A New Algorithm for Generative Planning

Matthew L. Ginsberg
CIRL
1269 University of Oregon
Eugene, OR 97403-1269

Abstract

Existing generative planners have two properties that one would like to avoid if possible. First, they use a single mechanism to solve problems both of action selection and of action sequencing, thereby failing to exploit recent progress on scheduling and satisfiability algorithms. Second, the context in which a subgoal is solved is governed in part by the solutions to other subgoals, as opposed to plans for the subgoals being developed in isolation and then merged to yield a plan for the conjunction. We present a reformulation of the planning problem that appears to avoid these difficulties, describing an algorithm that solves subgoals in isolation and then appeals to a separate NP-complete scheduling test to determine whether the actions that have been selected can be combined in a useful way.

1 INTRODUCTION

One of the insights arising from the theoretical analysis of the complexity of domain-independent planning [1, 4, 14] is that planning, involving both the selection and sequencing of actions, is in general properly more difficult than scheduling, which involves sequencing problems in isolation. My goal in this paper is to introduce a new type of planning algorithm that responds to these results by separating the problem of action selection from that of action sequencing. By doing this, we can expect to incorporate into planners fast existing algorithms for solving NP-complete problems such as scheduling.

We will achieve this computational split by introducing a new planning language $\mathcal{L}$ that mediates between the action selection and action sequencing phases. Our intention is that a planner concerned with action selection in isolation could easily produce a sentence of $\mathcal{L}$ as its output, and that a scheduler could accept such a sentence as its input. In solving the Sussman anomaly, for example (Figure 1), the planner would realize that it needs to move block C off block A before moving block A onto block B, and also that it needs to move block B onto block C. The scheduling algorithm would then identify the ordering needed to actually achieve the goal (namely, that it is necessary to move blocks C, B and A in that order).

The language itself is introduced in Section 2, and we demonstrate its relevance to scheduling at a theoretical level in Section 3 by showing to be NP-complete the problem of deciding if a particular sentence in the language corresponds to a nonempty set of plans. We turn to planning in Section 4, developing an algorithm that generates instances of our planning language in response to a particular planning problem. This algorithm differs from existing partial-order causal link (POCL) planners in at least three fundamental ways.

First, it solves subgoals separately, merging the results to construct a plan for the overall goal itself. This property is recursive: Subgoals are solved separately at all levels, with the plans being merged to construct one that achieves the overall goal.

Second, it does no backtracking. Instead, the algorithm works by gradually refining a candidate set of plans that might achieve the goal. If a particular approach is tried for a goal g, and this approach involves developing plans to solve a subgoal g', any solutions found for the subgoal will remain available to the planner even if a different approach is eventually selected for g itself.

Finally, the planner needs to plan for distinct goals only once. Even if a particular goal g must be achieved as a precondition for each of two separate actions,

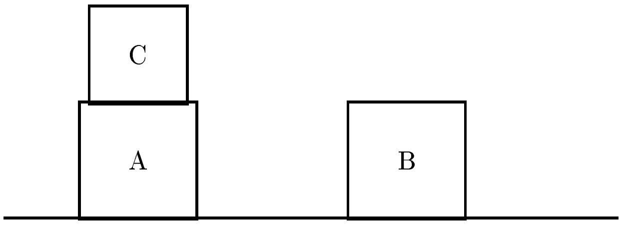

Figure 1: The Sussman anomaly: Get A on B on C

there will typically be only a single appearance of g in the planning search space.

We present an example in Section 5, where we also discuss the general computational properties of our procedure. Section 6 discusses an implementation of our work, and Section 7 contains a variety of concluding remarks, connecting our work to a variety of results that have appeared elsewhere. Proofs appear in an appendix.

2 THE PLAN LANGUAGE

We assume throughout that we have some fixed set of actions A.

Definition 2.1 *By a* linear plan *we will mean a finite sequence of elements of A. Such sequences will typically be denoted $\langle a_1, \ldots, a_n \rangle$.*

Of greater interest to us than specific linear plans will be *sets* of such plans. To this end, we define a language describing those sets in which we will be interested:

Definition 2.2 *The* plan language $\mathcal{L}$ *is the smallest set of sentences that contains the distinguished elements $\emptyset$ and $[\,]$ along with every element of A, and is closed under three binary operators $\cup$, $|$ and π.*

The set of plans $\emptyset$ is, as usual, the empty set. The plan $[\,]$ is intended to correspond to the sequence of no actions, and $a \in A \subseteq \mathcal{L}$ is the singleton plan where action a alone is taken. The operator $P|Q$ concatenates the plan Q to the end of the plan P, and $\pi(P,Q)$ is intended to capture the frame axiom in our setting. Let us give the formal definition of π first and then explain its meaning:

Definition 2.3 *Let $P \in \mathcal{L}$ be a set of plans. Membership in P is defined recursively as follows:*

1. *$\langle a_1, \ldots, a_n \rangle \notin \emptyset$.*
2. *$\langle a_1, \ldots, a_n \rangle \in [\,]$ if and only if $n = 0$.*
3. *$\langle a_1, \ldots, a_n \rangle \in a$ for an action a if and only if $n = 1$ and $a_1 = a$.*
4. *$\langle a_1, \ldots, a_n \rangle \in P|Q$ if and only if for some $i \leq n$, $\langle a_1, \ldots, a_i \rangle \in P$ and $\langle a_{i+1}, \ldots, a_n \rangle \in Q$.*
5. *$\langle a_1, \ldots, a_n \rangle \in P \cup Q$ if and only if $\langle a_1, \ldots, a_n \rangle \in P$ or $\langle a_1, \ldots, a_n \rangle \in Q$.*
6. *To determine if $\langle a_1, \ldots, a_n \rangle \in \pi(P,Q)$, let i be the greatest integer such that $\langle a_1, \ldots, a_i \rangle \in P \cup Q$. Then $\langle a_1, \ldots, a_n \rangle \in \pi(P,Q)$ if and only if such an i exists and $\langle a_1, \ldots, a_i \rangle \notin Q$.*

It is presumably only the last of the above clauses that requires explanation. To understand it, suppose that we have some goal that is established by the plans in P, but disestablished by the plans in Q (clobbered, in Chapman's more picturesque terms [2]). Now the goal will hold after an action sequence $\langle a_1, \ldots, a_n \rangle$ has been executed if and only if it is established by some initial subsequence of $\langle a_1, \ldots, a_n \rangle$ and not subsequently clobbered – in other words, if the longest subsequence $\langle a_1, \ldots, a_i \rangle$ that either establishes or disestablishes the goal in fact establishes it. By writing the definition to require that the action a_i not disestablish the goal, we cater to the possibility that $P \cap Q \neq \emptyset$.

Lemma 2.4 *$\pi(P_1, Q_1) \subseteq \pi(P_2, Q_2)$ if $P_1 \subseteq P_2$ and $Q_1 \supseteq Q_2$.* ■

We now make the following notational definition:

Definition 2.5

1. *$U \equiv \pi([\,], \emptyset)$.*
2. *$\neg P \equiv \pi(U, P)$.*
3. *$P \cap Q \equiv \neg(\neg P \cup \neg Q)$.*

Lemma 2.6 *U is the set of all linear plans. Negation and intersection have their conventional meanings.*

In the Sussman anomaly, for example, the goal of getting A on B is established by moving A to B, which succeeds in at most the plans given by $U|\texttt{move}(A,B)$. Moving C to B succeeds for at most the plans in $U|\texttt{move}(C,B)$. For the preconditions of the various actions, we define

$$\texttt{cover}(B) \equiv [U|\texttt{move}(A,B)] \cup [U|\texttt{move}(C,B)] \qquad (1)$$

the set of all plans that *might* clobber B's being clear, and similarly for $\texttt{cover}(A)$ and $\texttt{cover}(C)$. We can now apply Lemma 2.4 to conclude that since B is clear in the initial situation, it will remain clear after executing at *least* the plans in the set

$$\texttt{clear}(B) \equiv \pi([\,], \texttt{cover}(B)) \qquad (2)$$

since we have perhaps overestimated the set of clobberers in (1). The above set is the set of all action sequences that do not contain either $\texttt{move}(A, B)$ or $\texttt{move}(C, B)$, and B will be clear after any such sequence is executed, since it is clear initially.

In a similar way, C is clear for at least the plans in the set $\texttt{clear}(C) \equiv \pi([\,], \texttt{cover}(C))$ and C is on A for at least

$$\texttt{on}(C, A) \equiv \pi([\,], U|\texttt{move}(C, B) \cup U|\texttt{move}(C, \texttt{table}))$$

It follows that A will be made clear by the plans in the set

$$[\texttt{clear}(C) \cap \texttt{on}(C, A)]|\texttt{move}(C, \texttt{table})$$

and therefore that A will be clear for plans in the set

$$\texttt{clear}(A) \equiv \pi([\texttt{clear}(C) \cap \texttt{on}(C, A)]|\texttt{move}(C, \texttt{table}), \texttt{cover}(A))$$

If C is clear and on A, A will be clear in a persistent way after C is moved to the table. Covering A (the second argument to π above) defeats the persistence.

We can now conclude that A will be on B for at least the plans in the set

$$\texttt{on}(A, B) \equiv \pi([\texttt{clear}(A) \cap \texttt{clear}(B)]|\texttt{move}(A, B), [U|\texttt{move}(A, C)] \cup [U|\texttt{move}(A, \texttt{table})])$$

In a similar way, B will be on C for at least the plans

$$\texttt{on}(B, C) \equiv \pi([\texttt{clear}(B) \cap \texttt{clear}(C)]|\texttt{move}(B, C), [U|\texttt{move}(B, A)] \cup [U|\texttt{move}(B, \texttt{table})])$$

The overall goal will be achieved by plans in the set

$$\texttt{on}(A, B) \cap \texttt{on}(B, C) \tag{3}$$

Consider now the successful plan

$$\langle \texttt{move}(C, \texttt{table}), \texttt{move}(B, C), \texttt{move}(A, B) \rangle \tag{4}$$

This plan is in $\texttt{on}(A, B)$ because the plan itself is in $[\texttt{clear}(A) \cap \texttt{clear}(B)]|\texttt{move}(A, B)$. To see this, we must show that

$$\langle \texttt{move}(C, \texttt{table}), \texttt{move}(B, C) \rangle \tag{5}$$

is in $\texttt{clear}(A) \cap \texttt{clear}(B)$. To see *this*, note that no subsequence of (5) is in $\texttt{cover}(A)$ or $\texttt{cover}(B)$. As a result, we can show that (5) is in $\texttt{clear}(A)$ by showing that some subsequence of (5) is in

$$[\texttt{clear}(C) \cap \texttt{on}(C, A)]|\texttt{move}(C, \texttt{table})$$

In a similar way, we can show that (5) is in $\texttt{clear}(B)$ by showing that some subsequence of (5) is in $[\,]$.

The second of these is obvious. The first follows because $[\,]$ is in $\texttt{clear}(C) \cap \texttt{on}(C, A)$. As a result, we can conclude that A and B are clear after (5), so that A is indeed on B after executing (4). It is not hard to show that B is on C as well, so that (4) does indeed succeed in solving our problem.

What about the failing plan

$$\langle \texttt{move}(C, \texttt{table}), \texttt{move}(A, B), \texttt{move}(B, C) \rangle \tag{6}$$

where A is moved onto B too early? In this case, we can show that (6) is not known to be in $\texttt{on}(B, C)$. Specifically, it has no subsequence known to be in

$$[\texttt{clear}(B) \cap \texttt{clear}(C)]|\texttt{move}(B, C)$$

because $\langle \texttt{move}(C, \texttt{table}), \texttt{move}(A, B) \rangle$ is not known to be in $\texttt{clear}(B) \cap \texttt{clear}(C)$. Still more specifically, $\langle \texttt{move}(C, \texttt{table}), \texttt{move}(A, B) \rangle$ is not in $\texttt{clear}(B)$ because the sequence itself is in $\texttt{cover}(B)$.

3 COMPLEXITY RESULTS

The analysis of the previous section shows that the successful plan

$$\langle \texttt{move}(C, \texttt{table}), \texttt{move}(B, C), \texttt{move}(A, B) \rangle$$

is a member of (3) while the failing one

$$\langle \texttt{move}(C, \texttt{table}), \texttt{move}(A, B), \texttt{move}(B, C) \rangle$$

is not. In some sense, this means that membership in (3) resolves the question of whether a specific linear plan satisfies the ordering constraints implicit in the original problem. Our main goal in this section will be to show the following:

Proposition 3.1 *Let $P \in \mathcal{L}$ be a set of plans. Then showing that $P \neq \emptyset$ is NP-complete.*

The proof involves two stages. First, we must show that the problem in question is in NP by demonstrating that it is possible to validate in polynomial time a witness that $P \neq \emptyset$. Second, we need to show that the problem is NP-hard by embedding some known NP-hard problem in it. For the first part, we have:

Definition 3.2 *For any $P \in \mathcal{L}$, we define the* length *of P to be the length of the syntactic expression P itself. The length of P will be denoted $|P|$.*

Lemma 3.3 *Let $P \in \mathcal{L}$ be a nonempty set of plans. Then there is a plan $\langle a_1, \ldots, a_n \rangle \in P$ with $n \leq 1 + |P|$.*

Lemma 3.4 *Let $P \in \mathcal{L}$ be a set of plans, and p a specific linear plan. Then the question of whether $p \in P$ can be resolved in time polynomial in the lengths of p and P.*

Lemma 3.4 tells us that we can validate a witness that $P \neq \emptyset$ in time polynomial in both the size of P and the length of the witness; Lemma 3.3 guarantees that the witness itself is of polynomial size in P.

Corollary 3.5 *Let $P \in \mathcal{L}$ be a set of plans. Showing that $P \neq \emptyset$ is in NP.* ∎

To show that the problem is NP-hard, consider the known NP-complete problem of satisfiability. In this problem, we have a set of clauses such as $x_1 \vee \neg x_2 \vee x_3$ and need to find an assignment of true or false to each of the x_i's such that every clause is satisfied. To achieve the embedding, we will view the x_i's as actions as well, taking x_i to mean that x_i is an action in some hypothetical plan p. Now the clauses can all be satisfied if and only if a plan can be found meeting the given conditions.

Definition 3.6 *Let X be a set of atomic variables. Now if $x \in X$ is an atom, we define $p(x)$ to be $\pi(U|x, \emptyset)$. For a literal that is the negation of an atom, we take $p(\neg x) = \neg p(x)$. If $c = \bigvee_i l_i$ is a clause (a disjunction of literals), we take $p(c) = \bigcup_i p(l_i)$, and if $C = \bigwedge_i c_i$ is a theory (a conjunction of clauses), we take $p(C) = \bigcap_i p(c_i)$.*

Lemma 3.7 *A theory C is satisfiable if and only if $p(C) \neq \emptyset$.*

Corollary 3.8 *Let $P \in \mathcal{L}$ be a set of plans. Showing that $P \neq \emptyset$ is NP-hard.* ∎

Proposition 3.1 now follows.

4 PLANNING

In this section, we present an algorithm that works from a domain description to compute the set of plans that achieve a particular goal. We will assume that the domain is described in terms of STRIPS-like add, delete, and precondition lists, but this assumption is not central to our methods. We will also assume the existence of an "initializing" action i that sets up the initial state. This action has no preconditions, adds all fluents true in the initial state, and deletes all others.

Let us begin with some fairly high-level remarks. In any realistic situation, it is unlikely that we be able to compute exactly the set of plans achieving a goal g. The axiomatization of the domain can be expected to include information implying the existence of a wide range of exceptions to any specific plan or plan schema that achieves g. These exceptions will themselves have exceptions, plans that achieve g even though they belong to "schema" that are expected not to, and so on. If we denote by $L(g)$ the set of plans that achieve g, we see that it will in general be impractical to attempt to compute $L(g)$ exactly.

Instead, we will give a procedure that computes a set of increasingly fine approximations to $L(g)$. Somewhat more specifically, we will produce sequences of plans

$$L_0^+(g) \supseteq L_1^+(g) \supseteq \cdots$$

and

$$L_0^-(g) \subseteq L_1^-(g) \subseteq \cdots$$

such that

$$L_0^-(g) \subseteq L_1^-(g) \subseteq \ldots \subseteq L(g) \subseteq \ldots \subseteq L_1^+(g) \subseteq L_0^+(g)$$

In other words, $L(g)$ is always bracketed by the L^- and L^+ approximations. Since $L(g) \supseteq L_k^-(g)$ for any k, we can stop and return *some* successful plan as soon as $L_k^-(g) \neq \emptyset$. (Note that the check that $L_k^-(g) \neq \emptyset$ is NP-complete.) Alternatively, we might stop and return when L^+ and L^- are "close" in some sense, as is suggested in the work on approximate planning [9].

My interest here, however, is simply in constructing the approximations L^+ and L^- themselves. To that end, we will think of a node in the planning search tree as corresponding either to a goal or subgoal (a **goal** node) or to a specific action that we intend to take (an **action** node). In addition, we will label nodes with a *parity* in the sense that "positive" nodes support plans to achieve the goal and "negative" nodes correspond to exceptions to those plans or other difficulties. A node can also be *expanded* or *unexpanded*.

Definition 4.1 *A* planning tree *is a singly rooted, directed acyclic graph where no node has paths back to the root of both even and odd length. Nodes at even depths (including the root) will be called* goal *nodes and denoted $(g, \pm)$; nodes at odd depths will be called* action *nodes and denoted $(a, \pm)$. Each fringe node in the graph can be* expanded *or* unexpanded.

If T is a planning tree and n is an unexpanded fringe node in T, the result of expanding n in T *is the planning tree obtained by labeling n as expanded and adding as children of n the following nodes:*

1. If $n = (g, \pm)$ is a goal node, the children are action nodes $(a, \pm)$ for each action that adds g and action nodes $(a, \mp)$ for each action that deletes g.

2. If $n = (a, \pm)$ is an action node, the children are goal nodes $(g, \pm)$ for each precondition g of a.

A legal tree *for a goal g is any tree that can be obtained by successively expanding the tree containing the single node $(g, +)$.*

Of course, it is not sufficient to simply build the trees; we also need to label the nodes with the plans that achieve them. To this end, we define:

Definition 4.2 *In a* conservative *labeling, every unexpanded node in a planning tree with positive parity is labeled $\emptyset$ and every unexpanded node with negative parity is labeled U. In a* liberal *labeling, every unexpanded node with positive parity is labeled U and every unexpanded node with negative parity is labeled $\emptyset$.*

The label for any expanded action node (a, p) is

$$\bigcap_i l(c_i)|a \tag{7}$$

where the c_i are the children of (a, p) and the labels $l(c_i)$ are the labels attached to those children. Any expanded goal node (g, p) is labeled with

$$\pi\left(\bigcup_i l(c_{i,p}), \bigcup_i l(c_{i,\neg p})\right)$$

where the $c_{i,p}$ are the children of the same parity as (g, p) and the $c_{i,\neg p}$ are the children of opposite parity.

The conservative *(respectively* liberal*) labeling of a tree T is the label assigned to the root node by a conservative (respectively liberal) labeling of T. These labelings are denoted $L^-(T)$ and $L^+(T)$ respectively.*

Theorem 4.3 *Let T_0 be a planning tree, and T_1 an expansion of it. Suppose that T_0 is labeled L_0^+ in a liberal labeling and L_0^- in a conservative one, and that T_1 is labeled L_1^+ and L_1^-. Then*

$$L_0^- \subseteq L_1^- \subseteq L(g) \subseteq L_1^+ \subseteq L_0^+$$

where g is the goal at the root of T_0.

The above theorem in isolation tells us little; it would be satisfied if $L_k^- = \emptyset$ and $L_k^+ = U$ for all k! We still need to know that the planning algorithm we have proposed actually converges on the correct answer:

Theorem 4.4 *Let g be a goal, and T_0 the planning tree consisting of the single goal node $(g, +)$. Now if p is a plan that achieves g, there is some expansion T of T_0 such that $p \in L^-(T)$. Conversely, if p fails to achieve g, there is an expansion such that $p \notin L^+(T)$.*

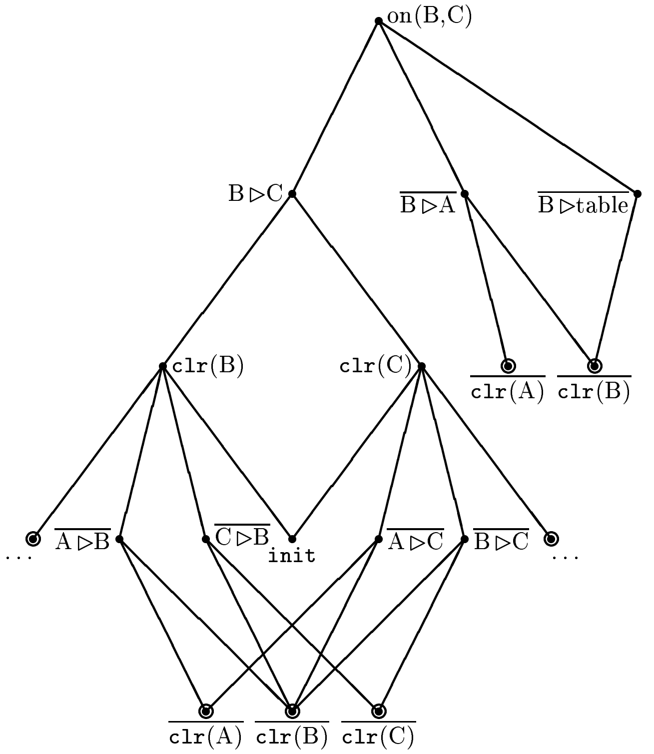

Figure 2: Planning to get B on C

5 COMPUTATION

The results of the previous section sanction the following planning algorithm:

Procedure 5.1 To find a plan for a goal g:

> $T :=$ the planning tree consisting of the single node $(g, +)$
> **while** $L^-(T) = \emptyset$
>   **if** T contains an unexpanded node, expand it
>   else **fail**
> **return** $L^-(T)$

Theorem 5.2 *If Procedure 5.1 returns a set of plans P in response to a goal g, every plan in P achieves g. In addition, if the nodes are expanded in order of increasing depth, the procedure will always terminate for any achievable goal g.* ■

Procedure 5.1 can be modified to check at each step whether $L^+(T) = \emptyset$. Should this ever happen, it is possible for the planner to report failure even if the tree has not yet been fully expanded.

As an example of the procedure at work, consider the simple goal of getting B onto C in the Sussman anomaly. A partial expansion of the planning tree for this problem appears in Figure 2. The nodes in the figure are labeled as follows:

1. Action nodes are labeled with the action in question; the action of moving A to B, for example, is denoted $A \rhd B$. The initializing action is denoted `init`.

2. Goal nodes are labeled with the fluent that the node is trying to achieve. A node labeled $\mathtt{clr}(B)$, for example, is trying to get B clear.

3. Nodes of negative parity appear with an overline. Nodes of positive parity are not so marked.

4. Unexpanded nodes are marked with a bullseye.

The reasoning behind the arcs in the graph is as follows:

1. The root node is labeled with the goal $\mathtt{on}(B, C)$.

2. The root can be achieved by moving B to C. It can be clobbered by moving B to A or to the table. These three children are added at depth 1, with the nodes corresponding to the clobbering actions having negative parity.

3. Moving B to C requires that B and C be clear; moving B to A (a clobbering action) requires that B and A be clear. Since the two nodes at depth two with goal $\mathtt{clear}(B)$ have opposing parities, they cannot be combined in the planning tree. The two appearances of $\mathtt{clear}(B)$ under the clobbering actions *are* combined.

4. B is made clear by the initializing action, and also potentially by a variety of other actions that have been lumped into a single node labeled ... in the figure. (Moving A off the top of B will make B clear, and so on.) B being clear will be clobbered by moving either A or C onto B. The goal node labeled $\mathtt{clr}(C)$ is expanded similarly.

5. The negative parity action node $A \rhd B$ at depth 3 has as children negative parity goal nodes trying to get A and B clear. These nodes could be identified with analogous nodes at depth 2, but haven't been in the interests of maintaining the clarity of the figure.

What about the labels assigned to the nodes? Assuming that we are interested in a conservative labeling of the tree, we begin by assigning either $\emptyset$ or U to the unexpanded nodes depending on their parity. The fringe nodes at the bottom of the tree (depth 4) and those at depth 2 are all of negative parity and are labeled U, while the unexpanded ... nodes at depth 3 are labeled $\emptyset$.

We can now compute the labels for the nodes at depth 3. These labels include expressions of the form $U|\mathtt{init}$, indicating that the initial situation needs to be established before the plans involved can be effective. We abbreviate $U|\mathtt{init}$ to [ ] to get the tree in Figure 3. In the figure, we have dropped nodes labeled $\emptyset$ that cannot contribute to the values of nodes at depth 2, and have abbreviated $U|\mathtt{move}(A, B)$ (for example) to $U|$A$\rhd$B.

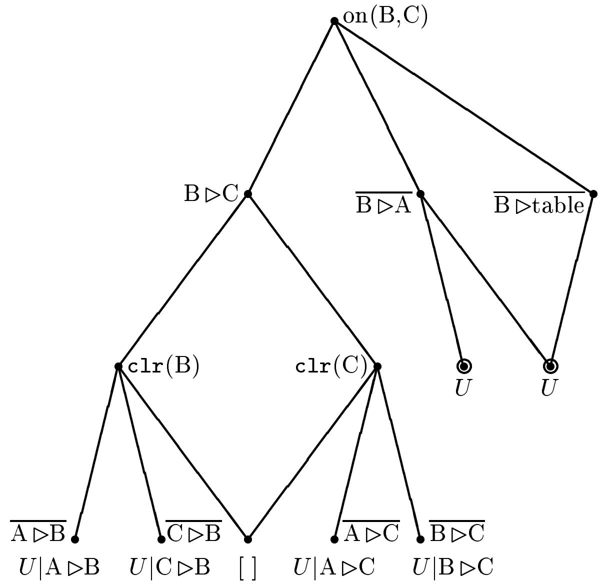

Figure 3: The conservative labeling, continued

Consider now the node at depth 2 in Figure 3 dealing with the goal $\mathtt{clear}(B)$. The label to be assigned to this node is

$$\pi([\,], U|\mathtt{move}(A, B) \cup U|\mathtt{move}(C, B))$$

which is exactly the set of plans that we denoted $\mathtt{clear}(B)$ in (2) of Section 2.

Continuing, the label assigned to the action node $B \rhd C$ in the figure is seen to be

$$[\mathtt{clear}(B) \cap \mathtt{clear}(C)]|\mathtt{move}(B, C) \tag{8}$$

and the labels assigned to the two action nodes of negative parity are $U|\mathtt{move}(B, A)$ and $[U|\mathtt{move}(B, \mathtt{table})$ respectively. We can finally label the root node as

$$\begin{aligned}\pi([&\mathtt{clear}(B) \cap \mathtt{clear}(C)]|\mathtt{move}(B, C),\\ &[U|\mathtt{move}(B, A)] \cup [U|\mathtt{move}(B, \mathtt{table})])\end{aligned} \tag{9}$$

This plan set is nonempty (the sequence $\langle \mathtt{move}(B, C)\rangle$ is an element of it), and the planner can return at this point.

Note that the planner cannot return until the two negative parity nodes at depth 1 have been expanded. If these were not expanded, (9) would be replaced with

$$\pi([\mathtt{clear}(B) \cap \mathtt{clear}(C)]|\mathtt{move}(B, C), U)$$

which is empty.

After the negative parity nodes at depth two are expanded, we see that specific actions are needed to clobber the goal $\texttt{on}(B, C)$; if these actions are not taken, the goal is achieved. The plan set (9) is nonempty because it is indeed possible to instantiate the plan without clobbering the overall goal.

Note also that although we chose to fully expand the planning tree in this example, there was no intrinsic need for us to do so. If we had identified only some of the actions that might achieve or clobber a particular goal, we would still have been able to construct plan sets in a similar fashion, although the bounds we obtained on the set of plans that actually *do* achieve our goal would necessarily have been somewhat looser.

If we had been working with some other (and presumably more sophisticated) semantics of action, our overall approach would have been unchanged. It would still have been possible for us to construct a tree describing the actions that might achieve or clobber the various goals that were of interest to us, and we would be able to employ quite similar techniques to construct and to then evaluate a sentence in the planning language.

Finally, we remarked in the introduction that unlike conventional planners (POCL or otherwise), Procedure 5.1 plans for subgoals separately. What this means is that once a plan is found for a subgoal anywhere in the tree that is "successful" in that it can be incorporated into a successful plan for the goal at the root, the node corresponding to the subgoal need not be revisited.

Formalizing this is clearest if we assume that nodes with identical parities and labels have not been identified (a conventional planner would be incapable of identifying them in any event). As a preliminary, we make the following definition:

Definition 5.3 *Suppose that T_1 and T_2 are two planning trees with identical roots, and that both are trees as opposed to directed acyclic graphs. We will say that T_1 and T_2* match *at a node n appearing in each of them if (1) the path from n back to the root is the same in the two trees, and (2) the expansions of the two trees under n are identical.*

Proposition 5.4 *Suppose that T is a partially expanded planning tree for a goal g, and that n is a positive node in T. Now if there is any planning tree T' for g that matches T at n and for which $L^-(T') \neq \emptyset$, there is an expansion T_e of T with $L^-(T_e) \neq \emptyset$ such that T_e and T match at n.*

In other words, once we have found an expansion of a node n that participates in any solution of the original planning problem (as witnessed by the tree T'), we need never examine a node below n again.

6 IMPLEMENTATION

We have built a prototype implementation of our ideas that conforms to the theoretical description that we have presented. The planner is called COPS (COmbined Planner and Scheduler).

COPS accepts as input a goal and a set of action descriptors. Each descriptor includes the name of the action, its preconditions, add and delete lists, and a list of resources used by the action. COPS begins by building a planning tree using the mechanisms we have described, ignoring the resource information as it does so. The user can have the system construct the tree in its entirety, or can control the node expansions so that only the portion of interest is built.

Either after the planning tree is complete or while it is being constructed, the user can select any node in the tree for either liberal or conservative evaluation. A sentence in the planning language is then constructed and translated into an equivalent sentence in a finite, sorted first-order logic. This translation incorporates additional constraints that handle potential conflicts due to resource over-utilization by the various actions.

The first-order theory is converted into a (once again equivalent) predicate theory and passed to a predicate logic prover to check for satisfiability. The implementation uses a modification of TABLEAU [3] that incorporates procedural reasoning capabilities [11]. While the original sentence in the planning language directly captures only some of the ordering constraints on the actions involved (and therefore might best be thought of as partial-order), the theory in predicate logic describes conditions that linear plans need to satisfy in order to achieve the overall goals.

The performance of the system is as one would expect. Interactions corresponding to single actions establishing or disestablishing multiple subgoals correspond to entanglements in the planning tree much like those present in Figure 2; these entanglements are eventually resolved by the predicate prover. Interactions that are a consequence of resource contentions are handled by the predicate prover in isolation, with the structure of the planning tree remaining simple in this case.

In the future, we will work to extend COPS in a variety of ways. First, we will exploit the similarity between the planner's existing architecture and the theorem proving system MVL [7] so that actions can be described using declarative methods instead of a STRIPS-like representation. This will also allow us to treat actions with variables in their description; the current implementation is limited to the ground case.

We will also work to apply COPS to a variety of different domains, and to evaluate its scaling behavior in each. In cases where subgoal interactions correspond to resource contentions, we expect the system's scaling properties to match those of the underlying predicate prover/scheduler; in cases where the interactions correspond to multiple uses of single actions, COPS will doubtless suffer from combinatorial difficulties similar to those encountered by conventional methods. We expect these difficulties to be mediated by the facts that some of the complexity will be deferred to the first-order prover and COPS can recognize repeated subgoals independent of the contexts in which they appear.

7 CONCLUDING REMARKS

The COPS algorithm is only a beginning; there is, for example, no guidance with regard to the selection of the planning node to be expanded next. In this section, we draw connections between our ideas and others that may shed light on this or other issues.

7.1 EXISTING PLANNING WORK

We begin by briefly comparing our ideas and two existing planners, Joslin's DESCARTES [12] and Etzioni's STATIC [5, 6]. DESCARTES is in the POCL class, while STATIC is not.

The DESCARTES planner works to identify the constraint-satisfaction problems corresponding to action scheduling and to then schedule the actions using a dedicated constraint-satisfaction or scheduling algorithm of some sort [12]; by taking this approach, DESCARTES attempts to use the search facilities of the scheduling engine to minimize the search undertaken by the planner per se. This idea is similar to ours, although DESCARTES is capable of making tentative and unforced commitments when it appears that the associated constraint-satisfaction problem has become underconstrained. These "early commitments" appear to have no counterpart in our method.

In the absence of early commitments, the behavior of the two systems is strikingly similar. Both use plans for achieving single subgoals in a variety of contexts, and both suffer from the potential difficulty that problems repeatedly passed to the scheduler share many features with their predecessors. In DESCARTES's case, it would be useful if the results of prior scheduling searches could inform subsequent ones; in our case, the determination of whether $L^-(T) = \emptyset$ can be expected to be similar from one expansion of the planning tree to the next and it will clearly be important to exploit these similarities computationally.

The other connection we would like to draw is between our work and other attempts to combine identical subgoals in planning. At an intuitive level, the development of macro operators [13] is an attempt in this direction. By identifying specific sequences of actions that achieve common goals, the cost of planning can be reduced. Our approach realizes similar advantages if a single sequence of actions can be used to achieve a repeated goal multiple times and can also deal with situations where differing sequences of actions are needed. In this latter case, however, there is some computational cost to our methods, since both sequences will need to be evaluated in the check to see whether $L^-(g) = \emptyset$.

Another author who has attempted to identify recurring subgoals is Etzioni [5]. Etzioni's "problem space graphs" (PSG's) are strikingly similar to our planning trees, consisting as they do of "alternating sequence[s] of subgoals and operators" [5, page 262].[1] The PSG's are not used to solve any planning problem specifically, but instead capture the dependencies between goals and subgoals that are part of the overall problem formulation.

Etzioni describes as "fortuitous" recursions that are of a structural form that can be identified by an analysis of the PSG for the domain in question, and he shows that the blocks world is indeed fortuitous. Unfortunately, no criteria are presented whereby fortuitous recursions can be identified analytically.

We can do a bit better than this. If a node in a planning tree is identical to one of its ancestors, we can denote the label assigned to the node by $L(n)$ and then produce a fixed-point equation that $L(n)$ needs to satisfy. The actual value for $L(n)$ will be the smallest set of plans satisfying the equation. In many cases, it may be possible to show that the label for the ancestor node is in fact independent of the label for the child, in which case we will have shown that the recursion is fortuitous and the child can be pruned.

[1] Smith and Peot's *operator graphs* [15] are also similar, as Dan Weld has pointed out to me. The principal differences between PSG/operator graphs and our approach are that (1) PSG's (but not operator graphs) are constructed statically from the domain description, and not in response to a particular problem instance, and (2) PSG's and operator graphs both include information about goal establish-

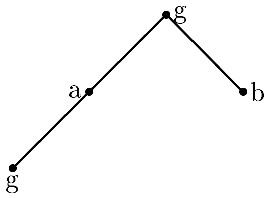

Figure 4: A fortuitous recursion

An example of this phenomenon appears in Figure 4. The goal g has no clobberers and can be achieved by either a or b. The preconditions for a are g, while b has no preconditions.

Suppose that we denote the eventual label for g by L. Now the label for a is $L|a$, while the label for b is $U|b$. The label for g at the root of the tree is therefore

$$\pi(L|a \cup U|b, \emptyset) = L \tag{10}$$

and it is not hard to see that the least solution to (10) is $L = \pi(U|b, \emptyset)$; g is achieved if we ever execute the action b, but in no other cases. In this case, we see that the repeated subgoal g can be pruned without affecting the label of the original. In fact, we can show:

Proposition 7.1 *Any subgoal with no clobberers in its subtree is fortuitous.*

7.2 BILATTICES AND MODALITY

The planning tree structure that we have introduced is similar in many ways to existing work on truth-functional interpretations of modality [8, 10]. Underlying this interpretation is the notion that sentences should be labeled not merely as "true" or "false" but instead with a richer set of truth values selected from a *bilattice* [7]. A bilattice is a set equipped with two partial orders, one indicating how true or false a particular sentence is, and the other partial order indicating how complete one's state of knowledge is.

In this setting, it is possible to respond to a modal query like, "I know it is raining outside," by invoking a theorem prover recursively on the embedded sentential argument ("It is raining outside") and then manipulating the truth value returned as a result [8]. One of the interesting features of naturally occurring modalities is that they tend to either preserve or to invert the "truth" partial order. As an example, "true" (t) is more true than "unknown" (u). The modal operator of knowledge (or necessity), applied to t, returns t. Applied to u, it returns f (false). We might write $K(t) = t$ and $K(u) = f$.

---

ment only, and ignore deletions and disestablishment.

With regard to the truth partial order, $t >_t u$ and $K(t) >_t K(u)$. The "knowledge" partial order is not preserved; $t >_k u$ but $K(t)$ and $K(u)$ are incomparable because neither corresponds to a more complete state of knowledge than the other. It is not clear why natural modalities should be either t-monotonic or t-antimonotonic in this way.

Now consider our planning language. A set of plans P can be thought of as a function ϕ from the set U of all plans into the two-point set $\{t, f\}$. A specific plan p is in P if and only if $\phi(p) = t$. The set of such functions can be interpreted as a bilattice by first embedding the set $\{t, f\}$ in the four element bilattice $F = \{t, f, u, \perp\}$ and then realizing that F^U, the set of functions from U to F, inherits a bilattice structure from the bilattice structure on F. Suppose that we denote this "planning" bilattice by $\mathcal{P}$.

An operator on sets of plans such as $|$ or π is now nothing more than a function that itself operates on functions from U to $\{t, f\}$. In other words, $|$ and π are binary functions from $\mathcal{P} \times \mathcal{P}$ to $\mathcal{P}$. As such, they can be interpreted as modal operators on the planning bilattice. The bilattice-based modal theorem prover [10] is, in fact, little more than a lifting of Procedure 5.1 to a more general setting. Once again, the modalities preserve or invert the t partial order. This is clear for $|$; increasing the set of plans in P or Q increases the set of plans in $P|Q$. The operator π, on the other hand, is monotonic in its first argument and antimonotonic in its second. Perhaps the planning results will shed some light on the bilattice theory here, since we know from results in this paper that the monotonicity and antimonotonicity underlie the convergence properties of the overall algorithms.

7.3 APPROXIMATE PLANNING

Finally, consider the termination criteria in Procedure 5.1: We stop as soon as $L^-(g) \neq \emptyset$, showing that the set of plans $L(g)$ for achieving the goal g is nonempty. There is an optional check to see if $L^+(g) = \emptyset$, in which case $L(g) = \emptyset$ as well and we can terminate and report that the goal is not achievable.

There are other possibilities also. One is to stop whenever the bracketing sets $L^+(T)$ and $L^-(T)$ are in some sense "close" to one another; this is similar to suggestions made in the work on *approximate planning* [9].

The basic idea here is to define conditions under which one set of plans is "small" relative to another. As an example, the set of plans $\pi(P|a, \emptyset)$ will typically be small in the set $\pi(P, \emptyset)$ since the first set involves a commitment to take the action a while the second set does not.

We can now go on to say that two sets $P \subseteq Q$ are *approximately equal*, writing $P \approx Q$, if their difference $Q - P$ is small relative to Q. Consider, for example, the plan

$$\pi(U|\texttt{move}(B, C), \emptyset) \tag{11}$$

for getting B onto C in the Sussman anomaly. In order for this plan to fail, one of a handful of specific actions must be taken: Something (A or C) needs to be moved onto B, something needs to be moved onto C, or B needs to be moved away after it is moved onto C successfully. Each of these "exception" plans is small relative to the overall plan schema (11), so (11) itself is approximately equal to the set of plans that actually achieve the goal.

We can modify our basic procedure to return these approximate plans by changing the termination condition from $L^-(T) \neq \emptyset$ to $L^-(T) \approx L^+(T)$. Doing so produces a procedure that implements the approximate planning ideas that have appeared elsewhere, although the complexity of checking whether $P \approx Q$ for $P \subseteq Q$ and $P, Q \in \mathcal{L}$ is not yet known.

All told, the ideas and algorithms that we have described seem to raise as many new questions as they answer. The new questions are, however, quite different from those that have typically been considered by the generative planning community. Perhaps the differences themselves are an indication of progress.

A PROOFS

Lemma 2.6 *U is the set of all linear plans. Negation and intersection have their conventional meanings.*

Proof. To see that $\langle a_1, \ldots, a_n \rangle \in U$ for any plan, note that the longest subsequence of $\langle a_1, \ldots, a_n \rangle$ in $[\,] \cup \emptyset$ is the initial empty subsequence, which is not in $\emptyset$.

For negation, Suppose that $\langle a_1, \ldots, a_n \rangle \in P$. Now it is in $U \cup P$, and also in P, so $\langle a_1, \ldots, a_n \rangle \notin \neg P = \pi(U, P)$. Alternatively, if $\langle a_1, \ldots, a_n \rangle \notin P$, it is in $U \cup P = U$ anyway, so $\langle a_1, \ldots, a_n \rangle \in \neg P$. The result for intersection follows from this. ■

Lemma 3.3 *Let $P \in \mathcal{L}$ be a nonempty set of plans. Then there is a plan $\langle a_1, \ldots, a_n \rangle \in P$ with $n \leq 1+|P|$.*

Proof. Let $\langle a_1, \ldots, a_k \rangle$ be a linear plan. We will show that there is some subset S of the indices $\{1, \ldots, k\}$ with the size of S at least $k - 1 - |P|$ and such that we can remove any subset of the actions in $\langle a_1, \ldots, a_k \rangle$ whose indices are in S without affecting membership in P. The result follows; given any element of P containing k actions, we can drop at least $k - 1 - |P|$ of them to get an element of P containing at most $1+|P|$ actions.

The proof is by structural induction on the set of plans P. For any of the base cases $\emptyset$, $[\,]$ or $a \in A$, the result is clear. (At most two actions are needed to resolve the question of membership in P, and $|P| = 1$.)

For the compound cases, consider $P|Q$ first. There is now a set S_p of appearances in P where actions can be dropped, and a set S_q of similar appearances in Q. We take $S = S_p \cap S_q$, so that membership in both P and Q is unaffected. Now the size of S is at least

$$\begin{aligned} |S_p| + |S_q| - k &\geq k - 1 - |P| + k - 1 - |Q| - k \\ &= k - 1 - (|P| + |Q| + 1) \\ &= k - 1 - |(P|Q)| \end{aligned}$$

since the size of $P|Q$ includes an additional count for the concatenation symbol. $\bigcup$ and π can be handled identically. ■

Lemma 3.4 *Let $P \in \mathcal{L}$ be a set of plans, and p a specific linear plan. Then the question of whether $p \in P$ can be resolved in time polynomial in the lengths of p and P.*

Proof. The only subtlety involves the clauses defining $P|Q$ and $\pi(P, Q)$ in Definition 2.3, since these clauses ostensibly involve a search over subsequences of p. We take a dynamic programming approach, investigating all of the subsequences of p simultaneously and working from the component plans of P outward.

Somewhat more specifically, suppose that Q is a set of plans, $p = \langle a_1, \ldots, a_n \rangle$, and for each top level plan Q_i appearing in Q and each $j \leq k \leq n$, we know whether $\langle a_j, \ldots, a_k \rangle \in Q_i$. Now we can determine in time $o(k)$ whether $\langle a_1, \ldots, a_k \rangle \in Q$. If Q is the concatenation of two plans, we consider each point at which the first plan might stop and the second begin. If Q is the result of applying π, we consider each initial subsequence of p starting with the longest (i.e., p itself).

Since there are at most n^2 subsequences of the plan p, it follows that we can determine membership in Q for each of them in time at most n^3 overall. If l is the length of the original set of plans P, it follows that we can determine in time $o(n^3 l)$ whether $\langle a_j, \ldots, a_k \rangle \in P$ for each j and k, and thus whether $p \in P$. ■

Lemma 3.7 *A theory C is satisfiable if and only if $p(C) \neq \emptyset$.*

Proof. It is clear from Definition 3.6 that the result will follow if we can show that for any atom x_j and linear plan $\langle x_1, \ldots, x_n \rangle$, $\langle x_1, \ldots, x_n \rangle \in p(x_j)$ if and only if $x_j \in \{x_i\}$ This is immediate from the definition, however. ■

Theorem 4.3 *Let T_0 be a planning tree, and T_1 an expansion of it. Suppose that the root of T_0 is labeled L_0^+ in a liberal labeling and L_0^- in a conservative one, and that the root of T_1 is labeled L_1^+ and L_1^-. Then*

$$L_0^- \subseteq L_1^- \subseteq L(g) \subseteq L_1^+ \subseteq L_0^+ \tag{12}$$

where g is the goal at the root of T_0.

Proof. The proof consists of two parts. In the first, we show that $L_0^- \subseteq L_1^-$; in the second, that $L_0^- \subseteq L(g)$. Since the argument is independent of the actual tree T_0, it will follow that $L_1^- \subseteq L(g)$ as well. The right hand inclusions in (12) follow using dual arguments.

To see that $L_0^- \subseteq L_1^-$, suppose that T is a planning tree, and T' an expansion of it. Suppose also that n is any node in T. If a conservative labeling of T labels n with P while a conservative labeling of T' labels n with P', we claim that $P \subseteq P'$ if the parity of n is positive, and $P \supseteq P'$ if the parity of n is negative.

We show this by induction from the fringe of the tree. For n an unexpanded node in T', the labels are identical in the two cases. If n is the node expanded to produce T' from T, the result follows immediately from the definition of a conservative labeling, since n will be labeled either with U (negative parity) or with $\emptyset$ (positive parity) in T.

For the inductive step, suppose that n is an internal node in the tree. If n is an action node, then the appropriate inequality holds for each of n's children by virtue of the inductive hypothesis, and therefore for n itself because of the definition (7). If n is a goal node, the result is a consequence of Lemma 2.4.

This completes the proof that $L_0^- \subseteq L_1^-$. To show that $L_0^- \subseteq L(g)$, we show the following:

1. The conservative labeling of T labels an action node $(a, +)$ with a plan that is a subset of the set of plans in which a has just been executed successfully.

2. The conservative labeling of T labels an action node $(a, -)$ with a superset of the set of plans in which a has just been executed successfully.

3. The conservative labeling of T labels a goal node $(g, +)$ with a subset of the set of plans in which g holds.

4. The conservative labeling of T labels a goal node $(g, -)$ with a superset of the set of plans in which g holds.

Once again, the proof proceeds by induction from the fringe of the tree; once again, that the above results hold for the fringe itself is obvious.

For an internal action node, the results hold because the set of plans in which a has just been executed successfully is exactly the intersection of the plans for which each of the preconditions of a holds, followed by the execution of the action a itself. For an internal goal node $(g, \pm)$, the set of plans for which g holds can be obtained by propagating the actions that successfully add or delete g. ■

Theorem 4.4 *Let g be a goal, and T_0 the planning tree consisting of the single goal node $(g, +)$. Now if p is a plan that achieves g, there is some expansion T of T_0 such that $p \in L^-(T)$. Conversely, if p fails to achieve g, there is an expansion such that $p \notin L^+(T)$.*

Proof. In fact, we show something stronger. Suppose that the length of p is k. Now we prove that if T is any tree with no unexpanded nodes shallower than depth $2k+2$, $p \in L^-(T)$ if and only if $p \in L^+(T)$. Since $L^-(T) \subseteq L(g) \subseteq L^+(T)$, the result will then follow.

The proof is by induction on k. If $k = 0$, so that $p = [\,]$, expanding the root node of T will produce an action node requiring *some* action (perhaps the initializing action i) to be taken to establish g. It follows that $[\,]$ will be in either both $L^-(T)$ and $L^+(T)$ or in neither.

For the inductive step, suppose that T has been fully expanded to depth $2k+2$. Now if n is a node in T of depth 2, we can apply the inductive hypothesis to conclude that L^- and L^+ agree on n for plans of length k or shorter. It follows that they will also agree on all (action) nodes of depth 1 and plans of length $k+1$ or less, since each of the depth 1 nodes requires the inclusion of a new final action. Since determining whether a plan p is in the set corresponding to the label for the root of the tree involves considering subsequences of p, the result for the root now follows from the result for the children at depth 1. ■

Proposition 5.4 *Suppose that T is a partially expanded planning tree for a goal g, and that n is a positive node in T. Now if there is any planning tree T' for g that matches T at n and for which $L^-(T') \neq \emptyset$, there is an expansion T_e of T with $L^-(T_e) \neq \emptyset$ such that T_e and T match at n.*

Proof. We can take T_e to be the union of T and T'. Since T_e is also an expansion of T', the result follows immediately from Theorem 4.3. ■

Proposition 7.1 *Any subgoal with no clobberers in its subtree is fortuitous.*

Proof. The situation is quite like that of Figure 4. Suppose that we label the repetition of the goal g with $\emptyset$ and that after doing so, conclude that $L(g) = P$. These are the plans that can be used for achieving g without recursion.

Now suppose we label the repeated appearance of g with L. Then the action node a_1 above g will be labeled with some subset of $L|a$. If the goal node above a_1 was previously labeled $Q = \pi(Q^+, \emptyset)$, it will now be labeled with a subset of

$$\begin{aligned} Q' &= \pi(Q^+ \cup L|a_1, \emptyset) = \pi(Q^+, \emptyset) \cup \pi(L|a_1, \emptyset) \\ &= Q \cup \pi(L|a_1, \emptyset) \end{aligned}$$

Continuing up the tree, if the path from the repeated occurrence of g back to the original one involves the actions $a_1, \ldots, a_n$, we see that the eventual label assigned to the root g will be a subset of

$$\pi(L|a_1|a_2|\ldots|a_n, \emptyset) \cup P$$

We therefore need to show that

$$\pi(P|a_1|a_2|\ldots|a_n, \emptyset) \cup P = P$$

or that

$$\pi(P|a_1|a_2|\ldots|a_n, \emptyset) \subseteq P$$

To see this, note that P itself is of the form $\pi(P^+, \emptyset)$ for some P^+, so that if we denote $\langle a_1, \ldots, a_n \rangle$ by a, we are trying to show

$$\pi(\pi(P^+, \emptyset)|a, \emptyset) \subseteq \pi(P^+, \emptyset) \tag{13}$$

But now let p be any element of the left hand side of (13). Since some initial subsequence of p is in $\pi(P^+, \emptyset)|a$, it follows that some initial subsequence of p is in $\pi(P^+, \emptyset)$, and therefore that some initial subsequence of p is in P^+. Thus p itself is an element of $\pi(P^+, \emptyset)$, and (13) follows as desired. The proof is complete. ■

Acknowledgment

This work has been supported by ARPA/Rome Labs under contracts F30602-91-C-0036 and F30602-93-C-00031. I would like to thank David Etherington, Oren Etzioni, David Joslin, Bart Massey, David McAllester, Dan Weld, and the members of CIRL for discussing these ideas with me.

References

[1] T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence*, 69:165–204, 1994.

[2] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–377, 1987.

[3] J. M. Crawford and L. D. Auton. Experimental results on the crossover point in random 3sat. *Artificial Intelligence*, 81:13–59, 1996.

[4] K. S. Erol, D. S. Nau, and V. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76:75–88, 1995.

[5] O. Etzioni. Acquiring search-control knowledge via static analysis. *Artificial Intelligence*, 62:255–302, 1993.

[6] O. Etzioni. A structural theory of explanation-based learning. *Artificial Intelligence*, 60:93–140, 1993.

[7] M. L. Ginsberg. Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316, 1988.

[8] M. L. Ginsberg. Bilattices and modal operators. *Journal of Logic and Computation*, 1:41–69, 1990.

[9] M. L. Ginsberg. Approximate planning. *Artificial Intelligence*, 76:89–123, 1995.

[10] M. L. Ginsberg. Modality and interrupts. *J. Automated Reasoning*, 14:43–91, 1995.

[11] A. K. Jonsson and M. L. Ginsberg. Efficient reasoning using procedures. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1996.

[12] D. Joslin and M. Pollack. Passive and active decision postponement in plan generation. In *European Workshop on Planning*, 1995.

[13] R. E. Korf. Macro-operators: A weak method for learning. *Artificial Intelligence*, 26:35–77, 1985.

[14] B. Selman. Near-optimal plans, tractability, and reactivity. In *Proceedings of the Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Bonn, Germany, 1994.

[15] D. E. Smith and M. A. Peot. Postponing threats in partial-order planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 500–506, 1993.