules
18	Total rules learned
19	Problems solved correctly
Run 4: Self-explanation, transformational analogy	
8	Rules learned during example studying (same rules as Run 1)
9	Rules learned during problem solving
6	Same as correct rules learned during Run 3
3	Incorrect rules
17	Total rules learned
21	Problems solved correctly

Explaining the Self-Explanation Findings

Cascade was expected to be able to explain the four differences between Good and Poor solvers observed by Chi et al. (1989). Assuming that the number of self-explanatory utterances was directly proportional to the number of lines explained during example studying, the job facing Cascade was to demonstrate that explaining more lines caused better scores on problem solving (Finding 1), more accurate self-monitoring (Finding 2), less frequent reference to the examples (Finding 3), and more specific reference to the examples (Finding 4).

The contrast between Runs 1 and 2 indicates that Cascade was able to reproduce the positive correlation between the number of example lines explained and the number of problems solved correctly. On Run 1, it explained all the example lines and got all 23 problems correct; on Run 2, 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 (Runs 1 and 2) plus Cascade's deterministic design are sufficient to demonstrate the main correlation of the self-explanation effect.

Several mechanisms contributed to this result, and each is examined in turn. 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 causes EBLC and analogy abduction to construct new rules during example explaining. Of the 20 rules that were learned during Run 1 and not Run 2, 8 (40%) were learned while explaining examples. As the domain knowledge becomes more complete, performance on problem solving rises. Thus, the more self-explanation, the more rules learned during example studying, and hence the more improvement in problem solving.

The acquisition of rules during example studying helps produce contexts that allow EBLC to learn more rules during problem solving even without the aid of analogical search control. For instance, one rule learned during example studying selects a body for resolving forces about. This rule is necessary for traversing the correct solution path for some problems, which in turn is necessary for acquiring certain rules. Learning this rule during example studying allows EBLC to learn new rules during problem solving, and some of these new rules can be learned even without the guidance of analogical search control. Of the 20 rules, Run 3 shows that 3 (15%) were acquired in this fashion. These new rules also contributed to the improvement in problem solving.

Analogical search control raises the test scores both directly and indirectly. When more lines are explained, more derivational triples are stored and available for analogical search control. Because analogical search control prevents Cascade from going down some dead ends, it directly raises the score during problem solving. There is an indirect effect as well. Analogical search control causes impasses to occur at places where knowledge is truly missing, rather than at local dead ends in the search space, so EBLC is more often applied to appropriate impasses and thus more often generates correct domain rules. The remaining 9 of the 20 rules (45%) require analogical search control for their acquisition.

There is a simple explanation for the finding that Good solvers made more accurate self-monitoring statements. We assume that negative self-monitoring statements (e.g., "I don't understand that") correspond to impasses and positive self-monitoring statements (e.g., "Ok, got that") occur with some probability during any nonimpasse situation. When more example lines are explained, there are more impasses, and hence the proportion of negative self-monitoring statements will be higher. In the extreme case of Run 2, in which no example lines were explained, all the

self-monitoring statements during example processing would be positive, which is not far off from Chi et al.'s (1989) observation that 85% of the Poor solver's self-monitoring statements were positive.

Chi et al. (1989) observed that during problem solving, the Good solvers made fewer references to the examples than did the Poor solvers (2.7 vs. 6.7 references per problem). These were mostly physical references, wherein the solver turned to the example and reread part of it. Currently, Cascade does not distinguish memory references from physical references. However, it does have two different kinds of analogical references. Analogical search control searches for a sought quantity in the derivation of a problem. Transformational analogy reads consecutive lines in an example, looking for one that contains the sought quantity. Suppose we assume that all of the transformational analogy references are physical and that a small proportion, say P, of the references due to analogical search control are physical. On the Good solver run, Cascade made 551 references for analogical search control and 40 for transformational analogy. Using the preceding assumption, Cascade would make $551P + 40$ physical references. On the Poor solver run, Cascade could not use analogical search control because no derivations were available from explaining examples. However, it made 91 references for transformational analogy. If $P < .092$, then $551P + 40 < 91$ and Cascade would correctly predict that the Good solvers make fewer physical references than Poor solvers.

Chi et al. (1989) observed that the Good solvers read fewer lines when they referred to examples than did the Poor solvers (1.6 vs. 13.0 lines per reference). Cascade can model this effect, although an assumption is again needed about the percentage of analogical search control references that are physical. Suppose we assume that P of the analogical search control references are physical and that a physical reference by analogical search control reads only one line. On the Good solver run, Cascade read 340 lines during transformational analogy and $551*P$ lines during analogical search control, for a total of $(551P + 340) / (551P + 40)$ lines per reference. On the Poor solver run, Cascade read 642 lines, for $692 / 71 = 7.1$ lines per reference. If $P > .017$ then $(551P + 340) / (551P + 40) < 7.1$ and Cascade correctly predicts that the Good solvers read fewer lines per reference than the Poor solvers.

Notice that the lower bound (.017) on P does not have to be beneath the upper bound (.092). If the P had to be above, say, .1 in order to get the lines-per-reference finding correct and below .05 in order to get the reference frequency finding correct, then Cascade could not model both these findings. Thus, these findings jointly have the power to test Cascade, and it passed their test.

Figure 2 summarizes the preceding arguments. It shows the major

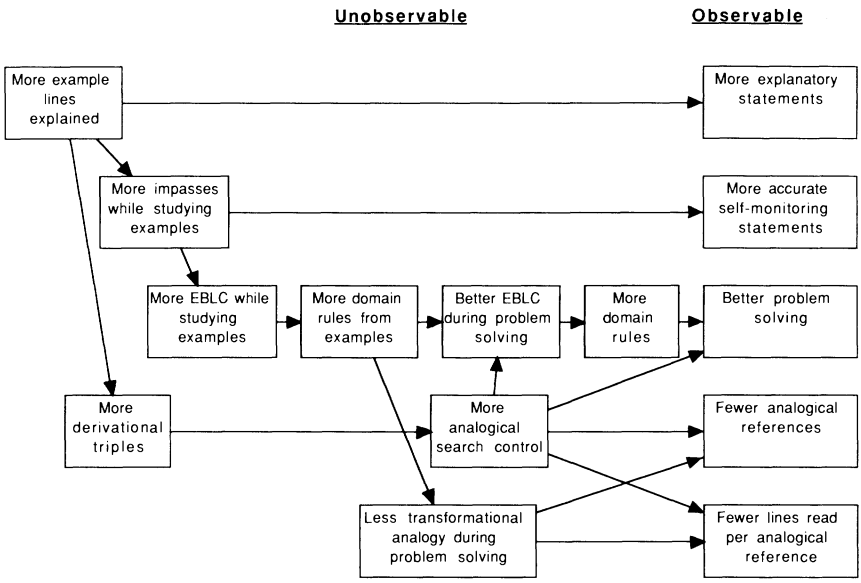

FIGURE 2 Causes of the self-explanation effect. EBLC = explanation-based learning of correctness.

processes and effects in the Cascade model and how they account for the self-explanation correlations.

Testing Cascade

Because Cascade was built to simulate the self-explanation effect, it probably seems unsurprising that it succeeded. In this section, we argue that it should be surprising because Cascade could easily have failed to simulate the study's findings.

The hardest test to pass was to get Cascade to learn as much as the Good solvers learned. One subject got all the problems right, so it is likely that she learned all 23 to-be-learned rules. To get Cascade to learn as much required overcoming two hurdles. The first was to get EBLC to occur on the right impasses. This is not so hard to achieve during example studying, but it is very hard to achieve during problem solving. We were surprised and relieved to see that analogical search control sufficed. The second hurdle was to supply overly general rules that created the right sort of domain rules when they were specialized. The new rules must neither be too specific nor too general. We were surprised to find that appropriate transfer was

obtained with an obvious generalization heuristic: Instantiate the rule then substitute variables for constants that are specific to problems.

The next hardest test to pass was to get Cascade to learn as little as the Poor solvers did. Two thirds of the to-be-learned rules occurred in the problems. The Poor solvers worked just as hard as the Good solvers on solving problems, yet they did not seem to learn as much during problem solving. Why do two sets of students learn different amounts from the same training material? To put it in terms of the Cascade model, the Poor solver simulation reaches even more impasses than the Good solver simulation during problem solving, so why does it not learn more than the Good solver simulation?

In fact, the Poor solver learned lots of rules during problem solving, but most of them were deleted because the Poor solver failed to answer most problems. The Poor solver simulation spent most of its time floundering because it lacked key rules that were acquired by the Good solver during example studying and because it lacked analogical search control. As it floundered about, it reached many impasses, but they were not the right impasse in that the rules learned at these impasses were not correct. Moreover, resolving these impasses let the Poor solver continue along a garden path that never terminated in a solution. When the Poor solver finally quit (actually, runs were halted by the experimenter after 20 min), the rules it learned were deleted. This behavior is consistent with a preliminary analysis by Chi, VanLehn, and 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 30 of these resulted in impasses. (An impasse was identified as an outcome of an episode whenever the student believed that the next step that should have been executed could not be performed. Some 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"). Many of these impasses seemed to result in acquiring incorrect beliefs. In contrast, the protocol of the Good solvers was divided into 31 episodes, only 7 of which 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 have proportionally more impasses (39%) than do the Good solvers (23%) while problem solving and that the resulting knowledge is more often incorrect. This is just what Cascade does, too.

The third test to pass was to get Cascade to simulate findings on analogical reference. Because Cascade lacks a model of memory, only partial success can be claimed here. However, the calculations in the Testing Cascade subsection show that it had a chance of failing the test, but succeeded nonetheless.

Although Cascade passed these tests, it is clear that the amount of testing

was small relative to the number of assumptions that underlie Cascade's design. It is not at all clear from these tests whether all the assumptions are necessary. In the next section we delineate the major hypotheses upon which Cascade is built and try to give empirical support to each one. Whenever possible, Cascade's hypotheses are compared with alternative hypotheses. Thus, the next section forms a partial competitive argument for Cascade. Although readers can skip it and go directly to the article's final section, it reveals much more of the model and its empirical support than any of the material presented so far.

MAJOR HYPOTHESES

The major hypotheses that together constitute our account for the self-explanation effect also function as design principles for the implementation of Cascade. Thus, in this section we have the dual task of introducing the major assumptions about cognition that underlie the Cascade's design as well as arguing on the basis of the protocol data that these assumptions are reasonable ones to make for this study. Unfortunately, many of the hypotheses that could be supported with quantitative protocol analysis have not been. We are frequently reduced to making statements like "There are many cases of such-and-such" or "No subject said such-and-such." Such statements should be supported by coding the protocols and counting the number of codes. Because there are about 3,000 pages of protocol, we have done this in only a few cases and relied on our memory of the protocols for the others. The memory-based statements should be understood as disclosing our motivations for choosing the hypotheses rather than providing solid empirical warrants for them. So far, the major formal empirical support for the hypotheses comes from the demonstration that Cascade can model the self-explanation effect findings, although this too is not as constraining as one would ideally like. (The final section of this article presents plans for further testing.)

This section has one subsection for each Cascade hypothesis. The first subsection introduces and justifies the hypothesis that the self-explanation effect is due to knowledge acquisition methods that occur during both example explaining and problem solving. The next subsection argues that the acquired knowledge is small relative to the size of the problems being solved. That is, the students learn rules rather than cases. This opens two issues, which are addressed in the subsequent subsections. First, how do students detect when a rule is missing and needs to be learned? Second, what methods are used to acquire the new rule? The last few subsections consider important details about the representation of rules and the explanation processes used by Good and Poor solvers.

Hypothesized Sources of the Self-Explanation Effect

Cascade is based on the hypothesis that the self-explanation effect is caused by knowledge acquisition that occurs as the students explain examples and solve problems. Let us introduce this hypothesis by first examining competing hypotheses.

A plausible hypothesis is that the two groups of students accumulated different knowledge of physics just before studying the examples. This difference could be due to either reading the text of the chapter more carefully or having knowledge of physics. The students who had more prior knowledge solved more problems correctly and thus were classified as Good solvers. Under this prior knowledge hypothesis, all subjects try to explain the text and the example lines, but those who have more prior knowledge are better able to explain the example and so produce more self-explanations (Finding 1 in the earlier list). Moreover, because they produce more derivations during example processing, they use fewer references (Finding 3) and more specific references (Finding 4) during analogical problem solving. Thus, the prior knowledge hypothesis is consistent with three of the four findings.

There are, however, three sets of evidence against the prior knowledge hypothesis.

1. The prior knowledge hypothesis predicts that Poor solvers would utter more negative self-monitoring statements because they more often fail to explain a line. In fact, they uttered fewer negative self-monitoring statements (Finding 2).

2. After reading the text of the target chapter, the students in the Chi et al. (1989) study took a test on their knowledge of Newton's laws (Phase 4 in the earlier description). The mean scores of the Good and Poor solvers on this test were exactly the same. Although affirming the null hypothesis with so few subjects is risky, taking the results at face value suggests that both groups of students had roughly the same prior knowledge.

3. Chi and VanLehn (1991) conducted a finer grained analysis of all the self-explanations in the protocols, reducing them to a set of 173 distinct propositions. For each proposition, they attempted to determine whether it was inferred from (a) the example line, (b) commonsense knowledge, (c) knowledge acquired from previous example lines, or (d) the text. If the Good solvers had more prior knowledge at the time they began studying the examples, more of their propositions would be encoded as coming from the text. However, the proportion of text sources (Category d) was the same for both Good and Poor students, which is inconsistent with the prior knowledge hypothesis.

Although it is unlikely that all students had exactly the same prior knowledge, the aforementioned difficulties indicate that variations in prior knowledge cannot be the sole source of the self-explanation effect. There must have been some kind of learning going on.

Because the subjects were explaining examples, a plausible type of learning is explanation-based learning (EBL; Mitchell, Keller, & Kedar-Cabelli, 1986). Like proceduralization (Anderson, 1983) and chunking (Newell, 1990), EBL is a kind of knowledge compilation, in that all the knowledge is assumed to be present in some form before the learning begins. Learning consists of making the knowledge more efficiently usable. However, the hypothesis that self-explanation is caused by knowledge compilation has four difficulties.

1. When the subjects took an untimed test on Newton's laws after reading the text (Phase 4), their mean score was only 5.5 out of a possible 12. This suggests that students did not know much physics after studying the text. After studying the examples and solving the problems, the Good solvers' mean score increased to 8.5 and the Poor solvers' mean score remained at 5.75. This suggests that additional knowledge was acquired by the Good solvers from studying the examples and working the problems.

2. The text did not contain all the information needed by the subjects to explain the examples or solve the problems. As explained earlier, a target knowledge base of 62 rules was developed and two judges determined which of the rules were covered in the text. Of the 62 rules, only 29 (47%) were judged to be present in the text before studying the examples. Thus, 33 rules representing more than half the knowledge required for explaining the examples and solving the problems were not presented in the text and presumably were not known by the subjects before they explained the examples and solved the problems.

3. Students took the same test on Newton's laws after studying the examples and solving the problems. Chi and VanLehn (1991; Table 6) showed that the aspects of Newton's law that were learned were the ones emphasized in the examples. This result is difficult to explain if students were merely recalling aspects of Newton's law learned during text processing but is quite consistent with acquiring knowledge of Newton's laws via example studying.

4. When Chi and VanLehn (1991) classified each of the 173 propositions found in the students' self-explanations as coming from text or from nontext sources, they found that at most 31.5% could be deduced from information presented in the text. This result is hard to explain given the knowledge compilation hypothesis, which predicts that propositions would be deduced from prior knowledge, of which the most important component is information presented in the text.

These results suggest that the major prerequisite of knowledge compilation was not met, because the students did not seem to have complete knowledge before the example studying and problem solving began. Thus, some kind of knowledge acquisition must have been going on during the explanation of examples and the solving of problems.

The text presented universal laws of mechanics, such as $F = ma$. Solving problems required more than these laws, however. It required many specific rules (or "constituent knowledge pieces," as they are called by Chi and VanLehn, 1991), such as

> When a string pulls on an object, there is a force on the object. The force is parallel to the string at the point of contact and directed away from the object. The magnitude of the force is equal to the tension in the string.

The text discussed only a few of these rules and only in a cursory manner. The others must have been acquired somehow from the examples and exercises. One might view this as a rare defect, an omission in this edition of the textbook that will surely be corrected in the next edition. However, this textbook was already in its third edition. Other science and engineering textbooks follow the same conventions for what to explain in the text and what to leave unsaid. Mathematics textbooks are even less complete. They often make no attempt to present informal rules necessary for solving word problems. We believe that knowledge acquisition during example studying and problem solving is endemic in formal schooling, and not at all idiosyncratic to this particular experimental study.

Because the examples contained more information than the problems, a plausible hypothesis is that all knowledge acquisition occurred during the explanation of examples. However, using the 33 rules that did not occur in the text, we estimate that only 11 of the rules were used during the examples. The other 22 were first used during the problems because they dealt with situations and objects (e.g., springs) that did not appear in the examples. This suggests that two thirds of the rules were acquired during problem solving. Thus, it appears that some kind of knowledge-level learning is going on during both example explaining and problem solving. This is the hypothesis on which Cascade is based.

Derivation Completion

A prototypical knowledge acquisition task is concept formation, wherein the learner is presented with examples and is expected to generate a concept that describes them. For instance, a learner might be presented with examples of a certain species of flower and be asked to form an operative definition of the species (e.g., has four blue petals, a 5 mm stamen, etc.). In concept formation, the piece of knowledge to be learned is about the same

size as the elaborated example. However, in learning physics, the knowledge to be learned while studying an example is much smaller than the example's derivation. (We use *derivation* to stand for all the reasoning required to produce a correct answer to an example or a problem. We use *example lines* to stand for that part of the derivation that is printed as the example's solution.) A derivation might involve hundreds of rules, most of which are quite familiar to the learner because they were used in earlier derivations. A derivation may require only one or two new, unfamiliar rules.

Even though the knowledge to be learned is much smaller than the example's derivation, the learner could in principle use a concept formation approach and generate a description that covers the whole example. For instance, a physics student might say, "Not all of that solution makes sense, so I'll just remember that whenever the problem has two blocks attached to a rope that goes over a pulley, the solution is to write down those equations and solve them." Although we never saw such a statement in the protocols, this approach to learning appears computationally viable.

The Good solvers rarely take such an approach. Instead they rederived each line of the solution from the preceding lines. When they encountered a line that they could not derive, they tried to find the gap in their knowledge that was causing them trouble. For instance, one subject, P1, could not explain the line "Fax = $-$Fa cos 30°." Although she knew Fax was the projection of force Fa onto the x-axis and that the cos 30° was due to projecting a 150° vector onto the x-axis, she could not explain the negative sign. She wondered, "How did they get that negative in there?" After much work, she eventually concluded, "The reason the negative is there is because the x-component [of force Fa] is in the negative direction on the x-axis." This bit of explanation allowed her to finish explaining the line and eventually the whole example. Subject P1 was typical of the other Good solvers. They tried to localize the defect in their knowledge and then invented as small a piece of knowledge as necessary for overcoming the defect and completing the explanation.

Earlier work has also shown this approach to be computationally viable, and it has been independently invented many times (Ali, 1989; Anderson, 1977, 1990; Bergadano, Giordana, & Ponsero, 1989; Berwick, 1985; Danyluk, 1989; Fawcett, 1989; Genesereth, 1983; Hall, 1988; Lewis, 1988; Martin & Redmond, 1988; Pazzani, 1990; Pazzani, Dyer, & Flowers, 1986; Schank, 1986; Sleeman, Hirsh, Ellery, & Kim, 1990; Smith, 1982; VanLehn, 1987; VanLehn, Ball, & Kowalski, 1990; Widmar, 1989; Wilkins, 1988). The approach has no standard name, so we suggest *derivation completion,* because the essential similarity is that the learner guesses a new piece of knowledge that allows a derivation to be completed.[3] We hypoth-

[3]VanLehn (1987) used "learning by completing explanations" and Hall (1988) used "learning by failing to explain" for roughly the same class of learning systems. However, Cascade can

esize that derivation completion is the major approach used by our physics students for acquiring new knowledge and thus have built Cascade to embody it.

There are two major steps in derivation completion. First the learner must locate a gap in a derivation that warrants filling, and then the learner must find a new piece of knowledge that will bridge the gap. These two steps are discussed in turn.

Locating a Knowledge Gap

Localizing a gap in one's knowledge is difficult. The first sign of missing knowledge is an impasse: A goal cannot be achieved with any rule in the knowledge base. For instance, subject P1 could not achieve the goal "Show that the sign of the projection formula is negative." The existence of an impasse always indicates that some knowledge is missing, namely, the rules needed for achieving the goal. However, it is not immediately clear whether this defect is worth fixing. It could be that one has wandered off the solution path or made an unintentional error (a slip), which would make this impasse occur even if one had complete knowledge of the domain. In this case, the right response to the impasse is to back up and try again. P1 entertained this possibility explicitly. Just after she reached the minus sign impasse, she went back and checked her earlier work. Only when she had assured herself that there were no slips and no other ways to explain the line did she proceed with hunting for a new rule. In principle, there is no way to know whether a given goal is part of the derivation of a correct answer until one has actually generated the whole derivation. Thus, one can never tell in principle whether an impasse is worth inventing a rule for until one has tried it and obtained a derivation. All learners, computational as well as human, must use heuristics for guessing whether to back up at an impasse or invent a new rule.

We hypothesize that all subjects used the same heuristic as P1, and therefore we built Cascade to use this heuristic exclusively. Whenever

---

learn from problem solving as well as from example explaining, so the broader term "derivation completion" is more appropriate. The terms "impasse-driven learning" (VanLehn, 1986) and "failure-driven learning" (Schank, 1982) have similar extensions but exclude systems such as Sierra (VanLehn, 1987), which collect several incomplete derivations and compare them before deciding how to complete them. Although some derivation completion systems use plausible reasoning to fill in the gaps in an explanation, the term is meant to exclude systems, such as Cohen's (1990), DeJong and Oblinger's (in press), and Eskey and Zweben's (1990), that build an explanation with plausible reasoning then convert the whole explanation into new domain knowledge. These systems would be derivation completion systems if they located the weakest links in their chain of plausible reasoning and built small pieces of knowledge that are relevant to just those gaps; however, this is not what they do.

Cascade encounters an impasse, it backs up and explores all alternatives for generating a solution. (Currently, Cascade does not make slips, so it does not check for them.) Only when it fails to find an alternative route to a solution does it return to processing the impasse. Notice that Cascade returns to the original impasse. In trying to find alternative routes, Cascade may encounter other impasses. If one of these alternative routes is a correct solution path, then its impasse should be the one to resolve. But Cascade has no way to know with certainty which of all the routes it explored is most likely to be a correct solution path. It guesses that the first route it explored is most likely to be correct, so the first impasse it encountered is the best one to resolve.

Computational experiments show that this heuristic works well during example explaining, but works well during problem solving only if search control heuristics ensure that the first path explored is likely to be a solution path. When Cascade is explaining an example, the lines of the printed solution tend to keep it on a correct solution path even without the help of search control heuristics. When Cascade is solving a problem, there are no solution lines, so, without search control, it tends to wander off the solution path rather quickly. The first impasse reached is usually caused by being on a wrong path. Nonetheless, Cascade resolutely applies its heuristic, finds that all other paths are blocked, and sets about fixing the impasse by inventing a new rule. At best, this is a waste of effort. At worst, the newly acquired rule is not a correct rule of physics even though it caused the derivation to go through. Subjects do not run amok like this, so some kind of search control is needed to keep Cascade on the solution path. Such search control is needed only for solving problems because the example lines provide equivalent constraint during example explanation.

We hypothesize that the missing search control during problem solving is provided by analogies with examples' solutions. The protocols provide ample evidence of the use of analogy. Subjects often turned to the page with the example on it or mentioned the example as they worked. All 8 subjects used analogy some of the time (Chi et al., 1989). Example–exercise analogies are heavily used by subjects in other task domains as well (Anderson et al., 1984; Pirolli & Anderson, 1985). Most models of analogical problem solving divide the process into three phases: retrieving an example, forming a mapping between the example and the problem, and applying information from the example to the problem. These phases will be discussed in turn.

Retrieving an example seems to be governed by visual processing. All the examples in the study had a diagram, such as the one shown in Figure 1. All but five of problems had a diagram as well. Subjects seemed to use these diagrams to help them locate an appropriate example for the problem they were working on. For instance, one subject said, "This looks very much like

the one I had in the examples. Okay. Should I just go right to the problem [example], which I distinctly remember? I mean, even the angle is the same here. Or should I try to do it without looking at the example?" The subject said this before reading the text of the problem, so apparently her retrieval of the example was based solely on the diagrams. Even when a subject's memory for the diagrams failed, it was not difficult to find an appropriate example because there were only three examples to search through and they had very distinct diagrams. In all of the protocols, there was only one case where a subject tried and failed to find an appropriate example. Thus, it is not the case that Good solvers were better at analogical retrieval than were Poor solvers, because they were all at ceiling. Cascade does not model the processes involved in retrieval because they seem somewhat specific to this study and the resulting retrievals were not a source of differences between Good and Poor solvers. Cascade was simply given a function, analogical_retrieval, which takes the name of a problem as input and delivers one or more example names as output.

Forming a mapping between an example and a problem means deciding which objects in the example correspond to which objects in the problem. We noticed that the subjects' first look at an example was more extended than other references to the example during the same problem. We believe that during the first reference, the subject built an analogical mapping as well as checked to see if the example was analogous enough with the problem to warrant using it. The mapping was used during this initial reference to the example and all subsequent ones during the solution of the current problem. Because the problems were often quite similar to the examples, the subjects always found the same, correct mapping. The lack of individual variation made it difficult to infer the heuristics used by subjects to select mappings. Cascade uses a set of heuristics based on the types of the objects (e.g., physical objects can only be paired with other physical objects, quantities with quantities, etc.). These yield the mappings that subjects chose, but there is no way to tell from these data whether these heuristics are the ones actually used by the subjects.

The retrieval and mapping processes could be used to import many kinds of information from the example to the problem. In this case, the subjects needed search control information: Which rule should be used to achieve the current goal? We hypothesize that they used the mapping to convert the goal from the problem into an equivalent goal for the example and then searched the example's derivation for that goal. They might have said, for instance, "My goal is to find the tension of String A, and String 1 in the example is analogous to String A, so I'll look for the tension of String 1 in the example." The search for an equivalent goal required recall or reconstruction of the example's derivation because the example's printed solution did not contain goals, although perusal of the printed lines may have

stimulated the recall of the derivation. Having found an equivalent goal, subjects next needed to recall which rule they had used to achieve it. If they succeeded in this, then they were nearly done, because no further analogical mapping was required. The rule was presumably a generic structure whose variables could simply be instantiated in order to apply it to the current goal of the problem.

This whole process amounted to a search control heuristic: To achieve a goal, it is wise to use a rule that achieved an equivalent goal in an example that is analogous to this problem. We call this heuristic mechanism *analogical search control* (cf. Jones, 1989).

Methods for Filling Gaps in Derivations

After learners had located a gap in their knowledge, the next step in derivation completion was to find a piece of knowledge that would bridge the gap. Our hypotheses are that finding an appropriate piece of knowledge became a goal in itself, that subjects had multiple methods for achieving such goals, and that if a method succeeded in acquiring an appropriate piece of knowledge, the knowledge was stored in memory. P1 provides a clear illustration of this process. After she detected that she could not explain the minus sign in "$Fax = -Fa \cos 30°$," she still had to figure out how to bridge that gap by recalling or constructing knowledge that would produce the minus sign in this particular case. First she consulted a table of trigonometric identities, because, as she put it, "I remember them doing strange things with the trig functions being negative and positive for no apparent reason." When this approach failed, she next looked up cos 30 in a table, hoping that it would come out to be a negative number. When this failed, she began her third approach. The protocol reads:

> P1: Hmmm, negative cosine 30, why would they say, ahhh, ummm. . . . The, ohh, okay maybe it's just because the A component is the X component of force A is negative. So they just. . . . Well, okay, I'll, I'll try that for a while. Let's see if that works, 'cause that makes sense.
> E: What makes sense?
> P1: The reason the negative is there is because the X component is in the negative direction on the x-axis.

P1 did produce the correct rule, but it is not clear how she did it. Although she could have recalled it from her mathematics courses, we believe she was constructing it. She probably noticed that the vector lay above the negative part of the x-axis and then applied an overly general rule for mathematical calculations, which could be called *conservation of*

negativity. When a negative quantity is transformed, the resulting quantity is often negative even though the negativity might be expressed somewhat differently than it was in the original quantity. In this case, the negativity of the x-location of the vector was preserved as it was projected and became a formula. The negativity changed from a locative encoding to an explicit negative sign. The point is that P1 bridged the gap by adopting an explicit goal of finding knowledge that would complete the derivation. She tried three methods, and the last one succeeded. This allowed her to complete the derivation. Evidence is presented later showing that she actually learned a new rule from the experience.

The hypothesis is that subjects seek knowledge when they detect that they are missing some and that they use multiple methods to achieve their knowledge acquisition goals. This hypothesis is hardly novel—virtually all the derivation completion learning models use it. The models differ primarily in the knowledge acquisition methods they use. For the sake of exposition, methods found in the literature are grouped into several broad categories and discussed next.

Acquiring knowledge by reading. One way to acquire knowledge is to seek it in the textual part of the instructional materials. For instance, one subject could not explain the units in an example's equation because she did not know the American unit of mass. She looked it up in the text and presumably stored a rule in memory stating that slugs are the American unit of mass. This method of filling gaps in one's knowledge is the main method of knowledge acquisition in early versions of ACT* (Anderson, 1983).

To find out how much our subjects used this method, we counted all references to the text or the examples made by the subjects. Of 433 references, 129 (30%) were to the chapter's text. Few of these 129 references were as focused and successful as the aforementioned slugs episode. Most frequently, students hunted through the textbook for an equation containing the currently sought quantity. Any equation containing a quantity of that type will do. It could occur in the middle of an apparently irrelevant example or even in a different chapter of the textbook. Most of the subjects seemed to know that this method for bridging gaps was not likely to yield correct knowledge. They often made comments such as, "I hate doing this." We doubt that they believed the rules (if any) acquired from this activity were correct rules of physics. Because searching the textbook for equations occurred rather frequently in the protocols, Cascade has a model of it, called *transformational analogy*. Transformational analogy does not produce new rules when it occurs.

Acquiring knowledge by syntactic induction. A common technique for concept formation is to compare multiple instances of the concept and

conjecture that their common features are the defining properties of the concept. Using this and related techniques, students construct concepts by making syntactic comparisons of the instances; thus this method is often called *syntactic induction* or *similarity-based learning*. Syntactic induction can be used for constructing knowledge to fill gaps. The basic idea is to collect several instances of the same gap and compare them. For instance, P1 could have found several cases in which a minus sign appeared in a projection formula, compared them, and discovered that they possessed a common feature: The vector being projected was over the negative part of the axis onto which it was being projected.

Syntactic induction techniques for filling gaps have been used by many derivation completion programs (Ali, 1989; Danyluk, 1989; Fawcett, 1989; Hall, 1988; VanLehn, 1987; Wilkins, 1988). Often the basic mechanisms of syntactic comparison are supplemented by syntactic heuristics, such as preferring the smallest rule that will fill the gap. In some derivation completion programs, syntactic heuristics alone induce a rule from a single instance of the gap (Anderson, 1977; Berwick, 1985; Genesereth, 1982; Martin & Redmond, 1988; Sleeman et al., 1990; Smith, 1982).

A version of syntactic induction was implemented in an early version of Cascade, but it did not perform well. At an impasse, the inducer sees if there is a new rule whose conclusion matches the current goal but whose antecedent is false in the current situation. If it finds such a rule, it drops the mismatching parts of its antecedent, thus generalizing it. This allows the inducer to achieve the current goal and thus resolve the impasse. If no such rule is found, a new rule is created by making its conclusion be the current goal and its antecedent be the current situation, with variables substituted for problem-specific constants (objects and numbers).

We had not even fully implemented this method of knowledge acquisition before it became clear that it would have severe problems. First, most of the new rules are learned during problem solving, but this technique can invent a new rule only during example explaining. Second, the rules it invents are limited in the kind of conclusions they can draw. For instance, if the goal is to find the tension of string_A and the example shows that string_A has a tension of 5, then the new rule's conclusion will have the form tension(X) = Y, because variables X and Y have been substituted for the object and number. However if 5 does not appear anywhere in the given situation because it is the result of some arithmetic calculation, then the variable Y will not appear in the rule's antecedent. Thus, the rule draws a nearly useless conclusion: The tension of an object is something, but I do not know what. Although there are ways to syntactically induce arithmetic formulas (e.g., VanLehn, 1987), they require many more examples than the one or two available for formulating the new rules in this instructional situation.

The conclusion is that there do not appear to be enough examples of new

rule applications for syntactic induction methods to work in this instructional situation.

Acquiring knowledge by goal–product analogies. Another knowledge acquisition method is based on drawing analogies between problems and worked examples. Anderson's (1990; Anderson & Thompson, 1989) latest model of skill acquisition is typical of this technique. Anderson's model assumes that derivations of examples are available in memory and, in particular, that every goal in the derivation is paired with the external product generated by achieving it. Because most of Anderson's examples involve LISP programming, the external products of most goals are small pieces of LISP code. During problem solving, if the learner cannot find a rule to achieve a goal, it seeks a similar goal in the derivation of an example. If a goal is found, the learner converts the old goal's product into the terms of the current problem, thus creating a product for the current problem's goal. Proceduralization then creates a rule that summarizes the results of this analogical knowledge acquisition process. Similar knowledge acquisition methods have been used in other models (e.g., Lewis, 1988; Pazzani, 1990; Pazzani et al., 1986). They all map a goal–product pair from an example to a current problem. They ignore the derivation of the product from the goal. The difference between this technique and Cascade's analogical search control is that this technique imports the external product generated by processing a goal, whereas analogical search control imports the name of the rule used to achieve a goal.

A version of goal–product analogy was implemented in Cascade but proved to have limited utility. As with syntactic induction, this technique fails when the to-be-learned rule first appears during problem solving. Because there is no earlier application of it during example studying, there is no early goal–product pair to which to refer. Even when the technique finds an appropriate goal–product pair, it often fails anyway. In physics, the most common goal is to seek the value of a quantity. The external product of such a goal is usually a number or a vector, and such atomic entities do not usually map successfully. For instance, suppose the problem's goal is to find a tension for String A, and the example's derivation says that String 1's tension is 5 Newtons. An analogical map can pair the two strings, the two goals, and the units (Newtons), but what should it pair with the 5? The 5 was calculated by simplifying an arithmetic expression, weight(block1) / [sin(3) − cos(30)/sin(45)], where weight(block1) = 10 Newtons appears in the example's givens. This expression can be mapped from the example to the problem by substituting the problem's block for block1 and the problem's angles for 30 and 45. However, after the expression is simplified to 5, there is nothing left to map. Simplification destroys information that is needed for analogical mapping. We conjecture

that goal–product analogy works adequately only when the external product of a goal is nearly isomorphic with the derivation used to produce it (cf. Carbonell, 1986). That is why it works so well for LISP but not for physics.

Explanation-based learning of correctness. Another knowledge acquisition technique, which we call *explanation-based learning of correctness* (VanLehn, Ball, & Kowalski, 1990), fills a gap by applying an overly general rule, which is not normally used during reasoning. If this application leads ultimately to a successful derivation, then a specialization of the rule is created and inserted into the set of rules that is normally used during reasoning. The analysis of P1 presented at the beginning of this section is an illustration of explanation-based learning of correctness, where "conservation of negativity" is the overly general rule. The same basic idea appears in many forms in the literature. Schank's (1986) explanation patterns are a kind of overly general rule used to bridge gaps in explanations of human interest stories. Causal attribution heuristics are used by many theorists to explain how subjects bridge gaps in explanations of the physical world (Anderson, 1990; Lewis, 1988; Pazzani, 1990). Several authors use determinations (Davies & Russell, 1987) as constraints on learning (Bergadano et al., 1989; Widmar, 1989). Goodman (1956) uses over hypotheses to explain scientific induction.

Any version of this method for filling gaps requires distinguishing between knowledge that is normally used and knowledge that is reserved for bridging gaps. Explanation patterns and causal attribution heuristics are expressed in a different format from the knowledge used normally in making explanations. Explanation-based learning of correctness uses the same representation for both types of knowledge but keeps them distinct by marking the rules that are normally used for solving problems in the task domain with the name of the domain (e.g., "physics"). This will make it easier to augment Cascade with a module that acquires overly general rules by syntactic generalization of normal rules (Ram, 1990; VanLehn & Jones, in press).

The hypothesis that domain rules are marked seems necessary to account for some common aspects of classroom problem solving. If students acquire an incorrect rule and then later learn that it is incorrect, they probably do not forget the rule even though they stop using it. Thus, there must be some way of indicating which rules should not be used even though they are potentially applicable. This could be represented by removing that task domain's mark from the rule. Similarly, subjects rarely use rules from other task domains even when they are potentially applicable. For instance, while explaining an example in which a block was sliding on a surface, one subject apparently knew from common sense that the block will not jump up or sink into the surface, but she could not prove it with her current physics

knowledge. After trying several approaches, she gave up and commented, "The only way you could have known that there's no acceleration in the y-direction is not from equations but from just knowing something about the situation." We believe that subjects prevent themselves from considering such knowledge by marking only some of the knowledge they have about blocks, surfaces, etc. as formal physics knowledge. We suspect that this system of marking is used only for task domains taught in school. It may be something that students learn to do early in school because the resulting reduction in search makes their problem solving more efficient and more often correct. Nonschool problems can be solved with knowledge of any kind.

As shown earlier, explanation-based learning of correctness works quite well. In particular, it is able to fill gaps that occur during problem solving, which is something that syntactic induction and goal–product analogy cannot do. However, it is unable to handle one learning event, wherein a knot is declared to be the body, so analogy abduction was added. Analogy abduction is similar to goal–product analogy.

Summary. The following list indicates which of the aforementioned learning methods are modeled in Cascade:

1. Reading: Searching the textbook for equations is modeled, but produces no new rules.
2. Syntactic induction: Was modeled, but currently turned off.
3. Goal–product analogies: Is modeled, but is rarely used.
4. Explanation-based learning of correctness: Is modeled and is frequently used.

There are other methods for bridging gaps in derivations, such as scientific discovery (VanLehn, 1991a), that do not appear in this list because we saw no signs of them in the protocols.

Goal Specificity of New Knowledge Pieces

Most of the goals in physics reasoning are to find a value for a quantity, and most of the inferences involve equations. Thus, a typical goal might be to find the tension in a certain string, and an equation to achieve that goal is tension(S) = magnitude(force(B,S)), which says that the tension in string S is equal to the magnitude of a tension force acting on body B due to string S. Suppose that a student acquires this rule at an impasse where he or she is trying to achieve the tension goal. Would this piece of knowledge be invertible, so that the student can use it to achieve a goal of finding the

magnitude of a tension force? Although the protocols are silent on this point, we believe that students are able to invert new pieces of knowledge. That is, we believe that when subjects have learned an equation, they can use it to find any of the quantities that the equation mentions. For instance, if they learn F = ma in a context in which net force is sought, they can nonetheless apply it in a context in which mass is sought. Although this particular claim is untested, there are some related transfer findings that provide indirect support.

Singley and Anderson (1989) reviewed experiments from Anderson's group that suggest that there are two kinds of transfer, procedural transfer and declarative transfer. Procedural transfer is use specific but develops only after practice. That is, when a person uses a piece of knowledge in one context several times and thus gets faster and more accurate at using it, this practice does not make it easier for them to use it in a new context. In terms of Cascade's task domain, suppose one group of subjects is taught that w = mg with examples and exercises in which weight is always the sought quantity. Another group of subjects is taught that w = mg with examples and exercises in which mass is the sought quantity. They are given an hour of practice, during which time their performance (speed and accuracy) increases. After the practice period, the group's tasks are switched and their performance is measured. Singley and Anderson would predict that the group who practiced seeking weights would do as poorly on seeking masses as the mass-seeking group did at the beginning of their practice period. Similarly, the mass-seeking group would look like weight-seeking novices. Thus, substantial practice causes procedural transfer, which is specific to the particular goals for which a piece of knowledge (w = mg, in this case) is put.

Declarative transfer is not use specific and does not require practice to develop. As an illustration of declarative transfer, suppose the practice periods of the preceding two groups is reduced to about a minute or two, so that each group uses w = mg only once or twice before being switched to the transfer task. Singley and Anderson (1989) would predict that both groups would perform about the same on their transfer task as their opposites did during training and, moreover, that both would do better on their transfer tasks than a control group who received no practice at all on w = mg before being tested. This illustrates declarative transfer: Knowledge of the equation w = mg, regardless of whether it is learned in the weight-seeking condition or the mass-seeking condition, is necessary and sufficient for the initial few uses of the equation for any purpose.

Cascade's derivation completion methods are knowledge acquisition methods, rather than knowledge compilation mechanisms. Thus, Singley and Anderson's (1989) results suggest that the knowledge constructed by these methods can be declaratively transferred. This suggests that they be

represented as equations, because equations are not specific to the goal that was present at the time the knowledge was acquired. Therefore, instead of representing w = mg as three production rules

If body(B), problem(P), mass(B) = M and grav_constant(P) = G, then weight(B) = M*G.

If body(B), problem(P), mass(B) = M and weight(B) = W, then grav_constant(P) = W/M.

If body(B), problem(P), grav_constant(P) = G and weight(B) = W, then mass(B) = W/G

Cascade should use one equation:

weight(B) = mass(B)*grav_constant(P).

However, this drops the condition that B be a body and P be the current problem, making it necessary to add applicability conditions. Thus, w = mg should be expressed as

If body(B) and problem(P), then weight(B) = mass(B)*grav_constant(P).

Cascade's interpreter must be more complex than a typical rule interpreter in order to use knowledge expressed in this form, because it must use algebraic transformations. As an illustration, suppose that Cascade is given the goal of finding mass(block4). To apply the preceding conditioned equation, it first shows that body(block4) and problem(prob3) hold, then it sets the subgoals of finding the other quantities in the equation, weight(block4) and grav_constant(prob3). Achieving these subgoals means that the values of the two quantities become known. Suppose the weight is 98 and the gravitational constant is 9.8. The interpreter must combine the values for the subgoal quantities to form a value for the goal quantity. Substituting the subgoals' values into the equation yields 98 = mass(block4)*9.8. Solving the equation yields mass(block4) = 98 / 9.8 = 10. Cascade has found a value for the sought quantity, thus achieving the goal. Notice that it was necessary to use algebraic transformations to solve the equation. This is inevitable when the knowledge is expressed in a format, such as conditioned equations, that allows the same piece of knowledge to be used for achieving multiple goals. When the knowledge is expressed as multiple single-goal rules, such as the three production rules mentioned earlier, then algebraic knowledge is not needed during interpre-

tation. For instance, in ACT*, conditioned equations would be represented as structures in declarative memory, and production rules would implement an interpreter for them. The production rules would embed the algebraic knowledge necessary for using the conditioned equations.

In short, in order to obtain the type of declarative transfer that we believe is common in this task domain, it is useful to represent knowledge as conditioned equations and to embed algebraic knowledge in the interpreter. Because procedural transfer develops only with practice, modeling it would require a model of memory, which will be added to later versions of Cascade (see Step 1d in Table 3).

Local Explanation

Cascade explains each line of an example individually, but it does not try to find a plan or schema that spans all the lines. That is, Cascade does local but not global explanation (plan recognition). This design is motivated by examination of the protocols. Of the 204 self-explanation statements analyzed by Chi and VanLehn (1991; Table 4), only 13 (6%) related goals to groups of actions. Plan recognition appears not to be a common process in this instructional situation.

Good Versus Poor Explanation of Examples

The simulation rules are based on the hypothesis that the only difference between Good and Poor solvers is that the Good solvers explain more example lines than do the Poor solvers. This assumption is consistent with several observations that show that the contents of Good and Poor solvers' self-explanations are not significantly different (see Tables 4 and 7 in Chi & VanLehn, 1991). The Good solvers just produced more self-explanations than the Poor solvers did.

Summary

The preceding argument may be summarized as follows. First, students invented new knowledge during example studying and problem solving, rather than recalling and operationalizing knowledge acquired by reading the text. Moreover, much of this knowledge was acquired during problem solving, and not just during example explaining.

Next, when a derivation used previously unpresented knowledge, only a few small pieces of knowledge were new. Although students could have simply stored a whole derivation whenever they detected that it involved some new knowledge, they instead tried to find the gap in their own

derivation and infer a piece of knowledge that filled it. This technique is called *derivation completion.*

Detecting a gap was difficult because some impasses were caused by poor search control decisions or slips. We believe that subjects usually checked their partial derivation for slips and to determine if an alternative solution path existed. Only when they were satisfied that the impasse was inevitable given their current knowledge did they proceed to search for knowledge to fill it. Computational experiments showed that this was insufficient in itself to account for subject's' behavior during problem solving, so we conjectured that subjects used the derivations produced while explaining examples to constrain their generation of derivations while problem solving. This technique is called *analogical search control.*

Subjects had multiple methods for finding new knowledge. The most productive one for our subjects seems to have been EBLC, wherein new domain knowledge was created by specialization of overly general knowledge.

Subjects can probably perform a type of *declarative transfer,* to use Singley and Anderson's (1989) term, wherein an equation acquired while seeking one quantity can be used later when another quantity in the equation is sought. This suggests that knowledge be represented as conditioned equations and that algebraic equation-solving knowledge be built into their interpreter.

When students explained an example's solution, they rederived each line but did not try to find an overall plan that spanned all the lines. This same process was used by both Good and Poor solvers. The Good solvers merely chose to explain more example lines than did the Poor solvers.

DISCUSSION

What Was Discovered While Developing Cascade

We had originally thought that EBLC and analogical search control were completely independent. However, in trying to simulate the Good solvers, we discovered that (a) most of the rules that need to be learned were first used during problem solving, and (b) Cascade tended to learn at the wrong impasses when analogical search control was turned off during problem solving. In retrospect, this result is an obvious, inevitable, general principle of machine learning. If missing knowledge is required to solve a problem or explain an example, then all paths from the initial state are blocked—they terminate in an impasse. There is no way in principle for the learner to know which of these impasses, if resolved, would lead to a solution. However, if the learner is explaining an example, the example lines often permit only one partial solution path and thus only one impasse. Because the example's

problem is solvable, resolving the impasse will probably lead to a solution. Thus, the learner can assume that this impasse probably was caused by a missing piece of knowledge, so inventing a rule that resolves it is likely to (re)construct a correct domain rule. On the other hand, if the learner is solving a problem, there are no printed solution lines to guide generation of a derivation, so there tend to be many partial solution paths that terminate in impasses. In order to increase the probability that a correct rule will be learned, the learner needs some way to intelligently select one of these partial solution paths/impasses. Ample search control knowledge must be learned before encountering the impasse. We thus arrive at the novel result that search control learning is required for all kinds of derivation completion, including EBLC and analogy abduction, that occur during problem solving. Derivation completion during example studying requires less search control knowledge, if any. This is consistent with the finding that examples cause faster learning than equivalent problems (e.g., Pirolli, in press; Sweller & Cooper, 1985). This line of argument is backed by computational experiments with Cascade. It learned 15 rules during problem solving when analogical search control was turned on but only 6 when it was turned off. Thus, 60% of the rules learned during problem solving required analogical search control.

Another major surprise was that the increased learning of the Good solvers was not due to a single learning mechanism, but rather to a variety of interacting mechanisms. According to the Cascade analysis, the ideal Good solver learned 23 rules and the ideal Poor solver learned 3. The 20 rules that were learned by the Good solver and not by the Poor solver came from several sources:

- 8 rules were learned as the examples were explained. Because the Poor solver did not explain the examples, it did not learn these rules.
- 3 rules were learned during problem solving simply because the 8 rules learned during example studying set up contexts that allowed them to be learned by EBLC, even without the aid of analogical search control.
- 9 rules were learned by EBLC during problem solving using analogical search control. Because the Poor solver did not generate derivations for the examples, it could not use analogies to them and thus could not learn these 9 rules.

We had originally expected that all rules would be learned during example studying, but this turned out to be the source for only 8 of the 20 rules.

Another major surprise was that self-explanation raised the learning rate during problem solving. This result is consistent with the conjecture by Pirolli and Anderson (1985) that the way students study examples causes some students to learn more while solving problems than other students.

Cascade provides an explicit model of this. As far as we know, this is the first computational model to show how one kind of training can increase the learning rate of a different kind of training. Most models predict only additive interactions, in which the amount learned by the combined training is the sum of the amounts learned by each training in isolation. This unusual prediction of a nonlinear interaction warrants further empirical testing.

Because we invented overly general rules whenever Cascade encountered an impasse that it could not resolve with the existing overly general rules, we feared that the resulting collection of overly general rules would be terribly ad hoc with no interesting themes or patterns. Fortunately, the result was otherwise. The collection of overly general rules fell neatly into two classes. Rules in the first class (Rules 1–11 in Table 4) relate property values of two objects whenever those two objects are assigned compatible property values by the example and the objects themselves have some intrinsic relationship. As described in VanLehn and Jones (in press), these rules fall into an interesting hierarchy that could be learned by simple generalization and strengthening techniques and thus predict a learning-to-learn phenomenon. The second class of overly general rules (Rules 39–44 in Table 4) implements the basic idea that it is okay to substitute common sense quantities (e.g., pulls and pushes, accelerations, and decelerations) for formal quantities, but only if one is really stuck and the resulting substitution leads to a successful derivation. In short, computational modeling taught us that most rules in this instructional situation could be learned by using two basic assumptions: Property–value coincidences are sometimes not accidental, and commonsense quantities can sometimes be treated as formal physics quantities.

We had a surprisingly hard time finding a way to transfer knowledge from the knot-is-a-body impasse to later problem-solving situations. We first attempted to use syntactic induction techniques, but the resulting rules either were too specific and did not apply where we saw subjects applying their rules or were too general and applied inappropriately. Eventually we discovered that a rule could invoke the analogical problem-solving machinery directly. This rather unusual type of rule gave us the right combination of selectivity and generality. As far as we know, this analogy abduction technique is unique in machine learning.

We did not initially realize that memory plays an important role in explaining the findings on analogical references. The protocols showed that the Good solvers referred to the examples less than did the Poor solvers, and yet analogical search control required that they refer often to the derivations of examples. The only way to resolve this apparent conflict is to assume that the Good solvers were able to retrieve most of the derivational information from memory and thus did not need to look at the examples as often as the Poor solvers did.

Analogical search control solves a nagging problem in the expert–novice literature. When subjects solved a problem whose diagram appears in Figure 3A, they drew analogies to two earlier problems whose diagrams are shown in Figures 3B and 3C. In some schema-based problem-solving systems, it is difficult to get the solver to use more than one schema to solve the problem. The solver tends to let one schema dominate the problem solving and invokes the other schema as a subordinate. This does not reflect the quality of human problem solving (Holland et al., 1986; VanLehn, 1989). Thus, the problem is supposedly to form a compound schema from two component schemas of equal stature. When Cascade solves the problem of Figure 3A, it is told that the diagram is similar to two examples' diagrams. It retrieves both examples during analogical search control and refers to goals from both of them. This produces the mixture of inferences that seems required for simulating human problem solving. We were surprised that analogical search control solved the so-called schema compounding problem. This suggests that a collection of derivational triples plays the role of a schema.

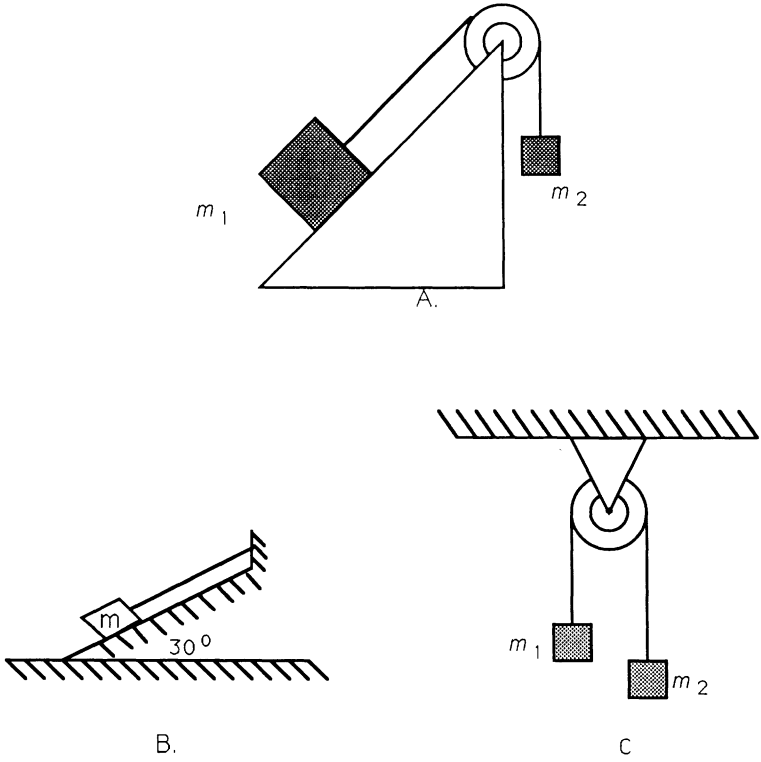

FIGURE 3 A problem solved by analogy to two other problems.

Other Models of the Self-Explanation Effect

Two other models of the self-explanation effect are under development. Reimann (in press) is developing a case-based model of the self-explanation effect. Example studying is modeled as plan recognition. Existing schemas/plans are used to analyze an example and create an annotated case that explains how the various parts of the example fit together to form a whole. During problem solving, such cases are adapted and replayed. Students who make more self-explanations acquire better annotations, which permit better adaptation during problem solving. Like Cascade, the Reimann model is based on the assumption that study habits and not prior knowledge are the source of the self-explanation effect. The main difference between the two models is the grain size of their knowledge representations, which in turn governs how they acquire knowledge during example study and use it during problem solving. The Reimann model learns example-sized units of knowledge (cases), whereas Cascade learns smaller units (rules). There are both computational and empirical reasons for using the smaller grain size. Computational experiments suggest that transfer is increased by storing knowledge as parts of cases (snippets) rather than whole cases (Hinrichs, 1988; Kolodner & Simpson, 1984; Redmond, 1990). The Chi et al. (1989) protocols suggest that subjects in that study attended more to individual lines than to the overall plan of an example, and their attention became especially focused when they detected that they needed to learn new knowledge. For instance, when P1 could not explain a minus sign, after a brief review of her work to this point, she concentrated almost exclusively on finding a rule or rules that would explain the minus sign.

On the other hand, schemas have often been used to explain phenomena in the expert–novice literature (e.g., Elio & Sharf, 1990). Because Cascade does not have schemas, it is not immediately clear how it can explain, for instance, the finding by Chi, Feltovitch, and Glaser (1981) that experts classify problems according to the solution method while novices classify problems according to their surface characteristics. However, we believe this finding can be explained within the Cascade framework if one assumes that experts have a vast store of derivations that they use to quickly plan a solution to the given problem. This allows them to determine the main solution method and to use that as a basis for classification. Novices cannot determine solutions quickly enough to do this, so they use surface features for classification. Consistent with this explanation, Chi et al. (1981) found that experts actually took longer than novices to classify problems (45 sec vs. 30 sec per problem). Perhaps other phenomena that have been explained with schemas can also be explained in the Cascade framework.

Pirolli and Recker (1991) are developing a model that can understand text as well as examples. Following Kintsch (1986), they model understanding as

a process that may involve many levels of elaboration and abstraction. Poor students do verbatim processing of the text and examples, leading to memory traces that are retrieved and used in a rote fashion during problem solving. Good students make deeper explanations. For instance, a model of a Good student might explain a sample LISP program by constructing a mental model of how the program satisfies its specifications, which would in turn involve constructing mental models of program abstractions, computations, and functions.

The Pirolli–Recker (1991) model is similar to Cascade in that both are built on the assumption that the self-explanation effect is caused by the students' study habits rather than their prior knowledge. Moreover, both models use small-grained representations of knowledge rather than cases or schemas. There are two main differences, however. Although Pirolli and Recker are grounding their model's development on data collected from a study of students learning to code LISP by reading chapters from a textbook, studying examples, and solving problems, their text, which is based on a cognitive task analysis of LISP coding, is probably more complete and easier to understand then the physics text used by Chi et al. (1989). Because the LISP text clearly states almost all the to-be-learned rules, the primary knowledge acquisition method in the Pirolli–Recker model appears to be interpretation of text, whereas most rules in Cascade are acquired during example studying because they are not mentioned in the text. In this respect, the models are complementary rather than antagonistic. A second difference between the models is that Cascade currently has no model of memory, whereas the Pirolli–Recker model has a detailed model of the encoding, indexing, and retrieval of mental information. It uses Soar's data-chunking facility (Newell, 1990) to implement a version of the ACT* declarative/procedural distinction. Much of the learning during example studying and problem solving appears to be knowledge compilation wherein knowledge is reformatted and reindexed to make it more useful. Again, the two projects are complementary rather than antagonistic.

Cascade's Weaknesses and Plans for Further Research

The current version of Cascade, Cascade 3, models knowledge acquisition and not knowledge compilation. Like most theorists, we believe that knowledge compilation is intimately related to human memory mechanisms. Clearly, Cascade needs to be augmented with a model of human memory in order to be a more complete model of learning. We are interested not in neurologically or computationally plausible mechanisms of memory, but only in creating a model that will yield pedagogically useful predictions about initial knowledge acquisition, transfer, and practice (a

"pseudo-student"; VanLehn, 1991b). Therefore, we plan to use a "black box" mathematical model of memory that delivers an external performance that is an accurate description of human memory. Table 3 shows where the black boxes go.

Another area where Cascade is weak is its model of Poor solvers' explanation of examples. The protocols did not reveal much, because the subjects just paraphrased the lines, perhaps adding, "Ok, that makes sense." When they did explain a line, they said the same kinds of things as Good solvers (Chi & VanLehn, 1991). Currently, Poor solvers' processing of examples is modeled by making the lines available for transformational analogy but not rederiving the lines. However, this fails to explain why the subjects thought they understood the lines. We suspect their explanation was just like the Good solvers' explanation but they took the example's word about the details. For instance, in explaining the line "$Fax = -Fa \cos 45°$," the subject would use the rule that recognized that this was a projection equation and produced the four correspondences shown in Table 1. The Good solvers went on to explain each of these four assertions, whereas the Poor solvers may have just stopped at this point and assumed that each of the four assertions held. Thus, their explanation did not go as deep as the Good solvers' but was otherwise the same. This version of Poor solver behavior explains why Poor solvers thought they had successfully explained the example. With this addition to the model, our hypothesis about the key difference between Good and Poor solvers is twofold: The Good solvers rederived more lines than Poor solvers did, and their derivations were more complete.

Cascade's sharp distinction between domain rules and other rules is an idealization. When a new rule is acquired, subjects probably do not immediately believe that the rule is just as valid as rules they have been using successfully for many problems. In the next version of Cascade, rules will bear a degree of belief that increases whenever the rule is used in a successful derivation (cf. Rosenbloom & Aasman, 1990). The initial degree of belief given to a new rule will be a function of the amount of backing up that has gone on before the rule's creation. This should capture the following intuition: Suppose problem solving has been going smoothly when one encounters a resolvable impasse. One might say, "I don't know of any kind of force here, but if there were one, that would balance the other two forces and explain why the object is at rest." In such circumstances, one might believe one has discovered something about physics and form a new rule. On the other hand, if one has been floundering about for some time and feeling utterly lost, one is unlikely to react to impasses in the same way. Cascade should keep a running count of the number of impasses, especially ones that could not be resolved and required backing up. As these counts

get higher, the degree of belief accredited a new rule is reduced. When the counts pass some threshold, rules are no longer formed. The type of derivation completion used to induce a new rule should also affect its initial degree of belief. EBLC should produce higher degrees of belief than analogy abduction. Transformational analogy, which currently does not produce rules, should produce rules with even lower degrees of belief. As discussed in VanLehn and Jones (in press), representing degrees of belief plays a key role in a syntactic induction method for learning overly general rules, which may also be added to Cascade. In fact, adopting an explicit representation of degrees of belief has many implications for the overall design of Cascade that require thorough exploration.

The next milestone will be to fit Cascade's behavior to the protocols of each individual subject. Cascade will be made to explain exactly the example lines that the subject seems to have explained, as indicated in the protocol, and to explain them to roughly the same depth. When given problems to solve, Cascade should reach impasses in the same places that the subject does. However, the subject will probably display more impasses than Cascade. Currently, Cascade's initial knowledge contains every rule mentioned anywhere in the text before the examples. Because subjects probably have a less thorough understanding of the text, they will probably reach more impasses than Cascade. Therefore, Cascade's initial domain knowledge will be adjusted to fit the impasses that it cannot explain as deficiencies in example processing. This computational experiment should help us understand how much of the self-explanation effect is due to missing prior knowledge and how much is due to shallow self-explanation.

At places where Cascade did perform EBLC, the subjects in the Chi et al. (1989) study generated pauses and other signs of intensive processing, but their comments were too vague to indicate whether they were actually using overly general rules to resolve their impasses. For instance, in the protocol from P1 quoted earlier, EBLC seems to have occurred while the subject was saying, "Hmmm, negative cosine 30, why would they say, ahhh, ummm. . . . The, ohh, okay maybe it's just because the A component is the X component of force A is negative. So they just. . . . Well okay I'll, I'll try that for a while." This is typical of other places in the protocols where EBLC was supposedly occurring. Clearly, something is happening here, but it is far from clear that EBLC is a good characterization of it. It could, for instance, be syntactic induction of some kind. The only good argument for EBLC over other proposals for processes that handle these impasses is that EBLC is computationally sufficient and our implementations of the others were not. It would be better to settle the issue empirically. For instance, training materials could

be designed so that several gaps can be spanned by one overly general rule and several others can be spanned by a second. EBLC would predict a specific pattern of rule learning events because the subject either learns all or none of the rules corresponding to each overly general rule.

We are currently working on modeling learning in two other task domains: conceptual physics problems (e.g., Which direction does a pendulum bob fall when you cut its string at the apex of its swing?) and combinatorics word problems (e.g., If a professor has four graders to grade eight examination questions, how many different ways can she assign graders to questions so that each grader grades two examination questions?). These efforts have already revealed inadequacies in the equation-based representation used in Cascade 3, but it is not yet clear how serious they are.

The processes of derivation completion and analogical search control may be the key to learning in many kinds of situations. For instance, Palinscar and Brown (1984) showed that reciprocal teaching increases learning. Results from 19 published studies on small peer groups (Webb, 1989) indicate that giving explanations almost always improves learning, whereas receiving explanations is seldom correlated with increased learning. Because reciprocal teaching increases the amount of explanation giving, and explanation giving is similar to self-explanation, reciprocal teaching may succeed just because it encourages EBLC and the other processes modeled by Cascade.

A Final Comment

A good theory of knowledge acquisition methods could improve the design of instructional situations and the training of teachers, because teachers and instructional designers need to know how students will react to the examples, exercises, explanations, and other information to which they are exposed. Research on knowledge compilation mechanisms is useful too, but primarily for determining how much and what kinds of practice to assign. The Cascade project is one of a small but growing number of efforts aimed at providing descriptive theories of knowledge acquisition (e.g., Badre, 1972; Glidden, 1991; Mayer, 1990; Neves, 1981; Martin & Redmond, 1988; Ohlsson, in press; Ohlsson & Rees, 1991; Pirolli & Recker, 1991; Reimann, in press; VanLehn, 1990). Because people use many different knowledge acquisition methods, we expect these efforts to be complementary rather than competitive accounts of cognition, and we look forward to some distant future when they can all be unified to provide an encompassing model of human knowledge acquisition.

ACKNOWLEDGMENTS

Research for this article was supported by the Cognitive Sciences Division (N00014-88-K-0086) and the Information Sciences Division (N00014-86-K-0678) of Office of Naval Research.

We appreciate the help of Rolf Ploetzner, Janet Kolodner, Renee Elio, and an anonymous reviewer in clarifying the exposition.

REFERENCES

Ali, K. M. (1989). Augmenting domain theory for explanation-based generalizations. In A. M. Segre (Ed.), *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 40–42). Los Altos, CA: Kaufman.

Anderson, J. R. (1977). Induction of augmented transition networks. *Cognitive Science, 1,* 125–157.

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

Anderson, J. R. (1990). *The adaptive character of thought.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

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.* Cambridge, England: Cambridge University Press.

Badre, N. A. (1972). *Computer learning from English text.* Berkeley: University of California at Berkeley, Electronics Research Laboratory.

Bergadano, F., Giordana, A., & Ponsero, S. (1989). Deduction in top-down inductive learning. In A. M. Segre (Ed.), *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 23–25). Los Altos, CA: Kaufman.

Berwick, R. (1985). *The acquisition of syntactic knowledge.* Cambridge, MA: MIT 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: Kaufman.

Carbonell, J. G. (1983). Learning by analogy: Formulating and generalizing plans from past experience. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine learning: An artificial intelligence approach* (pp. 137–161). Los Altos, CA: Kaufman.

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* (pp. 371–392). Los Altos, CA: Kaufman.

Carey, S. (1985). *Conceptual change in childhood.* Cambridge, MA: MIT Press.

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. (in press). Conceptual change across ontological categories: Implications for learning and discovery in sciences. In R. Giere (Ed.), *Cognitive models of science: Minnesota studies in the philosophy of science.* Minneapolis: University of Minnesota Press.

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

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., Feltovich, P. J., & Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science, 5,* 121–152.

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

Cohen, W. W. (1990). Learning from textbook knowledge: A case study. In T. Dietterich & W. Swartout (Eds.), *Proceedings, Eighth National Conference on Artificial Intelligence* (pp. 743–748). Los Altos, CA: Kaufman.

Danyluk, A. P. (1989). Finding new rules for incomplete theories: Explicit biases for induction with contextual information. In A. M. Segre (Ed.), *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 34–36). Los Altos, CA: Kaufman.

Davies, T. R., & Russell, S. J. (1987). A logical approach to reasoning by analogy. In J. McDermott (Ed.), *Proceedings of the Tenth International Joint Conference on Artificial Intelligence* (pp. 264–270). Los Altos, CA: Kaufman.

De Jong, G., & Oblinger, D. (in press). Steps toward a theory of plausible inference and its use in continuous domain planning. In S. Minton & P. Langley (Eds.), *Machine learning methods for planning and scheduling.* Los Altos, CA: Kaufman.

DiSessa, A. A. (1988). Knowledge in pieces. In G. Forman & P. B. Pufall (Eds.), *Constructivism in the computer age* (pp. 49–70). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Elio, R., & Scharf, P. B. (1990). Modeling novice-to-expert shifts in problem solving strategy and knowledge organization. *Cognitive Science, 14,* 579–639.

Eskey, M., & Zweben, M. (1990). Learning search control for constraint-based scheduling. In T. Dietterich & W. Swartout (Eds.), *Proceedings, Eighth National Conference on Artificial Intelligence* (pp. 908–915). Los Altos, CA: Kaufman.

Fawcett, T. E. (1989). Learning from plausible explanations. In A. M. Segre (Ed.), *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 37–39). Los Altos, CA: Kaufman.

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.

Genesereth, M. R. (1982). The role of plans in intelligent teaching systems. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 137–155). New York: Academic.

Glidden, P. L. (1991). Towards developing a model of mathematics learning: Modeling knowledge restructuring in learning school algebra. In R. Lewis & S. Otsuki (Eds.), *Advanced research on computers in education* (pp. 271–276). Amsterdam: Elsevier.

Goodman, N. (1956). *Fact, fiction, and forecast.* Cambridge, MA: Harvard University Press.

Hall, R. J. (1988). Learning by failing to explain: Using partial explanations to learn in incomplete or intractable domains. *Machine Learning, 3,* 45–78.

Halliday, D., & Resnick, R. (1981). *Fundamentals of physics.* New York: Wiley.

Hinrichs, T. R. (1988). Towards an architecture for open world problem solving. In J. Kolodner (Ed.), *Proceedings of a workshop on case-based reasoning* (pp. 182–189). Los Altos, CA: Kaufman.

Holland, J. H., Holyoak, K. J., Nisbett, R. E., & Thagard, P. R. (1986). *Induction: Processes of inference, learning and discovery.* Cambridge, MA: MIT Press.

Jones, R. (1989). *A model of retrieval in problem solving.* Unpublished doctoral dissertation, University of California at Irvine.

Kintsch, W. (1986). Learning from text. *Cognition and Instruction, 3,* 87–108.

Kolodner, J. L., & Simpson, R., Jr. (1984). A case for case-based reasoning. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society* (pp. 239–243). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Larkin, J. (1981). Enriching formal knowledge: A model for learning to solve textbook physics problems. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 311–334). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Larkin, J. (1983). The role of problem representation in physics. In D. Gentner & A. Stevens (Eds.), *Mental models* (pp. 75–98). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

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.

Martin, J. D., & Redmond, M. (1988). The use of explanations for completing and correcting causal models. In V. L. Patel & G. J. Groen (Eds.), *Proceedings of the Tenth Annual Conference of the Cognitive Science Society* (pp. 440–446). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Mayer, J. H. (1990). *Explanation-based knowledge acquisition of schemas in practical electronics* (TR-90/ONR32). Ann Arbor, MI: University of Michigan, Technical Communications Program.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. (1986). Explanation-based generalization: A unifying view. *Machine Learning, 1,* 11–46.

Neves, D. M. (1981). *Learning procedures from examples.* Unpublished doctoral dissertation, Carnegie-Mellon University.

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

Newell, A., & Simon, H. A. (1972). *Human problem solving.* Englewood Cliffs, NJ: Prentice-Hall.

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: Kaufman.

Ohlsson, S. (1990). Trace analysis and spatial reasoning: An example of its implications for testing. In N. Frederiksen, R. Glaser, A. Lesgold, & M. Shafto (Eds.), *Diagnostic monitoring of skill and knowledge acquisition* (pp. 251–296). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Ohlsson, S. (in press). Artificial instruction: A method for relating learning theory to instructional design. In P. H. Winne & M. Jones (Eds.), *Foundations and frontiers in instructional computing systems.* New York: Springer-Verlag.

Ohlsson, S., & Rees, E. (1991). An information processing analysis of the function of conceptual understanding in the learning of arithmetic procedures. *Cognition and Instruction, 8,* 103–179.

Palinscar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and monitoring activities. *Cognition and Instruction, 1,* 117–175.

Pazzani, M. (1990). *Creating a memory of causal relationships.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Pazzani, M., Dyer, M., & Flowers, M. (1986). The role of prior causal theories in generalization. In T. Kehler, S. Rosenschein, R. Filman, & P. Patel-Schneider (Eds.),

Proceedings, Fifth National Conference on Artificial Intelligence (pp. 545–550). Los Altos, CA: Kaufman.

Pirolli, P. (1991). Effects of examples and their explanations in a lesson on recursion: A production system analysis. *Cognition and Instruction, 8,* 207–259.

Pirolli, P., & Anderson, J. R. (1985). The role of learning from examples in the acquisition of recursive programming skills. *Canadian Journal of Psychology, 39,* 240–272.

Pirolli, P., & Bielaczyc, K. (1989). Empirical analyses of self-explanation and transfer in learning to program. In G. M. Olson & E. E. Smith (Eds.), *Proceedings of the 11th Annual Conference of the Cognitive Science Society* (pp. 450–457). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Pirolli, P., & Recker, M. (1991). *A model of self-explanation strategies of instructional text in the acquisition of programming skills* (Tech. Rep. No. CSM-1). Berkeley: University of California, School of Education.

Pople, H. E. (1973). On the mechanization of abductive logic. In N. J. Nilsson (Ed.), *Proceedings of the Third International Joint Conference on Artificial Intelligence* (pp. 147–152). San Mateo, CA: Kaufman.

Ram, A. (1990). Incremental learning of explanation patterns and their indices. In B. Porter & R. Mooney (Eds.), *Machine learning: Proceedings of the Seventh International Conference* (pp. 313–320). Los Altos, CA: Kaufman.

Redmond, M. (1990). Distributed cases for case-based reasoning: Facilitating use of multiple cases. In T. Dietterich & W. Swartout (Eds.), *Proceedings, Eighth National Conference on Artificial Intelligence* (pp. 304–309). Los Altos, CA: Kaufman.

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

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.

Rosenbloom, P. S., & Aasman, J. (1990). Knowledge level and inductive uses of chunking (EBL). In T. Dietterich & W. Swartout (Eds.), *Proceedings, Eighth National Conference on Artificial Intelligence* (pp. 821–827). Los Altos, CA: Kaufman.

Schank, R. (1982). *Dynamic memory: A theory of learning in computers and people.* Cambridge, England: Cambridge University Press.

Schank, R. (1986). *Explanation patterns: Learning creatively and mechanically.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Singley, M. K., & Anderson, J. R. (1989). *Transfer of cognitive skill.* Cambridge, MA: Harvard University Press.

Sleeman, D., Hirsh, H., Ellery, I., & Kim, I. (1990). Extending domain theories: Two case studies in student modeling. *Machine Learning, 5,* 11–38.

Smith, D. E. (1982). *Focuser: A strategic interaction paradigm for language acquisition* (LCSR-TR-36). New Brunswick, NJ: Rutgers University, Laboratory for Computer Science Research.

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.

VanLehn, K. (1986). Arithmetic procedures are induced from examples. In J. Hiebert (Ed.), *Conceptual and procedural knowledge: The case of mathematics* (pp. 133–180). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence, 31,* 1–40.

VanLehn, K. (1989). Problem solving and cognitive skill acquisition. In M. I. Posner (Ed.), *Foundations of cognitive science* (pp. 527–579). Cambridge, MA: MIT Press.

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* (pp. 17–26). New York: North-Holland.

VanLehn, K., Brown, J. S., & Greeno, J. G. (1984). Competitive argumentation in computational theories of cognition. In W. Kintsch, J. Miller, & P. Polson (Eds.), *Methods and tactics in cognitive science* (pp. 235–262). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

VanLehn, K., Ball, W., & Kowalski, B. (1989). Non-LIFO execution of cognitive procedures. *Cognitive Science, 13,* 415–465.

VanLehn, K., Ball, W., & Kowalski, B. (1990). Explanation-based learning of correctness: Towards a model of the self-explanation effect. In M. Piattelli-Palmarini (Ed.), *Proceedings of the 12th Annual Conference of the Cognitive Science Society* (pp. 717–724). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

VanLehn, K., & Jones, R. (in press). Integration of analogical search control and explanation based learning of correctness. In S. Minton & P. Langley (Eds.), *Machine learning methods for planning and scheduling.* Los Altos, CA: Kaufman.

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

Webb, N. M. (1989). Peer interaction and learning in small groups. *International Journal of Educational Research, 13,* 21–40.

Widmar, G. (1989). A tight integration of deductive and inductive learning. In A. M. Segre (Ed.), *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 11–13). Los Altos, CA: Kaufman.

Wilkins, D. C. (1988). Knowledge base refinement using apprenticeship learning techniques. In R. G. Smith & T. M. Mitchell (Eds.), *Proceedings, the Seventh National Conference on Artificial Intelligence* (pp. 646–651). Los Altos, CA: Kaufman.