Learning Several Lessons from One Experience

Bruce Krulwich, Lawrence Birnbaum, and Gregg Collins
Northwestern University, The Institute for the Learning Sciences
1890 Maple Avenue, Evanston, IL 60201
Telephone: (708) 491-3500
Electronic mail: krulwich@ils.nwu.edu

Abstract

The architecture of an intelligent agent must include components that carry out a wide variety of cognitive tasks, including perception, goal activation, plan generation, plan selection, and execution. In order to make use of opportunities to learn, such a system must be capable of determining which system components should be modified as a result of a new experience, and how lessons that are appropriate for each component's task can be derived from the experience. We describe an approach that uses a self-model as a source of information about each system component. The model is used to determine whether a component should be augmented in response to a new example, and a portion of the model, *component performance specifications*, are used to determine what aspects of an example are relevant to each component and to express the details of the lessons learned in vocabulary that is appropriate to the component. We show how this approach is implemented in the CASTLE system, which learns strategic concepts in the domain of chess.

Cognitive tasks and components

In the course of pursuing its goals, an intelligent agent must notice opportunities, devise plans of action, and select among such plans. In domains that involve interactions with other agents, including such games as chess (in which our system operates), an agent must additionally notice threats posed by the other agents and develop plans to respond to them. It is useful for a variety of reasons to model such an agent as a collection of components, each of which is responsible for one of these planning tasks. In particular, such an approach to modeling the agent is useful in learning [Collins *et al.*, 1991c; Krulwich, 1991]. In this view of a problem-solving agent, learning involves three steps. The first is recognizing situations in which there is a lesson to be learned, such as when the system experiences an expectation failure. The second is determining which component is implicated in the lesson. The third is determining how that component should be modified.

Of course, the tasks in which an agent engages are not completely disjoint: Generating goals requires the agent to reason about its own plans and abilities as well as other agents; plan generation requires the agent to reason about its perceptual and mechanical abilities, as well as reasoning about its future decision-making processes; selecting among plans requires the agent to reason about its ability to execute them. The interrelations between these tasks require that the system's components be correspondingly intertwined. This interconnection of the system's components in turn requires the agent's learning module to be able to reason about interactions between components in formulating new concepts.

Reasoning about such interactions requires that the agent have a degree of *self-knowledge* in order to properly assimilate new knowledge. This paper investigates a *model-based* approach to handling these issues [Collins *et al.*, 1991a]. Our system, named CASTLE,[1] uses an explicit model of its decision-making mechanism to diagnose planning errors [Birnbaum *et al.*, 1990] and to repair its faulty components [Krulwich, 1991; Krulwich, 1992].

An every-day example

Consider the case of a person cooking rice pilaf for the first time. The last step in the directions says to "cover the pot and cook for 25-30 minutes." Suppose the person starts the rice cooking and then goes off to do something else—say, clean up the house. In the interim, the pot boils over. When the person returns to the kitchen a half-hour later, the rice pilaf is ruined.

What should be learned from this sequence of events? This depends on which of the generic decision-making tasks involved in planning the agent chooses to modify. For each task there will be a concept that operationalizes the idea of pots boiling over, in terms that are meaningful in the context of carrying out that task. Figure 1 summarizes the following three ways of operationalizing the problem posed by pots boiling over:

1. Whenever a covered pot containing liquid is on the stove, keep an ear peeled for the sound of the lid bouncing or the sound of the water bubbling.
2. Do not put a covered pot with liquid in it over a high flame, because it will boil over. The flame should be turned down or the pot lid should be left ajar.

[1] CASTLE stands for *C*oncocting *A*bstract *S*trategies *T*hrough *L*earning from *E*xpectation-failures.

System task	What to learn
Perception	Listen for the bubbling sound that warns of the pot boiling over
Planning	Leave the lid ajar or turn down the flame
Plan execution and scheduling	Don't execute any other plans that involve leaving the kitchen

Figure 1: Learned concepts in the *rice pilaf* example

3. When cooking liquid in a covered pot, stay in the kitchen, because it's hard to hear a pot boiling over from the other rooms.

Which of these concepts the agent should learn depends on its perceptual and plan execution abilities, the plans that it typically generates, and the constraints under which it operates. If the agent would in general be able to hear the pot boiling over from the other room, but simply had not attended to the soft sounds that it heard in this instance, then tuning its perceptual attention apparatus when a pot is on the stove is a good way to adapt to the new concept of pots boiling over. This is the first lesson listed above. If the agent would not be able to hear the pot boiling over however hard it listened, another lesson must be learned, either to prevent pots from boiling over by changing the parameters of the cooking process (e.g., by turning down the flame), or to avoid leaving the kitchen when a pot is on the stove. If the recipe will work properly with the flame turned very low, as is the case with rice, or with the pot lid off (which is not usually the case with rice but is with other foods such as spaghetti), then the second lesson in figure 1 will suffice for the agent to plan properly in the future. If the recipe cannot be cooked uncovered or over a low flame, the third lesson is the one that should be learned, that it should not leave the kitchen while a pot is on the stove.

We see, then, that the agent could learn several things in response to the rice pilaf boiling over. The first lesson, that the problem can be averted if the lid can be left ajar or if the flame is lowered, should be learned regardless of the agent's abilities, but should only be applied as appropriate. Which of the other lessons the agent should learn, the idea of staying in the kitchen, or of tuning its perceptual apparatus, depend on the agent's knowledge of its hearing abilities. Other possible lessons, such as the need to compute the ideal height for the flame under the pot, are ruled out due to the inability of the system to perform this computation accurately.

Constraints on concept formulation

Our discussion of learning about pots boiling over while cooking rice pilaf demonstrates several elements of the agent's self-knowledge that affect the formulation of learned concepts:

- *Knowledge of the agent's components:* Different aspects of the example will be relevant to different components in the agent's decision-making architecture (e.g., the agent could learn concepts relating to planning, plan execution/scheduling, and perception).
- *Knowledge of the agent's physical abilities:* Some formulations of the concept will not be effective due to limitations in the agent's physical abilities (e.g., whether to learn to listen harder depends on the physical capability of the agent to hear the bubbling from the other room).
- *Knowledge of the agent's cognitive abilities:* Limitations could also be cognitive (e.g., the agent could attempt to learn to compute the precisely optimal flame height, but limitations in the agent's ability to perform this reasoning make this untenable).
- *Knowledge of typical planning situations:* The possible alternative plans (e.g., lowering the flame and leaving the lid ajar) must be selected based on the situations in which the agent expects to find itself.

Each of these is a type of self-knowledge that an intelligent agent must possess in order to assimilate learned knowledge effectively. This self-knowledge will enable the agent to relate new concepts to relevant components of its decision-making architecture.

The CASTLE system

Our research is an investigation of a *failure-driven* approach to acquiring new planning knowledge [Birnbaum *et al.*, 1990; Collins *et al.*, 1991c]. Our system, CASTLE, detects situations that are contrary to its expectations, and responds to these expectation failures by repairing the faulty planner components which were responsible for the failure. We approach this learning task in a knowledge-intensive fashion, in which the system uses knowledge of its own planning components to assimilate events which led to expectation failures. This knowledge is expressed in the form of a planner self-model, which is used to diagnose and repair expectation failures [Davis, 1984; deKleer and Williams, 1987]. More specifically, the system first examines an explicit *justification structure* that encodes the the reasoning that led to its belief in the incorrect expectation [deKleer *et al.*, 1977; Doyle, 1979]. This justification is used to isolate the components of its architecture that are responsible for the failure [Collins *et al.*, 1991b]. It then uses a *specification* of the faulty components to guide the learning of new rules to embody the concept which must be learned in response to the failure [Krulwich, 1991].

The CASTLE system carries out the tasks we have been discussing in the domain of chess. CASTLE is broken up into a number of components, which reflect a functional decomposition of the decision-making process [Collins *et al.*, 1991]. Each component is dedicated to a particular cognitive task, and is implemented as a set of rules which provide different methods for performing

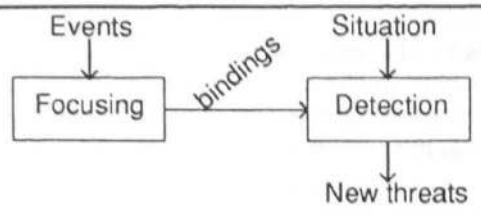

Figure 2: Incremental threat detection

the task. These rules in turn invoke other components as necessary.

We will now examine more specifically some of the knowledge that CASTLE has regarding its components. One cognitive task in which CASTLE engages is that of noticing threats and opportunities as they arise. Rather than recomputing these at each turn, CASTLE maintains a set of active threats and opportunities that is updated over time. To accomplish this incremental threat detection, the system uses a *detection focusing* component, which consists of *focus rules* that specify the areas in which new threats may have been enabled. Then, a separate *threat detection* component, consisting of rules for noticing specific types of threats, detects the threats that have in fact been enabled. The relationship between the two components is shown pictorially in figure 2. A sample focus rule is shown in figure 3, which embodies the system's knowledge that the most recently moved piece, in its new location, may be a source of new threats. Another focus rule, not shown, specifies that the most recently moved piece can also be a target of newly enabled attacks. Using focus rules such as these, the actual threat detector rules will only be invoked on areas of the board that can possibly contain new threats.

Another task in which CASTLE engages is *plan generation*. One of the system's components for doing plan generation is a *schema applier*. This component retrieves schemata, which are generalized sequences of actions, that will achieve a particular goal in the current situation. In CASTLE these schemata are called *offensive strategies*, and the system's offensive strategy component consists of rules that encode the actions in a schema, its conditions of applicability, and the goal which it satisfies. A strategy rule for the classic chess

```
(def-brule focus-new-source
 (focus focus-moved-piece ?player
   (move ?player ?move-type ?piece ?loc1 ?loc2)
   (world-at-time ?time))
<=
 (move-to-make (move ?player ?prev-move-type
                     ?piece ?old-loc ?loc1)
               ?player ?goal (1- ?time)) )
```

Figure 3: Focusing on new moves by a moved piece

```
(def-brule strategy-fork-sample
  (strategy fork ?player (world-at-time ?time)
            (goal (capture ?target2))
    (plan (move ?player non-capture ?piece
                ?loc1 ?loc2 ?time)
      (next (move ?player (capture ?target2)
                  ?piece ?loc2 ?loc4 (1+ ?time))))
<=
 (and (at-loc ?player ?piece ?loc1 ?time)
      (at-loc ?opponent ?target1 ?loc3 ?time)
      (at-loc ?opponent ?target2 ?loc4 ?time)
      (not (at-loc ?anyone ?any-piece ?loc2 ?time))
      (move-legal ?player ?piece ?loc1 ?loc2)
      (move-legal ?player ?piece ?loc2 ?loc3)
      (move-legal ?player ?piece ?loc2 ?loc4)
      (> (value ?target1) (value ?target2))
    (no (and (counterplan ?cp-meth1 ?opponent
              (goal-capture ?target ?loc3
                (move ?player (capture ?target1)
                      ?piece ?loc2 ?loc3))
              ?time ?counterplan)
             (counterplan ?cp-meth2 ?opponent
              (goal-capture ?target ?loc4
                (move ?player (capture ?target2)
                      ?piece ?loc2 ?loc4))
              ?time ?counterplan))) ))
```

Figure 4: A strategy rule: The *fork*

strategy the *fork* is shown in figure 4.[2] This rule says roughly that *one way to capture an opponent piece is to find a piece that can move to a location from which it can capture two opponent pieces, if the opponent will have no one counterplan against both the attacks.* Other such strategies are *pin* and the *sactifice*. Issues in acquiring such strategies have been discussed previously [Birnbaum *et al.*, 1990; Freed, 1991].

Learning focusing and a new strategy

Consider the partial chess situations shown in figure 5. Initially CASTLE (playing black) uses its offensive strategy component for plan recognition [Schank and Abelson, 1975; Cullingford, 1978], that is, to see if the opponent can be expected to have any good strategies to apply. Since none of its strategy rules apply from the perspective of the opponent, CASTLE assumes that the opponent will not be able to make a move that will enable a guaranteed capture on the following turn. In other words, the opponent would presumably like to make a situation in which no matter what CASTLE does, the opponent will capture a piece on the next turn. CASTLE believes that since none of its strategy rules apply for the opponent, the opponent will not be able to create such a situation.

[2] In practice this rule is specialized to use geometric reasoning. Alternatively, it could evaluate the results of a projection engine without duplicating its computation.

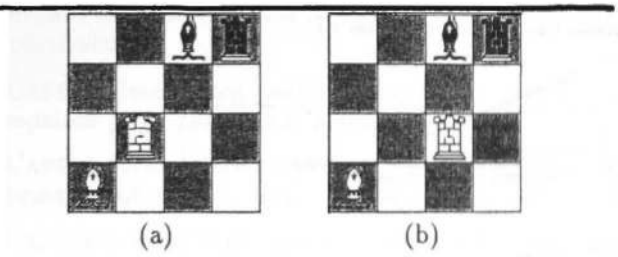

Figure 5: Example: Opponent (white) to move

The opponent (playing white) then moves its rook, resulting in the situation in figure 5(b). By making this move, the opponent has enabled two attacks simultaneously, the bishop attack on the computer's rook, and the rook attack on the computer's bishop, and one of the attacks is sure to succeed. CASTLE's lack of an offensive strategy rule for this type of *simultaneous attacks* resulted in its being unable to counterplan early enough. CASTLE should learn a new strategy rule as a result of the loss.

The planning failure in our example is also relevant to another of CASTLE's components, namely the detection focusing component. Initially CASTLE is only equipped with the two focus rules discussed earlier, for the new threats by and against the most-recently moved piece, and does not have a rule for focusing on the *discovered attack* that was enabled in our example by moving the rook out of the line of attack between the bishop and the computer's rook [Collins *et al.*, 1991c]. Because of this, CASTLE is at first unable to detect the threat against its rook, and believes that the threat against its bishop is the only threat on the board. Because of this error, in figure 5(b) it moves its bishop, and thinks that all of its pieces are safe as a consequence. When the opponent executes the capture of CASTLE's rook, the computer realizes the extent of its error.

These two concepts, *simultaneous attacks* and *discovered attacks*, should both be learned from the sequence of events that we have seen. Each concept involves a different aspect of the situation, and each must be characterized in a way that can be effectively used by the relevant components of the agent's architecture.

Component	Learned concept
Planning	Make a successful attack by moving a piece off a line of attack to a new location which can also make a second attack
Perception	Threats can be enabled by moving a piece off of a line of attack that is otherwise open

Figure 6: Concepts in the chess example

Developing these characterizations requires the system to use knowledge of the functions and interactions of its components. A characterization of simultaneous attacks that can be used by the offensive strategy component must mention the simultaneous enablement of the two attacks, one through the vacated square and one from the new location of the moved piece, and must encode the fact that the opponent must be unable to react to both attacks in a single move. Additionally, it must be predictive, because it will be invoked before any move has been made, and so it must refer not to moves that have already been made, but rather to moves that can potentially be made. A characterization of discovered attacks that can be used effectively to focus the detection rules must generate a set of constraints describing all the possible moves through a vacated square, without referring to the legal moves themselves which will be checked subsequently by the detection component (see figure 2). This rule must be expressed in terms that can be applied after the enabling move has been made, but before the discovered attack is made.

CASTLE detects the opportunity to learn by observing an expectation failure when its rook is captured by the opponent's bishop. As we have discussed above, CASTLE uses a model-based reasoning approach to diagnosing expectation failures, in which the system diagnoses the failure by examining an explicit justification structure which encodes the basis for its belief in its expectation that its pieces were safe. The system's diagnosis engine traverses this justification, which is shown in figure 7, to find the underlying beliefs of the system that were responsible for the failure. In our example there were two such incorrect assumptions: that *the system's set of threat detection focusing rules is complete*, and *the system's set of strategy schemata is complete*. The task now at hand is for the system to repair its planning

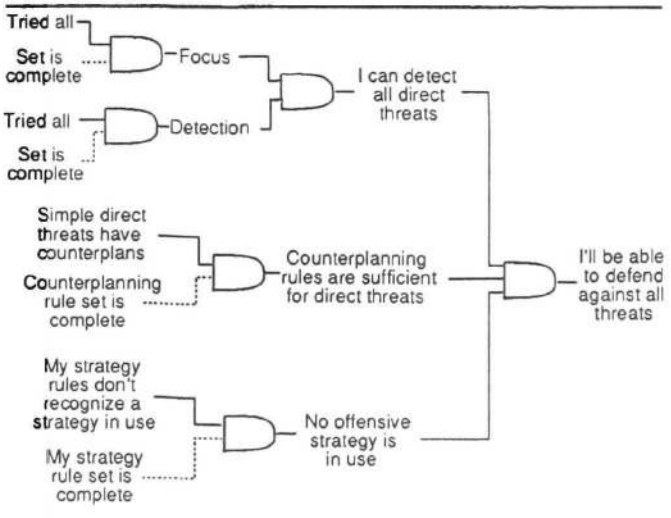

Figure 7: Justification for the failed expectation

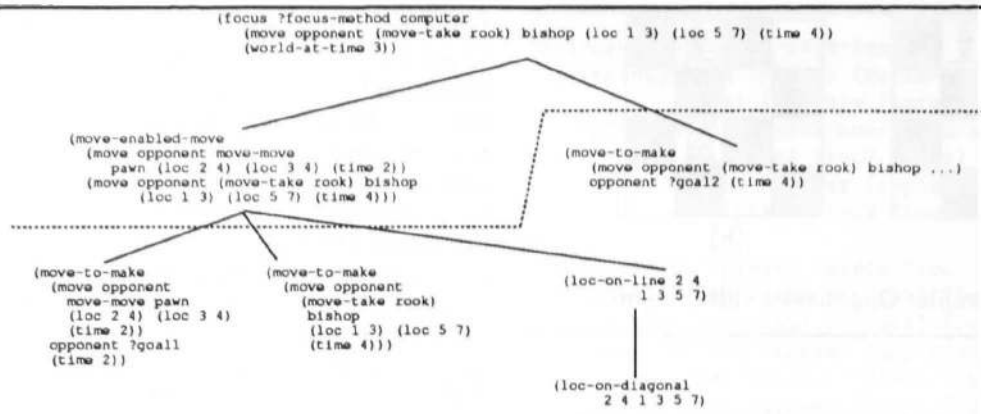

Figure 8: Explanation of desired detection focusing performance

mechanism. Each of the two faults can be repaired by augmenting a corresponding rule set, one for the focusing component and one for the strategy component.

To construct the new rules, CASTLE retrieves a *component performance specification* for each component [Krulwich, 1991]. These performance specifications, a form of planner self-knowledge, describe the correct behavior of each component. These specifications can be used to recognize correct behavior that was not produced by the component's rule sets. The specification of the detection focusing component says roughly that *the focusing component will generate bindings that include any capture that is enabled by a given move.* This specification enables CASTLE to focus on the details of the example that are relevant to the component being repaired, by serving as an explanation-based learning target concept. After retrieving the specification, CASTLE invokes its deductive inference engine to construct an explanation of why the possible capture of the rook should have been in the set of constraints generated by the focusing component. This explanation, shown in figure 8, says roughly that the opponent's move should have been generated by the focusing component, *because* the opponent's previous move enabled the attack, *because* it was on a square between the bishop and the rook, *and* there were no other pieces along the line of attack, *and* emptying the line of attack is an enabling condition for the capture to be made. CASTLE then uses explanation-based learning [Mitchell *et al.*, 1986; DeJong and Mooney, 1986] to generalize this explanation and to construct the new detection focusing rule shown in figure 9.

The same mechanism is used to construct the new offensive strategy rule. A specification of the offensive strategy component is retrieved, which says roughly *the offensive strategy component will generate any plans which are sure to result in a successful attack.* CASTLE then explains why the opponent's move resulted in a certain capture. This explanation says roughly that the opponent's move was a good offensive strategy, *because* it enabled one attack through the vacated square, *and* it enabled a second attack from the new location of the moved piece, *and* there was no counterplan for the opponent that could disable both attacks. The crucial inference in constructing this explanation is that the existence of two attacks, in a situation where they cannot both be counterplanned against simultaneously, means that one of them will necessarily succeed. One approach is for this knowledge to be built in, as has been done implicitly by others. Our approach is for this knowledge to be inferred from more primitive axioms of plan execution [Birnbaum *et al.*, 1990]. After this explanation is constructed it is generalized to form a rule for simultaneous attacks.

Discussion

In section we saw several types of reasoning in which an agent might engage in the course of learning from a sequence of events. In our example in section we saw

```
(def-brule learned-focus-method25
  (focus learned-focus-method25 ?player
     (move ?player (capture ?taken-piece)
        ?taking-piece (loc  ?row1 ?col1)
        (loc  ?row2 ?col2))
     (world-at-time ?time2))
<=
  (and (move-to-make
          (move ?other-player move ?interm-piece
                (loc ?r-interm ?c-interm)
                (loc ?r-other ?c-other))
          ?player ?goal ?time1)
       (loc-on-line ?r-interm ?c-interm
                    ?row1 ?col1 ?row2 ?col2)
       (at-loc ?player ?taking-piece
               (loc ?row1 ?col1)
               (- gen-time2.24 2)) ))
```

Figure 9: Learned focus rule for *discovered attacks*

how CASTLE performs some of these types of reasoning, in particular:

- CASTLE determined which components should be repaired *(e.g., perception, planning)*
- CASTLE formulated a concept for each component being repaired *(e.g., focus rule, strategy schema)*

CASTLE is capable of carrying out several other types of reasoning about the state of its knowledge. Consider, for example, a situation in which the system applied its *simultaneous attacks* strategy against an opponent, and the opponent is able to counterplan against both attacks in a way that CASTLE does not know about. One such counterplanning method might, for example, be to move one of the attacked pieces to a square along the line of attack against the second piece. Since CASTLE does not initially have a rule for counterplanning by interposing pieces, it would think that its simultaneous attack would be successful. When the strategy is seen to be unsuccessful, CASTLE must determine whether the fault lies with the strategy rule or with another component. In this case CASTLE should realize that the strategy is sound, but that it needs to learn a new counterplanning rule for interposition.

This learning process requires that CASTLE have a *self-model* which describes the functions and interactions of its components. This self-model consists of several forms of self-knowledge. *Belief justifications* relate expectations and other beliefs to the reasons that CASTLE believes them to be true. *Implicit assumptions* describe the assumptions that CASTLE is making (such as rule set completeness) that underly the validity of its decision-making mechanisms. *Performance specifications* describe the performance that CASTLE expects from each of its components. Using these forms of self-knowledge, CASTLE can reason about the state of its knowledge and abilities in order to effectively assimilate new knowledge.

Our research has involved extending our model of planning and decision-making to include a variety of tasks and components. To date we have developed models of threat detection, counterplanning, schema application, goal regression, lookahead search, and execution scheduling. Future research will determine the degree to which our theory of model-based knowledge assimilation through diagnosis and learning applies to other decision-making tasks and to planning in other domains.

Acknowledgements: Thanks go to Matt Brand, Michael Freed, Menachem Jona, Eric Jones, and Louise Pryor for discussions on this paper and on the research presented. This work was supported in part by the Air Force Office of Scientific Research under grant number AFOSR-91-0341-DEF, and by the Defense Advanced Research Projects Agency, monitored by the Office of Naval Research under contract N-00014-91-J-4092. The Institute for the Learning Sciences was established in 1989 with the support of Andersen Consulting, part of The Arthur Andersen Worldwide Organization. The Institute receives additional support from Ameritech, an Institute Partner, and from IBM.

References

[Birnbaum *et al.*, 1990] L. Birnbaum, G. Collins, M. Freed, and B. Krulwich. Model-based diagnosis of planning failures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 318–323, Boston, MA, 1990.

[Collins *et al.*, 1991a] G. Collins, L. Birnbaum, B. Krulwich, and M. Freed. A model-based approach to learning from planning failures. In *Notes of the AAAI Workshop on Model-Based Reasoning*, Anaheim, CA, 1991.

[Collins *et al.*, 1991b] G. Collins, L. Birnbaum, B. Krulwich, and M. Freed. Model-based integration of planning and learning. *SIGART Bulletin*, 2(4):56–60, 1991. Originally in Working Notes of the AAAI Spring Symposium on Integrated Intelligent Architectures.

[Collins *et al.*, 1991c] G. Collins, L. Birnbaum, B. Krulwich, and M. Freed. Plan debugging in an intentional system. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 353–358, Sydney, Australia, 1991.

[Cullingford, 1978] R. Cullingford. *Script Application: Computer Understanding of Newspaper Stories.* PhD thesis, Yale University, 1978. Technical Report 116.

[Davis, 1984] R. Davis. Diagnostic reasoning based on structure and function: Paths of interaction and the locality principle. *Artificial Intelligence*, 24(1-3):347–410, 1984.

[DeJong and Mooney, 1986] G. DeJong and R. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1:145–176, January 1986.

[deKleer and Williams, 1987] J. deKleer and B.C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–129, April 1987.

[deKleer *et al.*, 1977] J. deKleer, J. Doyle, G.L. Steele, and G.J. Sussman. Explicit control of reasoning. *SIGPLAN Notices*, 12(8), 1977.

[Doyle, 1979] J. Doyle. A truth maintenance system. *Artificial Intelligence*, 12(3):231–272, 1979.

[Freed, 1991] M. Freed. Learning strategic concepts from experience: A seven-stage process. In *Proceedings of the Thirteenth Annual Conference of The Cognitive Science Society*, pages 132–136, Chicago, IL, 1991.

[Krulwich, 1991] B. Krulwich. Determining what to learn in a multi-component planning system. In *Proceedings of the Thirteenth Annual Conference of The Cognitive Science Society*, pages 102–107, Chicago, IL, 1991.

[Krulwich, 1992] B. Krulwich. *Learning New Methods for Multiple Cognitive Tasks.* PhD thesis, Northwestern University, Institute for the Learning Sciences, 1992. (in preparation).

[Mitchell *et al.*, 1986] T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1), January 1986.

[Schank and Abelson, 1975] R. C. Schank and R. P. Abelson. *Scripts, Plans, Goals, and Understanding.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1975.