Plan Debugging in an Intentional System

Gregg Collins, Lawrence Birnbaum, Bruce Krulwich, and Michael Freed

Northwestern University
The Institute for the Learning Sciences
Evanston, Illinois USA

Abstract

We have developed a model-based approach to learning from plan failures in which an agent uses a model of itself to determine where in its planning or execution the cause of a failure lies. We believe that such an approach constitutes the most promising basis for developing learning models that are capable of deciding for themselves what needs to be learned from a given experience. In addition, such methods appear capable of learning about planning at a very abstract level, and thus of supporting the transfer of knowledge from experience in one domain or task *to* other domains or tasks.

1 Introduction

Any approach to learning by debugging, or *failure-driven* learning (see, e.g., Sussman, 1975; Schank, 1982; Hayes-Roth, 1983; Kolodner, 1987; Birnbaum and Collins, 1988; Simmons, 1988; Hammond, 1989a), must begin by answering a rather obvious question: What *exactly* is being debugged? When this paradigm has been applied to the problem of learning to plan, it has generally been taken for granted that the answer is, simply, "plans" (see, e.g., Sussman, 1975; Simmons, 1988).[1] This has historically made a great deal of sense, inasmuch as classical planning work in AI has taken plans to be self-contained, monolithic structures that completely specify the sequence of actions that must be carried out in *order* to achieve a *goal. That* is, within the classical tradition, plans have been assumed to be something very much like computer programs. Concomitantly, *planners* have been viewed as devices that take goals and situational constraints as inputs and produce these sorts of complete, program-like structures as outputs; and *plan executors* have been taken to be essentially general-purpose mechanisms for the execution of such program-like structures.

However, this conception of planning has become increasingly untenable as the role of reactivity in goal-directed behavior has become more clearly understood (see, e.g., Hayes-Roth and Hayes-Roth, 1979; Brooks, 1986; Agre and Chapman, 1987; Firby, 1989; Hammond, 1989b). This shift towards reactive models of planning is in part motivated by the recognition that, since the conditions under which an agent's plans will be carried out cannot be completely anticipated, much of the responsibility for determining the particular actions that the agent will perform at a given time must lie in the plan execution component of that agent, rather than resting exclusively with the plans themselves. In order to be capable of carrying out the additional responsibilities required by these models, however, the plan execution component can no longer be taken to be a simple, general-purpose program interpreter. Rather, it must be seen as a highly articulated set of components, each devoted to controlling a particular aspect of behavior.

Consider, for example, a simple plan for keeping a piece safe in chess, formulated as a self-contained, program-like structure:

```
while the piece is on the board do
  if a threat against the piece is detected
  then either  a. move the piece
               b. guard the piece
               c. interpose another piece
               d. remove the threat
               e. ...etc.
```

There are two key points to notice about this plan. First, an agent cannot yield total control of its behavior to a plan of this sort, because the plan will never relinquish control unless a threat is successfully executed and the piece is taken, and the agent cannot

[1] An important exception to this concentration on the individual plans themselves, albeit not within a failure-driven framework, is Carbonell (1986).

afford to execute such a plan to the exclusion of all others. Second, the details of the actions to be carried out in service of this plan cannot be specified in very much detail in advance of detecting a particular threat.

A standard approach to problems of this sort involves *timesharing* or *multitasking.* Applying a similar approach to the current example entails that the above plan must relinquish control of the agent's computational, perceptual, and behavioral resources until such time as a threat against the piece is actually detected. This in turn implies that some executive component of the agent must be charged with the responsibility of returning control to the plan at the appropriate time, i.e., when such a threat is detected. Thus, a task that was formerly the responsibility of individual plans—threat detection—now becomes the responsibility of some specialized component of the agent's architecture.

The increased specialization entailed by this approach also offers opportunities to increase efficiency in much the same way that manufacturing efficiency exploits specialization of workers and equipment on an assembly line: By breaking plans up into constituent pieces, and distributing responsibility for those pieces among components of the agent specialized for those purposes, we can optimize each component for its particular purpose.

In light of the above discussion, we need to reconsider our original question of what is being debugged in a failure-driven approach to learning how to plan. Since a great deal of responsibility for determining what to do has now been shifted to the agent's plan executor, it follows that any adequate approach to learning by debugging must be capable of determining the causes of, and repairs for, performance errors arising from the operation of the plan executor. Approaches that consider only the plans themselves as the objects to be debugged are obviously incapable of making such determinations. In other words, as more responsibility is shifted to the plan executor, the focus of debugging must be shifted there as well.

Changing the focus of debugging in this way yields another benefit: Errors that arise from the operation of the plan executor are the very sorts of errors that are most likely to yield lessons of broad applicability. Because all plans depend upon the plan executor, repairing bugs in this resource has the potential to improve the execution of any plan, regardless of the domain in which it is intended to function. Furthermore, this argument applies not only to the plan executor, but to *any* component of the intentional architecture: Once ubiquitous functions such as threat detection have been assigned to specialized components of the agent's architecture, any improvement in a particular component benefits all plans utilizing that component. Thus, learning that occurs in the context of one task or domain may subsequently yield improved performance in other tasks and domains. In other words, this model serves as the basis for a straightforward approach to achieving *transfer* of learning across tasks and domains, one of the key issues in learning (see, e.g., Krulwich *et al.,* 1990). To enlarge upon our assembly line analogy, when a faulty item is discovered coming out of a factory, one might simply repair that item and continue on; but it is obviously more sensible to determine where in the manufacturing process the fault was introduced, and to see whether anything can be done to avoid such problems in the future. Our thesis is that a similar approach can be applied when learning how to plan. In other words, *when a plan fails, debug the planner, not just the plan.*

The process of debugging an intentional system must, from this perspective, involve determining which component of that system is responsible for the fault. This is a difficult problem inasmuch as the architecture of the agent is, as we have argued above, a rather complex mechanism. Our approach to this problem utilizes *model-based reasoning,* a methodology that has been developed in AI for reasoning about and debugging complex mechanisms such as electronic circuits (see, e.g., Stallman and Sussman, 1977; Davis, 1984; deKleer and Williams, 1987). In this paradigm for debugging, the diagnostic system uses a model of the device being debugged to generate predictions about what the behavior of the device would be if it were functioning properly. These predictions are then compared with measurements performed on the device itself. When a discrepancy is detected, the diagnostic system attempts to determine which of a set of possible device faults is the underlying cause of the discrepancy.

Applied to our problem of learning to plan by debugging, this paradigm becomes the following: A model of the agent's intentional architecture is used to generate predictions about the performance of its plans; deviations from these predictions can then be used to pinpoint where in the mechanism the fault lies. In other words, an intentional agent needs a model of itself in order to adequately diagnose and repair its failures (Birnbaum *et al.,* 1990; Collins and Birnbaum, 1990). Given such a model, the system is able to determine for itself what it needs to learn from a given failure.

The need for a self-model can be justified on psychological grounds as well. Many plans, such as cooking, require waiting for the appropriate moment to perform a particular task. As in the case of detecting and blocking threats described above, an agent cannot afford to idle its resources while waiting for this moment to arrive. A standard technique in such cases is for the agent to set an *alarm* that will return its attention to the activity when the task needs to be performed. To set up such an alarm, the agent must have some notion of the kinds of events that its control allocation component is sensitive to—in other words, a model of the properties of this component. Mnemonic devices, for example the proverbial string tied around one's finger, employ similar techniques to attack a slightly different problem, that of retrieving a piece of information at the appropriate time. To develop and employ such techniques, the agent must not only have a model of its attention focussing mechanisms, but also of how its memory works. To the extent that these techniques work, the naive psychological models upon which they are based must reflect something real about the underlying structure of the agent.

2 Modelling the agent: Threat detection

A central issue in our approach is the development of explicit models for intentional agents that can be used in debugging their performance. We have developed simple models *of a* number *of* important planning components, including threat detection, execution scheduling, projection, and case retrieval and adaptation, and applied them to learning within the context *of competitive* games such *as* chess and checkers (see, e.g., Collins *et al.*, 1989; Birnbaum *et al*, 1990; Birnbaum *et al.*, 1991). In this section we will describe our model of threat detection in a simple agent.

We model threat detection as a simple rule-based process (Birnbaum *et al*, 1990). The planner's threat detection knowledge is encoded as a set of condition-action rules, with each rule being responsible for recognizing a particular type of threat. In chess, for example, the planner could be expected to possess rules specifying, among other things, the various board configurations that indicate that an attack on a piece is imminent.

A set of threat recognition rules does not, in itself, completely determine the behavior of a threat detection mechanism, since the planner must still decide how and when these rules should be evaluated, and what will be done with the results. In particular, the planner can vary the particular rules to be evaluated, the frequency with which they are checked, the domain over which they are evaluated, and the extent to which previously detected threats are cached in memory as opposed to being recomputed each time the rules are evaluated.

Threats are placed on a threat queue when detected:

$$\forall x \exists t \;\; t \le \text{ert}(x) \,\&\, \text{detect}(x, t) \rightarrow \text{added-to}(x, T, t)$$

Threats remain on the threat queue until explicitly *removed:*

$$\begin{aligned} \forall x, t_1, t_2 \quad & \text{added-to}(x, T, t_1) \,\&\, t_2 \le \text{ert}(x) \,\& \\ & \sim\exists t_r \;\; (t1 \le t_r \le t2 \,\&\, \text{removed-from}(x, T, t_r)) \\ \rightarrow\, & \text{member-of}(x, T, t_2) \end{aligned}$$

A rule is capable of detecting a threat at a particular time if there exist bindings such that evaluating the rule with those bindings at that time would result in the detection of the threat:

$$\begin{aligned} \forall x, r, t, \theta \quad & \text{could-detect}(r, x, t, \theta) \leftrightarrow \\ & \text{evaluate-rule}(r, \theta, t) \rightarrow \text{detect}(x, t) \end{aligned}$$

A rule can detect a threat in time if there is a time while *that threat is active, but before it can be realized,* when the rule could detect that threat:

$$\begin{aligned} \forall x, r, \theta \quad & \text{could-detect-in-time}(r, x, \theta) \leftrightarrow \\ & \exists t \; \text{active}(x, t) \,\&\, t \le \text{ert}(x) \,\&\, \text{could-detect}(r, x, t, \theta) \end{aligned}$$

Our rules are capable of detecting all active threats in time:

$$\forall x \;\exists r, \theta \;\; \text{could-detect-in-time}(r, x, \theta)$$

Any threat that could be detected without attention focussing can still be detected with such focussing:

$$\begin{aligned} \forall x \;\; & (\exists r, \theta \;\; \text{could-detect-in-time}(r, x, \theta)) \rightarrow \\ & (\exists r, \theta, t \;\; \text{subset-of}(\theta, \text{focus-results}(t)) \,\& \\ & \quad \text{active}(x, t) \,\&\, t \le \text{ert}(x) \,\&\, \text{could-detect}(r, x, t, \theta)) \end{aligned}$$

The bindings available at a given time are the results of applying all of the focus rules at that time:

$$\begin{aligned} \forall t \;\; & \text{focus-results}(t) = \\ & \text{union-over}(F, \text{focus-rule-evaluation-result}(f, t)) \end{aligned}$$

At each turn, all of the threat rules are applied using bindings provided by the focus rules:

$$\forall r, t \;\; \text{evaluate-rule}(r, \text{focus-results}(t), t)$$

A threat is removed from the threat queue when it is no longer active:

$$\begin{aligned} \forall x, t \;\; & \text{remove-from}(x, T, t) \leftrightarrow \\ & \text{member-of}(x, T, t) \,\&\, \sim\text{active}(x, t) \end{aligned}$$

Figure 1: Partial model of threat-detection[2]

2 The predicate "ert" stands for *earliest realization time, i.e.,* the earliest time at which the threatened action can be expected to occur.

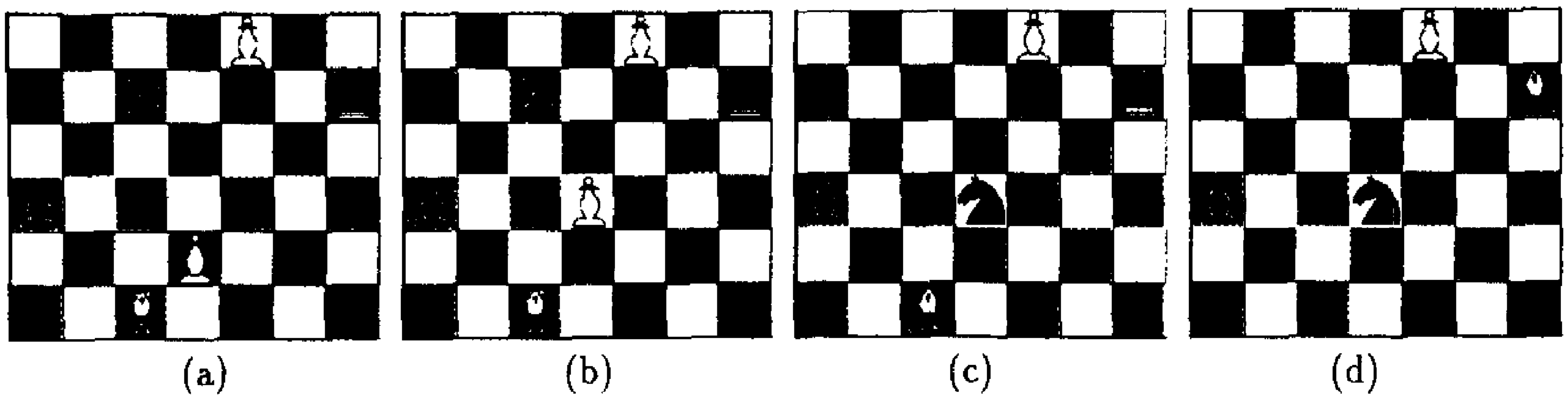
Figure 2: A discovered attack

Let's consider this last issue in more detail. Broadly speaking, the planner has a choice between two strategies for threat detection: It can recompute the set of outstanding threats each time it checks its rules, or it can incrementally compute this set by noting the changes that have occurred since the previous check. In tunvtaking games, such as chess, the latter approach is arguably more efficient, and indeed appears to be the one employed by humans.

In order to benefit from the incremental approach, however, a planner must find a way to ensure that it can detect new threats without having to re-detect all of the old threats, since otherwise it is in effect doing all of the work entailed by the recomputation approach. Given the rule-based framework for threat-detection described above, focussing on threats resulting from changes can be implemented as a set of restrictions on the domain of application of the threat detection rules. In our model these restrictions are themselves implemented as a set of *focus rules* that specify the domain over which the threat detection rules will be applied.

The above is a rather brief outline of the threat detection model we have developed. A portion of the model is shown in figure 1,

3 A case study: Discovered attacks

The model of threat detection described above was developed as part of an account of learning in competitive situations—in particular, chess. A central prediction of the model is that if a threat exists, it will be detected. In our application of the model to competitive planning, this prediction is crucial, inasmuch as the decision to block a particular threat depends upon the assumption that all outstanding threats have been taken into consideration, in order to rule out the possibility that a higher priority threat exists. There are, however, a number of reasons why this prediction might fail, and each of these presents an opportunity to learn a different lesson.

Consider, in particular, the example of *discovered attacks* in chess, in which the movement of one piece opens a line of attack for another piece. Novices often fall prey to such attacks, and the key point is that this is *not* because they fail to understand the mechanism of the threat, i.e., the way in which the piece can move to make the capture. Instead, the problem appears to be one of focus: Novices simply fail to consider new threats arising from pieces other than the one just moved. In other words, the problem lies not in their ability to detect the given threat in principle, but rather in their decisions about where to look for threats in practice. In fact, without such a distinction—e.g., if the method for detecting threats were non-incremental in the sense discussed earlier, and entailed scanning the entire board after each move—the problem of discovered attacks, and the need to distinguish them from other sorts of attacks, would not even arise. The model of threat detection described in the last section reflects this distinction by introducing a notion *of focus rules* that limit the application of *threat detection rules.* The former embody knowledge of where to look; the latter, of what to look for. Our system is capable of learning both sorts of rules, depending upon the cause of the failure in a given situation.

The following example depicts a scenario in which the system falls prey to a discovered attack, and thereby learns to improve the attention-focusing portion of its threat detection component. In figure 2{a), the system's opponent is to move. In figure 2(b), we see the result of the opponent's move, which is to advance one of its pawns one square. This opens a discovered attack by the opponent's bishop on the system's rook. However, because of overly restrictive attention-focusing, the system fails to notice this threat, despite the fact that it has a threat detection rule which, if applied to the appropriate portions of the board, would have in fact detected the threat. In figure 2(c), we see that the system has chosen to capture the opponent's pawn with its knight, leaving the threat on the rook unaddressed. Finally, in figure 2(d), the opponent carries out the threat and captures the rook.

Let's examine the system's decision-making in some detail. Any decision to make a particular move must be based upon some assessment of how well that move compares to the available alternatives. Such a comparison, in turn, depends upon the ability to project the implications of each alternative considered. Once an alternative is selected, relevant aspects of the projection made in service of that choice become, in effect, predictions about the future course of events upon which the rationality of the system's chosen action depends. These predictions, in turn, depend upon assumptions stemming from the system's self-model, for example, the assumption that its threat detection component is capable of detecting all outstanding threats. The failure of one of these predictions leads the system to attempt to determine which component of its decision-making process was at fault, as described above. Our approach to this problem entails determining how and why the faulty prediction was originally formulated. A record of the reasoning by which the model gave rise to the predictions is kept in the form of explicit *justification structures* (see, e.g., deKleer, Doyle, Steele, and Sussman, 1977; Doyle, 1979). Diagnosis, then, consists of tracing back through these justification structures in order to determine where the fault lies.

In this particular case, the relevant prediction is that capturing the opponent's pawn would be of greater value than blocking any extant threat. When the opponent takes the system's rook, this prediction is violated. The justification structure underlying the prediction is roughly the following: The system believes that no such threat exists because it has not detected such a threat and it assumes that it can detect all threats. The latter assumption, in turn, is justified by the assumption that there is a threat rule capable of detecting any given threat, and the assumption that the threat rules have been evaluated using appropriate bindings. Finally, the latter is justified by the assumption that the system's focus rules have generated the appropriate bindings. This justification structure, and the assumptions it contains, are derived from the model of threat detection described above.

There are two components of interest that might be at fault in this case: the *threat rules,* and the *focus rules.* Determining which component is at fault means exonerating all of the alternatives. In general, exonerating an alternative can require an arbitrary amount of inference. However, in this case the model's formulation of attention focussing includes the assumption that any threat which can be detected without using the focus rules to delimit the bindings utilized by the threat rules can also be detected using only those bindings (sec figure 1 above). Attempting to fault this assumption means attempting to show its negation, which suggests the following experiment: First, evaluate all threat rules over the situation without using any focus rules to delimit their bindings. If this process results in the system detecting the threat, then the fault lies in the set of attention focusing rules; if not, it lies in the set of threat rules. In this instance, the problem lies in the focus rules. Once that has been determined, the system retrieves a specification for focus rules, and uses this specification in conjunction with the current circumstances to acquire a new focus rule via explanation-based learning (see, e.g., Dejong and Mooney, 1986; Mitchell *et al.,* 1986). This example has been implemented in our test-bed system (described in Collins, Birnbaum, and Krulwich, 1989)- Once the system has uncovered and repaired the deficiency in its attention focussing, it no longer falls prey to discovered attacks, regardless of the particular pieces involved or their location on the board.

4 Conclusion

We have developed a model-based approach to learning from plan failures in which an agent uses a model of itself to determine where in its planning or execution the cause of a failure lies. We believe that such an approach constitutes the most promising basis for developing learning models that are capable of deciding for themselves what needs to be learned from a given experience. In addition, such methods appear capable of learning about planning at a very abstract level, and thus of supporting the transfer of knowledge from experience in one domain or task to other domains or tasks.

Acknowledgments

We thank Matt Brand and Louise Pry or for many useful discussions. 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.

References

Agre, P., and Chapman, D. 1987. Pengi: An implementation of a theory of activity. *Proceedings of the 1987 AAA1 Conference,* Seattle, WA, pp. 268-272,

Birnbaum, L., and Collins, G. 1988. The transfer of experience across planning domains through the acquisition of abstract strategies. *Proceedings of the 1988 Workshop on Case-Based Reasoning,* Clearwater Beach, FL, pp. 61-79.

Birnbaum, L., Collins, G., Brand, M., Freed, M., Krulwich, B., and Pryor, L. 1991. A model-based approach to the construction of adaptive case-based planning systems. To appear in *Proceedings of the 1991 Workshop on Case-Based Reasoning,* Washington, DC.

Birnbaum, L,, Collins, G„ Freed, M., and Krulwich, B. 1990. Model-based diagnosis of planning failures. *Proceedings of the 1990 AAA1 Conference,* Boston, MA, pp. 318-323.

Brooks, R. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation,* vol. 2, no. 1.

Carbonell, J. 1986. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In R. Michalski, J. Carbonell, and T. Mitchell, eds., *Machine Learning: An Artificial Intelligence Approach, Volume 11,* Morgan Kaufmann, Los Altos, CA, pp. 371-392.

Collins, G., and Birnbaum, L. 1990. Problem-solver state descriptions as abstract indices for case retrieval. *Working Notes of the 1990 AAAI Spring Symposium on Case-Based Reasoning,* Stanford, CA, pp. 32-35.

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

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 Williams, B. 1987. Diagnosing multiple faults. *Artificial Intelligence,* vol. 32, pp. 97-130.

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

Firby, R. 1989. Adaptive execution in complex dynamic worlds- Research report no. 672, Yale University, Dept. of Computer Science, New Haven, CT.

Hammond, K. 1989a. *Case-Based Planning: Viewing Planning as a Memory Task,* Academic Press, San Diego.

Hammond, K. 1989b. Opportunistic memory. *Proceedings of the Eleventh IJCAl,* Detroit, MI, pp. 504-510.

Hayes-Roth, F. 1983. Using proofs and refutations to learn from experience. In R, Michalski,]. Carbonell, and T. Mitchell eds., *Machine Learning: An Artificial Intelligence Approach, Vol. 1,* Tioga, Palo Alto, CA, pp. 221-240.

Kolodner, J. 1987. Capitalizing on failure through case-based inference. *Proceedings of the Ninth Cognitive Science Conference,* Seattle, WA, pp. 715-726.

Krulwich, B,, Collins, G., and Birnbaum, L. 1990. Cross-domain transfer of planning strategies: Alternative approaches. Proceedings *of the Twelfth Cognitive Science Conference,* Cambridge, MA, pp. 954-961.

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

Schank, R. 1982. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People.* Cambridge University Press, Cambridge, England.

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

Stallman, R., and Sussman, G. 1977. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence,* vol. 9, pp, 135-196.

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