In S. Minton (Ed.), Machine Learning Methods for Planning (pp. 273-315). Los Altos, CA: Morgan Kaufmann. (1993).

CHAPTER 9

Integration of Analogical Search Control and Explanation-Based Learning of Correctness

KURT VANLEHN
RANDOLPH M. JONES

1. Introduction

A commonly studied problem in machine learning is to extend an incomplete first-order theory to make it consistent with a given set of examples. In this chapter we describe a theory revision system, named CASCADE, that addresses this problem. Although the system was developed as a simulation of human students learning college physics, we treat it here as a multistrategy machine-learning system and evaluate it in terms of its computational properties. For evaluations of it as a psychological model, see VanLehn, Jones, and Chi (1992).

A widely used approach to solving the incomplete theory revision problem is based on a simple idea that has been "invented" independently many times (Ali, 1989; Anderson, 1977, 1990; Bergadano, Giordana, & Ponsero, 1989; Berwick, 1985; Danyluk, 1989; Fawcett, 1989; Genesereth, 1982; Goodman, 1956; Hall, 1988; Lewis, 1988; Martin & Redmond, 1988; Pazzani, Dyer, & Flowers, 1986; Schank, 1986; Sleeman, Hirsh, Ellery, & Kim, 1990; Smith, 1982; VanLehn, 1987; VanLehn, Ball, & Kowalski, 1990; Widmar, 1989; Wilkins, 1988). Suppose the learner is attempting to explain an example. If it reaches an impasse, a new rule is invented that allows it to resolve the impasse and continue the explanation. If a successful explanation results, the new rule is added to the domain theory. This same basic process can also occur during problem solving. When the problem solver fails to achieve a goal, instead of backing up it invents a new rule that allows it to continue. If the resulting derivation is successful, the new rule is added to the domain theory. The basic idea in both the example-explaining and problem-solving cases

is that a derivation that cannot be completed with the current domain theory is completed by adding a new rule to the domain theory. Thus, the term *derivation completion* seems appropriate.[1]

Two major problems confront any system that does derivation completion. The first is deciding whether to handle an impasse by inventing a new rule or by backing up. Most derivation completion systems try to resolve all impasses and back up only if they fail. CASCADE is based on the heuristic that an impasse is worth repairing only if the system is on a solution path when the impasse occurs. In an attempt to stay on a solution path, CASCADE uses a form of search control based on drawing analogies to previously solved problems. Before starting to solve a problem, it selects one or more previously solved problems and forms analogical mappings between them and the current problem. As it solves the problem, it uses these mappings to refer to the old problems' derivations whenever there are two or more rules for achieving a goal. If it can find an old goal that is analogous to the current goal, it chooses the rule that was used to achieve the old goal. If it can find no goal that is analogous to its current goal, then it uses heuristics to choose a rule. Thus, it follows the old derivations whenever it can, but falls back on its default search control when it cannot. This way of using past experience to guide problem solving is much simpler than derivational analogy, case-based planning, and plan replaying. More importantly, computational experiments show that analogical search control dramatically increases the effectiveness of derivation completion as compared to the standard technique of trying to resolve all impasses.

The second problem facing any derivation completion system is inventing a new rule that will resolve an impasse that the system has decided is worth resolving. The systems in the literature use several approaches:

1. Some systems use abduction (O'Rorke, Morris, & Schulenburg, 1990; Pople, 1973). The impasse is resolved by creating a fact (a unit

1. 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 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 they exclude systems, such as SIERRA (VanLehn, 1987), that collect several incomplete derivations and compare them before deciding how to complete them.

clause, or a rule of the form "if true then <literal>") that unifies with two or more goals that cannot be achieved.

2. Some systems create a new rule using syntactic heuristics, such as selecting the smallest or simplest rule that will complete the derivation (Anderson, 1977; Berwick, 1985; Genesereth, 1982; Martin & Redmond, 1988; Sleeman, Hirsh, Ellery, & Kim, 1990; Smith, 1982).

3. Some systems use induction over several examples (Ali, 1989; Cain, 1991; Danyluk, 1989; Fawcett, 1989; Hall, 1988; Ourston & Mooney, 1990; VanLehn, 1987; Wilkins, 1988). Some of these systems compare gaps in the partial explanations of several examples in order to induce a rule that will fill them all. Other systems concentrate on filling a single gap, but they use induction over multiple examples to provide the body (antecedents) of the rule. Often, this induction is accomplished with standard concept formation algorithms, such as ID3 (Quinlan, 1986) and Foil (Quinlan, 1990).

4. Some systems use analogy to resolve impasses (Anderson, 1990; Lewis, 1988; Pazzani, 1990). They assume that achieving a goal produces some result and that goal-result pairs from old derivations can be retrieved. When the system reaches an impasse, it retrieves an old goal that is similar to the current, failed goal, draws an analogy between the surrounding contexts of the old goal and the current goal, maps the old goal's result over to current problem context, and thus produces a result for the current goal. The learner may create a new rule that is general enough to cover both the old goal-result pair and the new one.

5. In order to resolve an impasse, some systems use knowledge that they normally avoid using, such as explanation patterns (Schank, 1986), causal attribution rules (Pazzani, Dyer, & Flowers, 1986; Lewis, 1988; Anderson, 1990), determinations (Bergadano, Giordana, & Ponsero, 1989; Widmar, 1989), or overly general rules (VanLehn, Ball, & Kowalski, 1990). This knowledge is normally avoided because it is so general that it could explain anything when used to explain examples, and it could produce incorrect results when used during problem solving. Thus, the system is right to mistrust it and use it only when the domain theory fails to achieve a goal. If the

overly general knowledge resolves the impasse and allows a derivation to be completed, then the system creates a new rule that is midway in generality between the overly general knowledge and the instantiation of it that resolves this impasse. This role can be created with any of the many EBL methods that create rules that are midway between existing general rules and their instantiations.

CASCADE uses the last method. Its version is called explanation-based learning of correctness (VanLehn, Ball, & Kowalski, 1990) or EBLC.[2] This is an apt name because the system uses EBL techniques to increase the number of problems that it can solve correctly (this will be demonstrated later), whereas standard EBL techniques increase the efficiency of the reasoning. That is, EBLC does knowledge-level learning, and EBL does symbol-level learning (cf. Flann & Dietterich, 1989; Cohen, 1990).

However, EBLC relies on a library of overly general rules, which raises two questions. First, is this kind of learning trivial in that one can always invent an ad hoc overly general rule for learning any kind of domain rule? Second, for a given learning problem, what would a fairly complete library of overly general rules contain? Would some common themes or structures be evident in the rules, or would the collection be just ad hoc? These issues arise for any system that uses overly general knowledge (method 5 above) to complete derivations.

One way to find out about the content of the library of overly general rules is to choose a particular learning problem and develop a library that allows the system to learn as many domain rules as possible. It may prove impossible to invent overly general knowledge that can learn some of the desired rules. The library may turn out to be unstructured, or it may have interesting common themes. Perhaps the best way to find out is to try the approach on one or more learning tasks. This chapter reports such a computational experiment.

To summarize, the issues discussed herein are:

2. The name was coined by VanLehn, Ball, and Kowalski (1990) before we knew of the other systems that learn by completing explanations. Ironically, the name is overly general—for EBLC would be a good name for the basic idea behind all these systems. Rather than add to the confusion by renaming CASCADE's mechanism, we continue to use the old name, even though it is now clear that EBL can be used in many ways to learn correctness.

1. How can a derivation completion learner intelligently decide which impasses should be resolved and learned from? Does analogical search control suffice?

2. Can EBLC learn any domain rule, or are there limits to its power?

3. What would a library of overly general rules contain for a particular task domain?

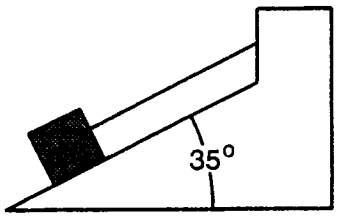
In the next section we describe the particular learning problem and CASCADE. Following that, we present results on the interaction between search control and derivation completion, concluding with a study of the power of explanation-based learning of correctness.

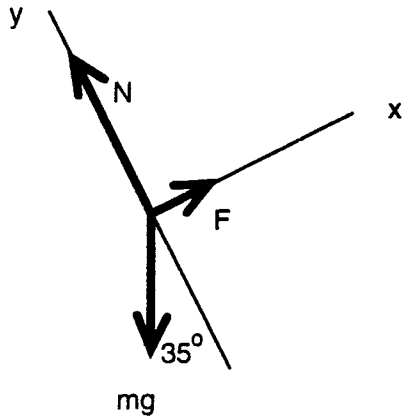
Throughout this chapter, we will emphasize CASCADE's main knowledge-level learning technique, EBLC, and the requirements it puts on the rest of the system. However, CASCADE is not a simple, single-strategy learning system whose power has been assessed on classic machine-learning problems. CASCADE is fundamentally a cognitive model, and people are fundamentally multistrategy learners. Our primary objective was to get CASCADE to learn as well as the best human subjects in a study of physics learning (Chi, Bassok, Lewis, Reimann & Glaser, 1989). This task required discovering and implementing several learning strategies, then ironing out their interactions. The emphasis on EBLC is meant only as an expositional device to simplify the presentation of an integrated, multistrategy cognitive model of learning.

2. The CASCADE System

The task domain is college physics, and in particular, Chapter 5 of Halliday and Resnick (1981), which teaches the application of Newton's laws to solve translational dynamics problems. An example from Chapter 5 is presented in Figure 9–1. Several kinds of knowledge are required for solving the problems in Chapter 5: (1) Newton's third law, $F = ma$; (2) force laws, which assert the existence of forces given specific configurations of objects, such as a taut string attached to a block; (3) mathematical knowledge, including projection of vectors onto axes and solution of systems of linear equations; and (4) heuristic knowledge about how to idealize a problem as a system, and in particular, how to choose the bodies that Newton's law will be applied to.

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





Solution:
(1) We choose the block as the body.

(2) The forces acting on the block are shown in the free-body diagram on the right.

(3) Because we wish to analyze the motion of the block, we choose all the forces acting on the block. Note that the block will exert forces on other bodies in its environment (the string, the earth, the surface of the incline) in accordance with the action-reaction principle: these forces, however, are not needed to determine the motion of the block because they do not act on the block.

(4) Since the block is unaccelerated, we obtain:

$$\mathbf{F} + \mathbf{N} + \mathbf{mg} = 0$$

(5) It is convenient to choose the x-axis of our reference frame to be along the incline and the y-axis to be normal to the incline (see figure above).

(6) With this choice of coordinates, only one force, **mg**, must be resolved into components in solving the problem.

(7) The two scalar equations obtained by resolving **mg** along the x- and y-axes are:

$$\mathrm{F} - \mathrm{mg}\sin 35 = 0 \qquad \text{and} \qquad \mathrm{N} - \mathrm{mg}\cos 35 = 0$$

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

Figure 9–1. A physics example.

The protocol data used to constrain the design of CASCADE came from a study by Chi et al. (1989). There were three phases in the study. During the first phase, subjects studied the first four chapters of Halliday and Resnick (1981), which review mathematical and other prerequisite

knowledge, then studied the text of Chapter 5, which presents Newton's laws. Most of the text describes experimental evidence for the laws and their histories. There is little discussion of how to solve physics problems. During phase 2 of the study, the subjects studied three examples, including the one shown in Figure 9–1. Verbal protocols of the subjects were taken as they studied these examples. During phase 3, the subjects worked 25 problems while giving protocols. Although subjects varied in their abilities to solve these problems, the best subject got all of them right.

The learning problem is to replicate the learning that occurred during phases 2 and 3 of the Chi, Bassok, Lewis, Reimann and Glaser (1989) study. CASCADE should start with an initial domain theory that represents the knowledge in Halliday and Resnick. It should study the three examples and solve the 25 problems. It should learn as much as the best subject, and so it should get all the problems right.

In the next few sections, we describe the CASCADE system. For expository purposes, this chapter works with the knowledge representation of an older version of CASCADE, CASCADE 2, that uses Horn clauses to represent its knowledge. The current version of CASCADE, CASCADE 3, uses conditioned equations instead of Horn clauses. See VanLehn, Jones, and Chi (1992) for a description of the equation-based knowledge representation.

In the first two sections, we present CASCADE's knowledge representation and illustrate its method for solving problems and explaining examples. Its search control mechanism, which is based on analogy to old derivations, is the focus of the next section. The fourth section is devoted to describing explanation-based learning of correctness.

2.1 Solving Problems

Figure 9–1 shows a physics example used in the Chi et al. study. (Actually, it was edited slightly for expository purposes.) Like all physics examples, it consists of a problem and a solution. The problem consists of some given information and some sought quantities. The givens of a problem are represented in CASCADE as a set of ground literals. For instance, the givens of the problem of Figure 9–1 are represented as follows:

```
givens(ix,
 [current_situation(ix),
  instance(ix,standard_situation),
```

```
instance(block_ix, block),
instance(string_ix,string),
instance(plane_ix,plane),
massless(string_ix),
slides_on(block_ix,plane_ix),
tied_to(string_ix,block_ix),
at_rest(block_ix),
above(block_ix,plane_ix),
above(string_ix,block_ix),
right(string_ix,block_ix),
value(incline(plane_ix),35),
value(incline(string_ix),35),
value(mass(block_ix),m)]).
```

In order to perform analogies, CASCADE must keep many problems in memory at the same time, so each problem is assigned a distinct name, such as `ix` in this case (which stands for "incline example"). All information about a problem is stored under its name.

Much of the reasoning involves `value(Q,V)` literals, where `Q` is a quantity and `V` is a value. Even free-body diagrams are represented this way, so a value can be a complex structure. For instance, the free-body diagram of Figure 9–1 is represented by the following literal:

```
value(fbd(block_ix),
   fbd([arrow(35,1),arrow(115,1),arrow(90,-1)],
      [axis(block_ix,x,35), axis(block_ix,y,115)])).
```

The above indicates that three arrows and two axes are drawn on the diagram. Their directions are indicated by an inclination (a number in the interval [0,180] indicating degrees from horizontal) and a sense (1 if the arrow points up, -1 if it points down).[3]

The sought quantities of the problem are represented as goals (literals) to be proved. The soughts of the problem of Figure 9–1 are represented as follows:

```
soughts(ix,
   [value(
```

3. This unusual convention for indicating direction is handy because many things, such as lines or segments, have inclinations but no sense. In the rule above, the fact that the vector and the axis have the same inclination means that they are parallel. The fact that they have the same sense indicates that they point in the same direction.

```
    magnitude(force(block_ix,string_ix,tension)),
    Tmag),
  value(
    magnitude(force(block_ix,plane_ix,normal)),
    Nmag)]).
```

Capitalized words are variables, so these goals seek values for the magnitudes of the forces on the block caused by the string and the plane. Solving a problem is simply proving a set of goals of the form `value(Q,V)` where `Q` is a specified quantity and `V` is a variable.

The domain theory is represented by PROLOG rules. For instance, one piece of domain knowledge is that the magnitude of a tension force is equal to the tension in the string causing the force:

```
value(magnitude(Force),V) :-
   instance(Force, tension_force),
   part(agent,Force,String),
   value(tension(String),V).
```

The head of this rule unifies with the first goal of Figure 9–1:

```
value(magnitude(force(block_ix,string_ix,tension)),
      Tmag)
```

Applying this rule produces subgoals corresponding to each literal in the body:

```
instance(force(block_ix,string_ix,tension),
         tension_force),
part(agent,force(block_ix,string_ix,tension),
      string_ix),
value(tension(string_ix),Tmag).
```

These subgoals are addressed in the order in which they appear in the rule. The two subgoals are satisfied by two simple rules:

```
instance(force(Body,Agent,tension),tension_force).
part(agent,force(Body,Agent,Type), Agent).
```

These rules simply access values embedded in the force data structure. They are examples of the kind of miscellaneous knowledge that is provided in the initial domain knowledge.

Several rules unify with the third subgoal. One rule is the inverse of the tension rule that produced the subgoal:

```
value(tension(String),V) :-
   instance(Force,tension_force),
   agent(Force,String),
   value(magnitude(Force),V).
```

This rule is marked as being an inverse of the other rule, and the interpreter will not apply a rule whose inverse was used in producing the current goal. Thus, short cycles are prevented in the derivation. (In the equation-based representation used by the current version of CASCADE, these two rules are represented by a single equation, so cycle detection is even simpler.) Another rule that applies to the goal is the following:

```
value(tension(Whole),V) :-
   instance(Whole,string),
   piece(Whole,Part),
   value(tension(Part),V).
```

This rule says that the tension of a whole string is the same as the tension of a piece of the string. It is useful for pulley problems, where a string is represented as having several parts of different inclinations. In this problem, the string does not have any pieces, so after producing the subgoal `piece(P,string_ix)`, CASCADE fails and backs up. CASCADE uses chronological backtracking, just as PROLOG does. In this case, the most recent goal with any alternatives left is the top-level goal

```
value(magnitude(force(block_ix,string_ix,tension)),
      Tmag)
```

so CASCADE returns to trying to satisfy it. CASCADE applies the following rule:

```
value(magnitude(Vector), V) :-
   patient(Vector,Body),
   value(fbd(Body),Fbd),
   axis_of_fbd(Fbd,Axis),
   value(incline(Vector),I),
   value(incline(Axis),I),
   value(sense(Vector),S),
   value(sense(Axis),S),
   value(projection(Vector,Axis),V).
```

This rule says that a magnitude of a vector is equal to its projection onto an axis of the appropriate free-body diagram (fbd) provided that

the vector lies along the axis. Applying this rule produces the following set of subgoals:

```
patient(force(block_ix,string_ix,tension),Body)
value(fbd(Body),Fbd)
axis_of_fbd(Fbd,Axis)
value(incline(force(block_ix,string_ix,tension)),I)
value(incline(Axis),I)
value(sense(force(block_ix,string_ix,tension)),S)
value(sense(Axis),S)
value(
   projection(force(block_ix,string_ix,tension),Axis),
   V)
```

The first subgoal is satisfied by an access function rule,

```
patient(force(Patient,Agent,Type),Patient).
```

which binds the constant `block_ix` to the variable `Body` in the goals listed above, so the next goal attempted by CASCADE is

```
value(fbd(block_ix),Fbd)
```

Achieving this goal leads CASCADE to recognize the forces acting on `block_ix`, to enter them on the free-body diagram, and to select axes for the fbd. The next five goals are all access functions that verify that there is an axis along the tension force vector. Achieving these goals sets up the last goal:

```
value(projection(force(block_ix,string_ix,tension),
                 axis(block_ix,x,35)),V).
```

This goal unifies with a version of Newton's first law, which states that the sum of forces on a body is zero if the body is at rest. Applying Newton's law leads CASCADE to finding the weight of `block_ix`, and eventually this produces a correct answer.

2.2 Explaining Examples

An example is a problem and its solution. Properly speaking, a solution should be a derivation, where a derivation consists of at least a complete proof tree and perhaps even a complete trace, including failures as well as successes. However, it is clear from Figure 9–1 that physics examples do not print a complete derivation. Instead, they print a partial

derivation in the form of a series of statements that often include equations and diagrams. Such statements are represented to CASCADE with a list of literals called lines. For instance, the solution of Figure 9–1 is represented with four lines:

```
lines(ix,
 [value(bodies(ix),[block_ix]),                          (1)
  value(                                                 (2)
    fbd(block_ix),
    fbd([arrow(35,1),arrow(115,1),arrow(90,-1)],
        [axis(block_ix,x,35),axis(block_ix,y,115)])),
  value(
    magnitude(force(block_ix,string_ix,tension)),        (3)
    0-0+(1*(m*g)*sin(35))),
  value(
    magnitude(force(block_ix,plane_ix,normal)),          (4)
    0-(1*(m*g)*cos(35))+0)]).
```

The lines of the formal representation do not quite match up with the statements in the examples. For instance, statement 7 of Figure 9–1 contains a system of equations, while lines 3 and 4 above contain the solution to that system of equations. This inaccuracy is mostly the result of the lack of expressive power of this simplified, expository representation. The equation-based representation used by the current version of CASCADE allows a more faithful formalization of example solutions.

Explaining a line is modeled as finding a proof for that line. Because the line gives a value for the sought quantity, finding a proof requires less search than it would if the value were unknown (i.e., a variable), as it would be during problem solving. The algebraic form of the value often eliminates rules that would otherwise be applicable. For instance, at one point in problem ix, CASCADE is trying to find the projection of the normal force onto the x-axis. Knowing that the value of this quantity is 0 prohibits it from applying the following projection rule,

```
value(projection(Vector,Axis),
      Sign*Mag*TrigFunction) :-
  off_axis(Vector,Axis),
  value(proj_sign(Vector,Axis),Sign),
  value(magnitude(Vector),Mag),
  value(proj_trigfn(Vector,Axis),TrigFunction).
```

because 0 will not unify with `Sign*Mag*TrigFunction`. In order to allow unification to prune the search properly, values mentioned in the lines are not simplified. For instance, the third line of problem `ix` states that the tension is `0-0+(1*(m*g)*sin(35))` instead of `m*g*sin(35)`.

2.3 Analogical Search Control

When the subjects in the Chi, Bassok, Lewis, Reimann and Glaser (1989) study began work on a physics problem, they typically looked at the diagram for a few seconds, remarked that the problem looked similar to an earlier example, and paged through the textbook in order to expose that example. They would then read the problem statement and begin solving the problem. As they solved the problem, they would often look at the example. Even if they did not overtly refer to the example, one could sometimes infer from their comments that they were referring to their memory of their explanation of the example. CASCADE's problem solver has been equipped to model this behavior with a mechanism called analogical search control.

When CASCADE begins to solve a problem, it retrieves one or more examples whose diagrams are similar to the problem's diagram. Since visual image retrieval is outside the scope of this research, the outcome of such retrievals are simply given to CASCADE as a table of facts, such as:

```
analogical_retrieval(s1,sx).
analogical_retrieval(q5,sx).
analogical_retrieval(q5,ix).
```

The first fact indicates that the diagram of problem `s1` is similar to the diagram of example `sx`. The next two facts cause both examples `sx` and `ix` to be retrieved for problem `q5`. Not all problems have such facts, so CASCADE cannot use analogical search control for those problems.

The next step is to find a mapping between the description of the problem and the description of the example's problem. Both are represented as lists of ground literals (facts). The map consists of pairs of atomic constants. Finding such a map is an NP-complete problem, so heuristics are used to solve it. The main heuristic is that the types of constants must match: Physical objects can only be paired with other physical objects, numerical values with other numerical values, situation tags with other situation tags, and so on. CASCADE only produces maps that maximize the number of facts that are put into correspondence. If

there are ties, then one map is chosen. If backtracking returns to this point, another map is chosen.

These steps initialize an analogy between the problem and the example. They are intended to correspond to the long pauses seen at the beginning of protocols before the subjects set pencil to paper and begin producing lines.

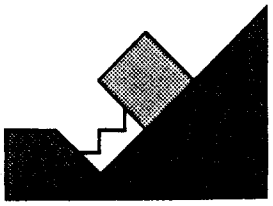
Once problem solving is in progress, CASCADE uses the analogical map to control its search. Whenever it must choose a rule for achieving a goal, it first tries to find a goal in the example's derivation that is isomorphic under the map to the current goal. If it succeeds, the rule that is used with the old goal is selected for use with the current goal. If it fails, CASCADE arbitrarily chooses a rule whose conclusion unifies with the goal (actually, it choose the first such rule in the file). Thus, the analogy is used solely for controlling search, which is why it is called analogical search control.

For an illustration of analogical search control, let us follow CASCADE as it solves problem i3, which is shown in Figure 9–2 along with the formalization of example ix's problem. CASCADE begins by retrieving a similar problem, ix. It generates mappings between the facts describing the example and the problem, and it selects the following one, which pairs the taut string with the compressed spring:

```
[(ix,i3),
 (standard_situation,standard_situation),
 (block_ix,block_i3),
 (block,block),
 (string_ix,spring_i3),
 (string,compressed_spring)
 (plane_ix,plane_i3),
 (plane,plane),
 (35,40),
 (m,10)]
```

This mapping is stored for later use. Now, CASCADE begins problem solving by making the sought quantity of i3 a goal. The mapping is applied to this goal to create an equivalent goal expressed in terms of the example's constants:

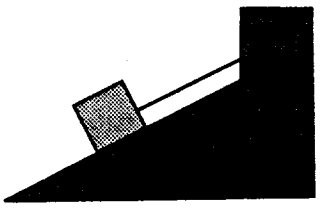
```
value(
  magnitude(force(block_i3,spring_i3,compression)),
  Cmag)
```



Problem: The figure above shows a block being held in place on an inclined, frictionless surface (theta = 40 degrees) by a massless spring. If the mass of the block is 10 kg, what force must the spring be exerting on the block?

Representation of the problem:

```
givens(i3a,
  [current_situation(i3a),
   instance(i3a,standard_situation),
   instance(bi3a,block),
   instance(spring_i3a,
      compressed_spring),
   instance(plane_i3a,plane),
   massless(sprint_i3a),
   slides_on(bi3a,plane_i3a),
   tied_to(sprint_i3a,bi3a),
   at_rest(bi3a),
   above(bi3a,plane_i3a),
   above(bi3a,spring_i3a),
   right(bi3a,spring_i3a),
   given(incline(plane_i3a),40),
   given(incline(spring_i3a),40),
   given(mass(bi3a),10)]).
soughts(i3a,
   [value(magnitude(force(bi3a,
      spring(i3a,compression)),
      Cmag)]).
```



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

Representation of the problem:

```
givens(ixa,
  [current_situation(ixa),
   instance(ixa,standard_situation),
   instance(bixa,block),
   instance(spring_ixa,
      spring),
   instance(plane_ixa,plane),
   massless(sprint_ixa),
   slides_on(bixa,plane_ixa),
   tied_to(sprint_ixa,bixa),
   at_rest(bixa),
   above(bixa,plane_ixa),
   above(bixa,spring_ixa),
   right(bixa,spring_ixa),
   given(incline(plane_ixa),35),
   given(incline(spring_ixa),35),
   given(mass(bixa),10)]).
soughts(ixa,
  [value(magnitude(force(bixa,
      spring_ixa,tension)),
      Tmag),
   value(magnitude(force(bixa,
      plane_ixa,normal)),Nmag)]).
```

Figure 9–2. Analogy between example and problem.

Applying the map to this goal fails because the atom `compression` is not in the map. CASCADE's analogical inferences are restricted so that it does not try to extend its map by transferring unmatched items directly from an example to a new problem. This failure leaves CASCADE back where it started, with no advice on which rule to pick. So it picks the rule that appears first in the file. After descending through some more subgoals without getting any help from analogical search control,

CASCADE starts encountering goals that contain only atoms mentioned in the givens (i.e., atoms that appear in the analogical mapping). One such goal is the following:

```
value(bodies(i3),BodySet)
```

This goal is mapped to become:

```
value(bodies(ix),BodySet)
```

CASCADE now searches its database for a fact of the form

```
derivation(ix,bodies(ix), RuleName)
```

and finds

```
derivation(ix,bodies(ix),1_body_problem).
```

Such derivational triples are stored at the conclusion of each goal during both problem solving and example explaining. A triple indicates the problem's name, the quantity sought by the goal, and the rule used to achieve the goal. Having found an old goal that is isomorphic to the current goal, CASCADE selects its rule, `1_body_problem` in this case, for trying to achieve the current goal. If it did not have this advice, CASCADE would first try to find three bodies and fail, then try to find two bodies and fail, and finally apply the rule `1_body_problem`.

The above illustrates how analogical search control can reduce search substantially. Moreover, when CASCADE fails to obtain advice from its experiences, it just goes on with its normal problem solving. There is no need for the kind of elaborate plan-repairing mechanisms typically found in case-based problem solving or derivation replaying.

Analogical search control first appeared in Jones's (1989) cognitive architecture, EUREKA. However, EUREKA did not view problems as first-class objects; only goals were first-class objects. It would do new analogical retrievals and mappings with every search control decision. That is, it would retrieve all goals that appeared similar to the current goal, draw up maps to each, decide which old goal most closely matched the current goal, and then use its rule for the current goal. This procedure slows EUREKA down—as several NP-complete problems (partial matches) must be solved for each rule selection decision. In contrast, CASCADE solves one NP-complete problem for each example-problem pair, then uses the resulting map throughout the attempt to solve that

problem. Making a search control decision is rather fast because converting the current goal to an isomorphic old goal is linear in the size of the goal, and finding an old goal in the derivation is linear in the size of the derivation (and with appropriate indexing, it could be even faster).

Even with this improvement, analogical search control does not always speed up problem solving, although it happens to speed up problem solving in the experiments described below. It reduces the time per problem from 308 cpu seconds to 170 seconds (on a DecStation 3100). In these experiments, the problems include about 5 to 12 objects in them, which makes the mapping process run in time proportional to 5! to 12!. The searches generally have branching factors (without analogical search control) of about 1.5 and depth of about 20 to 50, whereas the branching factor is significantly less when analogical search control is turned on. In this experiment, the cost of running the mapping process was less than the speedup obtained by reducing the branching factor. However, if the derivations were short and the number of objects in the problems were large, then analogical search control would actually slow the problem solver down. Notice that this trade-off has nothing to do with the details of analogical search control per se. Any analogical problem solver that solves the NP-complete mapping problem and uses the result to reduce search faces the same trade-off. Fortunately, if speed were the only consideration, it would be simple to implement a manager that calculates the expected times of analogical and non-analogical problem solving and chooses the one that maximizes speed. In CASCADE's case, speed is not the major concern, because analogical search control is needed for modeling the behavior of subjects.

2.4 Explanation-Based Learning of Correctness

As we mentioned in the Introduction, CASCADE's method for learning new rules is based on completing derivations. Its particular method is called explanation-based learning of correctness (EBLC). When the system cannot achieve a goal with its domain theory but can achieve it with overly general rules, the system saves a specialization of the overly general rule it used as a new rule in the domain theory. The specialization is generated with EBL-like techniques.

The basic idea of using EBL to increase the correctness of a domain theory is not new. Nelson Goodman (1956) speculates that scientists often use "overhypotheses" to constrain their inductions from data. The

overhypothesis "All gems of a given type have the same color" allows the scientist to induce from only a single example that "All emeralds are green." Goodman was not interested in computational models of learning, so it was not until Roger Shank's (1986) group implemented SWALE that the idea of using overly general rules to constrain single-example induction was linked to failure-driven learning.

If SWALE could not explain a fact in a story, it would try to explain it with an explanation pattern. An explanation pattern contains knowledge, such as "stress causes heart attacks," that is not always true. Thus, SWALE can be viewed as applying overly general knowledge to explain specific examples. The resulting explanation is stored in a case memory, where it can be used to assimilate and explain more stories. Adding a new case to memory could be seen as augmenting SWALE's domain theory. Another early system that uses the same basic idea is OCCAM (Pazzani, Dyer, & Flowers, 1986; Pazzani, 1990). It uses causal attribution rules, such as "The action immediately preceding a state is the cause of the state." Such rules express overly general knowledge. If OCCAM cannot explain an event with its domain theory and it can explain the event with its causal attribution rules, then it forms a tentative generalization of the explanation with EBL. If the tentative generalization explains other examples or is otherwise confirmed, then it becomes a full-fledged member of the domain theory. Causal attribution rules have also been used by other, similar learning systems (Anderson, 1990; Lewis, 1988).

Another popular type of overly general rule, called a determination, indicates that one property's value determines another's, but it does not specify the nature of the dependency (Widmar, 1989; Bergadano, Giordana, & Ponsero, 1989).

In all these systems, the basic idea is to explain examples with the domain theory if possible and use risky overly general knowledge if that fails. The resulting explanation is processed to extract some kind of knowledge that is midway in generality between the explanation itself and the overly general knowledge used to form it.

The rationale for this kind of learning is that overly general knowledge holds a grain of truth but is not trustworthy enough to be used regularly, so it is used only when domain knowledge is exhausted. If some overly general knowledge applies successfully to a particular situation, then it is likely that it can be used again for similar situations. Specializing it makes it apply only in similar situations. Thus, it makes sense to save a specialization of the overly general knowledge as domain knowledge if

that knowledge helps achieve a goal that domain knowledge could not achieve.

All cognitive models based on this idea assume that in some task domains, people believe that there is only a certain set of correct rules—the official rules of the game, so to speak—which we call the domain theory. In most systems, the domain theory and the overly general knowledge are represented in different formats. CASCADE uses the same format for both domain theory rules and overly general rules, and, thus, it must use some kind of marking to distinguish them. Since we eventually would like to use CASCADE to study concurrent learning in multiple task domains, its marking system allows rules from several domain theories to reside concurrently in memory. Each rule is marked with the names of the task domains it is relevant to. Thus, a rule might bear `[physics, chemistry]` as its mark to represent the belief that this rule is part of the official knowledge of both physics and chemistry. Problems are also marked with the names of the task domains. For instance, the problems currently solved by CASCADE are marked `[physics, math]`, which indicates that both physics and math knowledge is relevant. When solving a problem, CASCADE uses only rules whose mark intersects with the problem's mark. What we have been calling "domain theory" is just the set of rules that bear the problem's mark.

Many rules are not marked, because they represent reasoning that is not regulated by higher authorities, such as physicists and chemists. For instance, the rules for planning a birthday party are unmarked. Rules may also become unmarked if they were once thought to be proper rules of the task domain but then were found to be incorrect. This unmarking reflects the fact that people can often recall incorrect rules as well as the fact that they are incorrect. Overly general rules are incorrect and unmarked. As illustrations, consider three rules:

1. *The magnitude of a tension force is equal to the tension of the string exerting the force.* This rule is marked `[physics]`.

2. *The value of a property of an object is equal to the value of a property of one of the parts of the object.* This is an overly general rule in that it sometimes does not make physically correct inferences. It is unmarked.

3. *A string can only pull an object, not push it.* Although a correct rule in that it makes only physically correct predictions, this rule

belongs to commonsense physics. It uses the concepts "pull" and "push," which are not formal physics concepts. It is unmarked.

In order to express overly general rules conveniently, CASCADE uses a nonstandard syntax for functions and predicates. For example, the function `tension(P)` is written `f(tension,P)`, so that the functor appears in the first argument position of the dummy function `f`. There are similar conventions for writing selected predicates. Thus, overly general rules can be written with variables for functors; for example:

```
value(f(Property1,Whole),V) :-
    part(PartName,Whole,Part),
    value(f(Property2,Part),V).
```

This rule says that a property of the whole often has the same value as a property of one of its parts.

EBLC creates a new rule by extracting the instantiation of the overly general rule from the derivation and substituting variables for the constants that denote objects or values. For instance, in explaining the example of Figure 9–1, CASCADE uses the following rule:

```
value(f(magnitude,Force),V) :-
    instance(Force,tension_force),
    agent(Force,String),
    value(f(tension,String),V).
```

This rule says that the magnitude of a tension force is equal to the tension in the string causing it. If this rule is deleted from the initial domain knowledge, CASCADE will use the overly general rule mentioned above to resolve the resulting impasse. The instantiation of the overly general rule is as follows:

```
value(
  f(magnitude,f(force,block_ix,string_ix,tension),
  0-0+(1*(m*g)*sin(35))) :-
    agent(f(force,block_ix,string_ix,tension),
          string_ix),
    value(f(tension,string_ix),
          0-0+(1*(m*g)*sin(35))).
```

When variables are substituted for objects and values, the following results:

```
value(f(magnitude,f(force,B,S,tension), V) :-
    agent(f(force,B,S,tension),S),
    value(f(tension,S), V).
```

This rule is syntactically different from the deleted rule, but it performs identically.

This method of specializing overly general rules implements the idea that no two problems have the same objects, and they rarely have the same values, so rules should not mention such constants explicitly. We believe that people have many such generalization heuristics and that those heuristics are domain specific. For instance, a mathematical generalization heuristic is to substitute variables for all numerical constants except 0, 1, π, or other special constants. Most other EBLC-like systems use the standard EBL technique of replacing the constants that are introduced by the example. This technique would not work for CASCADE, because the examples often introduce constants (e.g., the names `string` or `tension_force`) that must appear in the resulting rules.

Most EBLC-like systems use an operationality criterion to extract an antecedent for the learned rule from the explanation. Often, the leaves of the explanation (proof tree) are extracted and become clauses in the new rule. CASCADE selects much higher nodes in the proof tree, which makes its rules more general. In fact, it chooses the highest possible nodes: those that are immediate descendants of the impasse goal. These nodes are always produced by an application of an overly general rule, which means that the learned rule is a specialization of the overly general rule. Clearly, this is also a heuristic that requires further testing even though it succeeds for the experiments below. It may even be that no single-example operationality criterion will suffice. Cohen (1990) and VanLehn (1987) demonstrate that the decision about which nodes to use as antecedents can be based on comparing explanations from multiple examples.

3. What Overly General Rules Are Needed for Learning Physics?

The major source of power of explanation-based learning of correctness is the collection of overly general rules. They serve two purposes. If they can resolve an impasse, they strongly determine the contents of the new

rule. Although the generalization mechanism discussed earlier determines whether the arguments in the clauses are constants or variables, all the rest of the rule is determined by the overly general rule. In particular, it determines which clauses will appear in the body of the new rule. The second role of the overly general rules is to reject impasses that are not worth learning from. If the rules cannot achieve a goal, and CASCADE is solving a problem, then the overall problem-solving attempt fails, and the system goes on to the next problem.

In order to study the power of overly general rules, we had CASCADE acquire a nontrivial amount of knowledge in physics. This knowledge takes good students about 4 hours to learn, and poor students fail to learn some parts of it. So the knowledge-acquisition problem is definitely nontrivial for human learners. We set up the system's learning problem to be as close as possible to the humans' learning problem. The humans were the eight subjects in the Chi, Bassok, Lewis, Reimann and Glaser (1989) study. They read a college physics textbook, then studied three example problems, and then worked 25 problems. CASCADE cannot read, so we gave it an initial domain theory that corresponds to the knowledge available by reading the textbook.

In order to determine the initial domain theory for this experiment, a target domain theory was developed that could solve 23 of the 25 problems (the remaining two required kinematic knowledge, which we did not bother to formalize) and explain all three examples. It was based on the analyses of Bundy, Byrd, Luger, Mellish, and Palmer (1979) and Larkin (1981, 1983), as well as our own analysis of the Chi protocols. The target domain theory had 62 rules. Two people who were not involved in developing the target domain theory were asked to judge whether each rule was mentioned in the text studied during phase 1. The judges agreed 95% of the time, and disagreements were settled by a third judge. Of the 62 rules, only 29 (47%) were judged to be present in the text. The learning problem is to learn the remaining 33 rules via explaining the examples and solving the problems. (Actually, the knowledge representation used by CASCADE differs from that used by this target domain theory, so the exact number of rules learned is not the same, although the knowledge content is.)

We had CASCADE explain the three examples, then solve the 23 problems. As it went along, it would attempt to learn from its impasses. If it could not learn a correct rule from an impasse, we tried to augment the stock of overly general rules until it could. In one case, we could find

no way to get EBLC to learn a correct rule. The first subsection below evaluates the power of EBLC by examining its successes and failures. The second subsection examines the contents of the overly general rules that were invented to see whether they are ad hoc or if they have an underlying structure.

3.1 The Power of EBLC Is Limited

It may seem trivial that EBLC constructs correct rules if one can propose any overly general rule one wants. If one knows what the right rule is and can get an impasse to occur in the right place, then it seems simple to write an overly general rule that will specialize to be the desired rule. In practice, things are not so simple, because the impasse often does not provide enough context for specializing the rule. Consider two contrasting cases, both of which occurred while solving a problem in which a helium balloon pulls upward on an object. CASCADE first reaches an impasse because it knows a force is needed (by analogy with an earlier example), but it knows nothing about buoyancy forces. It sees that the balloon is tied to the object and uses an overly general rule to infer that the force results from the balloon since the two objects are physically related. This first impasse was easy for EBLC. Moments later, CASCADE reaches another impasse. This impasse is problematic. The failing goal seeks a directional sense, either up or down. (If CASCADE were explaining an example's solution to this problem, then the goal would state what the sense is.) If one were to write an ad hoc overly general rule for this impasse, one would have to specify "vertically up" in the rule. Clearly, this is a totally arbitrary overly general rule. If CASCADE were given a "down" rule as well, then it would not know which to apply and would have essentially no guidance on resolving this impasse.

When the subjects encounter this impasse, they do not seem to have any special problems with it. Our intuition is that they use common sense to infer that the balloon is pulling up and therefore the force is directed upward. In order to model this resolution of the impasse and others like it, we gave CASCADE some commonsense physics knowledge about pulls, pushes, and the things that cause them. These were expressed as 28 unmarked rules. Since they are unmarked, they can be applied only when the marked (domain theory) rules have failed. They are not immediately relevant anyway, because the problem's goal asks for the direction of a force, and these rules provide directions for pulls

Table 9–1. Overly General Rules That Reference Commonsense Physics

	Rule	Description
1.	value(f(gen_force,B) f(force,B,A,T)) :- exist(f(cs_force(B,A,T))).	If there is a commonsense force of type T on a body B caused by an agent A, then there is an analogous formal physics force.
2.	value(f(Prop,f(force,B,A,T)), V) :- exists(f(cs_force,B,A,T)), value(f(Prop,f(cs_force,B,A,T)),V).	If a commonsense force has a property Prop with value V, then so does the analogous formal physics force.
3.	value(f(gen_accel,B f(accel,B)) :- exists(f(cs_accel,B))),	If there is a commonsense acceleration on a body B, then there is a formal physics acceleration on it.
4.	value(f(Prop,f(accel,B)),V) :- exists(f(cs_accel,B)), value(f(Prop,f(cs_accel,B)),V).	If a commonsense acceleration has a property Prop with value V, then so does the analogous formal physics acceleration.
5.	value(f(axes,B),B) :- exists(f(cs_axes,B)), value(f(cs_axes,B),V).	If there are commonsense axes about B, then there are formal axes about B of the same orientation.
6.	value(f(sense,f(force,B,A,T)), v(f(relpos,B,A))) :- exists(f(cs_force,B,A,T)), push(f(cs_force,B,A,T))	If there is a commonsense push analogous to a formal physics force, then the force has a positive sense iff the body is above the agent.
7.	value(f(sense,f(force,B,A,T)), v(f(relpos,A,B))) :- exists(f(cs_force,B,A,T)), pull(f(cs_force,B,A,T)).	If there is a commonsense pull analogous to a formal physics force, then the force has a positive sense iff the agent is above the body.

and pushes. To provide a bridge between the two types of quantities, CASCADE was given overly general rules that say that forces have the same property values as pushes and as pulls, provided they act on the same objects (see Table 9–1). With these rules, CASCADE resolves the second buoyancy impasse by using a commonsense physics rule that says that balloons pull upward and an overly general rule that says that forces have the same property values as commonsense pulls and pushes (rule 7 in Table 9–1). Applying both of these rules, it infers that its newly invented formal physics force is probably directed upward.

During example studying and problem solving, CASCADE encountered 23 impasses. Of these, 14 required commonsense physics reasoning for their resolution. Most of the time (12 of the 14 cases), commonsense physics was necessary during problem solving, when there is less context

than during example explaining. The moral is that overly general rules often do not provide enough constraint in themselves for learning. The extra constraint has to come either from the example, if it is explaining one, or from another knowledge source, which was commonsense physics in this case.

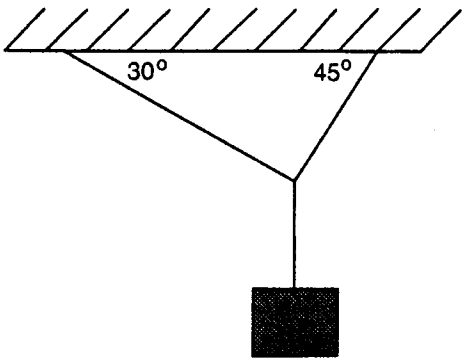
For all but one of the 23 impasses, we found overly general rules and commonsense physics rules that would cause appropriate learning. The single case in which EBLC would not work is important because it indicates that there are limits to the applicability of this approach, so it will be discussed in some detail.

The case occurred while explaining the second line of the example shown in Figure 9–3. The line says "Consider the knot at the junction of the three strings to be the 'body.' " The choice of a body in solving force problems is essentially a heuristic decision. Although one can resolve forces about any point one likes, which point one chooses can make a big difference in the success of the problem solving. Even expert physicists can guess wrong. Usually, they make a choice, then plan a solution based on it to see whether their choice works (Larkin, 1983). For instance, they might say for this problem, "If I choose the knot as the body, then I get three tension forces on it to resolve. I can get the magnitude of the vertical one from the block, and I can get the other two from that. So that's everything. Let's do it." Notice that the initial guess about the choice of body is just a heuristic guess. Although CASCADE 3 cannot do the kind of abstract planning that the experts use to check their guesses (CASCADE 2 could, but that facility was taken out when learning was added), it should be able to guess just as well as the experts. The learning problem is to have it acquire appropriate rules for guessing bodies. In this case, an appropriate heuristic might be the following:

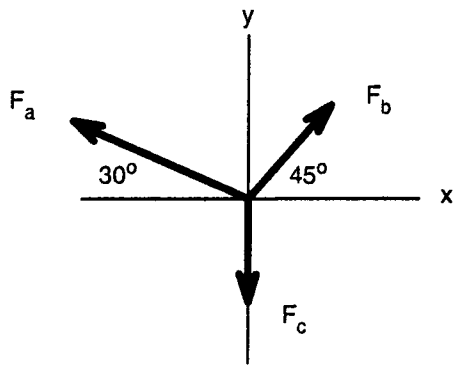
```
If there are three or more objects attached to a
     potential body B,
   and B is not the earth,
   and one or more of the attached objects is part
     of the definition of a sought or given quantity,
then choose B as a body.
```

We could not think of a rule that is a generalization of this and yet is at all plausible as something that people are likely to know before they learn mechanics.

Problem: Figure (a) shows an object of weight W hung by strings. Consider the know 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?



(a)



(b)

Solution:

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

$$\mathrm{Fax} + \mathrm{Fbx} = 0$$

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

$$\mathrm{Fax} + \mathrm{Fbx} = 0$$
$$\mathrm{Fax} + \mathrm{Fby} + \mathrm{Fcy} = 0$$

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

$$\mathrm{Faz} = \mathrm{Fbz} = \mathrm{Fcz} = 0$$

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

$$\mathrm{Fax} = -\mathrm{Fa}\cos 30^\circ = -0.866\mathrm{Fa},$$
$$\mathrm{Fay} = \mathrm{Fa}\sin 30^\circ = 0.500\mathrm{Fa},$$

and

$$\mathrm{Fbx} = -\mathrm{Fv}\cos 45^\circ = 0.707\mathrm{Fb},$$
$$\mathrm{Fby} = \mathrm{Fb}\sin 45^\circ = 0.707\mathrm{Fb}.$$

Also,

$$\mathrm{FCy} = -\mathrm{Fc} = -\mathrm{W}$$

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:

$$-0.866\mathrm{Fa} + 0.707\mathrm{Fb} = 0$$
$$0.500\mathrm{Fa} + 0.707\mathrm{Fb} - \mathrm{W} = 0$$

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 9–3. A physics example with a strange body.

When we turned to the protocols for inspiration, we found that all the subjects who bothered to explain this line also got stuck. For instance, one subject said, "Why should this be the body? I thought W was the body. OK, let's see later." The subject never finds an explanation for the line but goes on anyway, accepting the example's assertion as correct. This acceptance of an unproved assertion is often called abduction (Pople, 1973; O'Rorke, Morris, & Schulenburg, 1990). When the subjects later encountered problems that were isomorphic to this example, they would often explicitly mention the example as a warrant for assuming that the (new) knot should be a body. Clearly, they were reasoning by analogy. We built a specific abduction/analogy mechanism in CASCADE to model this behavior. It produces rules that say, "If the current problem is analogous to problem `sx` [the one shown in Figure 9–3], then the body is analogous to `knot_sx` [the knot shown in Figure 9–3]."

EBLC cannot directly learn expert-level heuristics such as the one mentioned earlier. The difficulty in this case is that EBLC forms a new rule at the time of the impasse, but some aspects of the rule to be learned can only be inferred later. If the learner analyzed the whole solution, it might discover that a key feature is that the attached objects (the strings in this case) occur in sought quantities (the forces in this case), but other examples would be needed to discover that they could occur in given quantities as well. Since the earth (e.g., the ceiling in Figure 9–1) is often attached to important objects but is rarely a body, some kind of discrimination learning might be necessary to learn the remaining condition, which is that the body not be the earth. The bottom line is that EBLC has limited powers despite the use of an open-ended collection of powerful rules. To learn some things requires information from problem solving that takes place after EBLC forms a rule.

In short, we have discovered two fundamental limitations on the power of EBLC. The first is that when it applies in the impoverished context of problem solving, it often needs another source of knowledge to specialize the overly general rules. In this experiment, that source was knowledge about commonsense physics. The second limitation is that forming a new rule at the time of the impasse's resolution may be too early. Crucial information needed for learning a good rule via EBLC may not appear until later, and perhaps not even in the same problem. Although a machine learning system might adopt a wait-and-see approach and form rules only when ample information is available, this is not what people do, perhaps because of the memory costs. People seem to use a

combination of abduction and analogy, and that is what CASCADE uses in the single case in which EBLC does not work.

3.2 The Contents of the Overly General Rule Base

The second issue to be discussed is the exact nature of the collection of overly general rules that was invented. This collection was invented bottom up, so to speak, in that all rules were used at one time or another to learn at impasses that actually occurred during problem solving.

Tables 9–1 and 9–2 show the overly general rules contained in CASCADE's current collection. Table 9–1 was discussed earlier, since its rules are responsible for linking formal physics reasoning to commonsense physics reasoning. This section focuses on the contents of Table 9–2. Rules 1, 5, 7, and 9 generated one rule each. Rule 11 generated two rules. CASCADE did not apply the others during these experiments, although they were used earlier in CASCADE's development.

Examination of Table 9–2 (and Table 9–1, too) reveals that the overly general rules are far from ad hoc. In fact, there appears to be a tacit generalization hierarchy among them. For instance, the first four rules all express the same idea, that parts and wholes share property values. Although the current syntax of the representation requires different rules for properties with differing numbers of arguments, a more flexible representation would allow writing a single more general rule that subsumes rules 1 through 4. Similarly, rules 5 and 6 could be generalized to become a single rule that says that a whole's property can be computed by a single function evaluation from a part's property. In this case, two particular functions, `perpendicular` and `opposite_sense`, are used instead of a variable functor. Similarly, rules 7 and 8 are also more specific than they could be. Rule 7, for instance, could have variables in functor locations:

```
value(f(P,B), f(F,B,A)) :-
    predicate(Pred,B,A),
    instance(Pred,physical_relation).
```

This rules says that the value of a property `P` of an object is a specific two-place function `F` if there is some physical relationship between the arguments of `F`.

It would be interesting to represent this generalization hierarchy explicitly and also to replace the binary distinction of domain versus nondomain with a continuous scale (see Rosenbloom & Aasman, 1990).

Table 9–2. General Overly General Rules

	Rule	Description
1.	value(f(Prop,Whole), V) :- part(PartName,Whole,Part), value(if(Prop,Part), V).	A unary property of a whole has the same value as the same property of a part.
2.	value(f(Prop,Part), V) :- part(PartName,Whole,Part), value(f(Prop,Whole), V).	A unary property of a part has the same value as the same property of the whole
3.	value(f(Prop1.Whole), V) :- part(PartName1,Whole,Part1), part(PartName2,Whole,Part2), not(PartName1 = PartName2), value(f(Part2,Part1,Part2), V).	A unary property of a whole has the same value as a binary property of two parts.
4.	value(f(Prop1.Part1,Part2), V) :- part(PartName1,Whole,Part1), part(PartName2,Whole,Part2), not(PartName1 = PartName2), value(f(Prop1,Whole), V).	A binary property of two parts has the same value as a unary property of the whole.
5.	value(f(Prop,Whole), perpendicular(V)) :- part(PartName,Whole,Part), value(f(Prop,Part),V).	The value of a whole's property is perpendicular to the value of the same property of a part.
6.	value(f(Prop,Whole), opposite_sense(V)) :- part(PartName,Whole,Part), value(f(Prop,Part),V).	The value of a whole's property is the opposite sense from the value of the same property of a part.
7.	value(f(gen_force,B),f(force,B,A,T) :- predicate(Pred,B,A), instance(Pred,physical_relation).	A force exists between body B and agent A if B and A are related by a physical relation.
8.	value(f(gen_force,B),f(force,B,A,T) :- predicate(Pred,A,B), instance(Pred,physical_relation).	A force exists between body B and agent A if A and B are related by a physical relation.
9.	value(f(P1,f(P2,X)), V) :- value(f(P3,f(P4,X)),V2), value(f(P3,f(P4,Y)),V2), derivable(f(P1,f(P2,_)), f(P3,f(P4,_))), value(f(P1,f(P2,Y)),V).	If P3(P4(X))=P3(P4(Y)) and P1(P2(_)) can somehow be derived from P3(P4(_)), then P1(P2(X))=P1(P2(Y)).
10.	value(f(P1,f(P2,X)), V) :- value(f(P3,f(P4,X)),V2), value(f(P3,f(P4,Y)),V2), derivable(f(P1,f(P2,_)), f(P3,f(P4,_))), value(f(P1,f(P2,X)),V).	If P3(P4(X))=P3(P4(Y)) and P1(P2(_)) can somehow be derived from P3(P4(_)), then P1(P2(Y))=P1(P2(X)).
11.	value(f(P,X), V) :- exists(f(force(P),B,A), value(f(magnitude,f(force(P),B,A))),V).	If a force of type P exists, and a property whose name is P is needed, then its value is the magnitude of the force.

In particular, rules could have an estimated probability of correctness associated with them. Rules that are currently marked as being domain rules would bear 1.0 as their estimated probability of correctness. Overly general rules would have estimated probability of correctness of less than 1.0. More general rules would have lower values than more specific rules. For instance, rule 1 in Table 9–2 is more specific than the overly general rule mentioned earlier,

```
value(f(Prop1,Whole), V) :-
    exists (Whole),
    part(PartName,Whole,Part),
    value(f(Prop2,Part),V).
```

which allows the whole and the part to have different properties. Presumably, the more specific rule of Table 9–2 would have a higher estimated probability of correctness than this more general rule. When EBLC forms a new rule, the rule would start with a low estimated probability of correctness, but the estimate would be increased every time the rule participated in a correct derivation. Moreover, whenever EBLC forms a new domain-knowledge rule, it would also form rules that are intermediate in the generalization hierarchy, such as rules 5 and 6. As the domain-level rules are used and confidence in their correctness increases, some of this confidence is passed along to the intermediate rules. Thus, the system would learn how to learn things in this domain. This generalization hierarchy might provide a model for the common belief that experts have better "intuition" than novices and can more reliably guess a correct explanation in situations in which they lack appropriate knowledge.

4. Which Impasses Should Cause Learning?

Many learning systems use the same basic idea: If you fail to achieve a subgoal, then work hard to resolve the failure (impasse) and store the resolution as a new rule for achieving that subgoal. Although this idea seems compellingly natural, it is only a heuristic solution to the problem of blame assignment. By placing the new knowledge at the failed subgoal, the learner is assuming that the domain theory is incomplete and that the missing rule should have been executed exactly where the impasse occurred. In fact, the impasse could have been caused by missing rules that should have been used much earlier than the impasse or

even by incorrect rules that were used earlier than the impasse. The impasse could also be caused by incomplete or incorrect search control knowledge. Inadequate search control knowledge proved to be a particularly fertile source of impasses in the experiment. For instance, when CASCADE tries to explain the following lines from Figure 9–1,

$$F_{a_x} = -F_a \cos 30$$
$$F_{a_y} = F_a \sin 30,$$

it always starts by resolving forces along the x-axis. This convention succeeds for the first equation, but it fails for the second, so CASCADE backs up and tries resolving along the y-axis, which succeeds. The cause of the impasse during the second equation is inadequate search control knowledge. If the system resolved this impasse with EBLC, it would learn an incorrect trigonometric rule. (The rule would assert that sine is the trigonometric function to use when the axis is adjacent to the angle.)

Basically, impasses can be caused either by inadequate search control or by inadequate domain knowledge. If an impasse is caused by an incorrect choice earlier along the current search path, the appropriate response is to back up to that point and try a different choice. (Some systems take this opportunity to modify their search control knowledge to prevent this kind of mistake from happening again. CASCADE does not, because it uses analogical search control.) If the impasse is caused by an inadequate domain theory, the appropriate response is to construct a new rule or modify existing ones.

CASCADE's heuristics for solving the assignment-of-blame problem are implemented as follows. The problem (or an example line) is attempted first with all impasses handled by backing up. If this attempt succeeds, the domain theory is adequate for this problem, so CASCADE merely goes on to the next problem or example line. If the attempt fails, thus indicating that the domain theory is incomplete, CASCADE tries again treating impasses as triggers for learning. If it succeeds in resolving the first impasse it encounters, it learns a new rule. This learning event changes the domain knowledge, so for the rest of the problem, impasses are again handled by backing up. If this second attempt finishes successfully, CASCADE goes on, retaining the rule that it learned. If the attempt fails, then CASCADE still retains the rule but tries the problem again treating impasses as triggers for learning. This process continues until it eventually succeeds, or until it is impossible to resolve one of the impasses with overly general rules.

This approach to the assignment-of-blame problem is based on several heuristic assumptions. First, it assumes that the search control knowledge and the domain theory are correct albeit incomplete. In particular, if the domain theory can generate a correct solution path, search will not block the system from following it. This makes it possible for the system to handle a special case that arises frequently. If the domain theory is adequate for solving a problem, but the search control is not good enough to choose a correct solution path on the first attempt, then impasses will occur when the system reaches the ends of nonsolution paths. To handle this case, CASCADE makes a first pass at solving the problem with all failures handled by backing up. If it solves the problem on this pass, then it does not need to learn any new domain theory, because the present theory is adequate. If it fails on this pass, it makes a second pass, wherein impasses cause learning of new domain knowledge.

Second, this approach to blame assignment assumes that the first impasse encountered during the second pass is worth resolving and learning from. This amounts to assuming that on the problems in which new domain theory needs to be learned, the first path generated by the search control mechanism is a solution path.

To demonstrate the interaction between EBLC and analogical search control, another run over the Chi problems was conducted with analogical search control turned off. In this run, CASCADE explains the three examples, learning new rules from impasses it encounters. To solve the problems, however, we disabled the analogical search control mechanism. Thus, CASCADE is able to use the domain knowledge acquired during example studying, but not the search control knowledge. While solving problems under these conditions, CASCADE learns ten rules, only 6 of which are correct. In comparison, when CASCADE solves the problems with analogy turned on, it learns 15 rules, all of which are correct. Thus, it appears that analogical search control (or some other kind of search control) is required if EBLC is to operate correctly during problem solving.

A particularly strong interaction between analogical search control and domain theory learning appears when universal quantification occurs in a knowledge base. By "universal quantification," we mean the kind of construct that says, "For all known objects x such that $\mathrm{P}(x), \ldots$" If the domain knowledge is incomplete, some of the objects that should be known may not be. However, the universal quantifier will not fail, since it does not care how many objects are known. Analogical search

control can cause the needed failure here, provided certain conditions are met. This process is best illustrated with the episode that led us to discover this beneficial interaction.

In the Chi et al. study, the training material introduces five force laws during problem solving, where a force law is a rule that asserts the existence of a force given that certain conditions hold. For instance, the problem of Figure 9–2 is the first place where compression forces are mentioned. When resolving forces, CASCADE uses a universal quantifier (similar to a PROLOG "setof") to generate all the forces on the given body. However, it can only generate forces that it knows about. During the processing of the problem of Figure 9–2, it does not know about compression force, so it finds only two forces, the normal force and the gravitational force. However, this satisfies its goal of finding all the known forces acting on the body, so it goes on applying Newton's laws, projecting forces and so on. In this problem, that attempt ultimately fails because the sought quantity explicitly asks for the magnitude of the compression force. However, in some problems, the missing-force solution goes through fine even though it produces an incorrect answer. Neither CASCADE nor the Chi, Bassok, Lewis, Reimann and Glaser subjects receive feedback on their answers' correctness, so there is no total failure on these problems and hence no impasse to drive the learning of the force laws. The problem is that when a rule is normally used under a universal quantifier to delineate all known elements of a set, and the content of the set is not known because CASCADE is solving a problem instead of explaining an example, then there is no simple way to detect that the set is missing elements and hence that some rules are missing from the knowledge base.

In order to get CASCADE to appropriately fail and learn, we made following the advice given by analogical search control mandatory. If analogical search control suggests a rule for a goal, and the goal cannot be achieved by the rule, then the goal fails without trying any other rule. Also, we eliminated the universal quantifiers and use different rules for sets of different sizes (see the discussion of the rule `1_body_problem` in the section introducing analogical search control). These changes caused CASCADE to correctly acquire the five force laws. For instance, on problem `i3`, analogical search control demands finding three forces. Since CASCADE knows only about two, it must invent a third. Rule 7 of Table 9–2 applies, a new force is postulated, and a new rule is learned. Essentially, this change means that CASCADE now trusts the analogy

with the example enough that it is willing to invent new rules to support it. Clearly, this is a risky design, but so far it seems to be working well. Perhaps the more lasting contribution, however, is the discovery of an interaction between a representational construct (universal quantification) and impasse-driven learning.

Some interesting properties of this blame assignment heuristic emerge when it is compared to heuristics used by predecessor systems. In SIERRA (VanLehn, 1987, 1990), blame could be assigned to any failure after a total failure indicates that domain knowledge is incomplete. SIERRA first tried to prove its top goal via backchaining, saving a complete trace of the goals and subgoals. Any failed goal (impasse) became a candidate for learning. Explaining an example often produced multiple candidates. Rather than using heuristics to select one for learning, SIERRA saved the whole set of failed goals and went on to explain the next example in the training sequence. Typically, the next example also produced a set of failed goals that were candidates for learning. SIERRA intersected the two sets of failed goals and kept only those that were common to both examples. It would continue in this way until it had narrowed the set down to a singleton. This impasse was selected for learning. (If it ran out of examples before reducing the set to a singleton, heuristics were used to select an element of the set.)

Interesting sets of failed goals meant that SIERRA could learn only from groups of examples that exemplified a common piece of missing knowledge. When human training materials for SIERRA's task domain were examined, it was found that they did indeed group examples so as to exemplify a single rule. Moreover, each group was clearly identified as a distinct lesson, which suggested that when human authors develop training material for human students, they conventionally introduce at most one rule per lesson. The fact that neither authors nor students seem aware of this convention suggests that it acts like a felicity condition (Austin, 1962), in natural language conversation. Thus, SIERRA's learning procedure works correctly when trained by materials that obey the felicity condition of only introducing one new rule per lesson.

CASCADE's heuristic for assignment of blame is different from SIERRA's, but there is a felicity condition for it as well. Because CASCADE triggers learning on the first local failure after a total failure, if there is some kind of local failure in the explanation prior to the point where one would like the impasse to occur, then CASCADE will let the local failure at its end trigger learning that will probably

learn an incorrect rule. Thus, CASCADE will learn correctly only if its training material ensures that there is no local failure in an explanation prior to the use of a new piece of knowledge. This is CASCADE's felicity condition. Hall (1988) discusses felicity conditions for his derivation completion learner.

The assignment-of-blame problem is very difficult, so it is natural that the machine learning community should try many heuristics. CASCADE and SIERRA use different heuristics, although both are based on the notion of a local failure following a total failure. They use different heuristics for selecting which local failures trigger learning. Different heuristics mean that different felicity conditions define the type of training material that will cause the heuristic to function appropriately.

A felicity-conditions approach to solving the assignment-of-blame problem seems better than the customary solution, in which the first impasse of any kind triggers learning and a human must engineer the initial domain knowledge so that impasses happen only at the right points. When a system learns only from local failures after a total failure, it can still search without being derailed by incorrect impasse-driven learning. Moreover, with a well-defined heuristic for selecting which local failure to learn from, felicity conditions can be specified so that trainers can develop appropriate training material.

The knowledge representation language turned out to have a significant impact on the location of impasses, and hence the success of impasse-driven learning. As mentioned earlier, this chapter focuses on the rule-based notation of CASCADE 2. CASCADE 3 uses conditioned equations, such as this:

```
If instance(F,tension_force),
  and agent(F,S),
then magnitude(F) = tension(S).
```

This equation says that an equation, `magnitude(F)=tension(S)`, holds whenever two conditions hold: that `F` is a tension force and `S` is the agent causing it. CASCADE 3 finds values for quantities by chaining backward through the equations. In using this equation, it would seek the magnitude of a force by setting up the subgoal of finding the tension of the string causing the force. It only considers chaining through equations whose conditions are met. For CASCADE 3, an impasse is failing to find an equation to chain through. EBLC builds new conditioned equations at such impasses by specializing overly general equations.

For instance, suppose CASCADE does not know the equation just mentioned and it is seeking the magnitude of a tension force. Suppose further that it has an equation for the magnitude of gravitational forces, namely:

```
If instance(F,gravitational_force),
  and acts_on(F,B),
  and instance(G,gravitational_constant),
then magnitude(F)=mass(B)*G.
```

This equation is inapplicable because CASCADE 3 cannot prove that the sought force is gravitational force. Hence, it reaches an impasse and uses EBLC to learn the tension equation it lacked. Now, consider what CASCADE 2 would do. Its representation of the gravitational force rule appears below in the same informal syntax:

```
If instance (F,gravitational_force),
  and acts_on(F,G),
  and instance(G,gravitational_constant),
  and value(mass(B),M),
then value(magnitude(F),M*G).
```

CASCADE 2 would chain backward through this rule and reach an impasse trying to prove that the sought force is a gravitational force. EBLC would not be able to learn a new rule from this impasse. Basically, it is too low level an impasse for impasse-driven learning.

CASCADE 2 did not have EBLC, so we initially did not know about this problem. CASCADE 3's conditioned equation representation was developed mostly to model certain empirical predictions about transfer (see VanLehn, Jones, & Chi, 1992), and EBLC was developed in that context. For the purpose of writing this chapter, we decided to put EBLC in CASCADE 2 and discovered, much to our surprise, that it hardly learned at all.

In retrospect, it is clear that any impasse-driven learner will reach impasses on the lowest level goals it can get to. If one uses a very fine-grained representation, perhaps even down to the perceptual-motor level, then there would probably be many impasses down at that level. Clearly, an effective learner has to know, somehow, which levels of knowledge are probably incomplete and which are probably complete. When an impasse occurs beneath rules that are part of the knowledge believed to be complete, then the learner should pop the goal stack back to goals at a level where knowledge is likely to be missing. Alternatively, it could

try repairing the impasse at the lowest level, then try higher goals if that fails. SIERRA (VanLehn, 1990) and Soar (Newell, 1990) take the latter approach.

5. Summary and General Discussion

CASCADE's reasoner is essentially a backward chaining theorem prover similar to pure PROLOG. The problem solver is augmented with three learning mechanisms, one for search control knowledge and two for domain knowledge. Search control learning is accomplished by recording the derivation (proof tree) of an example and referring to it analogically when solving problems for advice on which rules to choose. This technique, called analogical search control, is efficient because it computes only one partial mapping per problem-example pair and then uses that mapping at every search control decision in the problem.

CASCADE's second learning mechanism, explanation-based learning of correctness (EBLC), acquires new domain rules by specializing non-domain knowledge, which includes rules from other task domains (e.g., commonsense physics) and rules that are overly general and thus not totally correct. EBLC is invoked only when CASCADE reaches an impasse, and an impasse is defined to be the first goal failure (in the PROLOG sense) after a failed exhaustive search has demonstrated that the knowledge base is incomplete and cannot solve the current problem. This definition allows search, with its concomitant failures, to be used to solve problems for which the domain knowledge is sufficient.

The third learning mechanism acquires new domain knowledge on impasses that EBLC cannot handle because it lacks appropriate overly general rules. It builds rules that explicitly invoke analogical extensions. For instance, it built a rule that says, "If the current problem is analogous to `sx`, then let the body be an object analogous to `knot_sx`."

A major goal was to get CASCADE to learn as well as the best student in the Chi, Bassok, Lewis, Reimann and Glaser (1989) study. We succeeded, for we kept adding learning mechanisms until CASCADE could learn enough to solve all problems correctly, just as the best subject did. This methodology is different from the usual one in machine learning, which is to try a single new learning mechanism on a number of classic learning problems from the literature and thus come to understand it better. We are more interested in how multiple learning strategies interact to yield human-level performance. In order to understand the

interactions, two mechanisms were tested in isolation. Analogical search control doubles the problem solver's speed when it is used with EBLC turned off. The performance of EBLC when analogical search control is turned off depends on whether the system is explaining examples or solving problems. During example explaining, analogical search control is not operative (because there is no earlier analogous problem). The fact that EBLC acquired all seven of the rules that we hoped it would acquire shows that it works well in isolation during example explaining. During problem solving, analogical search control normally does operate and helps EBLC learn all 15 rules that we hoped it would acquire. When it is turned off, CASCADE learns only six of those 15 rules and learns four other incorrect rules as well. Thus, EBLC works fine in isolation during example explaining but works poorly during problem solving unless some source of search control is available. This is to be expected. When explaining an example, values are passed along with the goal quantities. The values help control search by reducing the number of rules that unify with the goals. This tends to keep CASCADE on a correct solution path, which ensures that the impasses are the desired ones. During problem solving, the values are not available, so another form of search control is needed. Without it, CASCADE wanders down dead ends in the search space and tries to "learn its way out" rather than backtracking as it should.

This result makes intuitive sense and suggests an improvement to CASCADE. Suppose you have been working successfully for a while, get stuck, but see that making an assumption would allow you to continue. For instance, you 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, you might believe you had discovered something about the task domain and form a new rule. On the other hand, if you have been floundering about for some time and are feeling utterly lost, then you are unlikely to react to impasses in the same way. This illustration suggests that CASCADE and other impasse-driven learning systems should keep a running measure of the probable effectiveness of their search so that they can turn off learning if they are floundering. For instance, they might count the average number of applicable rules per goal. If there are many rules left even after search control is applied, then the system is likely to be floundering and should not treat its failures as something to learn from. This learning heuristic could be applied in conjunction with CASCADE's current heuristics. The

revised policy would be that learning occurs on the first local failure after a total failure but only if the probability of floundering is low.

Another improvement to CASCADE would be to have it learn new overly general rules. Since overly general rules are syntactically homomorphic to domain theory rules, simply changing constants to variables and/or dropping antecedent conditions may suffice for creating overly general rules. One approach would be to use Flann and Dietterich (1989) inductive technique for substituting variables for constants.

An earlier discussion suggested replacing the current binary distinction between domain theory rules and rules from outside the domain theory with a continuous representation of belief, such as an estimated probability of correctness. For instance, when an overly general rule is created by syntactic induction, it would be given a very low probability of correctness. As it is used successfully by EBLC, its probability of correctness increases. Taking this idea one step further, the problem solver would try to maximize the correctness of the derivations it produces by preferring rules that have high estimated probabilities of correctness. If the best rule's probability is less than 1.0 and it successfully achieves the goal, then instantiations of the rule are created and assigned estimated probabilities that are somewhat higher than the original rule's estimated probability. Thus, there would be no impasses per se, and EBLC would be folded into the normal operation of the rule interpreter. However, this approach would work only if the system has excellent control knowledge so that it rarely travels down dead ends.

Once again we come back to a theme that has haunted both the current version of CASCADE and the proposed extensions: In order to learn correct domain knowledge via heuristic impasse-driven learning, one must have (or learn) good search control. This point was certainly not obvious to us prior to implementing CASCADE, although it seems rather obvious now. We were particularly surprised to find that a seemingly innocent representational construct, the universal quantifier implicit in PROLOG's set of predicates, thwarted learning of domain knowledge until analogical search control was made mandatory.

Ironically, we made the discovery that search control is necessary for domain theory learning in the same way that CASCADE would have discovered it, if CASCADE could design AI programs instead of solve physics problems. We fixed a bug in CASCADE's design using an overly general design rule: "More knowledge permits more constrained processing, which generates fewer bugs." After resolving this design impasse,

we formed a specialization of the rule: "Acquiring more search control permits more constrained impasse-driven learning, which generates fewer buggy new rules."

Acknowledgements

This research was supported by the Cognitive Sciences Division of the ONR (N00014-88-K-0086) and the Information Sciences Division of the ONR (N00014-86-K-0678). Micki Chi provided us with the data from her group's studies as well as many insightful comments on the research.

References

Ali, K. M. (1989). Augmenting domain theory for explanation-based learning. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 40–42). San Mateo, CA: Morgan Kaufmann.

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

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

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

Austin, J. L. (1962). *How to do things with words.* New York: Oxford University Press.

Bergadano, F., Giordana, A., & Ponsero, S. (1989). Deduction in top-down inductive learning. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 23–25). San Mateo, CA: Morgan Kaufmann.

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

Bundy, A., Byrd, L., Luger, G., Mellish, C., & Palmer, M. (1979). Solving mechanics problems using meta-level inference. *Proceedings of the Sixth International Joint Conference on Artificial Intelligence* (pp. 1017–1027). IJCAI. San Mateo, CA: Morgan Kaufmann (distributor).

Cain, T. (1991). The DUCTOR: A theory revision system for propositional domains. *Proceedings of the Eighth International Workshop on Machine Learning* (pp. 485–489). San Mateo, CA: Morgan Kaufmann.

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.

Cohen, W. W. (1990). Learning from textbook knowledge: A case study. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 743–748). Menlo Park, CA: AAAI Press.

Danyluk, A. P. (1989). Finding new rules for incomplete theories: Explicit biases for induction with contextual information. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 34–36). San Mateo, CA: Morgan Kaufmann.

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

Fawcett, T. E. (1989). Learning from plausible explanations. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 37–39). San Mateo, CA: Morgan Kaufmann.

Flann, N. S., & Dietterich, T. G. (1989). A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187–226.

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

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: John Wiley.

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

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

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

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. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society* (pp. 440–446). Hillsdale, NJ: Lawrence Erlbaum.

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

O'Rorke, P., Morris, S., & Schulenburg, D. (1990). Theory formation by abduction: A case study based on the chemical revolution. In J. Shrager & P. Langley (Eds.), *Computational models of scientific discovery and theory formation.* San Mateo, CA: Morgan Kaufmann.

Ourston, D., & Mooney, R. J. (1990). Changing the rules: A comprehensive approach to theory refinement. *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 815–820). San Mateo, CA: Morgan Kaufmann.

Pazzani, M. J. (1990). *Creating a memory of causal relationships: An integration of empirical and explanation-based learning methods.* Hillsdale, NJ: Lawrence Erlbaum.

Pazzani, M., Dyer, M., & Flowers, M. (1986). The role of prior causal theories in generalization. *Proceedings of the Fifth National Conference on Artificial Intelligence* (pp. 545–550). AAAI. San Mateo, CA: Morgan Kaufmann (distributor).

Pople, H. E., Jr. (1973). On the mechanization of abductive logic. *Proceedings of the Third International Joint Conference on Artificial Intelligence* (pp. 147–152). IJCAI. Los Altos, CA: William Kaufmann (distributor).

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning, 1,* 81–106.

Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine Learning, 3,* 239–266.

Rosenbloom, P. S., & Aasman, J. (1990). Knowledge level and inductive uses of chunking (EBL). *Proceedings of the Eighth National Conference on Artificial Intelligence* (pp. 821–828). Menlo Park, CA: AAAI Press.

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

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

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: Laboratory for Computer Science Research, Rutgers University.

VanLehn, K. (1986). Towards a theory of impasse-driven learning. In H. Mandl & A. Lesgold (Eds.), *Learning issues for intelligent tutoring systems*. New York: Springer.

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

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

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

VanLehn, K., Jones, R. M., & Chi, M. T. H. (1992). A model of the self-explanation effect. *Journal of the Learning Sciences, 2*, 1–60.

Widmar, G. (1989). A tight integration of deductive and inductive learning. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 11–13). San Mateo, CA: Morgan Kaufmann.

Wilkins, D. C. (1988). Knowledge base refinement using apprenticeship learning techniques. *Proceedings of the Seventh National Conference on Artificial Intelligence* (pp. 646–653). AAAI. San Mateo, CA: Morgan Kaufmann (distributor).