Determining What to Learn in a Multi-Component Planning System*

Bruce Krulwich
Northwestern University
The Institute for the Learning Sciences
1890 Maple Avenue
Evanston, Illinois 60201

Abstract

An intelligent agent which is involved in a variety of cognitive tasks must be able to learn new methods for performing each of them. We discuss how this can be achieved by a system composed of sets of rules for each task. To learn a new rule, the system first isolates the rule set which should be augmented, and then invokes an explanation-based learning mechanism to construct the new rule. This raises the question of how appropriate *target concepts* for explanation can be determined for each task. We discuss the solution to this problem employed in the CASTLE system, which retrieves target concepts in the form of performance specifications of its components, and demonstrate the system learning rules for several different tasks using this uniform mechanism.

Cognitive tasks and components

In the course of determining its behavior, an intelligent agent must activate goals, notice opportunities, and devise and select among plans of action that will accomplish its goals. In competitive planning domains, including such games as chess (in which our system operates), an agent must additionally notice threats posed by other agents and develop plans to respond to them. Such an agent should be able to learn new methods for accomplishing each of these cognitive tasks.

The CASTLE system[1] engages in these tasks while playing chess [Birnbaum *et al.*, 1990; Collins *et al.*, 1991]. In the course of its decision-making, CASTLE generates explicit *expectations* about future actions and abilities, and learns from failures of these expectations. A variety of decision-making methods are represented explicitly, and an explicitly represented model of these procedures is used to diagnose expectation failures.

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 the task. This decomposition of the agent into distinct components enables a uniform learning process to learn methods for any cognitive task in which the system is engaged.

For example, one such component has the task of noticing threats and opportunities as they become available. Rather than recomputing these at each turn, CASTLE maintains a set of active threats and opportunities which 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 been enabled. A sample focus rule is shown in figure 1, 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 move 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 which can possibly contain new threats.

A second system component has the task of generating responses to enemy threats. This *counterplanning*

```
(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 1: Focusing on new moves by a moved piece

*This work was supported in part by the Office of Naval Research under contract N00014-89-J-3217, and by the Defense Advanced Research Projects Agency, monitored by the Air Force Office of Scientific Research under contract F49620-88-C-0058. 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.

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

```
(def-brule counterplan-1
  (counterplan ?player
     (goal-capture ?piece ?loc
        (move opponent (capture ?piece)
                       ?opp-piece ?opp-loc ?loc))
     ?time ?the-response)
<=
 (and (move-legal
        (move ?player move ?piece ?loc ?new-loc))
   (no (and (at-loc ?opponent ?other-opp-piece
                    ?other-opp-loc ?time)
            (move-legal
             (move ?opponent (capture ?piece)
                   ?other-opp-piece ?other-opp-loc
                   ?new-loc))))
   (is-seq ?the-response
       (move ?player move ?piece ?loc ?new-loc))))
```

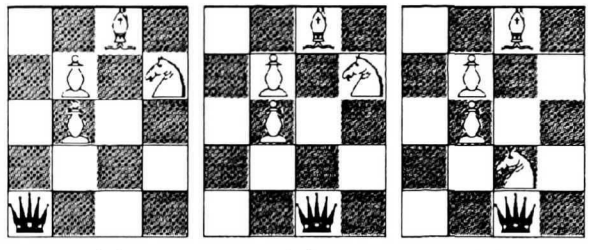
Figure 2: Reacting to a threat by running away

component is invoked whenever an enemy threat is noticed, and generates moves that the system can make in response. It is additionally invoked whenever the system has to reason about responses to an action, as we will see later. One such counterplanning rule is shown in figure 2, which says that a possible response to an enemy threat is to move the threatened piece to a safe location.

These two components, and others like them, span the spectrum of cognitive tasks, including monitoring the environment, reacting to external actions, planning computer actions, and selecting among possible actions. Each component consists of a set of rules which collectively achieve the desired task. For each of these components, learning new methods for achieving their tasks is accomplished by constructing new rules for their rule sets.

Counterplanning by interposition

Consider one way in which CASTLE can learn a new rule for the *counterplanning* component. In the chess situation shown in figure 3(a), the computer has two options — it can capture the opponent's knight or it can construct a plan to take the opponent's bishop. One plan to capture the opponent's bishop would be to move the queen two squares to the right and capture the bishop on the subsequent turn. In deciding whether this is a viable plan, the computer considers whether the opponent would have a response to its move that would disable the attack before it could be carried out. In other words, the computer checks whether there is a possible counterplan for the opponent [Krulwich, 1991].



(a) (b) (c)

Figure 3: *Interposition:* Computer (black) to move

It is clear in figure 3(b) that the opponent would in fact have a possible counterplan, namely moving its knight to block the attack. Suppose, however, that the system had an incomplete collection of counterplanning rules that included rules for moving a piece to a safe location (the rule shown in figure 2) and capturing the attacking piece, but did not include a rule for interposing a piece to block a threat. Under these circumstances, when considering the viability of its plan, the system would conclude that the opponent will not have a response to the threat. Given this assumption, the system would consider the plan to take the bishop foolproof, and would proceed to carry it out.

Of course, this plan will fail, because the opponent will block the attack by interposing its knight, as we see in figure 3(c). Because the plan failure came about due to the system's lack of counterplanning knowledge, our system should respond to the failure by constructing a rule for counterplanning against an attack by interposing a piece.

Learning from the component failure

To learn from this plan failure, the system must first determine which component is at fault. The system's diagnosis engine has this task of identifying the faulty component. CASTLE accomplishes this using a *model-based reasoning* approach [Davis, 1984; deKleer and Williams, 1987; Simmons, 1988]. Currently, our model is limited to the case in which only one component is at fault. Failures of more than one component could be handled by repairing the two components independently, if the failure reflects separable faults in the two components. Alternatively, the higher-level component which made use of the two faulty components may need to be repaired, to account for unanticipated interactions between the two components. In this paper we are limiting our discussion to the case of single-component faults.

CASTLE performs its diagnosis using explicit *justification structures* [deKleer *et al.*, 1977; Doyle, 1979], which record how the planner's expectations are inferred from the rules that constitute its decision-making mechanisms, in conjunction with the policies and underlying assumptions which it has adopted. Diagnosing the failure then involves "backing up" through the justification structures, recursively explaining the failure in terms of faulty rule antecedents [Smith, Winston, Mitchell, and Buchanan, 1985; Simmons, 1988; Birnbaum, Collins,

and Krulwich, 1989]. In the cases we are considering, this procedure will "bottom out" by faulting a completeness assumption for a component. In other words, many of the underlying assumptions of the system will refer to the completeness of its component rule sets, and faulting one of these assumptions is a signal that a rule set must be augmented. There are, of course, other types of assumptions that could be faulted. CASTLE examines the form of the faulted assumption to determine which type of repair is necessary. In our example, the diagnosis will indicate that the failure was due to the lack of a counterplanning rule to predict the opponent's response. The structure of the faulted assumption will direct CASTLE to repair the failure by constructing a new counterplanning rule.

The task now at hand is for the system to analyze the failure and encapsulate the relevant information into a new rule for counterplanning by interposition. One common approach to learning from failures is to equip the system with *critics*, which map directly from failure descriptions to repairs [Sussman, 1975]. CASTLE's more general approach is to use explanation-based learning ("EBL") [Mitchell, Keller, and Kedar-Cabelli, 1986; DeJong and Mooney, 1986]. In its general form, EBL constructs new rules for category membership by explaining why a particular instance is a member of a category, and then determining the most general preconditions under which the explanation will still be correct. To accomplish this, EBL is given a *target concept*, which is a description of the category being learned. As has been widely discussed, target concepts must relate to the purpose to which the acquired category membership rules will be put. In previous work, however, this observation has been applied in a framework in which the purpose of the rules to be learned has been fixed, such as recognizing objects for use in plans [Kedar-Cabelli, 1987] or speeding up a search process [Keller, 1987; Minton, 1988]. We would like to extend the notion of EBL target concept formulation for an intelligent agent which is involved in many cognitive tasks.

Constructing a counterplanning rule

Once the system has determined that the plan failure is due to its set of counterplanning rules being incomplete, it knows that it needs to construct a new rule to perform counterplanning, i.e., a rule whose consequent matches the generic form of counterplanner queries. The antecedent of this new rule should encode the type of counterplan that the opponent used in the situation that caused the failure. More specifically, the antecedent should be a generalized description of the counterplan (i.e., the move) that the opponent used. To construct this rule, CASTLE uses explanation-based learning to generalize a description of the move that the opponent made. The EBL target concept should thus be an expression which implies the assertion that the counterplanner should have generated.

CASTLE has expressions which fit these requirements, in the form of *component performance specifications*. These specifications are part of a larger planner self-model [Collins *et al.*, 1991], which is used for planning, expectation formulation, failure diagnosis, and, as described here, fault repair. In general, performance specifications describe the correct behavior of each component, in a way that can be recognized after the fact in a situation in which the component should have produced a particular behavior.[2] In other words, when the system has reason to believe that a component should have been executed in a given situation, based on information from the diagnosis module, this specification will enable the system to conclude that the component should have behaved in a particular way, and to construct an explanation of why it should have done so.[3] An explanation (proof) of the correctness of the specification as applied to a particular situation can then be generalized using EBL, and the new rule can be constructed.

For example, the system's performance specification for the *counterplanning* component says, simply, that a correct counterplan to an attack is a one-move sequence consisting of a move that disabled the attack. Since this specification is only used to explain why a particular move is a correct counterplan, MOVE-DISABLED-MOVE need not be implemented in a way which can be evaluated before the fact. In other words, even without a specific rule for disabling an attack by interposing a piece, the system has the ability to observe the game board sequence after the fact and use its domain physics knowledge to realize how the attack was blocked.

In our example from figure 3, after the opponent has responded to the computer's attack by interposing its knight, the diagnosis module determines that the counterplanning component must be augmented. To construct a new counterplanning rule, the system uses its counterplanner specification as a target concept, and invokes a deductive explanation mechanism to construct an explanation of the opponent's counterplan. This explanation, shown in figure 4, says roughly that *the opponent counterplanned against the queen-bishop attack*, because *the attack was disabled by the opponent's move*, because *the line of attack was no*

[2] CASTLE's component performance specifications serve the purpose of PRODIGY's *target concept specifications* [Minton, 1988], which specify EBL target concepts for each type of search-control rule. In PRODIGY terms, CASTLE is (a) generalizing from search-control rules to functional agent components, (b) embedding the specifications in a general planner self-model, and (c) replacing *example recognizers* with expectation-failure diagnosis.

[3] CASTLE's component performance specifications, along with the desired component result which is provided by the diagnosis module, can be viewed as a "performance improvement objective" as discussed in Chapter 5 of [Keller, 1987] and in [Keller, 1989]. The leverage gained from being *failure-driven* enables CASTLE to generate these performance objectives automatically as a consequence of failure diagnosis.

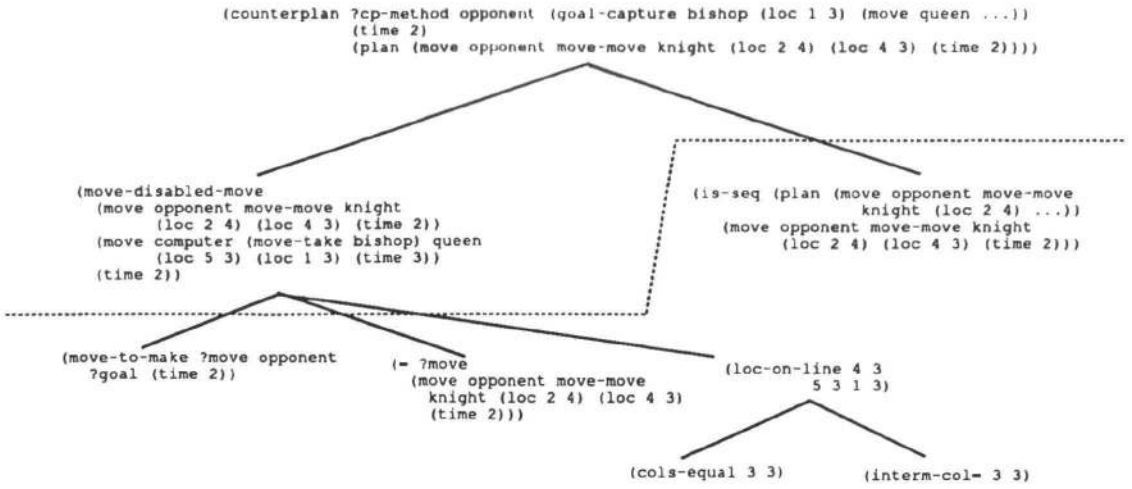


Figure 4: Explanation of desired counterplanner performance

longer clear, because *the opponent's knight was moved into the line of attack.*

Explanation-based learning is then invoked, which generalizes the details of the explanation which are not relevant to its correctness. For example, it is irrelevant what type of piece is interposed, because all links in the explanation which relate to the interposed piece will remain valid for any type of piece. The system then collects the most general operational expressions in the proof tree (operational assertions are shown below the dotted line). These expressions are then "backed up" in time so as to be predictive instead of explanatory. An example of a leaf which must be backed up is the MOVE-TO-MAKE assertion about the opponent's actual move, which took place after the counterplanner was invoked. CASTLE accomplishes this transformation using a set of rules for backing-up propositions in time.[4]

These generalized and adjusted proof-tree leaves will then form the antecedent of a new rule. Since the leaves imply the target concept, and the target concept implies the correct behavior of the counterplanner, the leaves are sure to specify a correct counterplanning method. The new counterplanning rule which the system constructs says, roughly,

TO COMPUTE: A reactive counterplan to a capture

DETERMINE: A location on the line of attack
that another piece can move to

When this rule is added to the set of counterplanning rules, the system will correctly predict the opponent's response in a situation where interposition is viable, such as the situation of figure 3(b). Furthermore, because the same set of rules are used for actual counterplanning against the opponent's threats, the system will be able to respond to threats in a new way [Krulwich, 1991].

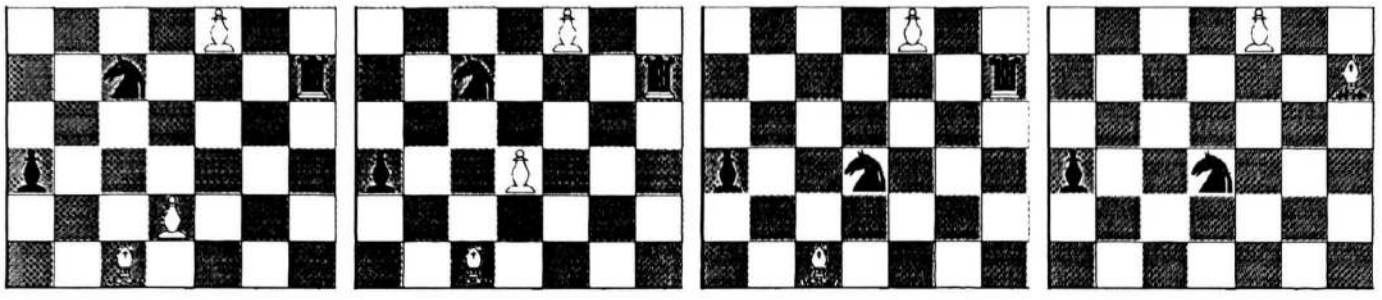
Tangentially, it should be noted that the explanation from which the new rule was generated will form a *justification* for the rule's correctness [Mitchell, Keller, and Kedar-Cabelli, 1986]. This justification structure could then be used for maintaining correctness when underlying assumptions change [deKleer *et al.*, 1977; Doyle, 1979; McDermott, 1989]. More importantly, CASTLE can use these justifications for constructed rules to facilitate further failure diagnosis. This will enable the system to further refine its learned rules, in the event that a constructed rule fails in the future.

Learning threat detection

The main point of our model of target concept retrieval is, of course, that it enables a system to learn new rules for a variety of cognitive tasks. Consider, for example, how CASTLE can learn a new rule for the *detection focusing* component which we discussed earlier. As we saw, this component determines the areas of the board in which threats may have been recently enabled.

Suppose CASTLE had an incomplete set of detection focusing rules, in particular that it knew about threats being enabled either to or from the new location of a piece that has just been moved (as we discussed earlier), but did not know that threats could be enabled through *discovered attacks*, namely, by moving a third piece out of the line of attack. If this were the case, there could be threats enabled through discovered attacks which CASTLE would not be aware of. In the situation shown in figure 5(a), the opponent advances its pawn and thereby enables an attack by its bishop on the computer's rook. When the system updates its set of active threats and opportunities, its threat focusing

[4] These rules are merely an approximation of the ideal behavior: A system should examine the *implicit* assumptions that underly an assertion's being true (e.g., that the legality of a move underlies its being made), and select from them the appropriate predictive tests.



(a) (b) (c) (d)

Figure 5: *Discovered attacks* example: Opponent (white) to move

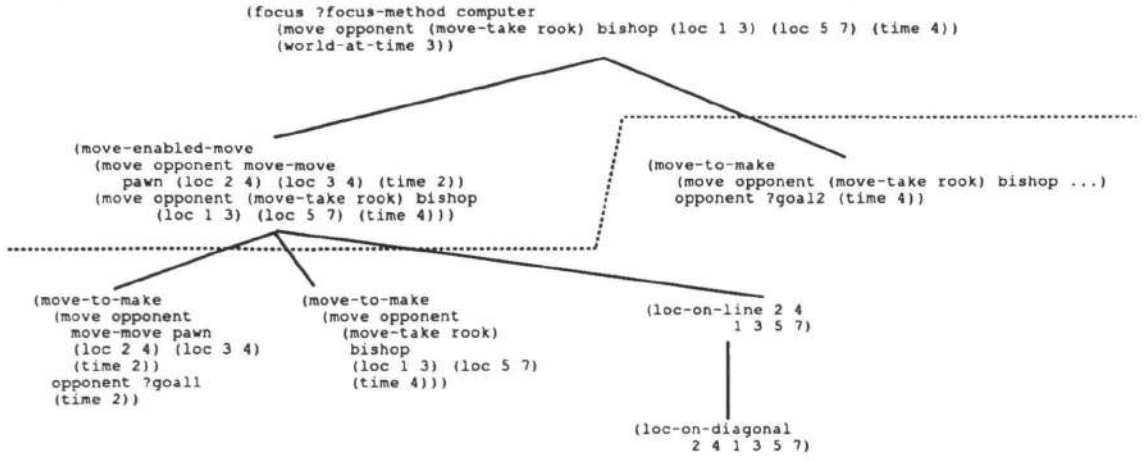


Figure 6: Explanation of desired detection focusing

rules will enable it to detect its own ability to attack the opponent's pawn, but it will not detect the threat to its rook. Because of this, when faced with the situation in figure 5(b), the computer will capture the opponent's pawn instead of rescuing its own rook, and it will expect that the opponent's response will be to execute the attack which it believes to the only one available, namely to capture the computer's pawn. Then, in the situation shown in figure 5(c), when the opponent captures the computer's rook, the system has the task of diagnosing and learning from its failure to detect the threat which the opponent executed.

As in the *interposition* example discussed earlier, the system invokes its diagnosis engine to determine the component which is at fault. In this case it will conclude that the failure resulted from having an incomplete set of detection focusing rules, and it will try to learn from the failure by constructing a new one. To accomplish this, the system retrieves a specification of the detection focusing component, which says that focus rules will indicate any moves which have been enabled. To learn from the failure, CASTLE uses this specification as a target concept to construct an explanation of what the focus rules should have computed. This explanation, which is shown in figure 6, says roughly that *the focus rules should have focussed on the bishop-rook attack,* because *a new threat was enabled,* because *the line of attack was opened up,* because *the opponent's pawn was moved out of the line of attack.*

As in the previous example, explanation-based learning is used to generalize this explanation and collect the operational leaves of the proof tree. CASTLE then constructs a rule from the proof tree leaves which will correctly focus on discovered attacks. This rule says roughly:

TO COMPUTE: Moves that may have been enabled

DETERMINE: Moves through the square
vacated by the most recent move

While the two rules that we have seen learned are very similar in terms of their structure and the computation which they perform, they fulfill very different purposes in the CASTLE system, and more generally they relate to very different cognitive tasks. Using component performance specifications, CASTLE can learn rules to focus perceptual detection by means of the same mechanism that it uses to learn counterplanning rules.

Conclusions

We have given a framework for determining appropriate explanation-based learning target concepts from explicit performance specifications of agent components. This framework enables explanation-based learning to be applied to the spectrum of cognitive tasks which are performed by intelligent agents. We have demonstrated this within the CASTLE system, which learns new rules for such tasks as counterplanning and threat detection focusing.

Applying this technique to learning rules for other cognitive tasks involves extending CASTLE's model in two ways. First, the explicit decision-making model must be extended to include the cognitive task in question. Secondly, a performance specification must be given for any new components. We are in the process of extending the model to include components for goal-regression planning, execution scheduling, and case-based planning. When these and other tasks are expressed in CASTLE's decision-making vocabulary, and performance specifications are given for them, CASTLE will be able to learn new methods for them in response to failures. Future work will determine how well the technique applies to other cognitive tasks, as well as to other types of intelligent agents.

Acknowledgements: The ideas presented here were developed in collaboration with Larry Birnbaum, Gregg Collins, and Mike Freed. Additional thanks go to Matt Brand, Eric Jones, and Louise Pryor for comments on this paper and many discussions on the research presented.

References

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

Birnbaum, L., Collins, G., and Krulwich, B. 1989. Issues in the justification-based diagnosis of planning failures. *Proceedings of the Sixth International Workshop on Machine Learning,* Ithaca, NY, pp. 194-196.

Collins, G., Birnbaum, L., and Krulwich, B. 1989. An adaptive model of decision-making in planning. *Proceedings of the Eleventh IJCAI,* Detroit, MI, pp. 511-516.

Collins, G., Birnbaum, L., Krulwich, B., and Freed, M. 1991. Plan debugging in an intentional system. To appear in *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence,* Sydney, Australia.

Davis, R. 1984. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence,* Vol. 24, pp. 347-410.

DeJong, G., and Mooney, R. 1986. Explanation-based learning: An alternative view. *Machine Learning,* vol. 1, pp. 145-176.

deKleer, J., Doyle, J., Steele, G., and Sussman, G. 1977. AMORD: Explicit control of reasoning. *Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages,* Rochester, NY, pp. 116-125.

deKleer, J., and Willams, B.C. 1987. Diagnosing multiple faults. *Artificial Intelligence* **32**, pp. 97-130.

Doyle, J. 1979. A truth maintenance system. *Artificial Intelligence,* vol. 12, pp. 231-272.

Kedar-Cabelli, S. 1987. Formulating concepts according to purpose. *Proceedings of the 1987 AAAI Conference,* Seattle, WA, pp. 477-481.

Keller, R. 1987. *The role of explicit contextual knowledge in learning concepts to improve performance.* Technical report ML-TR-7, Rutgers University Department of Computer Science, New Brunswick, NJ.

Keller, R. 1989. Compiling learning vocabulary from a performance system description. *Proceedings of the Sixth International Workshop on Machine Learning,* Ithaca, NY, pp. 492-495

Krulwich, B. 1991. Learning from deliberated reactivity. To appear in *Proceedings of the Eighth International Workshop on Machine Learning,* Evanston, IL.

Krulwich, B., Collins, G., and Birnbaum, L. 1989. Improving decision-making on the basis of experience. *Proceedings of the Sixth International Workshop on Machine Learning,* Ithaca, NY, pp. 55-57.

Krulwich, B., Collins, G., and Birnbaum, L. 1990. Cross-Domain Transfer of Planning Strategies: Alternative Approaches. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society,* Cambridge, MA, pp. 954-961.

McDermott, D. 1989. *A general framework for reason maintenance.* Technical report YALEU/CSD/RR-691, Yale University Department of Computer Science, New Haven, CT.

Minton, S. 1988. *Learning Effective Search Control Knowledge: An Explanation-Based Approach.* Technical report CMU-CS-88-133, Carnegie Mellon University School of Computer Science, Pittsburgh, PA.

Mitchell, T., Keller, R., and Kedar-Cabelli, S. 1986. Explanation-based generalization: A unifying view. *Machine Learning,* vol. 1, pp. 47-80.

Simmons, R. 1988. A theory of debugging plans and interpretations. *Proceedings of the 1988 AAAI Conference,* St. Paul, MN, pp. 94-99.

Smith, R., Winston, H., Mitchell, T., and Buchanan, B. 1985. Representation and use of explicit justifications for knowledge base refinement. *Proceedings of the Ninth IJCAI,* Los Angeles, CA, pp. 673-680.

Sussman, G. 1975. *A Computer Model of Skill Acquisition.* American Elsevier, New York.