LEARNING BY EXPERIMENTATION: THE OPERATOR REFINEMENT METHOD

Techinical Report AIP - 29

Jaime G. Carbonell & Yolanda Gil

Computer Science Department
Carnegie Mellon University
Pittsburgh, Pa. 15213

The Artificial Intelligence and Psychology Project

Departments of
Computer Science and Psychology
Carnegie Mellon University

Learning Research and Development Center
University of Pittsburgh

LEARNING BY EXPERIMENTATION: THE OPERATOR REFINEMENT METHOD

Techinical Report AIP - 29

Jaime G. Carbonell & Yolanda Gil

Computer Science Department
Carnegie Mellon University
Pittsburgh, Pa. 15213

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 contracts N00014-84-K-0345 and Computer Sciences Division, N00014-86K-0678, 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 the other members of the PRODIGY project at CMU: Oren Etzioni, Craig Knoblock, Dan Kuokka, Steve Minton, Henrik Nordin and Ellen Riloff. This is the text of a chapter appearing in *Machine Learning, Volume 3*, R.S. Michalski and Y. Kodratoff editors.

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AIP -29	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Carnegie-Mellon University; 6a. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: Computer Sciences Division, Office of Naval Research (Code 1133)
6c. ADDRESS (City, State, and ZIP Code): Department of Psychology, Pittsburgh, Pennsylvania 15213	7b. ADDRESS (City, State, and ZIP Code): 800 N. Quincy Street, Arlington, Virginia 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: Same as Monitoring Organization; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: N00014-86-K-0678
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO: N/A; PROJECT NO: N/A; TASK NO: N/A; WORK UNIT ACCESSION NO: N/A

11. TITLE (Include Security Classification)
Learning by Experimentation: The Operator Refinement Method

12. PERSONAL AUTHOR(S)
Jaime G. Carbonell and Yolanda Gil

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 86Sept15 TO 91Sept14	87 October 6	19

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Machine learning, planning, experimentation, problem solving.

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Autonomous systems require the ability to plan effective courses of action under potentially uncertain or unpredictable contingencies. Effective planning requires knowledge of the environment, and if the environment is too complex or changes dynamically, goal-driven learning with reactive feedback becomes a necessity. This paper addresses the issue of learning by experimentation as an integral component of PRODIGY, a flexible planning system augmented with capabilities for execution monitoring and dynamic replanning upon receiving adverse feedback. PRODIGY encodes its domain knowledge as declarative operators, and applies the *operator refinement method* to acquire additional preconditions or postconditions for its operators 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. 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 is presented as the basis for a systematic approach to extending an incomplete domain theory or correcting a potentially inaccurate one.

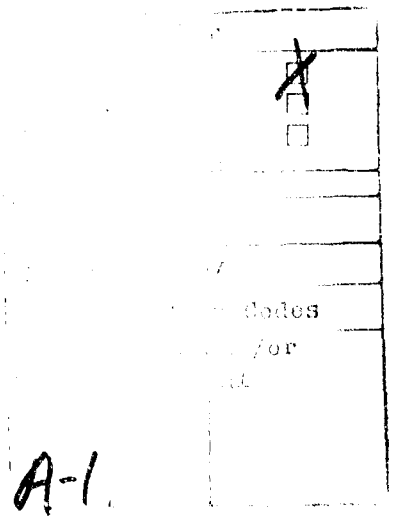
20. DISTRIBUTION / AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Learning by Experimentation: The Operator Refinement Method

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



30 October 1987

Abstract

Autonomous systems require the ability to plan effective courses of action under potentially uncertain or unpredictable contingencies. Effective planning requires knowledge of the environment, and if the environment is too complex or changes dynamically, goal-driven learning with reactive feedback becomes a necessity. This paper addresses the issue of learning by experimentation as an integral component of PRODIGY, a flexible planning system augmented with capabilities for execution monitoring and dynamic replanning upon receiving adverse feedback. PRODIGY encodes its domain knowledge as declarative operators, and applies the *operator refinement method* to acquire additional preconditions or postconditions for its operators 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. 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] 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 contracts N00014-86-K-0678 and Computer Sciences Division, N00014-86K-0678, 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 the other members of the PRODIGY project at CMU: Oren Etzioni, Craig Knoblock, Dan Kuokka, Steve Minton, Henrik Nordin and Ellen Riloff. This is the text of a chapter appearing in *Machine Learning, Volume 3*, R. S. Michalski and Y. Kodratoff editors. Reproduction in whole or in part is permitted for purposes of the US government. Approved for public release, distribution unlimited.

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, 1971, Minton, 1985, Cheng & Carbonell, 1986) and generalized chunking (Laird *et al*, 1986), to analogical transfer of problem solving strategies (Carbonell, 1983, Carbonell, 1986, 1986) and pure analytical or explanation-driven techniques (Mitchell *et al*, 1986, DeJong & Mooney, 1986, Minton & Carbonell, 1987). 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*, 1983)), or explicit search-control rules (as in PRODIGY (Minton *et al*, 1987)).

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 (with the exception of some learning work in robotics and the world modelers project (Carbonell & Hood, 1986)). 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. Of course, experimentation can yield search control preferences, as well as factual knowledge, as we see in our later example. The focus of this chapter is on experiment formulation and analytical interpretation in the context of PRODIGY (Minton & Carbonell, 1987, Minton *et al*, 1987), an interactive planning system, rather than on empirical interpretation of results from pre-formulated experiments in a single-pass learning-by-discovery approach typical of systems such as BACON (Langley *et al*, 1983) and ABACUS (Falkenhainer & Michalski, 1986).

In order to endow a problem solver with the capability to experiment on the external world, we start by interleaving planning and execution monitoring, so that external feedback is immediate. If the plan does not unfold as expected (e.g., unforeseen interactions take place, actions have unexpected consequences, etc.) the system replans dynamically using better-known methods, or suspends planning in order to determine the source of the discrepancy. Here is where experimentation is triggered: divergence from expected results that interfere with carrying out a plan for the active goal. The objective of the experiment is to augment the domain theory (e.g., record previously unknown consequences, after determining what conditions are needed to bring them about), or to correct that domain theory (e.g., deleting or altering the expected effects or applicability conditions of operators, in order to force the internal model to accord with external reality). Experimentation is used to isolate the cause of each discrepancy, and make the minimal modification possible to the internal model in order to establish external consistency. Moreover, this metaprinciple of "cognitive inertia" dictates that monotonic changes (adding new information) be preferred over non-monotonic ones (changing previous information) if both are of equivalent scope.

---

[2]The reader is referred to the two recent machine learning books for several good examples of inductive methodologies and systems built upon them (Michalski, Carbonell & Mitchell, [illegible], Michalski, Carbonell & Mitchell, [illegible]).

2. Other Research in Learning by Experimentation

Machine Learning has not yet addressed centrally the topic of learning by active experimentation, although there has been related work in scientific discovery and more recently some attempts to address directly the issue of experimentation.

The BACON and GLAUBER systems (Langley *et al*, 1986) are able to discover qualitative or quantitative empirical laws, focusing on the empirical interpretation of results from pre-formulated experiments. The authors have proposed combining these systems, having GLAUBER provide BACON with some qualitative information about the data. BACON would then be able to acquire data on its own by formulating experiments. FAHRENHEIT (Koehn & Zytkow, 1986) designs limited experiments in terms of quantitative values of the experiment's parameters to determine the scope of a law given by BACON.

Lenat's AM and EURISKO systems (Lenat, 1983) can be said to experiment, but in a limited sense. Both utilize heuristics that change internal concepts which are then tested for "interestingness", but not necessarily for external validity. In its symbiotic mode, however, EURISKO received feedback from the user (Doug Lenat), and was closer to a full experimentation system.

In LEX, Mitchell uses a limited form of experimentation in to generate problems in symbolic integration that formulate desirability conditions for when to select problem solving operators (Mitchell *et al*, 1983). His primary experiment generation method is to compose a problem that would maximally reduce the version space of possible desirable application conditions for the operator in question.

In some preliminary work, (Langley & Nordhausen, 1986) in the IDS system investigate experimentation in a qualitative physics framework. Also in initial stages of investigation, Kulkarni and Simon (Kulkarni & Simon, 1987) are developing general and domain-dependent heuristics for scientific experimentation, and Shen (Shen, 1987) is developing similar methods for naive experimentation.

The ADEPT system (Rajamoney, 1986) is concerned with experimentation in domains with incomplete or inconsistent theories. The domain knowledge is expressed in terms of qualitative physics. When a contradiction arises in the process of explaining an observation, the system uses a set of beliefs to propose some hypotheses. Several kinds of experiments are proposed to test these hypotheses. The design of an experiment is made following an algorithm that depends on the type of experiment, and that algorithm determines the necessary pieces of information associated with the experiment to be performed. Ultimately the system would design experiments that allow the construction of explanations in EBL with incorrect domain theories.

3. Background: The Role of Experimentation in PRODIGY

The PRODIGY system (Minton *et al*, 1987, Minton & Carbonell, 1987) is a general-purpose planner at CMU that serves as the underlying basis for much machine-learning research. In essence, PRODIGY learns incrementally through experience in solving increasingly more complex problems in a task domain, and gradually transitions from naive student, to apprentice, to journeyman, and eventually (we hope) to domain expert. Thus far we have experimented successfully with a version of explanation-based learning (EBL) (Mitchell *et al*, 1986) that can learn from failed instances (to avoid future failures that share the same underlying cause) and goal interactions, as well as the standard EBL based on deductively provable generalization from positive instances. We are also studying the role of case-based learning in PRODIGY, and are exploring interactive knowledge acquisition from a domain expert who looks over the proverbial shoulder of the planning system, making concrete suggestions on the current plan being synthesized, and occasionally providing more general advice.

Whereas experimentation in its broadest sense can be a very powerful and general learning method, here we confine our study to a very concrete type of experimentation: *operator refinement*. In essence, we assume that the domain knowledge is encoded as a set of declarative operators and inference rules.[3] Presently, learning is confined to the acquisition of new pre and post conditions for existing operators; which start as approximations of external reality and are refined to match that reality whenever discrepancies occur between internal expectations and external observation. Later we hope to extend the method to the acquisition of new domain operators.

Experimentation may be targeted at the acquisition of different kinds of knowledge, though augmentation of an incomplete domain theory (via refinement of operators) is our current focus of attention:

- ***Experimentation to acquire and refine control knowledge.*** When multiple sequences of actions appear to achieve the same goal, experimentation and analysis are required to determine which plan is the most cost-effective or robust one, and to generalize and compile the appropriate conditions so as to formulate the preferred plan in future problem solving instances where the same goal and relevant initial conditions are present. Thus, experimentation may be guided towards producing far more effective use of existing domain knowledge.

- ***Experimentation to augment an incomplete domain theory.*** Experiments may be formulated to synthesize new operators, learn new consequences of existing operators or determine previously unknown interactions among existing 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 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 (and are occasionally totally replaced 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 lea[illegible] not heretofore investigated in AI, and one where success is potentially much closer at hand than th[illegible] and seldomly encountered phenomenon of formulating radically new theories from [illegible] Thus, experimentation may be

[3] An operator has a conjunctive set of [illegible]

guided at incremental correction of a domain theory.[4]

Our central concern is the development of a method to generate operational hypotheses (those that can be tested through an external experiment) to account for unexpected divergence between predicted and observed consequences. Experimentation is invoked when such a divergence prevents the formulation of a plan to solve the problem at hand; thus "idle curiosity" is not our target. Moreover, the entire planning context is used to formulate and guide the experiment, in order to focus on the most direct and economical way of inferring the missing knowledge. Concessions must be made to other protected goals in the course of the experimentation: assuring safety of the experimenter, not consuming a resource in the experiment that will be required to carry out the rest of the plan, etc. 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).

4. The Base-Level System: Knowledge Required for Planning

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,[5] etc.). The operators in the domain include: GRIND-CONCAVE, POLISH, ALUMINIZE, and CLEAN. A complete domain theory would include, in addition to these four operators themselves, knowledge of:

- all the relevant *preconditions* for each operation to proceed successfully,
- all the *consequences* of applying each operator (stated as changes to the global world state),
- and all the *objects* to which these operators may be applied to achieve the desired effects (for instance, wood may be ground into a concave shape, but the result would not be an optical-quality telescope mirror).

In addition to the domain theory, an optimal-performance system needs to know control rules (hard and fast ones, as well as heuristic ones). These rules perform the following tasks:

- When multiple goals are present, determine which goals to work on first – or which ones to work on at all. For instance, if the goals *is-polished* and *is-ground-concave* are both present, it is better to work on the latter first so as not to undo polishing by later grinding. Similarly, if the goal of *reduce-weight* of the glass and *is-ground-concave* are both present, it may prove unnecessary to do more than grind, as that reduces weight as a side effect of grinding away some of the glass in the process of making it concave. Such interactions have been investigated before, albeit if not in a very systematic manner (Sacerdoti, 1977, Carbonell, 1981, Wilensky, 1983). Here we are focusing on an integrated architecture to acquire knowledge of plan interactions through observation of the consequences of its actions on the external environment, and when necessary through focused experimentation.
- When multiple operators may be chosen in order to make progress towards the active goal, determine which one(s) to apply. This is the standard role of a heuristic evaluation function (Nilsson, 1971), but

[4]We note that a totally incorrect theory, requiring wholesale reconceptualization, will not be addressed by our incremental methods. Such a paradigm shift, as Kuhn [illegible] requires a different approach, one along the lines of the more heuristic work of Machine Discovery (Langley et al, 1983, Lenat, 1977).

[5][illegible]

we propose to do the selection by compiling explicit symbolic reasoning, rather than *a-priori* numerical metrics. The notion of learning operator preferences in the context of an active goal was the central task of LEX (Mitchell *et al*, 1983), and is one of the major effects of chunking and universal subgoaling in SOAR (Laird *et al*, 1986). At one end of the spectrum one can view a string of purely deterministic preferences as equivalent to a linear macro-operator (Fikes, 1971, Minton, 1985, Cheng & Carbonell, 1986), and at the other extreme as guiding search in preferential directions based on past experience.

- When multiple objects may be chosen on which to apply the operators, determine which one(s) to select. Again, these can be categorical (polishing and aluminizing the wrong surface of a mirror will never yield desired results) or preferential (choosing a fast rough-grinding tool, vs choosing a slow fine-grinding one, vs choosing both – the former for rough shaping, followed by the latter for fine adjustment). Preferences may be stated in terms of achieving higher quality plans (more efficient ones to execute, or ones more likely to succeed), or in terms of minimizing planning effort (producing a working solution quickly, even if it may be far from an optimal plan).

These decision points serve a dual role in PRODIGY: Learning control rules to make the right decisions (Minton *et al*, 1987), and providing the handle for the experimentation module to direct the problem solver when it must perform actions to seek new knowledge before returning to the problem at hand.

5. Types of Knowledge Acquired

A domain theory of the world can be incomplete in several different senses:

- Factual properties of objects in the world could be missing (size, color, category, functional properties, etc.)
- Entire operators could be missing – the planner may not know all its capabilities.
- Operators could be partially specified – the planner may know only some of their preconditions and some of their consequences.
- Interactions among operators could be unknown, causing planning failures or planning inefficiencies.

Thus far we have worked on operator refinement addressing only the latter two categories of missing knowledge. Learning control knowledge to cope with certain kinds of operator interactions in PRODIGY is discussed in (Minton *et al*, 1987), and illustrated in our detailed example. Our methods for acquiring the missing pre and post conditions of operators are summarized in the table below, and elaborated in the detailed example that follows. 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.

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

6. Learning by Experimentation: A Detailed Example

Let us return to our telescope mirror example, and assume that we have only a partial domain theory and virtually no control knowledge. How can PRODIGY through its attempts to solve the problem learn to plan better the next time? Can learning be improved by formulating subtasks just for the sake of acquiring knowledge, in addition to pursuing externally given tasks? Suppose we start with the following greatly simplified knowledge base:

	OPERATORS	PRECONDITIONS	CONSEQUENCES
1)	GRIND-CONCAVE(<obj>)	ISA(<obj>, solid)	IS-CONCAVE(<obj>)
2)	POLISH(<obj>)	ISA(<obj>, glass) IS-CLEAN(<obj>)	IS-POLISHED(<obj>)
3)	ALUMINIZE(<obj>)	IS-CLEAN(<obj>) ISA(<obj>, solid)	IS-REFLECTIVE(<obj>)
4)	CLEAN(<obj>)	ISA(<obj>, solid)	IS-CLEAN(<obj>)

INFERENCE RULES:

1) IS-REFLECTIVE(<obj>) & IS-POLISHED(<obj>) --> IS-MIRROR(<obj>)

2) IS-MIRROR(<obj>) & IS-CONCAVE(<obj>) --> IS-TELESCOPE-MIRROR(<obj>)

Given the operators and inference rules above, let us suppose that the goal of producing a telescope mirror arises, and we have a 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 should first make a mirror, and then make its shape concave. 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 (see figure 6-1 for the object hierarchy), 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 successes enable the ALUMINIZE operator to apply successfully, and go on to the next goal in the conjunctive subgoal set: IS-POLISHED (see figure 6-2). Thus far there have been no surprises and no learning, just locally successful performance.

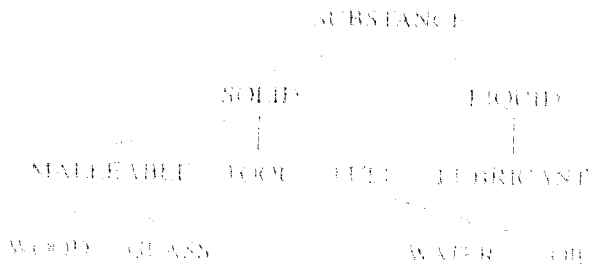
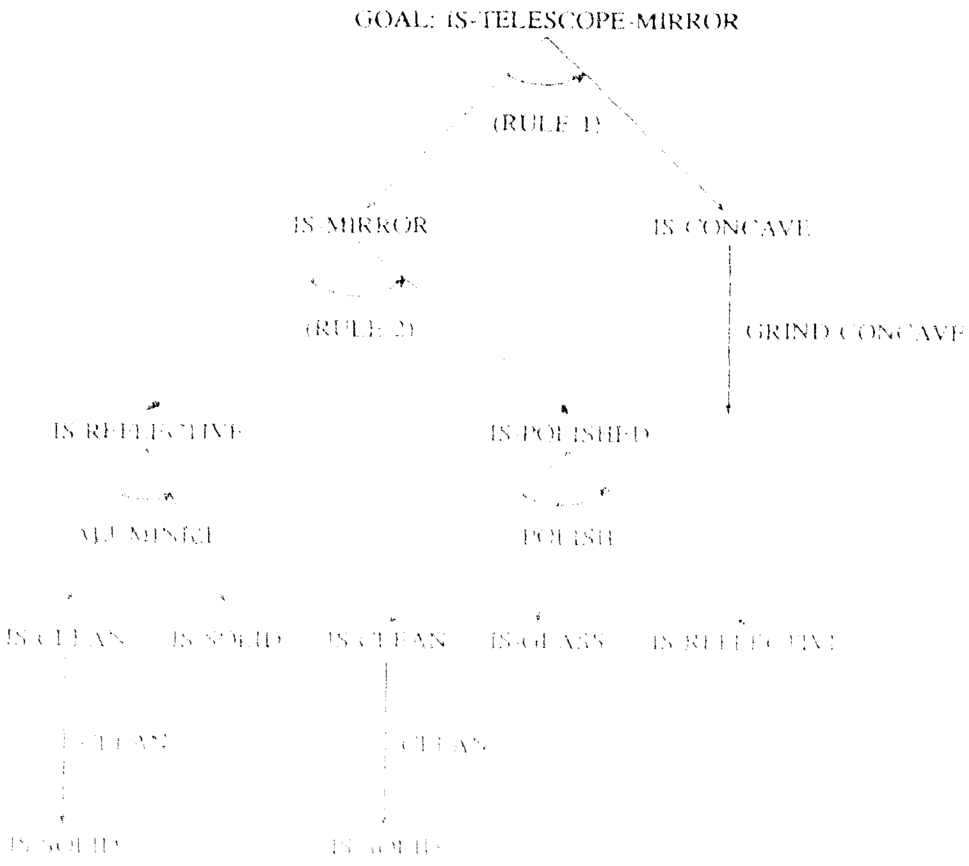


Figure 6-1: Fragment of Prodigy's Object Hierarchy

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 is not clean. The first learning step occurs in the attribution of this state change to one of the actions that occurred since the state IS-CLEANED 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. After applying the CLEAN operator 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?



[illegible]

[illegible]

The only possibilities are un-aluminized dirty glass blanks and dirty wood blanks. Only glass can be polished (see the precondition table), 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 choses a glass blank. After cleaning it, the POLISH operator succeeds, and once again it 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 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.[6] 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[7] 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, 1987). 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.

Now, once again, back to the problem at hand. The system tries again and succeeds in producing a mirror, but now needs to make it concave. The only operator to make IS-CONCAVE 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 on POLISHing and ALUMINIZing has disappeared. 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 would spawn off the subgoal to make the concave glass back into a mirror, and all that it learned when making the flat glass into a mirror applies (POLISH before ALUMINIZE, etc.) producing the plan more efficiently. Finally, the top level goal of IS-TELESCOPE-MIRROR is achieved.

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.

[6] [illegible]

[7] [illegible]

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-CONCAVE before achieving IS-MIRROR, resulting in a more efficient plan.[8] A proof process would again be invoked to determine whether to make it a reordering rule, concluding that it is always better to achieve IS-CONCAVE first. The chart below, summarizes the new knowledge acquired (in italics) as a result of the problem solving episodes, experiments, and proofs. Such is the process of fleshing out incomplete domain and control knowledge through experience and focused interaction with the task environment. Although in the example all the preconditions are consequences learned are negated predicates, the same process applies to acquiring simple atomic predicates. However, the process of acquiring logical combinations of atomic predicates is significantly more complex.

OPERATORS	PRECONDITIONS	CONSEQUENCES
1) GRIND-CONCAVE(<obj>)	ISA(<obj>, solid)	IS-CONCAVE(<obj>) *~IS-POLISHED(<obj>)* *~IS-REFLECTIVE(<obj>)*
2) POLISH(<obj>)	ISA(<obj>, glass) IS-CLEAN(<obj>) *~IS-REFLECTIVE(<obj>)*	IS-POLISHED(<obj>)
3) ALUMINIZE(<obj>)	IS-CLEAN(<obj>) ISA(<obj>, solid)	IS-REFLECTIVE(<obj>) *~IS-CLEAN(<obj>)*
4) CLEAN(<obj>)	ISA(<obj>, solid)	IS-CLEAN(<obj>)

INFERENCES:

1) IS-REFLECTIVE(<obj>) & IS-POLISHED(<obj>) --> IS-MIRROR(<obj>)

2) IS-MIRROR(<obj>) & IS-CONCAVE(<obj>) --> IS-TELESCOPE-MIRROR(<obj>)

NEWLY ACQUIRED CONTROL RULES for SUBGOAL ORDERING:

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

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

[8] In general [illegible]

7. The Current Implementation

The operator refinement strategy has been implemented in a subset of PRODIGY augmented with an execution monitoring component. We plan to integrate both execution monitoring and experimentation into the full PRODIGY system shortly. To handle operator and object hierarchies, we are representing operators and other domain knowledge using Framekit (Carbonell & Joseph, 1986), a frame-based knowledge representation system.

The planner, execution monitor, and experiment proposer combine three sources of dynamic knowledge:

- The state of the plan being developed and its partial execution.
- PRODIGY's expectations of the current status of the external world.
- The observed status of the external world, including divergences from expectations as determined by the execution monitor.

Since PRODIGY is not yet connected to an external robot or to the world modelers simulation environment (Carbonell & Hood, 1986), execution monitoring proceeds by interrogating the user about aspects of the external state it deems relevant. These aspects consist of expected changes brought about by the application of operators. For instance, the system checks that expected consequences of operators have come to pass, but not that all supposedly persistent states have remained untouched. Problems in the latter category come to light only when a presumably satisfied precondition to a later operator is found to be violated by the execution monitor. Then, the experimentation process is invoked to identify which of the candidate intervening actions could be the culprit operator, augmenting its postconditions so that next time the additional change to the external is recorded and expected.

8. The General Operator-Refinement Method

If the system is given complete and correct knowledge, it uses a standard problem solving approach. In particular, it employs means-ends analysis to select an operator. Then the system subgoals for every precondition of the operator that is not matched in the current state. Once all the preconditions are matched, the planner updates the state with the postconditions of the operator.

With incomplete knowledge, however, the system continually monitors the outside world to check for any discrepancy with the internal state. When a discrepancy arises, standard problem solving has to be modified as follows:

THE OPERATOR REFINEMENT METHOD

For every operator O selected

for every precondition P of operator O
if State(P) ≠ World(P)[9]
then One of the operators previously applied since P was established has a previously unknown postcondition. CASE 1

1) Select candidate operators. The candidate operators are all that were applied between the last time that P was checked in the World and the current check.
2) Identify responsible operator. Formulate experiments by selecting an operator in a binary search over the candidate operators, 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, P is a new postcondition of O_E.
3) Add P as a new postcondition of operator O_E.

for every postcondition P of operator O
if State(P) ≠ World(P)
then

if ∃ Q precondition of O such that State(Q) ≠ World(Q)
then One of the operators previously applied since Q was established should have had a postcondition affecting Q. CASE 2

1) Select candidate operators. The candidate operators are all that were applied between the last time that Q was checked and the current check.
2) Identify responsible operators. Formulate experiments by selecting an operator in a binary search over the candidate operators. Each experiment will consist of applying one of the operators and then check Q in the World. If as a result of an experiment with operator O_E Q is unexpectedly changed in the World, Q is a new postcondition of O_E.
3) Add Q as a new postcondition of operator O_E.

if ∀ preconditions Q of O State(Q) = World(Q)
then A precondition of operator O might be missing. CASE 3

1) Select candidate preconditions. The candidate set is formed with all the differences between any state in which O was applied successfully and the current state (unsuccessful application of O).
2) Identify missing precondition. Formulate experiments using a binary search over the set of candidate preconditions and choose the predicate R that the results show to be the significant difference as the new precondition of O.
3) Add R as a new precondition of operator O.

[9] State(P) is a predicate that checks if P is true in the current state of the planner's model. World(P) checks the same but for the actual world.

In all of the above cases, the system attempts to recover and fix the plan, using the new information learned. In addition we use 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.

9. Concluding Remarks: Beyond Simple Experimentation

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 9-1). 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.

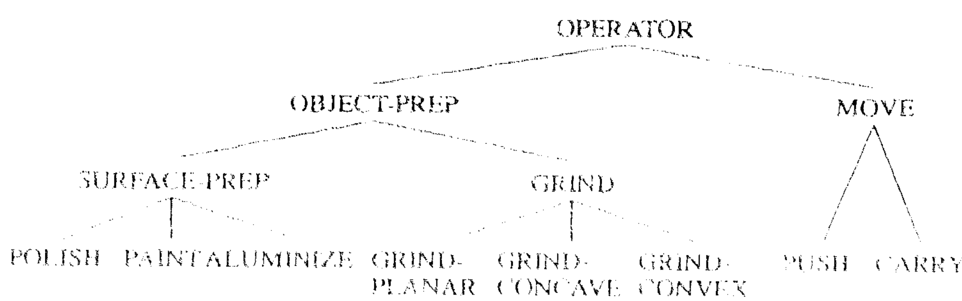


Figure 9-1: Fragment of operator "isa" hierarchy

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. Moreover, the experimentation methods so far have focused only on operator refinement (their application conditions and consequences) but not on acquiring new operators, new features of the domain, or new meta-level control structures. Our ultimate aim is to develop a set of general techniques for an AI system to acquire knowledge of its task domain automatically via experimentation, 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.

10. References

Carbonell, J. G. (1981). *Subjective Understanding: Computer Models of Belief Systems.* Ann Arbor, MI: UMI research press.

Carbonell, J. G. (1983). 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,* Palo Alto, CA: Tioga Press.

Carbonell, J. G. and Hood, G. (1986). 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,* Kluwer Academic Press.

Carbonell, J. G. and Joseph, R. (1986) The FrameKit+ Reference Manual. CMU Computer Science Department internal paper.

Carbonell, J. G. (1986). 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.

Cheng, P. W. and Carbonell, J. G. (1986). Inducing Iterative Rules from Experience: The FERMI Experiment. *Proceedings of AAAI-86,* Philadelphia, PA.

DeJong, G. F. and Mooney, R. (1986). Explanation-Based Learning: An Alternative View. *Machine Learning Journal, 1(2).*

Falkenhainer, B. C. and Michalski, R. S. (1986). Integrating Quantitative and Qualitative Discovery: The ABACUS System. *Machine Learning, 1,* 367-402.

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

Koehn, B. W. and Zytkow, J. M. (1986). Experimenting and Theorizing in Theory Formation. *Proceedings of the ACM Sigart International Symposium on Methodologies for Intelligent Systems,* Knoxville, TN.

Kuhn, T. S. (1977). *The Essential Tension: Selected Studies in Scientific Tradition and Change.* U. of Chicago Press.

Kulkarni, D. and Simon, H. A. (1987) The Process of Scientific Discovery: The Strategy of Experimentation. To appear in *Cognitive Science.*

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

Langley, P. adn Nordhausen, B. (1986). A Framework for Empirical Discovery. *Proceedings of the International Meeting on Advances in Learning,* Les Arcs, France.

Langley, P. W., Simon, H. A. and Bradshaw, G. L. (1983). Rediscovering Chemistry with the BACON System. In R. S. Michalski, J. G. Carbonell and T. M. Mitchell (Eds.) *Machine Learning, An Artificial Intelligence Approach,* Palo Alto, CA: Tioga Press.

Langley, P., Zytkow, J. M., Simon, H. A. and Bradshaw, G. L. (1986). The Search for Regularity: Four Aspects of Scientific Discovery. In Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. (Eds.) *Machine Learning, An Artificial Intelligence Approach, Volume II,* Morgan Kaufmann.

Lenat, D. B. (1983). The Role of Heuristics in Learning by Discovery: Three Case Studies. In R. S. Michalski, J. G. Carbonell and T. M. Mitchell (Eds.) *Machine Learning, An Artificial Intelligence Approach,* Palo Alto, CA: Tioga Press.

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

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

Minton, S. (1985). Selectively Generalizing Plans for Problem Solving. *Proceedings of AAAI-85*, 596-599.

Minton, S. N. and Carbonell, J. G. (1987). Strategies for Learning Search Control Rules: An Explanation-Based Approach. *Proceedings of IJCAI-87*, Milan, Italy.

Minton, S. N., Carbonell, J. G., Etzioni, O., Knoblock, C. A.,and Kuokka, D. R. (1987). Acquiring Search Control Rules: Explanation-Based Learning in the PRODIGY System. *Proceedings of the fourth Machine Learning Workshop*, Irvine, CA.

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

Mitchell, T. M., Utgoff, P. E. and Banerji, R. B. (1983). 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*, Palo Alto, CA: Tioga Press.

Nilsson, N. (1971). *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill.

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

Sacerdoti, E. D. (1977). *A Structure for Plans and Behavior*. Amsterdam: North-Holland.

Shen, W. M. (1987) . Department of Computer Science, Carnegie Mellon University. Personal communication.

Sussman, G. J. (1973). *A computational model of skill acquisition*. PhD thesis, Massachusetts Institute of Technology.

Wilensky, R. (1983). *Planning and Understanding*. Reading, MA: Addison Wesley.

Appendix: Annotated Program Trace

We include here a trace of our program. The example is the same as in section . The initial state is:

```
(initial-state ((is-glass glass1)
                (is-solid glass1)
                (is-planar glass1)
                (is-glass glass2)
                (is-solid glass2)
                (is-planar glass2)
                (is-wood wood1)))
```

and the goal is (is-telescope-mirror glass1).
The trace gives several pieces of information about every operator O:

- When the operator is selected: "Trying operator O".
- When a precondition P is checked in the internal state and external world: "Checking for precondition P".
- When all the preconditions have been matched: "All preconditions checked, the operator O is being applied".
- When a postcondition P is being checked in the internal state and in the external world: "Checking for postcondition P".

Every time a precondition or a postcondition is checked, the results of the checks with the internal state of the planner and the external world are shown. The system itself finds the information about the internal status, but the user has to provide the result of the check with the external world.

Only the interesting parts of the trace have been included. It has been commented at some points to make it more readable.

(goal-state (is-telescope-mirror glass1))

Trying operator (is-telescope-mirror glass1)

Checking for precondition (is-mirror glass1)
Internal State: n *Simulated World:* n

Trying operator (is-mirror glass1)

...

;;; Solving the subgoal (is-reflective)

...

Trying operator (polish glass1)

Checking for precondition (is-clean glass1)
Internal State: y *Simulated World:* n ;;; CASE 1

*** Experimentation triggered
*** New postcondition: (not (is-clean glass1))
*** Candidate operators: ((aluminize))

The postcondition (not (is-clean glass1))
is being added to the operator aluminize

...

;;; Solving (is-clean glass1)

...

All preconditions checked,
the operator (polish glass1) *is being applied*

Checking for postcondition (is-polished glass1)
Internal State: n *Simulated World:* y ;;; CASE 3

;;; Discrepancy between state and world (Case 2)

Checking again for precondition (is-glass glass1)
Internal State: y *Simulated World:* y

Checking again for precondition (is-clean glass1)
Internal State: y *Simulated World:* y

*** Experimentation triggered
*** Operator (polish)
*** Difference between current state and
*** Desired Successful State (is-reflective)
*** Would (polish) work if not (is-reflective glass1)?
*** Simulated result of the experiment: y

The precondition (not (is-reflective glass1))
is being added to the operator polish

Retrying operator (polish glass1)

Checking for precondition (is-glass glass1)
Internal State: y *Simulated World:* y

Checking for precondition (not (is-reflective glass1))
Internal State: n *Simulated World:* n

;;; There is no operator to achieve the goal (not (is-reflective glass1)).
;;; At this point the system hypothesizes that there is a goal interaction
;;; and applies the corresponding heuristic.

New goal preference: Prefer is-polished over is-reflective

;;; Since the system doesn't know how to make glass1 not reflective,
;;; it restarts the process with the glass that looks more like
;;; glass1, which is glass2.

;;; The subgoal is-mirror is solved again, but this time considering
;;; the new goal preference rule and using the refined operators.
;;; Then the subgoal is-concave is solved.

...

All preconditions checked,
the operator (is-telescope-mirror glass2) *is being applied*

Checking for postcondition (is-telescope-mirror glass2)
Internal State: y *Simulated World:* n ;;; CASE 2

Checking again for precondition (is-mirror glass2)
Internal State: y *Simulated World:* n

Checking again for precondition (is-concave glass2)
Internal State: y *Simulated World:* y

Checking again for precondition (is-reflective glass2)
Internal State: y *Simulated World:* n

Checking again for precondition (is-polished glass2)
Internal State: y *Simulated World:* n

*** Experimentation triggered
*** New postcondition: (not (is-reflective))
*** Candidate operators: (grind concave))

The postcondition (not (is-reflective glass2))
is being added to the operator grind-concave

*** New postcondition: (not (is-polished))
*** Candidate operators: ((aluminize) (grind-concave))

The postcondition (not (is-polished glass2))
is being added to the operator grind-concave

...

;;; The subgoal is-mirror is solved again,
;;; and the system can finally make a telescope mirror.

Operator (is-telescope-mirror glass2) successfully applied

Success!!

(plan ((clean glass2)
(polish glass2)
(aluminize glass2)
(is-mirror glass2)
(grind-concave glass2)
(clean glass2)
(polish glass2)
(aluminize glass2)
(is-mirror glass2)
(is-telescope-mirror glass2)))

Non optimal plan:

;;; The system applies the plan optimization heuristic:

New goal preference: Prefer grind-concave over is-mirror

End of trace.