Learning by Experimentation: The Operator Refinement Method

Article · August 1996

Source: CiteSeer

CITATIONS
54

READS
32

2 authors, including:

Jaime G. Carbonell
Carnegie Mellon University
439 PUBLICATIONS **20,123** CITATIONS

SEE PROFILE

Learning by Experimentation: The Operator Refinement Method

Jaime G. Carbonell and Yolanda Gil
Computer Science Department
Carnegie-Mellon University
Pittsburgh PA, 15213, USA

19 July 1996

Abstract

Autonomous systems require the ability to plan effective courses of action under potentially uncertain or unpredictable contingencies. Planning requires knowledge of the environment that is accurate enough to allow reasoning about actions. If the environment is too complex or very dynamic, goal-driven learning with reactive feedback becomes a necessity. This chapter addresses the issue of learning by experimentation as an integral component of PRODIGY. PRODIGY is a flexible planning system that encodes its domain knowledge as declarative operators, and applies the *operator refinement method* to acquire additional preconditions or postconditions when observed consequences diverge from internal expectations. When multiple explanations for the observed divergence are consistent with the existing domain knowledge, experiments to discriminate among these explanations are generated. The experimentation process isolates the deficient operator and inserts the discriminant condition or unforeseen side-effect to avoid similar impasses in future planning. Thus, experimentation is demand-driven and exploits both the internal state of the planner and any external feedback received. A detailed example of integrated experiment formulation in presented as the basis for a systematic approach to extending an incomplete domain theory or correcting a potentially inaccurate one.[1]

---

[1]In *Machine Learning: An Artificial Intelligence Approach*, Volume III, Michalski, R. S. and Kodratoff, Y. (Eds.), Morgan Kaufmann, 1990. This research was sponsored in part by the Defense Advanced Research Projects Agency (DOD), ARPA order No. 4976, monitored by the Air Force Avionics Laboratory under contract F33615-84-K-1520, in part by the Office of Naval Research under contract N00014-84-K-0345, and in part by a gift from the Hughes Corporation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, AFOSR, ONR, or the US government. The authors would like to acknowledge other past and present members of the PRODIGY project at CMU: Daniel Borrajo, Oren Etzioni, Robert Joseph, Craig Knoblock, Dan Kuokka, Steve Minton, Henrik Nordin, Alicia Perez, Santiago Rementeria, Hiroshi Tsuji, and Manuela Veloso, and the help of Dan Kahn, Michael Miller and Ellen Riloff in implementing the PRODIGY system.

1. Introduction: The Need for Reactive Experimentation

Learning in the context of problem solving can occur in multiple ways, ranging from macro-operator formation [Fikes 71, Cheng and Carbonell 86] and generalized chunking [Laird *et al.* 86], to analogical transfer of problem solving strategies [Carbonell 83, Carbonell, 1986 86] and pure analytical or explanation-driven techniques [Mitchell *et al* 86, DeJong and Mooney 86, Minton & Carbonell 87]. All of these techniques, however, focus on the acquisition of control knowledge to solve problems faster, more effectively, and to avoid pitfalls encountered in similar situations. Newly acquired control knowledge may be encoded as preferred operator sequences (chunks and macrooperators), improved heuristic left-hand sides on problem solving operators (as in LEX [Mitchell *et al* 83]), or explicit search-control rules (as in PRODIGY [Minton87a 87, --- 89]).

However important the acquisition of search control knowledge may be, the problem of acquiring factual domain knowledge and representing it effectively for problem solving is of at least equal significance. Most systems that acquire new factual knowledge do so by some form of inductive generalization[2], but operate independently of a goal-driven problem solver, and have no means of proactive interaction with an external environment.

When one observes real-world learners, ranging from children at play to scientists at work, it appears that active experimentation plays a crucial role in formulating and extending domain theories, whether everyday "naive" ones, or formal scientific ones. Many actions are taken in order to gather information and learn whether or not predicted results come to pass, or unforeseen consequences occur. Experimentation is a powerful tool to gather knowledge about the environment, both about properties of objects and about actions.

In general, experimentation may be targeted at the acquisition of different kinds of knowledge:

- ***Experimentation to augment an incomplete domain theory***. Experiments may be formulated to synthesize new operators, learn new consequences of existing operators, refine the applicability conditions of existing operators, or determine previously unknown interactions among different operators. Also, performing known actions on new objects in the task domain in a systematic manner, and observing their consequences, serves to acquire properties of these new objects and classify them according to pragmatic criteria determined by the task domain. Thus, experimentation may be guided towards acquiring new domain knowledge from the external environment.
- ***Experimentation to refine an incorrect domain theory***. No comprehensive empirical theory is ever perfect, as the history of science informs us, whether it be Newton's laws of motion or more ill-structured domain theories embedded in the knowledge bases of expert systems. However, partially correct theories often prove useful, and are gradually improved to match external reality (although they may be totally replaced on occasion by a newer conceptual structure). Here we deal only with minor errors of commission in the domain theory, which when locally corrected improve global performance. We believe automated knowledge refinement is a very important aspect of autonomous learning, and one where success is potentially much closer at hand than the far more difficult and seldomly encountered phenomenon of formulating radically new theories from ground zero. Thus, experimentation may be guided at incremental correction of a domain theory.
- ***Experimentation to acquire control knowledge in an otherwise intractable domain theory***. When multiple sequences of actions appear to achieve the same goal, experimentation and analysis are required to determine which actions to take in formulating the most cost-effective or robust plan, and to generalize and compile the appropriate conditions so as to formulate the preferred plan directly in future problem solving instances where the same goal and relevant initial conditions are present. Thus, experimentation may be guided towards making more effective use of existing domain knowledge.

---

[2]The reader is referred to the two previous machine learning books [Michalski, Carbonell and Mitchell 83, Michalski, Carbonell and Mitchell 86] and other chapters of this book for several good examples of inductive methodologies and systems built upon them.

- ***Experimentation to acquire or correct knowledge about the external state of the world***. Given a partial description of the external world[3], it often proves necessary to acquire a missing piece of knowledge in order to synthesize or elaborate a plan -- regardless of the accuracy, tractability, or completeness of the domain theory itself. This kind of observation (or "exploration" or "experimentation") is a common data-seeking behavior prototyped in systems such as MAX [--- 90], but missing from most idealized work on planning, including all theoretical treatments of non-linear constraint-based planning.

We have investigated a series of methods for learning by experimentation in the context of planning, that can yield factual knowledge, as well as search control preferences. The work described here is a method for refining the specifications of operators, and it has been implemented in a version of the PRODIGY system augmented with capabilities for execution monitoring and dynamic replanning.

2. The Role of Experimentation in PRODIGY

The PRODIGY system [Minton *et al.* 89, --- 89, Carbonell *et al.* 90] is a general purpose problem solver designed to provide an underlying basis for machine learning research. The appendix presents an overview of the basic architecture and the different learning mechanisms in the system. PRODIGY can improve its performance, by learning search control rules [Minton88 88, Etzioni 90], by storing and replaying derivational traces in an analogy/case-based reasoning mode [Veloso and Carbonell 89], by learning useful abstractions for hierarchical planning [Knoblock 89], and by acquiring knowledge from domain experts via graphically-oriented static and dynamic knowledge acquisition interfaces [Joseph 89]. Our work is focused on the acquisition of the domain theory through external feedback from targeted actions: execution monitoring of plans as they unfold and targeted experiments to resolve apparent indeterminacies in the environment.

Of the possible imperfections in a domain theory described in the previous section, we focus our work on the refinement of incomplete theories. The specification of a domain can be incomplete in several different ways:

- Attributes of objects in the world could be unknown – factual properties could be missing (size, color, category, functional properties, etc.) or even knowledge about to which objects the operators may be applied to achieve the desired effects. Totally new attributes could be learned, or the range of already known ones could be further specified. Additionally, attributes of objects can be combined to form new attributes. For example, density and volume under constant gravity define the attribute "weight." Inference rules can also define new attributes expressing more complex relations.
- Entire operators could be missing – the planner may not know all the capabilities of the performance component.
- Operators could be partially specified – the planner may know only some of their preconditions or some of their consequences.
- Interactions among operators could be unknown, causing planning failures or planning inefficiencies.

Our goal is to develop learning methods and experimentation strategies to acquire missing domain knowledge in general. This paper focuses on one central approach, the *operator refinement method*, to acquire missing pre and post conditions of operators in the domain theory. In a forthcoming paper [Carbonell *et al.* ng], we describe other techniques for learning by experimentation in the context of problem solving that address other types of incompleteness in the domain theory.

---

[3]All descriptions of a real robotic environment, for instance, are necessarily partial -- as are all computational models of a complex external reality.

We first present a detailed implemented example of the operator refinement method in action. Then, we describe the method itself more formally.

2.1. The Operator Refinement Method: A Detailed Example

Consider an example domain of expertise: crafting a primary telescope mirror from raw materials (such as pyrex glass, pure aluminum, distilled water, etc.) and pertinent tools (such as grinding equipment, aluminum vaporizers,[4] etc.). A telescope mirror will be considered here as a reflective and polished surface that has a parabolic shape. The operators in the domain include: GRIND-CONCAVE, POLISH, ALUMINIZE, and CLEAN and are presented in detail in figure 2-1 with the inference rules. As we will see through the example, this is an imcomplete specification of the domain.

OPERATORS

```
(GRIND-CONCAVE
  (params (<obj>))
  (preconds
      (is-solid <obj>))
  (effects (
      (add (is-parabolic <obj>)))))

(POLISH
  (params (<obj>))
  (preconds
      (and (is-clean <obj>)
           (is-glass <obj>)))
  (effects (
      (add (is-polished <obj>)))))

(CLEAN
  (params (<obj>))
  (preconds
      (is-solid <obj>))
  (effects (
      (add (is-clean <obj>)))))

(ALUMINIZE
  (params (<obj>))
  (preconds
      (and (is-clean <obj>)
           (is-solid <obj>)))
  (effects (
      (add (is-reflective <obj>)))))
```

INFERENCE RULES

```
(IS-TELESCOPE-MIRROR
  (params (<obj>))
  (preconds
      (and (is-mirror <obj>)
           (is-parabolic <obj>)))
  (effects (
     (add (is-telescope-mirror <obj>)))))

(IS-MIRROR
  (params (<obj>))
  (preconds
      (and (is-reflective <obj>)
           (is-polished <obj>)))
  (effects (
      (add (is-mirror <obj>)))))
```

Figure 2-1: Incomplete domain theory, as given initially to the system.

Let us suppose that the goal of producing a telescope mirror arises, and we have glass blanks and a wood pieces to work with, none of them with clean or polished surfaces. PRODIGY starts backchaining by matching the goal state against the right hand side of operators and inference rules, concluding that in order to make a telescope mirror it

[4]Aluminum is placed on the primary reflecting surface of a glass mirror blank by placing the blank in a vacuum chamber and passing a strong current through a thin pure aluminum strip, which then vaporizes and is deposited evenly, several molecules thick, on the glass surface to produce optical-quality mirrors. For simplicity in our discussion, these details of the aluminizing process are suppressed, as are internal details of the grinding and polishing processes. Hence, though the domain we have chose is very much a real one, we discuss it at suitable level of abstraction and simplification.

should first make a mirror, and then make its shape parabolic. Then seeing how to make a mirror, it concludes that it should make it reflective and then polish it (by matching IS-MIRROR against the right hand side of the second inference rule). Let us assume for now that PRODIGY correctly selected the glass blank (it was listed first) as the starting object. Now it must apply the operator ALUMINIZE to the glass, which requires that it be a solid, and that it be clean. The first precondition is satisfied (glass is a solid), and the second one requires applying the CLEAN operator, which succeeds because any solid thing may be cleaned. These conditions enable the ALUMINIZE operator to apply successfully, and go on to the next goal in the conjunctive subgoal set: IS-POLISHED. Thus far (as shown in figure 2-2), there have been no surprises and no learning, just locally successful performance.

However, whereas PRODIGY believed that the POLISH operator preconditions were satisfied (it believes in temporal persistence of states, such as IS-CLEAN, unless it learns otherwise), the environment states the contrary: the glass was not clean. The first learning step occurs in the attribution of this state change (the glass becoming dirty again) to one of the actions that occurred since the state IS-CLEAN was brought about. Since there was only one intervening operator invocation (ALUMINIZE), it infers that a previously unknown consequence of this operator is ~IS-CLEAN (meaning retracting IS-CLEAN from the current state). If there had been many intermediate operators, specific experiments to perform some but not other steps would have been required to isolate the culprit operator. The operator ALUMINIZE is corrected, and PRODIGY tries now to achieve its goal of making a telescope mirror with the new domain knowledge.

Since the glass is dirty, the CLEAN operator is applied once more. It again attempts to POLISH, but the operator does not result in the expected state: IS-POLISHED. This means that either it is missing some knowledge (some other precondition for POLISH is required), or its existing knowledge is incorrect (IS-POLISHED is not a consequence of POLISH). Always preferring to believe its knowledge correct unless forced otherwise, it prefers to examine the former alternative. But, how can it determine what precondition could be missing?

Figure 2-2: Initial planning attempts. PRODIGY learns that
~(IS-CLEAN <obj>) is a new postcondition of ALUMINIZE (from failure 1),
and ~(IS-REFLECTIVE <obj>) is a new precondition of POLISH (from failure 2)

It is time to formulate an experiment: Are there other objects on which it could attempt the POLISH operation?

The only possibilities are un-aluminized dirty glass blanks and dirty wood blanks. Only glass can be polished (see the preconditions of POLISH), and all the glass blanks are identical to each other, but different from the current object in that they are both dirty and unaluminized, so it chooses a glass blank. After cleaning it, the POLISH operator succeeds, and PRODIGY must establish a reason for the operator succeeding this time, but failing earlier: the only difference is the glass not being aluminized. Thus a new precondition for POLISH is learned as a result of a simple directed experiment: ~IS-REFLECTIVE(<OBJ>), meaning that once coated with aluminum, the substrate substance (e.g. the glass) cannot be polished.

Now back to the problem at hand. In order to POLISH the glass it must unaluminize it, but there is no known operator that removes aluminum (see figure 2-2).[5] So the IS-POLISHED subgoal fails, and failure propagates to the IS-MIRROR subgoal, with the cause of failure being that the IS-REFLECTIVE prevented POLISH from applying. Here there is a goal interaction[6] that can be solved by reordering the interacting components:

> *If the cause of failure of one conjunctive subgoal is a consequence of an operator in an earlier subgoal in the same conjunctive set, try reordering the subgoals.*

That heuristic succeeds by POLISHing before ALUMINIZing. Having obtained success in one ordering and failure in another, the system tries to prove to itself that this ordering is always required, and succeeds by constructing the proof: ALUMINIZE will always produce IS-REFLECTIVE which blocks POLISH, and since there are no other known ways to achieve IS-POLISHED, failure is guaranteed. The present version of PRODIGY is capable of producing such proofs in failure-driven EBL mode [Minton & Carbonell 87]. Thus, a goal-ordering control rule is acquired for this domain: always choose POLISH before ALUMINIZE, if both are in the same conjunctive goal set and both apply to the same object.

Figure 2-3: Second planning attempt: new postconditions of GRIND-CONCAVE are learned: ~(IS-REFLECTIVE <obj>) and ~(IS-POLISHED <obj>)

[5] If its domain knowledge were greater, it would know that grinding removes aluminum and well as changing shape and removing surface polish. In fact, this knowledge is acquired later in the example, as an unfortunate side effect of attempting to make a flat mirror into a parabolic one by grinding it.

[6] Sussman would call it a "clobber-brother-subgoal" interaction in HACKER [Sussman 73].

PRODIGY tries again to produce a telescope mirror. The system succeeds in producing a mirror, but now needs to make it parabolic. The only operator to make IS-PARABOLIC true is GRIND-CONCAVE. Its only precondition is that the object be solid, and so it applies. At this point the system checks whether it finally has achieved the top-level goal IS-TELESCOPE-MIRROR, and discovers (much to its dismay, were it capable of emotions), that all its work polishing and aluminizing has disappeared (see figure 2-3). The only operator that applied since the mirror was polished and aluminized was GRIND-CONCAVE, and so it learns two new consequences for GRIND-CONCAVE: ~IS-POLISHED and ~IS-REFLECTIVE. No explicit experiment was needed as only one operator (GRIND-CONCAVE) could have caused those changes.

At this point PRODIGY spawns off the subgoal to make the parabolic glass back into a mirror, using all it learned earlier (POLISH before ALUMINIZE, etc.) to produce the plan more efficiently. Finally, the top level goal of IS-TELESCOPE-MIRROR is achieved (see figure 2-4).

Figure 2-4: Final search tree after learning

The learning system, however, is seldom quiescent, and though global success was achieved, some states (IS-MIRROR, IS-REFLECTIVE, IS-POLISHED, IS-CLEAN) had to be achieved multiple times. Retrospective examination of the less-than-optimal solution suggests that another goal reordering heuristic applies:

> *If a result of a subgoal was undone when pursuing a later subgoal in the same conjunctive set, try reordering these two subgoals.*

So, PRODIGY goes off and tries the experiment of achieving IS-PARABOLIC before achieving IS-MIRROR, resulting in a more efficient plan.[7] A proof process is again invoked to determine whether to make it a reordering rule, concluding that it is always better to achieve IS-PARABOLIC first.

Figure 2-5 summarizes the new knowledge acquired (in italics) as a result of the problem solving episodes,

[7]In general we are measuring relative efficiency by requiring fewer total steps and no repeated subgoals. In the instance case we have a stronger condition: the leaf-node actions of the more efficient plan constitute a proper subset of the leaf-node actions of the previous less efficient plan.

experiments, and proofs. Such is the process of fleshing out incomplete domain and control knowledge through experience and focused interaction with the task environment. We present now a formal description of the method used through this example.

OPERATORS

```
(GRIND-CONCAVE
  (params (<obj>))
  (preconds
      (is-solid <obj>))
  (effects (
      (add (is-parabolic <obj>))
      (del (is-planar <obj>))
      (del (is-reflective <obj>))
      (del (is-polished <obj>)))))

(POLISH
  (params (<obj>))
  (preconds
      (and (is-clean <obj>)
           (is-glass <obj>))
           (~ (is-reflective <obj>)))
  (effects (
      (add (is-polished <obj>)))))

(CLEAN
  (params (<obj>))
  (preconds
      (is-solid <obj>))
  (effects (
      (add (is-clean <obj>)))))

(ALUMINIZE
  (params (<obj>))
  (preconds
      (and (is-clean <obj>)
           (is-solid <obj>)))
(effects (
      (add (is-reflective <obj>))
      (del (is-clean <obj>)))))
```

INFERENCE RULES

```
(IS-TELESCOPE-MIRROR
  (params (<obj>))
  (preconds
      (and (is-mirror <obj>)
           (is-parabolic <obj>)))
  (effects (
     (add (is-telescope-mirror <obj>)))))

(IS-MIRROR
  (params (<obj>))
  (preconds
      (and (is-reflective <obj>)
           (is-polished <obj>)))
  (effects (
      (add (is-mirror <obj>)))))
```

LEARNED CONTROL RULES

Select IS-POLISHED(<obj>) before IS-REFLECTIVE(<obj>) if both are present in the same conjunctive subgoal set.

Select IS-PARABOLIC(<obj>) before IS-MIRROR(<obj>) if both are present in the same conjunctive subgoal set.

Figure 2-5: Complete domain theory after experimentation. Items in italics denote new knowledge acquired through the operator refinement method.

2.2. The Operator Refinement Method

In the current implementation, PRODIGY continually monitors the outside world for external compliance when operator preconditions are matched in the internal state, and when new effects (adds and deletes) are asserted upon operator application. In doing this, for each precondition or efffect **P** we obtain a value of the predicate **Consistent(World, State, P)** as follows:

Consistent(World, State, P)
if P is satisfied in both the internal state and the external world
or P is not satisfied in either the internal state or external world
then True
else False

The predicate **Consistent** is false whenever **P** is satisfied in either **world** or **state** but not in the other, signifying a discrepancy between internal belief and external reality.

PRODIGY applies an operator **O** only after establishing that all its preconditions are satisfied in the internal state. If these are verified in the external world, planning proceeds normally, but if not, it attempts to extend the domain theory as follows:

For every operator O selected

for every precondition P of operator O
if Consistent(World, State, P)
then continue planning

if NOT(Consistent(World, State, P))
then one of the operators applied after P was established has a previously unknown postcondition. CASE 1

1) Select candidate operators. The candidate set consists of all operators applied since the consistency of P was last checked.
2) Identify responsible operator. Formulate experiments by selecting an operator in a binary search over the ordered candidate set, applying it and then checking P in the World. If as a result of an experiment with operator O_E, P is unexpectedly changed in the World, then O_E is incompletely specified.
3) Add P as a new postcondition of operator O_E.

This case corresponds to the first discrepancy discussed in our example: the planner's internal state contained the belief that the glass was clean, while in reality it was not. PRODIGY learned that ALUMINIZE should delete the literal IS-CLEAN from the internal state.

Whenever an operator **O** is applied, PRODIGY verifies that its postconditions have been realized in the external world. If not, the domain theory is refined as follows:

for every postcondition P of operator O
if Consistent(World, State, P)
then continue planning

if NOT(Consistent(World, State, P))
then

if ∃ Q precondition of O such that NOT(Consistent(World, State, Q))
then one of the operators applied after Q was established
should have had a postcondition affecting Q.] CASE 2

1) Select candidate operators. The candidate set consists of all operators applied since the consistency of Q was last checked.
2) Identify responsible operators. Formulate experiments by selecting an operator. Each experiment will consist of applying one of the operators and checking Q in the World. If as a result of an experiment with operator O_E Q is unexpectedly changed in the World, O_E is incompletely specified.
3) Add Q as a new postcondition of operator O_E.

if ∀ preconditions Q of O Consistent(World, State, Q)
then a precondition of operator O might be missing. CASE 3

1) Select candidate preconditions. The candidate set $\Delta(S_{old},S_{current})$ is formed by calculating all the differences between the most similar earlier state in the previous problem solving history in which O was applied successfully S_{old} and the current state $S_{current}$ (an unsuccessful application of O).
2) Identify missing precondition. Formulate experiments using a binary search over $\Delta(S_{old},S_{current})$ by generating new state $S_{experiment}$ which CONTAINS half of the differences between S_{old} and $S_{current}$ and determining whether O produces the desired effect. If so, continue the binary search over that half of of $\Delta(S_{old},S_{current})$, and if not over the other half until only one condition R is left in the Δ set.
3) Add R as a new precondition of operator O.

Case 2 corresponds to the last situation described in our example. After applying GRIND-CONCAVE the planner assumed that the glass was still a mirror (i.e., that it was still REFLECTIVE and POLISHED). Since the external world did not confirm this expectation, the planner acquired the previously unknown consequences of the grinding operator.

Case 3 also occurred in our example. When the system applied the operator POLISH, its effects were not realized in the external world. The method hypothesizes that a precondition must be missing from the operator. Through experimentation, the new precondition is found, and the hypothesis is confirmed.

Although in the example all the learned preconditions are negated predicates (absence tests) and the new consequences are deletions from the current state, the same basic process applies to acquiring non-negated preconditions and consequences that add assertions to the state.

In addition the system used the following heuristics for cases of goal interaction and plan optimization:

If the cause of failure of one conjunctive subgoal is a consequence of an operator in an earlier subgoal in the same conjunctive set, try reordering the subgoals.

If a result of a subgoal was undone when pursuing a later subgoal in the same conjunctive set, try reordering these two subgoals.

The method for acquiring the missing pre and post conditions of operators are summarized in the table below. In essence, plan execution failures trigger the experimentation and replanning process. Thus, each method is indexed by the failure condition to which it applies, encoded as differences between expected and observed outcomes. The first two cases are the focus of the current chapter. A forthcoming paper [Carbonell *et al.* ng] expands the method to address the last case on the table.

EXPECTED OUTCOME	OBSERVED BEHAVIOR	RECOVERY STRATEGY	LEARNING METHOD (EXPERIMENT GENERATOR)
all the known preconditions satisfied earlier	at least one precondition is violated at present	plan to achieve the missing precondition	binary search on operator sequence from establishment of precondition to present, adding negated precondition as postcondition of the culprit operator
all the known preconditions satisfied earlier	all the known preconditions satisfied but operator fails to apply; postconditions remain undone	attempt to plan without this operator, or failing that, suspend plan till the experiment is complete	compare present failure to the last time operator applied successfully, generating in a binary search intermediate world descriptions to identify the necessary part of the state, adding it to the operator preconditions
operator applies and all the postconditions are satisfied	at least one postcondition fails to be satisfied	if the unmet postcondition is incidental ignore it, but if it is a goal state try different operator(s)	compare to last time all postconditions were met, perform binary search on world state to determine necessary part to achieve all postconditions - then replace operator with two new ones: one with the new precondition and all the postconditions, the other with the new precondition negated and without the postcondition in question

Operator refinement is always applied in an active planning context: there is a goal, a state and a (partially) formulated plan. We are not modeling idle curiosity. Thus, we characterize our work as purposeful and task-driven experimentation. Experiments are always directed at overcoming a current impasse in the planning processes.

3. Related Work

Experimentation techniques have been used in recent work on various areas of Machine Learning, including learning from examples [Gross 88] and discovery programs [Langley *et al* 87, Nordhausen *et al*. 89]. Kulkarni and Simon [Kulkarni and Simon 89, Kulkarni 88] developed a system called KEKADA that simulates the reasoning process followed by scientists when they encounter surprising phenomena. In essence, they developed a set of heuristics to propose experiments to confirm, magnify and elaborate the extent of a previously unexpected observation.

In similar spirit to the work reported here, Rajamoney focused on on the problem of refining incorrect theories of qualitative physics in the ADEPT system [Rajamoney 86]. When a contradiction arises in the process of explaining an observation, ADEPT proposes hypotheses, and experimentation is used to confirm or reject a single hypothesis at a time. Several kinds of experiments are proposed to test these hypotheses. In COAST [Rajamoney 88], experimentation-based hypothesis refutation is also used to revise an incorrect theory. Experiments are designed using the predictions made by the current hypothesized theory, and their results are used to reject possible theories. Rajamoney proposes four dimensions to evaluate the design of experiments: efficacy, efficiency, tolerance in the presence of unavailable data, and feasibility.

In contrast with these systems, our work is focused on learning by experimentation to improve the domain theory of a planning system, and more specifically to overcome impasses when external reality differs from planning expectations. The LIVE system, by Shen and Simon [Shen and Simon 89, Shen 89] shares some of our objectives. LIVE acquires new operators and refines old ones by interacting with the environment in order to formulate indirectly observable features of objects in the domain, and uses these features in creating new preconditions to split overgeneral operators. This method differs from our work in several ways. First, in order to gather information about the world, it checks every instantiated predicate that is known to the system. In our system, the only predicates that are attended to in the external world are those that the planner checks or changes in the internal state. We consider this a more practical approach to larger domains. Second, the definition of new features causes a real overload for the system, since it must find out the value of every new feature for every object in order to apply its operators. Nevertheless, this capability for defining new features gives the system the capability to acquire a more powerful language to express the domain knowledge. Finally, LIVE keeps no history of its past behavior, retaining only the current set of operators, objects and features.

There is a significant amount of work on recovery from planning failures, both in the context of case-based reasoning and of reactive systems ([Hammond 89, Schoppers 87, Georgeff and Lansky 87, Kaelbling 86], and others). However our work is more focused on the techniques for learning from these failures rather than the process of plan recovery itself.

There are a number of systems that use different techniques to learn in the context of planning and interacting with an external environment. Robo-Soar [Laird *et al.* 89] is a system implemented in Soar that learns control knowledge from outside guidance. The Theo-Agent [Blythe and Mitchell 89] is an autonomous robot that starts out building plans to solve new problems and learns rules that allow it to have a reactive behavior.

4. Discussion and Further Work

The operators in the domains that we have used to test out methods are expressed using only conjunction and negation. Further work should expand these techniques to learn more complex expressions of the preconditions of operators. The method was also described assuming that only one condition is acquired in each learning episode. To increase efficiency, we are currently extending it to consider cases where the experimentation phase can find several unknown preconditions or postconditions.

More comprehensive learning could occur by attempting to generalize the newly acquired preconditions and consequences to other sibling operators in the operator hierarchy (see figure 4-1). For instance, the newly learned consequences of destroying a polished or aluminized surface apply not just to GRIND-CONCAVE, but to any GRIND operation (such as GRIND-CONVEX, GRIND-PLANAR). However, these consequences do not apply to other RESHAPE operations such as BEND, COMPRESS, etc. The process to determine the appropriate level of generalization again requires experimentation (or asking focused questions to a human expert). For instance, observing the consequences of GRIND-PLANAR on a previously aluminized mirror, provides evidence that all GRINDs behave alike with respect to destroying surface attributes, and observing the consequences of bending a polished reflective glass tube without adverse effects on surface attributes prevents generalization above GRIND.

Figure 4-1: Fragment of operator "isa" hierarchy

In addition to proposing experiments to guide generalization, we are starting to investigate tradeoffs between experimentation and resource consumption (minimizing the latter, while maximizing the information gained from the former), and tradeoffs between experimentation and other goals such jeopardizing safety of the robot or person conducting the experiment. The entire planning context can be used to formulate and guide the experiment, in order to focus on the most direct and economical way of inferring the missing knowledge. Thus, experiment formulation, once invoked with the appropriate constraints, becomes itself a meta-problem amenable to all the methods in the general purpose planner. The EBL method (or perhaps a similarity-based method – SBL) may then be invoked to retain not just the result of the instance experiment, but its provably correct generalization (or empirically appropriate one if SBL is used).

The experimentation methods discussed here focused only on operator refinement (both preconditions and consequences), but not on acquiring new operators, new features of the state or domain, or new meta-level control

structures. In a forthcoming paper [Carbonell *et al.* ng], we present other techniques for learning by experimentation in the context of problem solving. That paper describes how PRODIGY can acquire knowledge about the state, such as values of attributes not known when needed to expand the current plan. Another method allows the systen to learn multiple more specific versions of overgeneral operators that failed to predict outcomes consistently.

The methods described here and in [Carbonell *et al.* ng] apply when PRODIGY is given a correct but incomplete domain theory, and learning is always incremental: the initial knowledge is monotonically augmented. This metaprinciple of "cognitive inertia" dictates that monotonic changes (adding new information) be preferred over non-monotonic ones (changing previous information), so long as no overt inconsistencies are discovered. Further work should address the problem of modifying an incorrect domain theory.

In order to avoid the complexities of full interleaving of planning and execution, we constructed an expository domain where environmental feedback can be provided by a domain-knowledgeable user (that answers only yes-or-no questions about the state of the external environment), and one where the search space is of manageable size (e.g., there are no difficult decision points with multiple applicable operators). Moreover, we assume environmental feedback is correct and not deterministic. Clearly, not every domain permits such a limited manipulation and interchange of information as the one used to describe the *operator refinement method*. In other work on experimentation we connected PRODIGY to a full 3D newtonian kinematics robotic simulator [Carbonell and Hood 86, Carbonell *et al.* 89] for more realistic environmental feedback [Carbonell *et al.* ng]. The MAX system (a PRODIGY progeny) exhibits a richer communication channel [--- 90]. Finally, we assume that the environment is only affected by the actions of our system. There are no environmental changes unless PRODIGY produces them, although of course PRODIGY is not always aware of all the changes that each of its actions may produce.

Our ultimate aim is to develop a set of general techniques for an AI system to acquire knowledge of its task domain systematically under its own initiative, starting from a partial domain theory and little if any *a-priori* control knowledge. The impact of this work should be felt in robotic and other autonomous planning domains, as well as in expert systems that must deal with a potentially changing environment of which they cannot possibly have complete and accurate knowledge beforehand. The *operator refinement method* is but the first step in this long term endeavor.

5. References

[Blythe and Mitchell 89] Blythe, J. and Mitchell, T. M., "On Becoming Reactive," in *Proceedings of the Sixth International Workshop on Machine Learning*, Morgan Kaufmann, Ithaca, New York, 1989.

[Carbonell 83] Carbonell, J. G., "Learning by Analogy: Formulating and Generalizing Plans from Past Experience," in R. S. Michalski, J. G. Carbonell and T. M. Mitchell (eds.), *Machine Learning, An Artificial Intelligence Approach*, Tioga Press, Palo Alto, CA, 1983.

[Carbonell *et al.* 89] Carbonell, J., Gross, K., Hood, G., Shell, P., and Tallis, H., *The World Modeling System User Guide*, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Technical Report, 1989. Internal document

[Carbonell *et al.* 90] Carbonell, J. G., Knoblock, C. A., and Minton, S., "PRODIGY: An Integrated Architecture for Planning and Learning," in Kurt VanLehn (ed.), *Architectures for Intelligence*, Erlbaum, Hillsdale, NJ, 1990.

[Carbonell *et al.* ng] Carbonell, J. G., Gil, Y., and Rementeria, S., *Experimentation in PRODIGY: Acquiring Domain and State Knowledge*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Technical Report, Forthcoming.

[Carbonell and Hood 86] Carbonell, J. G. and Hood, G., "The World Modelers Project: Learning in a Reactive Environment," in Mitchell, T. M., Carbonell, J. G. and Michalski, R. S. (eds.), *Machine Learning: A Guide to Current Research*, pp. 29-34, Kluwer Academic Press, 1986.

[Carbonell and Veloso 88] Carbonell, J.G. and Veloso, M.M., "Integrating Derivational Analogy into a General Problem-Solving Architecture," in *Proceedings of the First Workshop on Case-Based Reasoning*, Morgan Kaufmann, Tampa, FL, May 1988.

[Carbonell, 1986 86] Carbonell, J. G., "Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition," in Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (eds.), *Machine Learning, An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, 1986.

[Cheng and Carbonell 86] Cheng, P. W. and Carbonell, J. G., "Inducing Iterative Rules from Experience: The FERMI Experiment," in *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, 1986.

[DeJong and Mooney 86] DeJong, G. F. and Mooney, R., "Explanation-Based Learning: An Alternative View," *Machine Learning* 1, (2), 1986, 145-176.

[Etzioni 90] Etzioni, O., *A Structural Theory of Search Control*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990. In preparation

[Fikes 71] Fikes, R. E. and Nilsson, N. J., "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence* 2, 1971, 189-208.

[Georgeff and Lansky 87] Georgeff, M. P. and Lansky, A. L., "Reactive Reasoning and Planning," in *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA, 1987.

[Gross 88] Gross, K. P., "Incremental Multiple Concept Learning Using Experiments," in *Proceedings of the Fifth International Conference on Machine Learning*, Ann Arbor, MI, 1988.

[Hammond 89] Hammond, K., *Case-based planning: Viewing planning as a Memory Task*, Academic Press, 1989.

[Joseph 89] Joseph, R.L., "Graphical Knowledge Acquisition," in *Proceedings of the 4th Workshop on Knowledge Acquisition For Knowledge-Based Systems*, Banff, Canada, 1989.

[Kaelbling 86] Kaelbling, L., *An Architecture for Intelligent Reactive Systems*, Artificial Intelligence Center, SRI International, Menlo Park, CA, Technical Report Technical Note 400, 1986.

[Knoblock 89] Knoblock, C.A., "Learning Hierarchies of Abstraction Spaces," in *Proceedings of the Sixth International Workshop on Machine Learning*, Morgan Kaufmann, Ithaca, NY, June 1989.

[Kulkarni 88] Kulkarni, D. S., *The Process of Scientific Research: The Strategy of Experimentation*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1988.

[Kulkarni and Simon 89] Kulkarni, D. and Simon, H. A., "The Role of Experimentation in Scientific Theory Revision," in *Proceedings of the Sixth International Workshop on Machine Learning*, Morgan Kaufmann, Ithaca, New York, 1989.

[--- 90] Kuokka, D. R., *The Deliberate Integration of Planning, Execution and Learning*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990.

[Laird *et al.* 86] Laird. J. E., Rosenbloom, P. S., and Newell, A., "Chunking in SOAR: The Anatomy of a General Learning Mechanism," *Machine Learning* 1, (1), 1986, 11-46.

[Laird *et al.* 89] Laird, J. E., Yager, E. S., and Tuck, C. M., "Learning in Tele-autonomous Systems using Soar," in *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, CA, 1989.

[Langley *et al* 87] Langley, P., Simon, H. A., Bradshaw, G. L., and Zytkow, J. M., *Scientific Discovery: Computational Explorations of the Creative Processes*, MIT Press, 1987.

[Michalski, Carbonell and Mitchell 83] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds), *Machine Learning, An Artificial Intelligence Approach*, Tioga Press, Palo Alto, CA, 1983.

[Michalski, Carbonell and Mitchell 86] Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (Eds), *Machine Learning, An Artificial Intelligence Approach, Volume II*, Morgan Kaufmann, Los Altos, CA, 1986.

[Minton & Carbonell 87] Minton, S. and Carbonell, J.G., "Strategies for Learning Search Control Rules: An Explanation-Based Approach," in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Milan, Italy, 1987.

[Minton *et al.* 89] Minton, S., Knoblock, C. A., Kuokka, D. R., Gil, Y., Joseph, R. L., Carbonell, J. G., *PRODIGY 2.0: The Manual and Tutorial*, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, Technical Report CMU-CS-89-146, 1989.

[Minton87a 87] Minton, S., Carbonell, J.G., Etzioni, O, Knoblock, C.A., Kuokka, D.R., "Acquiring Effective Search Control Rules: Explanation-Based Learning in the PRODIGY System," in *Proceedings of the Fourth International Workshop on Machine Learning*, Morgan Kaufmann, Irvine, CA, 1987.

[Minton88 88] Minton, S., *Learning Search Control Knowledge: An Explanation-based Approach*, Kluwer Academic Publishers, Boston, Massachusetts, 1988. Limited availability as Carnegie-Mellon CS Tech. Report CMU-CS-88-133

[Mitchell *et al* 86] Mitchell, T. M., Keller, R. M. and Kedar-Cabelli, S. T., "Explanation-Based Generalization: A Unifying View," *Machine Learning* 1, (1), 1986.

[Mitchell *et al* 83] Mitchell, T. M., Utgoff, P. E. and Banerji, R. B., "Learning by Experimentation: Acquiring and Refining Problem-Solving Heuristics," in R. S. Michalski, J. G. Carbonell and T. M. Mitchell (eds.), *Machine Learning, An Artificial Intelligence Approach*, Tioga Press, Palo Alto, CA, 1983.

[Nordhausen *et al.* 89] Nordhaussen, B. and Langley, P., *An Integrated Approach to Empirical Discovery*, Department of Information and Computer Science, University of California, Irvine, CA, Technical Report 89-20, 1989.

[--- 89] Minton, S., Carbonell, J.G., Knoblock, C.A., Kuokka, D.R., Etzioni, O., and Gil, Y., "Explanation-Based Learning: A Problem-Solving Perspective," *Artificial Intelligence* 40, (1-3), 1989, 63-118.

[Rajamoney 86] Rajamoney, S., *Automated Design of Experiments for Refining Theories*, M. S. Thesis, Department of Computer Science, University of Illinois, Urbana, IL, 1986.

[Rajamoney 88] Rajamoney, S. A., *Explanation-Based Theory Revision: An Approach to the Problems of Incomplete and Incorrect Theories*, Ph.D. thesis, University of Illinois at Urbana-Champaign, 1988.

[Schoppers 87] Schoppers, M. J., "Universal Plans for Reactive Robots in Unpredictable Environments," in *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Milan, Italy, 1987.

[Shen 89] Shen, W. M., *Learning from the Environment Based on Percepts and Actions*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1989.

[Shen and Simon 89] Shen, W. M. and Simon, H. A., "Rule Creation and Rule Learning through Environmental Exploration," in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, Detroit, Michigan, 1989.

[Sussman 73] Sussman, G. J., *A computational model of skill acquisition*, PhD thesis, Massachusetts Institute of Technology, 1973.

[Veloso and Carbonell 89] Veloso, M.M. and Carbonell, J.G., "Learning Analogies by Analogy - The Closed Loop of Memory Organization and Problem Solving," in *Proceedings of the Second Workshop on Case-Based Reasoning*, Morgan Kaufmann, Pensacola, FL, May 1989.

I. The PRODIGY Architecture

PRODIGY is a general problem solver combined with several learning modules. The PRODIGY architecture, in fact, was designed both as a unified testbed for different learning methods and as a general architecture to solve interesting problems in complex task domains. Let us now focus on the architecture itself, as diagrammed in Figure I-1.

Figure I-1: The PRODIGY architecture: multiple learning modules unified by a common representation language and a shared general problem solver

The operator-based problem solver produces a complete search tree, encapsulating all decisions -- right ones and wrong ones -- as well as the final solution. This information is used by each learning component in different ways, including the EBL component, which learns search control rules. In addition to the central problem solver,[8] PRODIGY has the following learning components:

- A user-interface that can participate in an apprentice-like dialogue, enabling the user to evaluate and guide the system's problem solving and learning. The interface is graphic-based and tied directly to the problem solver, so that it can accept advice as it is solving a problem (i.e., coaching) or replay and analyze earlier solution attempts, all-the-while refining the factual or control knowledge.
- An explanation-based learning facility [Minton88 88] for acquiring control rules from a problem-solving trace, as indicated in Figure I-1. Explanations are constructed from an axiomatized theory describing both the domain and relevant aspects of the problem solver's architecture. Then the resulting descriptions are expressed in control rule form, and control rules whose utility in search reduction outweighs their application overhead are retained.
- A method for learning control rules by analyzing PRODIGY's domain descriptions prior to problem

[8]The problem solver is an advanced operator-based planner that includes a simple reason-maintenance system and allows operators to have conditional effects. The problem solver's search (means-ends analysis) is guided by explicit domain-independent and domain-specific control rules. All of PRODIGY's learning modules share the same general problem solver and the same knowledge representation language, PDL.

solving. This investigation has culminated in the STATIC program [Etzioni 90], which produces control rules without utilizing any training examples. STATIC matches EBL's performance on some domains but exhibits one to two orders of magnitude faster learning rate. However, not all problem spaces permit purely static learning, requiring EBL to learn control rules dynamically.

- A derivational analogy engine [Carbonell and Veloso 88, Veloso and Carbonell 89] that is able to replay entire solutions to similar past problems, calling the problem solver recursively to reduce any new subgoals brought about by known differences between the old and new problems. As indicated in Figure I-1, both analogy and EBL are independent mechanisms to acquire domain-specific control knowledge. They coexist in PRODIGY and should be more tightly coupled than in the present architecture.
- A multi-level abstraction planning capability [Knoblock 89]. First, the axiomatized domain knowledge is divided into multiple abstraction layers based on an in-depth analysis of the domain. Then, during problem solving, PRODIGY proceeds to build abstract solutions and refine them by adding back details from the domain, solving new subgoals as they arise. This method is orthogonal to analogy and EBL, in that both can apply at each level of abstraction.
- A learning-by-experimentation module for refining domain knowledge that is incompletely or incorrectly specified (as described in the body of the paper). Experimentation is triggered when plan execution monitoring detects a divergence between internal expectations and external expectations. As indicated in the figure, the main focus of experimentation is to refine the factual domain knowledge, rather than the control knowledge.

The problem solver and EBL component of PRODIGY have been fully implemented, and tested on several task domains including the blocksworld domain, a machine shop scheduling domain, and a 3-D robotics construction domain. The other components, while successfully prototyped, are at various stages of development and implementation.

A more complete description of PRODIGY's architecture can be found in [Minton *et al.* 89].

Table of Contents

List of Figures