Using and Refining Simplifications: Explanation-based Learning of Plans in Intractable Domains

Steve A. Chien
Beckman Institute for Advanced Science and Technology
University of Illinois
Urbana, Ilinois 61801

Abstract

This paper describes an explanation-based approach lo learning plans despite a computationally intractable domain theory. In this approach, the system learns an initial plan using limited inference. In order to detect plans in which the limited inference causes a faulty plan the system monitors goal achievement in plan execution. When a plan unexpectedly fails to achieve a goal (or unexpectedly achieves the goal) a refinement process is triggered in which the system constructs an explanation for the expectation violation. This explanation is then used to refine the plan. By using expectation failures to guide search the learner avoids a computationally intractable exhaustive search involved in constructing a complete proof of the plan. This approach has the theoretical property of convergence upon a sound plan.

1 Introduction

Most *Explanation-Based Learning(EBL)* [DeJong86, Mitchell86] systems require that their domain theories be complete, sound, and computationally tractable[1]. But in real-world domains, large amounts of knowledge are necessary to adequately describe world behavior. With the necessary complex domain theory, complete reasoning becomes a computationally intractable task.

Consider getting to work in the morning. Your plan might be influenced by weather conditions, road construction, traffic conditions, your spouse's plans that day, small errands you might have to run, and the condition of your car. The number of potentially relevant factors is enormous.

Yet how do people plan in complex situations? One way in which this is accomplished is by using simplifying assumptions. By using simplifications, people can develop approximate plans despite complexity - and subsequently use and refine these approximate plans as the flaws in the plans become apparent.

The paper describes an approach to explanation-based learning using a complete and sound but intractable domain theory. In this approach the system uses limited inference in determining interactions between actions in the plan. Because this limited inference introduces the possibility of learning flawed concepts, the system monitors the execution of learned plans. When the system observes a discrepancy between the predicted and observed goal achievement, it begins a refinement process in which the violated expectation is explained. If the discrepancy is an unexpected plan failure, the explanation describes how a support in the plan is blocked. This explanation represents a set of inferences not previously explored by the system because of inference limitations. If the discrepancy is an unexpected plan success, the explanation describes how an anticipated failure was prevented by a method previously not considered due to inference limitations. In both cases the explanation is then used to modify the plan to avoid the incorrect prediction in the future.

By using expectation violations to direct the search for plan interactions the system avoids the computationally intractable blind search for interactions. Additionally, a concrete example of the failure aids the explanation process and subsequent failure analysis. However, these computational gains come at the cost of the failures used to guide the search.

2 Overview

Our approach to incremental explanation-based learning has three requirements. First, that a complete and sound domain theory exists. This means that in theory the system can generate sound explanations given unbounded time and computational resources. Second, that the system is given a set of default methods to use to determine approximate truth values[2]. Third, that the utility of a correctly learned concept is adequate to offset the cost of the failures required to learn it in our approach. This approach has been tested by implementation of an incremental EBL system. This model of problem-solving and learning consists of four steps: *Initial*

This research was supported by an IBM Graduate Fellowship, a University of Illinois Cognitive Science/Artificial Intelligence Fellowship, the Office of Naval Research under grant N-00014-86-K-0309, and the National Science Foundation under grant NSF-IRI-87-19766.

[1] For a more detailed description of domain theory requirements and problems see [Rajamoney87, Mitchell861.

[2] Development of methods for learning simplifications or a general theory of simplifications are areas for future work. An initial laxonomy of simplifications is described in [Chien87].

Learning, Expectation Violation, Explanation of Expectation Violation, and Knowledge Modification.

1. *Initial Learning:* The system can learn plans from internal problem-solving or from observation. In each case, the initial plan is an explanation for goal achievement using limited inference.
2. *Expectation Violation:* There are two types of expectation violations: unexpected successes and unexpected failures. Unexpected failures can result from problem-solving or observation and occur when a plan explanation for goal achievement (either observed or constructed by the planner) is applicable but the goal is not achieved. An unexpected success occurs when the system observes a plan from its plan library and predicts failure (due to an applicable failure explanation attached to the plan in the plan library) but observes the plan to succeed.
3. *Explanation of Expectation Violation:* The system constructs an explanation of the violated expectation.
4. *Knowledge Modification:* The system analyzes the explanation of the violated explanation to determine how to modify the preconditions and/or ordering constraints of the plan.

The system uses a representation based on situation calculus which significantly extends STRIPS operators. Facts can be asserted in four ways. First, a fact may be a direct effect of an operator. Second, unlike STRIPS operators, a fact may be asserted as an inferred effect of an operator, allowing representation of operators whose effects depend upon the state of the world in which they are executed (e.g. if I pull the trigger of a gun, the effects depend upon whether or not the gun is loaded, the direction in which the gun is pointed, etc.). Third, a fact may be inferred to be true by intra-situational rules (e.g. if A is above B and B is above C then A is above C). Finally, of key importance to this research, a fact may be assumed to be true by a persistence assumption.

In a complete inference model, as each new situation is caused by the execution of an operator, the system must exhaustively prove each fact in the new situation. In a complex domain, persistence of facts may depend on many other factors, making computation of updates expensive. This is the qualification problem [Shoham86] and is one aspect of the frame problem. In general, learning using complete reasoning requires determining the truth value of facts in a non-linear plan involving conditional effects which is NP-hard [Chapman87|.

In order to deal with this difficulty the system performs limited updating based on inferred effects. Concretely, the system examines inferred effects only when direct effects cannot account for observations. For example, if an operator is executed, and direct effects do not predict that one of its preconditions was true, the system attempts to prove the precondition involving inferred effects. Complete updating of direct effects is performed. The system also uses a defeasible persistence rule to allow potentially unsound updating of the model. A formal definition of the persistence simplification rule is:[3]

(V P,si,sj,sk)IN(P,si) ^ -(3 0) [contradicts(Q,P) ^ IN(0,SJ)A precedes(si,sj) A precedes(sj,sk)] ->IN(P,sk)

where IN(P,s) means that the fact P is believed by the system in situation s, A fact P is said to contradict a fact Q if P is ~Q or if P and Q specify conflicting attribute values for the same object, and precedesa,b) means that the situation a temporally precedes the situation b. Intuitively, this rule states that a fact P can be assumed to persist to a later situation provided it is not explicitly contradicted by a belief in an intervening situation. An important point to note is that because the system performs complete updating of direct effects, any incorrect persistence simplification must be due to an unconsidered inferred effect.

A plan is a constrained set of operators to achieve a desired world state and consists of a *goal state, plan preconditions, operators, ordering constraints,* and a *causal explanation.* The *goal state* is a partial situation that the system believes will be true after the execution of the plan. The *plan preconditions* are a partial world specification required for the proper execution of the plan. *Operators* is the set of the operators which cause the goal state to be true. *Ordering constraints* is a set of constraints upon the order in which the operators may be executed to achieve the goal. The *causal explanation* is a description of how the plan preconditions and operators achieve the goal state.

In the following section, the initial learning process is described and an example shown. Then the refinement process consisting of the detection, explanation and modification steps outlined above is described and several examples are discussed. Finally, comparisons to other work and future work are covered.

3 Initial Learning

There are three parts to learning an initial plan representation: 1) goal explanation, 2) explanation generalization, and 3) computation of ordering constraints. As the system processes operators in a training example, the system maintains a causal model. The causal model is the systems view of the world updated using limited inference as described above. After processing the operator sequence, the system constructs a causal explanation for goal achievement in a backward-chaining best-first manner with preference towards simplest explanations. This explanation is then generalized using the EGGS technique [DeJong86] to produce the plan preconditions.

3 These inferences arc currently implemented by specialized LISP code. Development of a single general inference engine for both defeasible and standard inferences and a declarative representation for defeasible inferences are areas for future work.

Next, the system computes constraints required by supports in a plan. For a fact to be true at a given state, it must be true in a previous state (achievement) and remain true until the state it is needed (protection). The achievement constraint requires that a fact be achieved before it is needed. Consequently when there is a chain of inferences, effects, or persistence assumptions from an operator A that establishes a precondition for operator B, A must be ordered before B. For example, suppose an object must be at the drilling station in order to be drilled. Then if an object is moved to the drilling station and then drilled, the move operator would support the drill operator through the location precondition of the drill operator. The system determines support relations by examining the explanation for goal achievement and posts these ordering constraints upon the plan.

Additionally, if a fact is achieved prior to immediately before it is needed, it must not be falsified by another action before it is needed. This is a *protection* constraint and is satisfied by ordering the clobbering action before the protected fact is achieved or after the protected fact is needed. Note that 1) if the fact is true from the initial state the clobberer cannot be moved before the protection interval and 2) if the fact is part of the goal specification the clobbering action cannot be moved after the protection interval. Some clobberings will be dependent upon how the plan is instantiated and can be prevented by posting constraints on variable bindings. For example, -(on ?x ?y) will clobber (on ?z ?y) only if ?z and ?y refer to the same object. This results in an ordering constraint dependent on the plan instantiation. Returning to the drilling station example, the location of the object to be drilled must be protected from the time it is achieved by the move until the time it is used by the drill operator. If at another time in the plan the object was moved to another location, this other move cannot be executed during the protection interval because it would clobber the fact that the object is at the drilling station.

The system computes protection intervals by tracing each precondition or goal support to the action which asserted it (or the initial state). These facts are then compared against effects of actions in the plan to determine potential clobberers. Each interval and action for which there exists a legal instantiation of variables which would cause the action to clobber the interval causes a protection constraint.

In order to clarify the initial learning process, an example from the system's workshop domain will now be discussed. In this example, the system learns a plan to construct a small assembly called a widget. In this plan, the system is shown an initial state with a metal rod, a gear, and a sheet. First the rod is heated and rolled. Next, the gear and sheet have holes drilled in them. Then the rod is inserted into the gear for a tight friction fit and through the sheet with a loose fit that allows the rod to spin. The system is told that this assembly is a widget. The initial state and operator sequence are:

```
INITIAL STATE:
(at rod101 bench2)              (shape gear212 gear)
(at gear212 bench2)             (state gear212 solid)
(at base4 bench4)               (shape base4 sheet)
(composition rod101 metal)      (state rod101 solid)
(composition gear212 metal)     (state base4 solid)

OPERATOR SEQUENCE:
m1 (move rod101 oven)
h1 (heat rod101)
m2 (move rod101 rolling-station)
r1 (roll rod101 12.0cm)
m3 (move rod101 bench3)
m4 (move gear212 drilling-station)
d1 (drill gear212 hole784 12.0cm)
m5 (move gear212 bench3)
m6 (move base4 drilling-station)
d2 (drill base4 hole785 12.1cm)
m7 (move base4 bench3)
i1 (insert rod101 gear212 hole784)
i2 (insert rod101 base4 hole785)
```

The system explains how the operators achieve the definition of a widget which involves a gear and rod spinning with respect to a sheet.

The simplified explanation for the plan uses 5 rules, 14 persistence assumptions, and 6 initial state facts. The simplifications covered facts persisting through 61 situations. Our exhaustive explanation used 68 rules to prove persistence, resulting in a total of 73 rules. Hence, the reduction in complexity (73 / 5 = 14.6, over an order of magnitude in proof size) by the persistence simplifications is significant. These savings are even more significant when considering that the explanation search space is exponential in the proof size.

Yet the temporal limited inference used in learning this plan is not without cost. In this example, the system makes the assumption that the shape of the gear does not change from the start state through the end of the plan. In the exhaustive formulation this persistence depends upon the fact that the gear is made of metal. However, the simplified plan does not account for place any restriction upon the composition of the gear used in the plan.

The learned plan abstracts many of the specifics of the example. For instance, in the example, the rod and gear were at the same initial location. Because it is not required by the explanation, the plan does not have this constraint. Additionally, the ordering of the actions in the plan is only partially constrained by support and protection constraints. For example, moving the rod to the oven (m1) must precede the heat operator (h1) because the move operator supports the heat operator through the location precondition. However, the plan does not include any interaction between the preparation of the gear, rod, and sheet before the insert.

4 Refinement

Refinement consists of three steps: expectation violation, explanation, and modification. The detection is done by monitoring goal achievement. The system monitors the top-level goal that is given to the problem-solver.[4] When there is a discrepancy between

expected and observed goal achievement, the system begins the refinement process.

Given this monitoring strategy, there are two possible cases: unexpected goal failure and unexpected goal achievement. In the case of unexpected goal failure the system encounters a plan execution that it expected to succeed but the plan fails. Because the domain theory used by the system is sound, a failure must be due to a violated persistence assumption which results in a protection interval violation. This failure explanation represents sufficient conditions for the failure and provides information on how the failure affects the plan applicability. There are three ways in which the failure can be prevented:

1. Ordering: the failure explanation describes an inferred effect of an operator clobbering a protected fact. If this operator is ordered before or after the interval it cannot clobber the protection and the failure will not occur.
2. Block Failure Constraints: the failure explanation has preconditions and ordering constraints. If any of these constraints arc not satisfied, the failure explanation is not applicable and the failure will not occur. For example, suppose a failure explanation has an operator A which establishes a condition (fact) causing operator C to clobber a protection in the plan through an inferred effect. This failure explanation would require A ordered before C. If C were ordered after A the failure would be prevented.
3. Block Persistence in Failure Explanation: If a persistence assumption in the failure explanation is blocked by an effect of an operator in the plan the failure will not occur. This blocking will impose additional constraints upon the plan in several ways. First, all of the constraints of the blocking explanation must be satisfied (ordering and preconditions). Second, the effect must occur within the protection interval in the failure explanation (the converse of protection).

Because of the potentially large number of persistence assumptions in the failure explanation and the expense of planning the system performs limited search to find Block Persistence in Failure Explanation methods of failure prevention in this situation. The applicability conditions of the refined plan arc formed as follows. Concretely, let P be the original conditions, O be the constraints on an ordering fix, C be the constraints to block a failure constraint, and S be a disjunct of the ways to block failure persistences found by limited search. The new constraints on the plan are (P A (O V C V S)).

In the case of unexpected goal achievement, the system observes a plan that it expects to fail but the plan succeeds. Because the limited inference always results in over-generalization, the incorrect prediction must be due to an over-general failure explanation. Since the system's failure analysis regarding Ordering and Block Failure Constraints is complete, the flaw must be due to an occurrence of Block Persistence in Failure Explanation as described above. Directed by this expectation failure, the system explains how a persistence in the failure explanation is blocked. The effective preconditions of the plan are now (P^ (O V C v S v N)) where N represents the constraints needed to produce the blocking persistence method occurring in the example.

In order to clarify the refinement process, two examples of refinement for the widget plan are now described. In the first example shows an unexpected failure resulting in a precondition refinement to the plan. In the second example an unexpected success is observed and the system learns a method for preventing the failure by adding an operator to the plan.

In the first example, the system attempts to use the widget plan and the plan fails. The system backtraces the support network in the plan and verifies supports for the goal which should be true in the goal stale. The system determines that all of these supports are met except that the gear is no longer gear-shaped. It determines that in the real world the gear is deformed in the final situation.

Next, the system explains why the gear is deformed in the final situation. The system derives the explanation shown in Figure 6 . The deformed shape of the

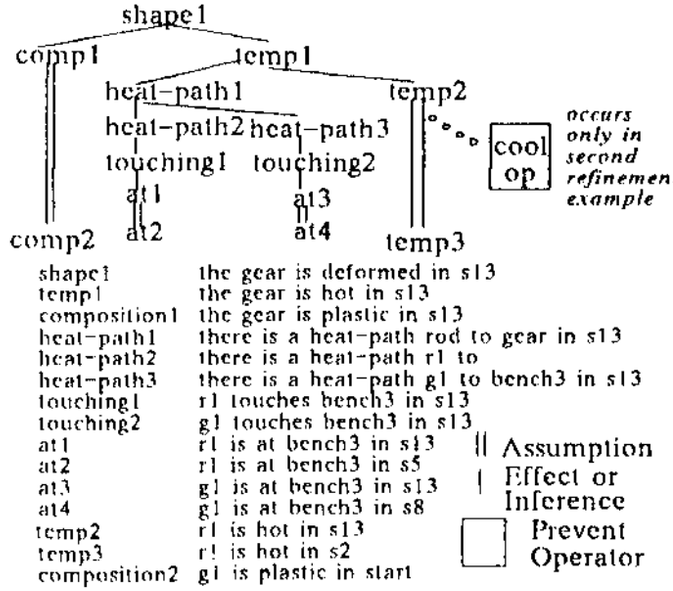

Figure 6: Failure and Prevention

gear is explained as follows. The gear gl is plastic; it became hot in situation s12 and melted, causing it to be deformed. The plastic composition property of the gear persisted from the start state. The gear became hot because it was at the same location as the rod (caused by the last moves of the gear and rod) and the rod was hot from being heated (from the heat operator). This failure explanation is generalized using the EGOS algorithm. The generalized explanation is then used to determine ways to prevent the failure using the three methods described above.

While it is possible to monitor subgoals, there is a tradeoff between monitoring effort and debugging effort. More detailed monitoring facilitates concept debugging because discrepancies may be delected closer to the underlying causes.

First, the system checks for an ordering fix. However, because the shape of the gear is an initial state fact used in the goal state, it is not possible to order the plan actions around the protected interval to prevent the clobbering. Thus there is no ordering fix to the failure.

Second, the system tries the block failure constraints method. The failure explanation has the precondition that the gear be plastic, hence if the gear is not plastic, the failure does not apply. The system also determines that the failure ordering constraints arc that the heat operator must be ordered before the move operators. However, this constraint is required by the plan, and hence cannot be denied.

Finally, the system tries the block persistence method but is unable to find a way to prevent the failure due to the limited search constraint. The first refinement example concludes with the plan being modified by adding the constraint that the gear not be plastic.

The second refinement example demonstrates the process of refinement due to unexpected successes. The system observes an initial state in which the gear is plastic and an operator sequence in which the metal rod is cooled after being rolled. The system predicts that the plan will fail because the failure conditions for the deformed gear failure from the previous example applies to the current example. However, the plan is observed to succeed, resulting in an unexpected success. Hence, a persistence assumption in the failure must have been blocked by the new plan. The system explains that the cooling step effect of the rod being cool blocks the persistence assumption regarding the hot temperature of the rod in the failure explanation, thus preventing the deformed gear failure as shown in Figure 6.

Because the cooling step did not appear in the general explanation for goal achievement, it was previously thought by the system to be an unimportant action. While in theory the system had the base-level knowledge necessary to understand this prevention upon initially learning the plan, this would have required the system to postulate the possibility of the melted gear failure which is a computationally intractable process. However, after the system has observed the failure and learned the corresponding refinement, the system predicts that the plan will fail and observed data is that the plan has succeeded. This expectation failure focusses the system's attention on portions of the failure that can be blocked; allowing tractable understanding of the preventive measure [Chien88].

The next step is to find the appropriate constraints on the plan required to prevent the expected failure. This is done by propagating the constraints between the plan and failure explanation (yielding the correspondences between the objects and values in the failure and the objects and values in the plan), computing the constraint that the effect of the cool action must block the persistence assumption in the failure explanation, and adding the supporting causal explanation for the cooling step to the plan. This process results in the constraint that the object that is cooled must be the rod. Next the system analyzes the ordering constraints. It finds that the cooling step must occur during the protection interval in the failure in order to block the failure. This requires that the cool step be ordered after the rod has been heated but not after both moves[5]. The new plan constraints are formed by adding these constraints in the disjunct for blocking preventions.

5 Discussion

Other related work includes techniques by Mooney [Mooney87] which addressed order generalization for STRIPS operators from a standard FBL framework and did not involve refinement. Additionally there have been numerous failure-driven refinement systems including [Doyle86, Gupla87, Hammond86] which also deal with intractability but address neither refinement from unexpected successes nor order generalization. PRODIGY [Minton87], FAILSAFE [Mostow87], and SOAR learn control rules from search failures (failures to find a solution) rather that failure to achieve the goal (as described in this paper) and hence do not make domain-level defeasible inferences. Abstraction work in SOAR [Unruh87] also does not make defeasible domain-level inferences. Other work by Chien [Chien89] also involves learning from unexpected successes but these successes due to multiple (disjunctive) methods to achieve goals not unsound simplifications. The general approach of refinement directed by violated expectations is similar to that described in [Hayes-Roth83, Schank82]. Another approach to learning in intractable domains involves constructing explanations in a transformed concept space such as [Tadepalli86] which learns goal structures of chess plans. A related research area addresses intractability in- using learned plans [Kellcr87] while our work focusses on intractability in learning plans.

The algorithm presented in this paper has the following theoretical properties:

1. Complete Model Convergence (soundness): As the plans are modified to prevent observed failures by methods which do not add operators the plan analysis converges on that dictated by the exhaustive analysis. Because we assume there are a finite number of inferred effects for each operator and operators in the plan, there are a finite number of potential clobberings. With each failure, the planner uncovers a previously undetected clobbering. As a result, the planner will eventually consider all clobberings.

2. Valid Explanation Generation (completeness): Suppose we have a set of examples for which there is a general plan for achieving a goal in our representation (e.g. no loops, conditionals, etc.). The learning algorithm described will eventually find an explanation which covers the examples in the complete analysis. Basically, the proof proceeds as follows. Suppose we

5 Note that the cool step must also follow the roll step to prevent clobbering the temperature protection interval which runs from the heat step to the roll step.

generate an explanation structure. Because of the complete model convergence property, any non-covering explanation cannot be refined to cover all the examples. Hence we can discard any explanation which fails in this manner. Because there are a finite number of operator effects and operators, there are a finite number of explanations for any given number of inferences. Because explanations are generated in order of increasing number of inferences we will eventually generate any explanation of finite length. Since there exists a covering explanation and we discard non-covering explanations, we will eventually generate a covering explanation.

6 Conclusion

This paper has described an approach to dealing with the complexity of learning plans involving inferred effects. In this approach, the system uses limited inference and defeasible inference rules in learning an initial plan. Because this limited inference may result in a flawed plan, the system monitors goal achievement. Incorrect prediction of goal achievement triggers a refinement process in which an explanation of the expectation violation is used to modify the plan. This refinement process is shown to produce a sound covering explanation. By using expectation violations to direct the search for inferred interactions, computationally intractable blind search through the space of possible interactions is avoided. Additionally, because the system has a concrete example of the expectation violation to explain, the explanation portion of the refinement process is simplified. Thus, by using the examples to guide the learning process, our approach can learn correct concepts despite the computational difficulties of a complex domain theory.

Acknowledgements

This work benefited from discussions with Ray Mooney. Comments and direction from my advisor, Gerald DeJong, and the rest of the CSL learning group are also gratefully acknowledged.

References

[Chapman87jD. Chapman, "Planning for Conjunctive Goals," Artificial Intelligence 32, 3 (1987), pp. 333-377.

[Chien87] S. A. Chien, "Simplifications in Temporal Persistence: An Approach to the Intractable Domain Theory Problem in Explanation-Based Learning/' Technical Report UILU-ENG-87-2255, AI Research Group, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, September 1987.

[Chien88] S. A. Chien and G. F. DeJong, "Recognizing Prevention in Plans for Explanation-based Learning," Proceedings of the American Association for Artificial Intelligence Workshop in Plan Recognition, St. Paul, MN, August 1988.

[Chien89] S. A. Chien, "Learning by Analyzing Fortuitous Occurrences," Proceedings of the Sixth International Workshop on Lachine Learning, Ithaca, NY, June 1989.

[DeJong86]G. F. DeJong and R. J. Mooney, "Explanation-Based Learning: An Alternative View," Machine Learning /, 2 (April 1986), pp. 145-176.

[Doyle86] R. J. Doyle, "Constructing and Refining Causal Explanations from an Inconsistent Domain Theory," Proceedings of the National Conference on Artificial Intelligence, Philadelphia, PA, August 1986, pp. 538-544.

[Gupta87] A. Gupta, "Explanation-based Failure Recovery," Proceedings of the National Conference on Artificial Intelligence, Seattle, WA, July 1987, pp. 606-610.

[Hammond86]K. Hammond, "Learning to Anticipate and Avoid Planning Failures through the Explanation of Failures," Proceedings of the National Conference on Artificial Intelligence, Philadelphia, PA, August 1986, pp. 556-560.

[Hayes-Roth83]F. Hayes-Roth, "Using Proofs and Refutations to Learn From Experience,"in Machine Learning: An Artificial Intelligence Approach, R. Michalski, J. Carbonell and T. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 221-240.

[Kcller87] R. Keller, "Concept Learning in Context," Proceedings of the Fourth International Workshop on Machine Learning, Irvine, CA, June 1987, pp. 91-102.

[Minton87]S. Minton, J. Carbonell, O. Etzioni, C. Knoblock and D Kuokka, "Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System," Proceedings of the Fourth International Workshop on Machine Learning, Irvine, CA, June 1987, pp. 122-133.

[Mitchell86]T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," Machine Learning I, 1 (January 1986), pp. 47-80.

[Mooney87]R. J. Mooney, "A General Mechanism for Explanation-Based Learning and Its Application to Narrative Understanding," PhD. Thesis , Department of Computer Science, University of Illinois, Urbana, IL, 1987.

[Mostow87]J. Mostow and N. Bhatnager, "Failsafe-A Floor Planner that Uses EBG to Learn from its Failures," Proceedings of the Tenth International Conference on Artificial Intelligence, Milan, Italy, August 1987.

[Rajamoney87]S. A. Rajamoney and G. F. DeJong, "The Classification, Detection and Handling of Imperfect Theory Problems," Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Milan, Italy, August 1987.

[Schank82]R. C. Schank, Dynamic Memory, Cambridge University Press, Cambridge, England, 1982.

[Shoham86]Y. Shoham, "Reasoning about Change: Time and Causation from the Standpoint of Artificial Intelligence," PhD. Thesis, Yale University, Dept. of Computer Science, New Haven, CT, 1986.

[Tadcpalli86]P. Tadepalli, "Learning in Intractable Domains,"in Machine Learning: A Guide to Current Research, T. Mitchell (ed.), Kluwer Academic Publishers, Hingham, MA, 1986, pp. 337-341.

[Unruh87] A. Unruh, P. Rosenbloom and J. Laird, "Dynamic abstraction problem-solving in Soar," Proceedings of the AOGIAAAIC Joint Conference, Dayton, OH, 1987.