Approximate Planning

Matthew L. Ginsberg

CIRL
1269 University of Oregon
Eugene, OR 97403
ginsberg@cs.uoregon.edu

Approximate Planning

Abstract

This paper makes two linked contributions. First, we argue that planning systems, instead of being correct (every plan returned achieves the goal) and complete (all such plans are returned), should be *approximately* correct and complete, in that most plans returned achieve the goal and that most such plans are returned. The first contribution we make is to formalize this notion.

Our second aim is to demonstrate the practical importance of these ideas. We argue that the cached plans used by case-based planners are best thought of as approximate as opposed to exact, and also show that we can use our approach to plan for subgoals g_1 and g_2 separately and to combine the plans generated to produce a plan for the conjoined goal $g_1 \wedge g_2$. The computational benefits of working with subgoals separately have long been recognized, but attempts to do so using correct and complete planners have failed.

1 Introduction

When we talk about a plan for achieving a goal, we typically mean not one plan but many. As an example, if I say on Thanksgiving that my plan for preparing a turkey involves stuffing and roasting it, I hardly mean that these are the only actions I will take between now and when the turkey is done. I may also plan on making sweet potatoes and pumpkin pie, buying a bottle of wine, calling family members to wish them happy holidays, and other actions even further removed from my stated goal of turkey preparation.

In fact, my plan "stuff the turkey and then roast it" might be represented something like this:

$$[\ldots\texttt{stuff}\ldots\texttt{roast}\ldots] \tag{1}$$

where the ellipses denote currently undetermined action sequences that I might intersperse into the above plan. If I need to roast the turkey *immediately* after stuffing it, I might write that as

$$[\ldots\texttt{stuff roast}\ldots] \tag{2}$$

where the second set of ellipses has been dropped.

There are, of course, many instance of (1) that are unsatisfactory. Perhaps I run the turkey through a paper shredder before beginning preparation, or unstuff it after stuffing it, or garnish it liberally with peanut butter before serving. In what sense can we say that (1) is our plan when so many things can go wrong?

The conventional approach to this problem is to deal not with plans such as that appearing in (1), but with far more specific plans such as

$$[\texttt{stuff yams telephone roast eat}] \tag{3}$$

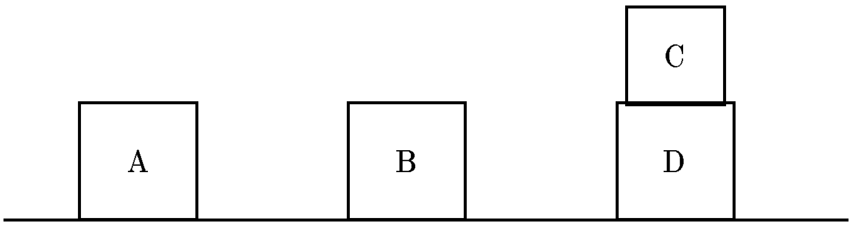

Figure 1: Get A on B on C without building a 4-block tower

where there are guaranteed to be no extraneous actions that might interfere with our achieving our goal. But from a practical point of view, the plan (3) is nearly worthless, since it is almost inconceivable that I execute it exactly as written.

There are many other examples of this phenomenon. If we intend to construct plans by retrieving them from a library of known solutions to similar problems (so-called *case-based* planning [14]), it is important that the plans in the library include some measure of flexibility. After all, it is unlikely that the new situation in which we find ourselves will be an exact match for the situation in which the plan was constructed.

Our ability to plan for conjunctive goals rests on similar ideas. When possible, it is important that we plan for conjuncts separately and then merge the results; this appears to require that the solutions to the individual conjuncts be plan schemas such as (1). Planning for conjuncts separately enables us to take computational advantage of the benevolence of our environment as reflected in the frame assumption – we can typically achieve one subgoal and then not worry about it while we work on other things.

Another example of a conjunctive planning problem appears in Figure 1. The goal is to get A on B and B on C, but there is a restriction to the effect that one cannot build a four-block tower.

For a human planner, the problem is easy. We realize that the general plan for getting B onto C is simply to move it there, and similarly for getting A on B. When we combine these two plans, however, we encounter a problem – the action of moving A to B will fail. We therefore modify the plan for getting B onto C, adding the additional action of moving C to the table.

I presented this problem to the authors of two generative planning systems – Minton (PRODIGY [17]) and Wilkins (SIPE [21]). Both reported (personal communication) that the problem would pose no significant difficulties for them and that they could solve it by adding an additional precondition to the action $\texttt{move}(x, y)$ to the effect that y had to be either on the table or on a block z that was on the table.[1]

The problem with this approach is that it doubles the branching factor for all planning problems. This will lead to prohibitive computational difficulties as the problems involved get larger; imagine having to move a block prior to constructing a 13-block tower in a domain that prohibits 14-block ones. As an example of the immediacy of these difficulties, Penberthy

[1] Wilkins made the alternative suggestion of creating two move operators. This is equivalent in practice, however; doubling the branching factor by introducing a second move operator is equivalent to doubling it by introducing a disjunction into the precondition.

and Weld's UCPOP system [18] proved incapable of solving the 4-block version of the problem in Figure 1 without the inclusion of domain-specific control information.[2]

Worse still is the fact that the branching factor is being increased on *all* problems, not just those that involve tall towers. Imagine, for example, that we can only put a blue block on a red one if the red block is on the table. The branching factor will still be doubled even if we are working in a domain without blue blocks![3]

Explicit control rules provide potential ways around these particular difficulties, but their use is problematic. What control rule are we to use if the previous domain includes painting actions, so that the colors of blocks can change? What control rule would allow us to efficiently solve the problem in Figure 1 if the constraint were changed so that only *five*-block towers were prohibited?

Related problems appear in plan debugging. If a human planner discovers a bug in one portion of a plan to achieve a complex goal, the typical response is to restrict the impact of the bug to a small portion of the analysis and to then plan around the problem. That we can make modifications that address the bug without destroying the effect of the original plan depends on our commonsense ability to construct and manipulate plans like (1) – plans that, while not holding universally, do hold in general.

My intention in this paper is to develop a formalization of the ideas that are implicit in the plan (1), and to then describe the use of these constructs in conjunctive planning. Please bear with me while we work through the mathematics, since there are a variety of fundamentally new ideas that we need to formalize.

1. We first need to describe plans that can have new actions added to them in arbitrary ways but that can still include the immediacy requirements of a plan such as (2). This is our goal in the next section, where we also present a variety of mathematical results about these new plans that will be needed later.

2. We next need to define conditions under which a plan approximately achieves a goal. The basic idea here is that a plan P is approximately correct if most instances of P that could actually be executed do indeed achieve the goal. We formalize this in Section 3 by introducing the idea of an exception to a plan and formalizing conditions under which plans hold sufficiently frequently that we are prepared to treat them as approximately correct.

3. The problem of building a planner around these ideas is discussed in Sections 4 and 5. Section 4 discusses the theoretical issues involved in the construction of the planner, showing that it is indeed possible to plan for conjuncts separately using our ideas. Section 5 discusses a preliminary implementation of our work.

[2]Will Harvey, personal communication. The problem is not one of time, but of space; UCPOP reported that it had exhausted its available memory while working on this problem.

[3]This is assuming that we treat color as a precondition and not as a filter. We would need to do this if there were actions available that changed blocks' colors.

4. Concluding remarks are contained in Section 6, and proofs have been deferred to an appendix.

Let me end this introduction with something of a disclaimer. I do not mean to imply that existing implemented systems are incapable of manipulating expressions such as (1). Tate's O-Plan system, for example [2, 20], appears to use ideas such as these routinely. But planners that behave in this fashion have thus far lacked formal foundation, and correcting *that* is my intention here. In providing a solid formal foundation for nonlinear planning, McAllester and Rosenblitt's paper [16] was both a step forward and a step back; although it formalized many ideas that had previously eluded precise description, it omitted many of the procedural tricks that make implemented planners effective. As a result, formally well-grounded planners such as that described by Penberthy and Weld [18] typically exhibit performance far worse than that of the informal systems that preceded them. My hope here is to shed some formal light on the ideas that have proven so effective in practice.

2 Plans

I will adopt the view that a plan is a partially ordered collection of actions, where an action is a functional expression such as $\texttt{move}(a, b)$:

Definition 2.1 *An* action *is either a variable or a functional expression, where the arguments to the function may themselves include variables. A* ground action *is an action that contains no variables.*

By an action such as $\texttt{move}(a, ?)$ we will mean the action of moving a to the location ? where ? will presumably be determined by other considerations.

We cannot now simply define a plan to be a partially ordered sequence of actions, since we need to be able to distinguish between (1) and (2). In some cases, we will want the action a to precede b *immediately*, while in others, there may be many actions interspersed between the two. We handle this as follows:

Definition 2.2 *A* plan *is a triple $(A, \leq, *)$ where A is a finite collection of actions and $\leq$ is a partial order on A; by $a \leq b$ for actions $a, b \in A$ we mean that a must precede b. $*$ is another binary relation on A with $* \subseteq <$, so that whenever $a * b$, $a < b$ as well. We will assume that $*$ and $\leq$ also satisfy the following conditions:*

1. *If $c * a$ and $c < b$, then $a \leq b$.*

2. *If $b * c$ and $a < c$, then $a \leq b$.*

We will take $a * b$ to mean that b is a successor of a for which no intermediate actions are permitted.

Note that the definition refers to A as a collection instead of as a set; this is to allow the same action to appear multiple times in the partial order. To understand the additional conditions, suppose that a is an immediate successor of c, so that $c * a$. Now if b is some other successor of c, then a must precede b as well. The second condition is the dual of the first.

In practice, plans are bounded by initial and terminal actions. We model this by requiring that plans contain dummy initial and terminal actions denoted by d_i and d_t respectively:

Definition 2.3 *A plan $(A, \leq, *)$ will be said to be* bounded *if $d_i \in A$ and $d_t \in A$ with $d_i \leq a$ and $d_t \geq a$ for all $a \in A$.*

We will assume throughout this paper that all plans are bounded.

The general turkey-roasting plan (1) corresponds to the partial order

$$d_i < \texttt{stuff} < \texttt{roast} < d_t$$

while the plan that roasts the turkey immediately after stuffing it corresponds to

$$d_i < \texttt{stuff} * \texttt{roast} < d_t$$

The second inequality has been made an instance of $*$.

Before proceeding, let me spend a moment discussing the difference between $*$, our annotation for the links in a partially-ordered plan, and the causal annotations introduced by McAllester and Rosenblitt [16].

McAllester and Rosenblitt's links serve more a bookkeeping function than anything else; the information they contain (which action is intended to achieve which precondition) could be recovered, if need be, from the plan being constructed. Recording the information on the arcs serves the computational purpose of making the plans more efficient to work with.

Our $*$ annotation is different. Annotating an arc with $*$ makes the associated plan a semantically different object, in the sense that we have added a new constraint to the set of linearizations of the given plan. Note also that our language is fundamentally more flexible than McAllester and Rosenblitt's – we allow the addition of arbitrary new actions to plans, while he does not. This is important, since it enables us to work with the flexible plan (1) instead of the far more restrictive (3).

Lemma 2.4 *If $(A, \leq, *)$ is a plan, then for any $a, b, c \in A$:*

1. *If $a * b$ and $a * c$, then $b = c$.*

2. *If $a * c$ and $b * c$, then $a = b$.*

An action can have at most one immediate predecessor or successor.

Suppose now that we have some plan $(A, \leq, *)$. If there is a chain

$$c = c_0 * \cdots * c_n = a$$

and $c \leq b$, then b must either be one of the c_i's or it must follow a; there is no other "room" between c and a. The following lemmas capture this, where we have written $\overline{*}$ for the transitive closure of $*$:

Lemma 2.5 *Let $(A, \leq, *)$ be a plan. Then for any $a, b, c \in A$:*

1. *If $c \leq b$ and $c \overline{*} a$, then either $c \overline{*} b$ or $a \leq b$.*
2. *If $b \leq c$ and $a \overline{*} c$, then either $b \overline{*} c$ or $b \leq a$.*

Lemma 2.6 *Let $(A, \leq, *)$ be a plan. Then for any $a, b, c \in A$, if $a \overline{*} b$ and $a \leq c \leq b$, then $a \overline{*} c$.*

It is also straightforward to define conditions under which two plans are equivalent or one is an instance of the other:

Definition 2.7 *Two plans P_1 and P_2 will be called* equivalent *if they are identical up to variable renaming.*

Definition 2.8 *A plan $(A_1, \leq_1, *_1)$ is an* instance *of another plan $(A_2, \leq_2, *_2)$ if there is a binding list σ and a 1-1 mapping $i : A_2 \to A_1$ with the following properties:*

1. *For each $a \in A_2$, $i(a) = a|_\sigma$. In other words, the mapping i maps a to an action that is the same as that constructed by applying the bindings in σ to a.*
2. *$i(\leq_2) \subseteq \leq_1$. Every ordering constraint on the second plan appears in the first as well.*
3. *$i(*_2) \subseteq *_1$.*

An example will probably help. If the `stuff` and `roast` actions accept an argument but the `eat` action doesn't,

$$[\ldots \texttt{stuff(turkey)}\ \texttt{roast(turkey)} \ldots \texttt{eat}] \tag{4}$$

is an instance of

$$[\ldots \texttt{stuff(?)}\ \texttt{roast(?)} \ldots] \tag{5}$$

The plan (4) corresponds to the partial order

$$d_i < \texttt{stuff(turkey)} * \texttt{roast(turkey)} < \texttt{eat} * d_t \tag{6}$$

while (5) corresponds to

$$d_i < \texttt{stuff(?)} * \texttt{roast(?)} < d_t \tag{7}$$

The required injection is now given by

$$i(x) = \begin{cases} x, & \text{if } x = d_i \text{ or } x = d_t; \\ \texttt{stuff(turkey)}, & \text{if } x = \texttt{stuff(?)}; \\ \texttt{roast(turkey)}, & \text{if } x = \texttt{roast(?)}. \end{cases}$$

and the binding list σ binds `?` to **`turkey`**.

It is clear that for each action x, $i(x) = x|_\sigma$. The image of $\leq$ in (7) under i is the partial order

$$d_i < \texttt{stuff(turkey)} < \texttt{roast(turkey)} < d_t$$

and is clearly included in the partial order of (6); the image of $*$ under i contains the single pair

$$\texttt{stuff(turkey)} * \texttt{roast(turkey)}$$

and is once again contained in the $*$ of (6).

Proposition 2.9 *The instance relation of Definition 2.8 is a partial order.*

We will write $P_1 \subseteq P_2$ to denote the fact that a plan P_1 is an instance of another plan P_2.

We have been careful in our definitions not to restrict the number of new actions that can be inserted between any two actions of the plan itself. When it comes time to actually execute the plan, however, we will need to select a specific action sequence.

Definition 2.10 *A plan $P = (A, \leq, *)$ will be called* linear *if the following conditions hold:*

1. *Every action in A is ground.*

2. $\overline{*} = \leq$.

A linear plan that is an instance of a plan P will be called a linearization *of P.*

In other words, a linearization of a plan replaces all of the variables with object constants and selects an ordering of the actions involved that can be derived solely from the immediacy conditions of $*$. This latter condition implies that no additional actions can be added to the plan.

As an example, the linear plan (3) corresponds to the partial order

$$d_i * \texttt{stuff} * \texttt{yams} * \texttt{telephone} * \texttt{roast} * \texttt{eat} * d_t$$

There is no way to add another action to this ordering without having it either precede d_i, follow d_t, or violate the conditions of Definition 2.2.

Lemma 2.11 *If $P = (A, \leq, *)$ is a linear plan, then $\leq$ is a total order.*

Proposition 2.12 *$P_1 \subseteq P_2$ if and only if the set of linearizations of P_1 is a subset of the set of linearizations of P_2.*

Given the above result, it makes sense to think of a plan in terms of its linearizations; each linearization is a way in which we might actually go about executing the actions in the plan.

Definition 2.13 *A* plan set *is a set of linear plans.*

3 Approximate correctness

Given now that there will almost inevitably be mistakes we can make, in the sense that there are linearizations of a given plan that do not actually achieve our intended result, how can we formalize the idea that the plan in (1) is correct "in general"?

The solution we will use is an extension of an idea I proposed in 1991 [10]. As an example, suppose that I am trying to get block A onto block B in the blocks world. A is clear, but C is currently on top of B and a wide variety of other blocks are scattered around the table. Here is my plan for achieving the goal:

$$[\texttt{move}(C, ?)\ \texttt{move}(A, B)] \tag{8}$$

I plan to move C out of the way, and then move A onto B. I've assumed just for the moment that no additional actions can be added; our interest here involves the variable ? that appears in (8).

Note that there is one location to which we should *not* move the block currently on top of B – if we relocate it to A, B will become clear but A no longer will be. Given this, in what sense is (8) a solution to our problem?

It is a solution in that there are many places to which we can move C, and the one that doesn't work is in some sense pathological – most locations *do* work. What we need to do is to capture the way in which the set of exceptions is small relative to the set of possibilities.

From a formal point of view, the exception involves a specific binding for the variable ?. This leads us to the following:

Definition 3.1 *Given a binding list σ and a plan $P = (A, \leq, *)$, the result of* applying σ to *P is defined to be that plan where the actions in A have had the binding list applied to them but the plan is otherwise unchanged. This plan will be denoted $P|_\sigma$.*

We can, for example, bind ? to A in (8) to obtain

$$[\texttt{move}(C, A)\ \texttt{move}(A, B)]$$

The following result is obvious:

Lemma 3.2 *Given a plan P and binding list σ, $P|_\sigma \subseteq P$.* ∎

We are now in a position to describe conditions under which one set of plans is "small" relative to another. We need to be careful, however, since plans generally have infinitely many linearizations and we can't simply say that Q is small relative to P if Q has many fewer linearizations than P does. Instead, we will say that Q is small relative to P if $Q = P|_\sigma$ but $Q \neq P$. The motivation behind this definition is that there are generally many ways to bind any particular variable and Q is committed to a specific choice.

In the following definition, we will say that Q is *of measure 0 in P* instead of simply saying that Q is small relative to P. The term is borrowed from real analysis, and we use it because the formal definition of smallness has many of the same properties as does the analytic definition on which it is modelled. (The finite union of small sets is small, for example.) The 0 means that the ratio of the size of Q to that of P is approximately 0; we will also say that Q is "of measure 1" in P if this ratio is approximately 1, so that Q and P are comparably sized.

Definition 3.3 *A plan Q will be said to be* of measure 0 *in a plan P if $Q \neq P$ but $Q = P|_\sigma$ for some binding list σ. A plan or plan set Q will also be said to be* of measure 0 *in a plan or plan set P if either of the following conditions is satisfied:*

1. *Q is the finite union of sets of measure 0 in P.*
2. *There exist plan sets R and S with $Q \subseteq R$ and both R and $S - P$ of measure 0 in S.*

The requirement that $Q \neq P$ ensures that the binding list σ is not trivial.

The second condition in the definition handles cases where Q is a subset of a set of measure 0 in P (take $S = P$), or where, for example, one specific linearization has been removed from P. Adding that linearization back to P should not impact the question of whether Q is of measure 0 in P (take S to be the union of P and the missing linearization).

As an example, the plan

$$[\ldots \texttt{move}(a, b) \ldots] \tag{9}$$

is of measure 0 in the plan

$$[\ldots \texttt{move}(a, ?) \ldots] \tag{10}$$

since the variable ? has been bound to a constant in (9). But the plan

$$[\ldots \texttt{move}(a, b)] \tag{11}$$

is *not* of measure 0 in the plan (9), since the difference between the two plans is not the binding of a variable but the fact that the action $\texttt{move}(a, b)$ is the final action in (11) (i.e., an immediate predecessor of the plan's terminal action) but not in (9). For similar reasons, a plan where two actions a_1 and a_2 are sequential is not of measure 0 in the plan where the actions are unordered.

Since (9) is of measure 0 in the plan set (10) and (11) is an instance (i.e., a subset) of (9), (11) is of measure 0 in (10) as well. Finally, (9) is also of measure 0 in the plan set given by removing

$$[\ldots \texttt{move}(a, c) \ldots] \tag{12}$$

from (10). After all, if binding ? to b reduces us to a set that is small relative to the set of all possible bindings, removing a single one of those possible bindings in advance shouldn't change this conclusion. The second condition in Definition 3.3 allows us to continue to measure the size of a plan set relative to (10) even after (12) has been removed.

What about adding new actions to a plan? If we add variable actions, the result will in general not be of measure 0 in the original plan; we are only committing to doing "something" and most of the linearizations of the original plan include additional actions of one form or another in any event. But if we add a specific action, the story is different:

Proposition 3.4 *Let P be a plan, and P' an instance of P with i the associated injection from A to A'. Then if there is any action in $A' - i(A)$ that is not a variable, P' is of measure 0 in P.*

Definition 3.5 *A plan set Q will be said to be* of measure 1 *in P if $P - Q$ is of measure 0 in P. Two plans sets will be called* approximately equal *if each is of measure 1 in the other.*

Lemma 3.6 *Q is of measure 0 in P if any of the following conditions holds:*

1. *Q is empty.*
2. *Q is a subset of a set of measure 0 in P.*
3. *Q is of measure 0 in a subset of P.*
4. *Q is of measure 0 in a superset S of P with P of measure 1 in S.*

Lemma 3.7 *Q is of measure 0 in P if and only if Q is of measure 0 in $P \cup Q$.*

Proposition 3.8 *Approximate equality is an equivalence relation.*

We also have the following:

Proposition 3.9 *Let $P \neq \emptyset$ be a plan set. Then provided that our language includes infinitely many object and action constants, there is no plan set that is both of measure 0 and of measure 1 in P.*

It is this result that gives teeth to the ideas we are proposing; if there were a plan of both measure 0 and measure 1 in P, we would be able to return as "generally correct" plans that in fact failed for large fractions of their linearizations.

The requirement that there be infinitely many constants is a necessary one. If, for example, there were only 37 places to which an object could be moved, we could use the fact that each specific choice is of measure 0 in the overall plan to conclude that the union of all of them was – thereby violating Proposition 3.9. Similarly, if the set of actions we could take were circumscribed in some way, we could use Proposition 3.4 to find a counterexample to the above proposition.

Finally, we present some technical results that we will need later. We begin by recalling the usual definition of convergence for a sequence S_i of sets:

Definition 3.10 *Let S_i be a sequence of (plan) sets. Then we will say that the S_i* converge *to a set S if for any x, there is some index $m(x)$ such that for $i > m(x)$, $x \in S_i$ if and only if $x \in S$.*

Lemma 3.11 *Suppose we have a sequence P_i of plan sets, where P_{i+1} is of measure 0 in P_i for each i. Then the P_i converge to the empty set.*

Every infinite descending chain where each element is of measure 0 in the previous one converges to the empty set.

It is *not* the case that there is no infinite descending chain of plan sets, each of measure 0 in the previous one. For any function constant f, we can get such a chain by considering

$$[\mathtt{move}(x, ?)] \supset [\mathtt{move}(x, f(?))] \supset [\mathtt{move}(x, f(f(?)))] \supset \cdots$$

Lemma 3.12 *Suppose S is of measure 1 in T and of measure 0 in U. Then T is of measure 0 in U.*

Suppose that we denote by $A \ominus B$ the symmetric difference of A and B, so that

$$A \ominus B = (A - B) \cup (B - A)$$

We now have:

Proposition 3.13 *Suppose that there is some set D that is of measure 1 in $A \ominus B$ and of measure 0 in A. Then A and B are approximately equal.*

4 Planning

Having introduced these notions, we need to use them to construct a planner. Before doing so, however, let me be clear about the problem that I am hoping to address. Our focus here is on planning itself, as opposed to reasoning about action or simulation. In other words, we will assume that the semantics of actions are somehow provided to us; somewhat more specifically, we assume that given a linear plan L and a goal g, we have some way to tell whether or not g holds after L is executed. From a formal point of view, we will assume that given a goal g, we can take $L(g)$ to be the set of all linear plans that achieve g. The analysis we are about to present is independent of the specific semantics of action underlying the function L.

In the examples, of course, we will need to rely on a specific semantics of action. For the blocks world, this semantics is presumably intuitive and corresponds to the usual STRIPS description. The only difference between our interpretation and the conventional one is that we need some way to interpret actions that are attempted even though their preconditions are not satisfied; we will take the view that such actions simply have no effect on the domain in question.

We now make the following definition:

Definition 4.1 *A* planning system $\mathcal{P}$ *accepts as input a goal g and a plan set P. It returns a plan set $\mathcal{P}(g, P) \subseteq P$ that is approximately equal to $L(g) \cap P$. In some cases, we will assume that the goal is fixed, writing $\mathcal{P}_g$ for the corresponding function that accepts the plan set P only.*

The plan set P can be used to focus the planner's attention on plans of a particular form. The condition that $\mathcal{P}(g, P)$ be of measure 1 in $L(g) \cap P$ means that almost all – but not necessarily all – of the plans in P that would achieve g are actually returned by the planner. In more picturesque terms, the planner is "approximately complete."

The condition that $L(g) \cap P$ be of measure 1 in $\mathcal{P}(g, P)$ means that almost all the plans returned by the planner achieve the goal; in other words, the planner is approximately correct. In a situation like this, where $L(g)$ is of measure 1 in a plan set P, we will often say that P "generally achieves" g.

In the remainder of this section, we will begin by discussing planning systems in general, describing implementation concerns that are likely to arise in their construction. There are then two technical issues that we will address. First, we will show that a planning system can be used to produce answers to planning queries that actually *are* correct and complete, at least in the limit. More precisely, we will show how a planning system can be used to construct a sequence of answers that converges on the actual set $L(g)$. Second, we will show how a planning system can respond to a conjunctive goal $g_1 \wedge g_2$ by invoking itself only on the subgoals g_1 and g_2 separately and then combining the results. More precisely, we will show how $\mathcal{P}_{g_1 \wedge g_2}$ can be constructed from $\mathcal{P}_{g_1}$ and $\mathcal{P}_{g_2}$.

We begin by discussing $\mathcal{P}_g$ itself. On an intuitive level, the way $\mathcal{P}_g$ works is as follows: Given the goal g, we find the actions a that might succeed in establishing g. If the preconditions to these actions are in general satisfied (perhaps because these preconditions hold in the initial situation), we can take $\mathcal{P}(G)$ to be the union of plan sets of the form

$$[\ldots a \ldots] \tag{13}$$

for each action a achieving g.

If the preconditions of a are not in general satisfied, we can invoke $\mathcal{P}$ recursively on each precondition, merge the results to obtain plans that enable a, and then append a to the end of such plans. The result (13) is in fact a special case of this observation; if the preconditions to a are known to hold in the initial state $[\,]$, these preconditions also hold in a plan set that is approximately equal to $[\ldots]$ (the set of all plans). The expression (13) is simply the result of appending a to the end of such plans.

We will see in what follows that we are often interested not only in $\mathcal{P}_g$, which constructs plans for achieving g, but also in $\mathcal{P}_{\neg g}$, which tells us which elements of a particular plan set *fail* to achieve g. This has obvious analogs in existing planners:

1. In a system like TWEAK [1], the exceptions involve finding what Chapman calls *clobberers.* New actions that make the plan work after all are called *white knights.*

2. McAllester and Rosenblitt [16] describe potential flaws in a plan (where one action might overturn the consequences of another) as *threats.* Overcoming the threats involves adding new actions to the plan that ensure that the consequences of the action hold after all.

3. Finally, we will discuss in Section 5 the use of a declarative system to construct $\mathcal{P}_g$ and $\mathcal{P}_{\neg g}$.

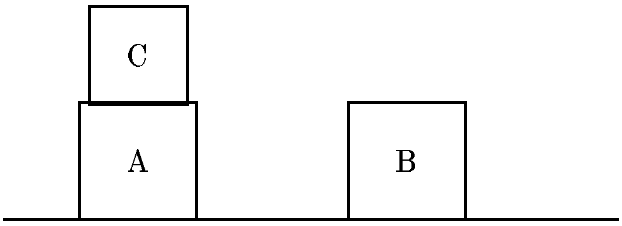

Figure 2: The Sussman anomaly

Before turning to technical issues, let us look at an example in a bit more detail. The problem we will consider is the well-known Sussman anomaly [19], shown in Figure 2. The goal is to get A on B and B on C. At this point, we consider the subgoals separately.

The first of these involves simply getting B on C. This is generally achieved by the plan

$$[\ldots \mathtt{move}(B, C) \ldots] \tag{14}$$

Although there are instances of (14) that do not succeed in getting B on C, there are only a finite number of ways for this to happen – something must be put on B or on C, or B has to be moved away from C at the end of the plan. Each of these exceptions is of measure 0 in (14), which is why the plan (14) generally achieves $\mathtt{on}(B, C)$. Furthermore, $\mathtt{move}(B, C)$ is the only action with $\mathtt{on}(B, C)$ in its add list, and the preconditions to this action hold in the initial situation. This implies that the plan set of (14) is approximately equal to the set of all plans for getting B onto C and we can take

$$\mathcal{P}(\mathtt{on}(B, C), [\ldots]) = [\ldots \mathtt{move}(B, C) \ldots] \tag{15}$$

The two conditions of approximate equality are satisfied: Most plans that achieve the goal are instances of the above plan (in fact, they all are), and the exceptions are a set of measure 0 in (15).

To continue the analysis, we compute

$$\mathcal{P}(\neg \mathtt{on}(B, C), [\ldots \mathtt{move}(B, C) \ldots])$$

in order to determine which elements of (15) are exceptions to the plan. There are two ways in which such exceptions might arise: One of the preconditions to the move action might fail to hold, or something might clobber the fact that B is on C after the action is executed.

The action $\mathtt{move}(B, C)$ has two preconditions, that B be clear and that C be. B is clear in the initial situation, and the most general plan that clobbers this fact is

$$[\ldots \mathtt{move}(?, B) \ldots]$$

It follows that the plan (14) will fail for the instance

$$[\ldots \mathtt{move}(?, B) \ldots \mathtt{move}(B, C) \ldots] \tag{16}$$

There are still more specific plans that do manage to get B on C, but (16) is one general failure possibility. (Recall that once we move something onto B, we are assuming that the failed action of moving B to C simply has no effect on the locations of the blocks.) Another way for (14) to fail is given by

$$[\ldots \mathtt{move}(?, C) \ldots \mathtt{move}(B, C) \ldots] \tag{17}$$

where something is moved onto the top of C.

The only remaining possibility is where B is not on C at the end of the plan because it is moved away. Combining this with (16) and (17), we see that we can take

$$\mathcal{P}(\neg \mathtt{on}(B, C), [\ldots \mathtt{move}(B, C) \ldots]) = e_1 \cup e_2 \cup e_3$$

to be the union of the following three sets of exceptions:

$$\begin{aligned} e_1 &= [\ldots \mathtt{move}(?, B) \ldots \mathtt{move}(B, C) \ldots] \\ e_2 &= [\ldots \mathtt{move}(?, C) \ldots \mathtt{move}(B, C) \ldots] \\ e_3 &= [\ldots \mathtt{move}(B, C) \ldots \mathtt{move}(B, ?) \ldots] \end{aligned} \tag{18}$$

Each of these sets is of measure 0 in (15), as is their union.

If we wish, we can continue the process, computing

$$\mathcal{P}(\mathtt{on}(B, C), e_1 \cup e_2 \cup e_3)$$

to find those elements of the exception set that achieve the goal after all, and so on. As we will see shortly, a sequence constructed in this fashion will eventually converge on the set of all plans that achieve the goal of getting B on C.

The second goal $\mathtt{on}(A, B)$ is more complicated, but only slightly so. It is generally achieved by

$$[\ldots \mathtt{move}(C, ?) \ldots \mathtt{move}(A, B) \ldots] \tag{19}$$

Once again, there are only finitely many ways for (19) to fail – the binding for `?` could be chosen poorly (A and B are bad choices), or additional actions could be added as in the previous case. (In keeping with Proposition 3.9, we are assuming that there are an infinite number of distinct locations on the table to which C could be moved.) The complete list of exceptions is as follows:

$$\begin{aligned} f_1 &= [\ldots \mathtt{move}(?1, C) \ldots \mathtt{move}(C, ?) \ldots \mathtt{move}(A, B) \ldots] \\ f_2 &= [\ldots \mathtt{move}(?1, ?) \ldots \mathtt{move}(C, ?) \ldots \mathtt{move}(A, B) \ldots] \\ f_3 &= [\ldots \mathtt{move}(C, ?) \ldots \mathtt{move}(?1, A) \ldots \mathtt{move}(A, B) \ldots] \\ f_4 &= [\ldots [\mathtt{move}(?1, B) \ \&\ \mathtt{move}(C, ?)] \ldots \mathtt{move}(A, B) \ldots] \\ f_5 &= [\ldots \mathtt{move}(C, ?) \ldots \mathtt{move}(A, B) \ldots \mathtt{move}(A, ?1) \ldots] \\ f_6 &= [\ldots \mathtt{move}(C, A) \ldots \mathtt{move}(A, B) \ldots] \\ f_7 &= [\ldots \mathtt{move}(C, B) \ldots \mathtt{move}(A, B) \ldots] \end{aligned} \tag{20}$$

In f_4, we have extended our notation somewhat, writing $a \ \&\ b$ for the plan of taking actions a and b without there being an ordering constraint between them.

In less formal terms, the plan (19) can fail for the following reasons:

1. The attempt to move C out of the way can fail. This may happen because something has been moved on top of C (f_1) or because something has already been moved to C's intended destination (f_2).

2. Something may be moved onto A (f_3 or f_6) or onto B (f_4 or f_7). We get two exceptions in each case here depending on whether the block moved into the way is C (f_6 and f_7) or not (f_3 and f_4).

3. A may be moved off of B after it is put there (f_5).

Because all of the exceptions are of measure 0 in (19), (19) itself is a satisfactory choice for $\mathcal{P}(\mathtt{on}(A,B),[\ldots])$. We also take

$$\mathcal{P}(\neg\mathtt{on}(A,B),[\ldots\mathtt{move}(C,?)\ldots\mathtt{move}(A,B)\ldots]) = \cup_i f_i$$

We needed to be careful when constructing the above exceptions to take f_3 as indicated instead of the more obvious choice

$$f_3' = [\ldots[\mathtt{move}(?1,A)\ \&\ \mathtt{move}(C,?)]\ldots\mathtt{move}(A,B)\ldots] \tag{21}$$

where we have allowed the action of moving something onto A to occur in parallel with the action of moving C out of the way. The reason is that if the action of moving `?1` onto A is to interfere with the rest of the plan, this action must succeed – and it won't unless C is moved out of the way first. Put more formally, the subset of f_3' that actually achieves $\mathtt{on}(A,B)$ includes a component that is approximately equal to

$$[\ldots\mathtt{move}(?1,A)\ldots\mathtt{move}(C,?)\ldots\mathtt{move}(A,B)\ldots] \tag{22}$$

and is therefore not of measure 0 in f_3'. Again, recall that our semantics of failed actions is that they have no effect at all. Since A is not clear in Figure 2, the first action of (22) fails and (22) effectively reduces to (19).

From a computational point of view, it may be more attractive to work with f_3' than to work with the more accurate f_3 appearing in (20). The reason is that f_3' is already of measure 0 in the original plan (19), and it is simpler than f_3 and therefore presumably easier to generate. It is obviously easier to stop as soon as a set of measure 0 in the original plan is encountered than to complete the analysis to obtain f_3 instead of f_3'. This is exactly what commonsense planners should do – when we plan for one of a set of conjuncts, we only worry about what might go wrong until we feel confident in dismissing it. In our running example, we know that something will go wrong with the plan of moving A to B if we move an additional block onto A. The need for this extra action ensures that we are looking at a set of measure 0 in our overall plan, so we don't think about it further. More specifically, we don't bother to draw the conclusion that we can only move something onto A after C is cleared off the top of it. It is to remain in keeping with this approach that may wish to work with f_3' instead of f_3.

The general version of this construction is similar. At each odd-numbered step (including the first), we look for plans that achieve the goal but that we have not yet identified. At even-numbered steps, we look for exceptions to the plans found thus far:

Definition 4.2 *Given a planning system $\mathcal{P}$ and a goal g, the* planning sequence generated by $\mathcal{P}$ for g *is given by*

$$\mathcal{P}_i(g) = \begin{cases} \mathcal{P}_{i-1}(g) \cup \mathcal{P}(g, P - \mathcal{P}_{i-1}(g)), & \text{if } i \text{ is odd;} \\ \mathcal{P}_{i-1}(g) - \mathcal{P}(\neg g, \mathcal{P}_{i-1}(g)), & \text{if } i \text{ is even.} \end{cases} \tag{23}$$

The sequence is initialized by $\mathcal{P}_0(g) = \emptyset$.

Some notation will make this definition easier to work with. If we write $\mathcal{D}_i$ for the symmetric difference between $\mathcal{P}_i$ and $\mathcal{P}_{i-1}$, it suffices to describe only how $\mathcal{D}_i$ is computed at each step. We know that $\mathcal{D}_i$ has to be added to $\mathcal{P}_i$ at odd steps and removed at even steps. Now (23) becomes:

$$\mathcal{D}_i(g) = \begin{cases} \mathcal{P}(g, P - \mathcal{P}_{i-1}(g)), & \text{if } i \text{ is odd;} \\ \mathcal{P}(\neg g, \mathcal{P}_{i-1}(g)), & \text{if } i \text{ is even.} \end{cases} \tag{24}$$

Further simplification is often possible as well. If i is even, for example, we can evaluate (24) recursively to get

$$\mathcal{D}_i(g) = \mathcal{P}(\neg g, \mathcal{P}_{i-2}(g) \cup \mathcal{D}_{i-1}(g)) \tag{25}$$

If we know that we caught all of the exceptions at the $i-2$nd step, we will know that $\mathcal{P}(\neg g, \mathcal{P}_{i-2}(g)) = \emptyset$ and we can replace (25) with the simpler

$$\mathcal{D}_i(g) = \mathcal{P}(\neg g, \mathcal{D}_{i-1}(g)) \tag{26}$$

In a similar way, if we know that $\mathcal{D}_{i-1}(g)$ includes all the plans that achieve g at an odd step, we can conclude

$$\mathcal{D}_i(g) = \mathcal{P}(g, \mathcal{D}_{i-1}(g)) \tag{27}$$

In both (26) and (27), the purpose of each step is to correct possible incorrectness in the previous step; possible incompleteness in the previous step is not an issue.

We are now in a position to achieve the first of our two technical goals in this section:

Theorem 4.3 *Given a planning system $\mathcal{P}$ and a goal g, the planning sequence generated by $\mathcal{P}$ for g converges to $L(g)$.*

This result shows us how to construct a planner that is correct and complete from one that is approximately correct and approximately complete.

Our remaining goal is that of showing how to combine the planner's results for g_1 and for g_2 to obtain a plan for $g_1 \wedge g_2$. Presumably, the semantics underlying L is such that a plan achieves the conjunctive goal $g_1 \wedge g_2$ if and only if it achieves both g_1 and g_2, so that

$$L(g_1 \wedge g_2) = L(g_1) \cap L(g_2)$$

The following result is now obvious:

Lemma 4.4 *Suppose that P_1 achieves a goal g_1 and that P_2 achieves g_2. Then $P_1 \cap P_2$ achieves $g_1 \wedge g_2$.* ■

The problem, of course, is that $P_1 \cap P_2$ may well be empty if P_1 and P_2 are specific linear plans that achieve the goals. If we could weaken the above lemma to require only that the plans *generally* achieve their goals, we could use the fact that (14) generally achieves $\texttt{on}(B, C)$ and that (19) generally achieves $\texttt{on}(A, B)$ to conclude that a general solution to the Sussman anomaly is

$$[\ldots \texttt{move}(B, C) \;\texttt{\&}\; [\texttt{move}(C, ?) \ldots \texttt{move}(A, B)] \ldots] \tag{28}$$

This plan involves three actions – moving B to C, moving C out of the way, and moving A to B. C must be moved before A is put on B, but there are no other ordering constraints involved. Unfortunately, this is not a solution to the Sussman anomaly, since it allows the action of moving B to C to precede the action of moving C out of the way. Here is the general problem:

Proposition 4.5 *There exist plans P_1, P_2, Q_1 and Q_2 with Q_i of measure 0 in P_i but $Q_1 \cap Q_2$ of measure 1 in $P_1 \cap P_2$.*

The Sussman anomaly isn't quite this bad; the correct plan

$$[\ldots \texttt{move}(C, ?) \ldots \texttt{move}(B, C) \ldots \texttt{move}(A, B) \ldots]$$

is neither of measure 0 nor of measure 1 in (28). If it were of measure 1 in (28), then (28) would generally achieve the original goal of getting A on B and B on C; if it were of measure 0, (28) would in general fail to achieve the goal. In fact, it succeeds some of the time and fails in others; it depends only on how we order the actions.

The tower-construction problem of Figure 1 is an example where Proposition 4.5 does hold. The plan

$$[\ldots \texttt{move}(B, C) \ldots]$$

gets B on C, and

$$[\ldots \texttt{move}(A, B) \ldots]$$

generally gets A on B. Nevertheless the plan for constructing the tower is of measure 0 in

$$[\ldots \texttt{move}(A, B) \;\texttt{\&}\; \texttt{move}(B, C) \ldots]$$

because we must take the additional action of moving C to the table. In terms of the proposition, the P_i are the plans for achieving the subgoals, and the Q_i are the exception sets for these plans.

We cannot necessarily merge specific plans for achieving the individual conjuncts. Nor, as Proposition 4.5 tells us, can we necessarily find a plan for the conjunctive goal by merging plans for *generally* achieving the conjuncts. But we do have the following, an immediate consequence of Theorem 4.3:

Corollary 4.6 *Given a planning system $\mathcal{P}$ and goals g_1 and g_2, denote by $\mathcal{P}_i(g)$ the planning sequence generated by $\mathcal{P}$ for g. Then the sequence*

$$\mathcal{P}_i(g_1) \cap \mathcal{P}_i(g_2) \tag{29}$$

converges to $L(g_1 \wedge g_2)$, the plan for achieving the conjunction of g_1 and g_2. ■

This result is evidence that we are on the right track, although we need to do a bit better – taking the limit in (29) will not be viable in practice. More precisely, we need a way to compute $\mathcal{P}_{g_1 \wedge g_2}$ that we can guarantee to satisfy the requirements of Definition 4.1. To see how to do this, let us look at the Sussman anomaly in a bit more detail.

We already know how to construct the first two terms in the planning sequences for the subgoals; they are

$$\begin{aligned}
\mathcal{P}_1(\texttt{on}(A,B)) &= [\ldots\texttt{move}(C,?)\ldots\texttt{move}(A,B)\ldots] \\
\mathcal{P}_2(\texttt{on}(A,B)) &= \mathcal{P}_1(\texttt{on}(A,B)) - \cup f_i \\
\mathcal{P}_1(\texttt{on}(B,C)) &= [\ldots\texttt{move}(B,C)\ldots] \\
\mathcal{P}_2(\texttt{on}(B,C)) &= \mathcal{P}_1(\texttt{on}(B,C)) - \cup e_i
\end{aligned}$$

where the e_i and f_i are given by (18) and (20) respectively.

Let us denote the conjoined sequence in (29) by simply $\mathcal{P}_i$. It now follows that the first element of this sequence is given by

$$\mathcal{P}_1 = [\ldots\texttt{move}(B,C)\ \texttt{\&}\ [\texttt{move}(C,?)\ldots\texttt{move}(A,B)]\ldots] \tag{30}$$

The second element of the sequence involves removing from $\mathcal{P}_1$ the exceptions in either (18) or (20). We can compute these by combining, for example, the plan

$$e_1 = [\ldots\texttt{move}(?1,B)\ldots\texttt{move}(B,C)\ldots] \tag{31}$$

which is one of the three sets removed from $\mathcal{P}_1(\texttt{on}(B,C))$, with

$$[\ldots\texttt{move}(C,?)\ldots\texttt{move}(A,B)\ldots] \tag{32}$$

which is the original $\mathcal{P}_1(\texttt{on}(A,B))$. (We have standardized apart the variables in e_1 and $\mathcal{P}_1(\texttt{on}(A,B))$, which is why the `?` in (18) has been replaced with `?1` in (31).) The result of this particular merge is the following set of three plans:

$$[\ldots[\texttt{move}(?1,B)\ldots\texttt{move}(B,C)]\ \texttt{\&}\ [\texttt{move}(C,?)\ldots\texttt{move}(A,B)]\ldots] \tag{33}$$
$$[\ldots\texttt{move}(C,?)\ldots\texttt{move}(A,B)\ldots\texttt{move}(B,C)\ldots] \tag{34}$$
$$[\ldots\texttt{move}(C,B)\ldots[\texttt{move}(B,C)\ \texttt{\&}\ \texttt{move}(A,B)]\ldots] \tag{35}$$

The first of the above plans is the "obvious" merge where the two separate plans are simply executed in parallel. In the second, the variable `?1` is bound to A and the first action in e_1 is identified with the second action in $\mathcal{P}_1(\mathtt{on}(A,B))$. The ordering on the resulting action sequence is accumulated from the orderings on e_1 and on $\mathcal{P}_1(\mathtt{on}(A,B))$. The third plan is similar, with ? being bound to B and ?1 bound to C.

Each of the three plans fails to achieve the subgoal of getting B onto C. In (33), a new (and currently unidentified) block is moved onto B. In (34), A is moved onto B before B is moved onto C. And finally, C itself is moved onto B in (35).

It is only (34) that is of interest to us. The plan (33) is of measure 0 in the set of exceptions because it involves an additional action, and (35) is of measure 0 because it binds the variable ?. As we have already remarked, (34) tells us that if we move A to B before moving B to C, we will not achieve our overall goal because B will be occupied when we try to move it.

We can continue in this fashion, accumulating all of the exceptions to the overall plan (30). In addition to (34), the only plan not of measure 0 in the set of all exceptions is

$$[\ldots\mathtt{move}(B,C)\ldots\mathtt{move}(C,?)\ldots\mathtt{move}(A,B)\ldots]$$

which is part of the result of merging $\mathcal{P}_1(\mathtt{on}(B,C))$ and f_1; this tells us that if we move B to C too early, our plan for getting C out of the way en route to moving A will fail.

We can conclude from all this that a generally valid plan for solving the Sussman anomaly is given by removing from the plan (30) the union of the two plans

$$[\ldots\mathtt{move}(C,?)\ldots\mathtt{move}(A,B)\ldots\mathtt{move}(B,C)\ldots]$$
$$[\ldots\mathtt{move}(B,C)\ldots\mathtt{move}(C,?)\ldots\mathtt{move}(A,B)\ldots]$$

The result is equivalent to the plan

$$[\ldots\mathtt{move}(C,?)\ldots\mathtt{move}(B,C)\ldots\mathtt{move}(A,B)\ldots] \tag{36}$$

which is indeed the usual solution to the original problem.

Here is the result dealing with the general situation:

Theorem 4.7 *Suppose that we have a conjunctive goal $g_1 \wedge g_2$ and a set P. Construct the plan sequences P_i converging to $L(g_1) \cap P$ and Q_i converging to $L(g_2) \cap P$. Now we can always find an i and a j such that both $P_i \ominus P_{i+1}$ and $Q_j \ominus Q_{j+1}$ are of measure 0 in $P_i \cap Q_j$. For any such i and j, $P_i \cap Q_j$ will be approximately equal to $L(g_1 \wedge g_2) \cap P$.*

In our analysis of the Sussman anomaly, we actually terminated the construction of the plans for the subgoals somewhat earlier than the points sanctioned by the above result. This early termination reflects some lookahead on our part; consider, for example, the fact that the exception

$$e_1 = [\ldots\mathtt{move}(?,B)\ldots\mathtt{move}(B,C)\ldots] \tag{37}$$

to the plan for getting B on C, when combined with the plan

$$[\ldots\mathtt{move}(C,?)\ldots\mathtt{move}(A,B)\ldots] \tag{38}$$

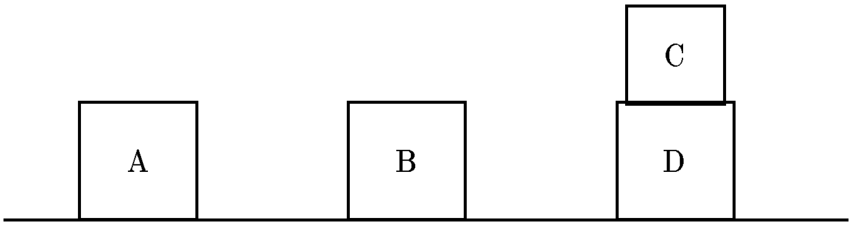

Figure 3: Get A on B on C without building a 4-block tower

for getting A onto B, led to the exception

$$[\ldots \mathtt{move}(C,?) \ldots \mathtt{move}(A,B) \ldots \mathtt{move}(B,C) \ldots] \tag{39}$$

to the general plan of moving B onto C in combination with (38).

It turns out, however, that one set of measure 0 in e_1 that actually achieves $\mathtt{on}(B,C)$ is

$$[\ldots \mathtt{move}(A,B) \ldots \mathtt{move}(B,C) \ldots] \tag{40}$$

The reason for this is that the initial action of moving A to B will fail (C is still in the way), so B will wind up on C after all. Now (39) is an instance of (40) and therefore might *not* be an exception to the general plan of getting B onto C.

The recognition that binding `?` to A in $[\ldots \mathtt{move}(?,B) \ldots \mathtt{move}(B,C) \ldots]$ is an exception to the overall plan is subtle. Roughly speaking, we need the action of moving A to B to succeed (in order to achieve the other subgoal), so moving B to C will indeed be blocked. In terms of Theorem 4.7, (40) isn't an exception to the plan for getting B on C, but

$$[\ldots \mathtt{move}(C,?) \ldots \mathtt{move}(A,B) \ldots \mathtt{move}(B,C) \ldots] \tag{41}$$

is an exception, and that's what matters. Once we have identified (41) as an exception to the original plan, the conditions of Theorem 4.7 are satisfied and we can construct the overall plan (36) with confidence.

As a final example, let us consider the tower-construction problem once again. The problem is repeated in Figure 3; recall that the goal is to get A on B and B on C without ever building a 4-block tower.

The planning sequence for getting B on C begins with

$$P_1 = [\ldots \mathtt{move}(B,C) \ldots]$$

and exceptions given by

$$\begin{aligned}
e_1 &= [\ldots \mathtt{move}(?,B) \ldots \mathtt{move}(B,C) \ldots] && (42)\\
e_2 &= [\ldots \mathtt{move}(?,C) \ldots \mathtt{move}(B,C) \ldots] \\
e_3 &= [\ldots \mathtt{move}(B,C) \ldots \mathtt{move}(B,?) \ldots] \\
e_4 &= [\ldots \mathtt{move}(?,?1) \ldots \mathtt{move}(C,?) \ldots \mathtt{move}(B,C) \ldots] && (43)
\end{aligned}$$

The final exception above involves situations where B cannot be moved to C because a 4-block tower is involved. In practice, however, the exception (43) is less likely to be generated than is

$$e'_4 = [\ldots\texttt{move}(C,?)\ldots\texttt{move}(B,C)\ldots] \tag{44}$$

We realize that putting B on C can never violate the 4-block constraint unless we first move C *somewhere*. We will temporarily work with e'_4 instead of e_4, just as we indicated the possibility of working with f'_3 in (21) instead of f_3 in (20) in the Sussman anomaly.

In a similar way, the planning sequence for getting A on B begins with

$$Q_1 = [\ldots\texttt{move}(A,B)\ldots] \tag{45}$$

and exceptions

$$\begin{aligned} f_1 &= [\ldots\texttt{move}(?,A)\ldots\texttt{move}(A,B)\ldots] \\ f_2 &= [\ldots\texttt{move}(?,B)\ldots\texttt{move}(A,B)\ldots] \\ f_3 &= [\ldots\texttt{move}(A,B)\ldots\texttt{move}(A,?)\ldots] \\ f'_4 &= [\ldots\texttt{move}(B,?)\ldots\texttt{move}(A,B)\ldots] \end{aligned} \tag{46}$$

When we combine this sequence with the previous one, we get

$$P_1 \cap Q_1 = [\ldots\texttt{move}(A,B)\ \texttt{\&}\ \texttt{move}(B,C)\ldots]$$

and the exceptions include

$$g_1 = [\ldots\texttt{move}(A,B)\ldots\texttt{move}(B,C)\ldots] \tag{47}$$

$$g_2 = [\ldots\texttt{move}(B,C)\ldots\texttt{move}(A,B)\ldots] \tag{48}$$

together with sets of measure 0 with respect to these. But

$$g_1 \cup g_2 = [\ldots\texttt{move}(A,B)\ \texttt{\&}\ \texttt{move}(B,C)\ldots]$$

since both possible orderings are eliminated. From a commonsense point of view, we can't put A on B first because we will then be unable to get B to C, and can't put B on C first because this might (and in fact does) make a 3-block tower to which A cannot be added.

There are two ways in which the analysis can be extended. The exceptions (47) and (48) are the result of intersections with (42) and (46), so one of these two sets must be analyzed further. Since f'_4 is an approximation, it seems natural to work on this first, leading to

$$\begin{aligned} f_1 &= [\ldots\texttt{move}(?,A)\ldots\texttt{move}(A,B)\ldots] \\ f_2 &= [\ldots\texttt{move}(?,B)\ldots\texttt{move}(A,B)\ldots] \\ f_3 &= [\ldots\texttt{move}(A,B)\ldots\texttt{move}(A,?)\ldots] \\ f_4 &= [\ldots\texttt{move}(B,C)\ldots\texttt{move}(A,B)\ldots] && (49) \\ f_5 &= [\ldots\texttt{move}(?,?1)\ldots\texttt{move}(B,?)\ldots\texttt{move}(A,B)\ldots] && (50) \end{aligned}$$

The final line (50) indicates that one way to make B the top block in a 3-block stack is to move `?` to `?1` and then B to `?`. This is of measure 0 in our prospective solution (45), however, so we need not worry about it.[4] The problem continues to be (49), which is still enough

[4] As in the Sussman anomaly, we actually need to be a bit more careful but the details are not of interest.

to invalidate our original plan. We now have to choose between finding exceptions to (49), finding a way to move B to C without creating a 3-block tower, and finding exceptions to (42), finding a way to move A to B before moving B to C.

Let us suppose that we choose wrongly, so that we now have to look for instances of

$$[\ldots \mathtt{move}(?, B) \ldots \mathtt{move}(B, C) \ldots]$$

for which B actually ends up on C after all. Here is the most general solution:

$$[\ldots \mathtt{move}(?, B) \ldots \mathtt{move}(?, ?1) \ldots \mathtt{move}(B, C) \ldots] \tag{51}$$

Unfortunately, this doesn't help us with our original difficulty, since we need to bind `?` to A and now

$$[\ldots \mathtt{move}(A, B) \ldots \mathtt{move}(A, ?1) \ldots] \tag{52}$$

is known to be an exception that fails to get A onto B.

So we turn our attention to (49); if we begin by moving C, then we will in fact be able to move A to B after all. So we can achieve $\mathtt{on}(A, B)$ using the plan

$$[\ldots \mathtt{move}(C, ?) \ldots \mathtt{move}(B, C) \ldots \mathtt{move}(A, B) \ldots] \tag{53}$$

Unfortunately, this still might not work, since moving C (potentially to the top of a 3-block stack) may cause a problem in getting B to C as indicated in the original plan (44) for achieving this subgoal. But now we finally replace (44) with the more appropriate (43), allowing us to conclude that (53) does indeed generally achieve the goal of getting A on B and B on C.

The analysis would be very different if we were to work with a more conventional planner. There, the fact that we cannot build a 4-block tower would be encoded by adding a new precondition to the `move` operator, saying that in order to move x to y, either y must be on the table, or it must be on a block that is on the table.

Now when we try to move B to C, we will naturally generate the plan of first moving C to the table (since C being on the table is one possible way to achieve the disjunctive precondition). This plan can then be extended to solve the problem but the plan itself has been constructed blindly instead of in response to an identified bug in the simple plan of putting B on C and then A on B.

In our approach, the problem is identified in (46), which is later refined into (49) and (53). A plan to overcome the 4-block difficulty is generated only when it is needed, and not as part of a general attempt to get B onto C.

There is another difference in the treatments of this example as well. Consider a conventional planner that proceeds by first planning to move B to C, and then planning to move A to B. When the difficulty is found and the plan for getting B onto C has to be modified, the ensuing backtrack will discard the plan for getting A onto B. Not much work is lost in our simple example, but in more complex problems it may be crucial to avoid replanning for the goal of getting A onto B. After all, the existing plan for achieving this subgoal is the correct one.

The approach that we have described behaves in this fashion. The successive refinements to the plan for getting A onto B, beginning with the basic plan

$$[\ldots \mathtt{move}(A, B) \ldots] \tag{54}$$

and eventually culminating in

$$[\ldots \mathtt{move}(C, ?) \ldots \mathtt{move}(B, C) \ldots \mathtt{move}(A, B) \ldots]$$

all continue to use the fundamental plan (54). In fact, work is in some sense *never* discarded in our approach, since we proceed by gradually refining plan sets in ways that are suggested by the merging computations and by corresponding interactions among subgoals. In our framework, the plan for constructing the tower is built up by starting with the basic plans of moving B to C and A to B, and then debugging the result. This leads to a much more focussed search process than that associated with conventional methods.

5 Implementation considerations

In order to actually build a planning system based on the ideas that we have described, there are three separate problems that need to be addressed. First, we need to discuss the manipulation of plan sets, including the underlying operation of plan intersection. Second, we need to describe the construction of a system that can produce the plan sets in the first place. And finally, we need to discuss implementation details surrounding results like Theorem 4.7; there are several simple ideas that can make this result substantially more effective in practice. (Witness the footnote in the previous section.) We will deal with these issues in this order.

5.1 Plan intersection and manipulating plan sets

Plan intersection is often known as plan *merging* and has already been discussed by a variety of authors; typical is the treatment of Foulser *et.al* [4]. The construction described there is related to ours, although not identical. Foulser *et.al* allow for the possibility that more than two plans be merged at once, but their plan description language is more restricted than ours. In keeping with conventional interests, they assume that the actions in a plan are sequential; no others can be interspersed as new information is obtained. As a result, they do not draw the distinction between $\leq$ and $*$ that was our focus in Section 2.

Nevertheless, the ideas introduced by Foulser and his coauthors [4] can be used to implement our somewhat more general notion of plan intersection. The method used continues to be that of treating sequences of actions (actions related by $\overline{*}$ in our notation) as atomic units, and then merging larger structures made up of these units. The details are not of any great theoretical interest and the interested reader is referred to the code itself.[5]

[5] The code described in this section is part of the MVL theorem proving system [5, 9, 11], which can be obtained by anonymous ftp from t.uoregon.edu.

One thing that does bear mention is that the result of intersecting two plans may be a plan set that cannot be represented as a single plan; witness the construction of (33), (34) and (35) from the intersection of (31) and (32). The implementation obviously needs to cater to this possibility.

Manipulating general plan sets is a bit more interesting. This is quite a difficult problem but is made simpler in practice by the recognition that the plan sets under consideration are generally of the form

$$D_1 - D_2 \cup D_3 - D_4 \cup D_5 - \cdots \tag{55}$$

where the D_i are the symmetric differences of successive $\mathcal{P}_i$ and are in general fairly simply represented. The evaluation here is intended to be from left to right, so that (55) is in fact

$$(((D_1 - D_2) \cup D_3) - D_4) \cup D_5 - \cdots$$

The implementation is constructed in just this way, representing any particular plan set as an alternating sum such as (55). Taking the union or intersection of these alternating sums is tedious but not terribly difficult.

Of course, the manipulations involved are fundamentally dependent on the plan intersection operation that we described earlier. Existing planners work with global data structures, gradually accumulating actions into an overall plan. This makes their plans somewhat brittle, since they will need to discard a great deal of existing work when a portion of the plan changes. A system such as we have described plans for subgoals separately but must frequently merge the plans involved in order to understand possible subgoal interactions.

The upshot of this is that the speed of a planner built on our ideas will be crucially dependent on the speed of the underlying mechanism for plan merging; as Foulser *et.al* point out, plan merging can be exponentially expensive. They also point out, however, that this exponential expense appears not to be incurred in practice because the plans being merged consist of small numbers of linear segments. This matches our experience. Finally, since specific plan segments tend to appear many times in the analysis, the overall computation can be speeded substantially by caching the results of the merging computation in some way.[6]

5.2 Constructing plan sets

Given an implementation that manipulates plan sets, from where are we to obtain these sets in the first place? There are three sources:

1. Information about the initial situation (corresponding to the plan [ ]) can be encoded in this fashion. Thus in the tower-construction example, we know that $\texttt{on}(C, D)$ is definitely achieved by the plan [ ]. It follows that $\texttt{on}(C, D)$ is generally achieved by the universal plan set $[\ldots]$.

[6]More effective still appears to be to cache the results of the plan *instance* computation.

2. Information about actions occurring (although perhaps not succeeding) is encoded similarly. For any action a, the statement "a has just occurred" is true for the plan $[\ldots a]$. What this says that that a occurs at the end of any sequence of actions that does indeed end in a.

3. Finally, there are operations that transform plan sets into new plan sets. We have already seen one of these in the form of intersection; another corresponds to the frame axiom in our setting.

Definition 5.1 *There exists a* frame operator $\mathcal{F}$ *that accepts as input two plan sets and returns another plan set. The operator is defined so that if a particular goal or fluent g is inserted into the database by those plans in the set $\mathcal{P}^+$ and deleted from the database by those plans in $\mathcal{P}^-$, then $L(g) = \mathcal{F}(\mathcal{P}^+, \mathcal{P}^-)$.*

Somewhat less formally, the goal is achieved by plans in $\mathcal{F}(\mathcal{P}^+, \mathcal{P}^-)$ and not achieved by plans outside this set. Note that the equality in the definition is exact, not approximate.

As an example, we would expect to have

$$\mathcal{F}([\,], \emptyset) = [\ldots] \tag{56}$$

What this tells us is that if a fluent is true in the initial situation, and we know of no reason for it to be false in other situations, we can expect it to be true at all times.

Here is another example:

$$\mathcal{F}([\ldots a], \emptyset) = [\ldots a \ldots]$$

If an action a succeeds in achieving a goal and no other actions delete that goal, then we can use the frame axiom to conclude that the goal continues to hold after additional actions occur.

Here is a slightly more interesting example. Suppose that some goal is added to the database by the plan $[\ldots a]$ but deleted by $[\ldots b]$. What should $\mathcal{F}([\ldots a], [\ldots b])$ be?

The result should be

$$[\ldots a \ldots] - [\ldots b \ldots] \cup [\ldots b \ldots a \ldots] \tag{57}$$

since if both actions a and b occur, the goal will be true only if a occurs after b does. This fits neatly into the implementation details already discussed; the plan set (57) is conveniently written as an alternating sum.

It is because of examples such as this one that the operator $\mathcal{F}$ is defined on *pairs* of plan sets. We cannot find a unary $\mathcal{F}'$ such that

$$\mathcal{F}(P, Q) = \mathcal{F}'(P) - \mathcal{F}'(Q)$$

because, as we see from (57), the plan sets P and Q interact in the construction of $\mathcal{F}(P, Q)$.

The reason the frame operator is important is because a planning database will typically indicate only which actions actually add and delete fluents from the domain description;

there will be no explicit description of the set of plans that achieve a given goal g. In order to construct these plans, we have to find the plans that add g to the database, and then use the frame operator $\mathcal{F}$ to actually construct the plan set in its entirety. Working with actions that delete g allows us to compute $\mathcal{P}_{\neg g}$ similarly.

In all of the examples we have encountered, if P and P' are approximately equal and Q and Q' are as well, then $\mathcal{F}(P, Q)$ is approximately equal to $\mathcal{F}(P', Q')$. This is as it should be if our ideas are to be usefully incorporated into systems that use the frame axiom; it also serves to provide loose confirmation of the utility of our measure-theoretic notion of when one set of plans is small relative to another.

The function $\mathcal{F}$ has a variety of analogs in earlier work. We have already remarked on its clear connection to the frame axiom; since it is accepts information about the plans where facts are added or deleted from the database and returns information about when those facts hold generally, it is also the analog in our setting of what Chapman and others have called the *modal truth criterion* [1].

The collection of all plan sets is a lattice under the subset relation; more formally, the set of plan sets is naturally isomorphic to the set of functions from the set S of linear plans into the two-point set $\mathbf{2} = \{t, f\}$. For a particular plan set P and plan p, $P(p) = t$ if $p \in P$ and $P(p) = f$ otherwise. The set $\mathbf{2}^S$ inherits a lattice structure from the lattice structure on $\mathbf{2}$, and it is not hard to extend this structure to embed $\mathbf{2}^S$ in a *bilattice* [5]. A function that maps bilattice elements to new bilattice elements is referred to as a *modal operator* [7] because there is a natural relationship between such functions and the existing notion of modality [15] in the philosophical community, and $\mathcal{F}$ is indeed a modal operator in this setting. I also suggest elsewhere [6] that the frame operator can be viewed modally.

The point of this embedding is that it allows us to treat $\mathcal{F}$ as a semantic object in a bilattice setting. We can use the declarative mechanisms that exist in the bilattice framework to manipulate the plan sets in question; we do not need to construct a special-purpose planner but can instead resort to a general multivalued theorem prover [9, 11]. All we need do is provide a declarative description of action.

The first axiom in this declarative description needs to capture a STRIPS-like frame axiom as in Definition 5.1. To do this, consider the statement "g has just been achieved." This sentence is true for plans sets that terminate with an action that adds g; "g has just been removed" is true for plan sets that terminate in actions deleting g.

To formalize this, we reify g and add a predicate `triggers`; we will take $\texttt{triggers}(g)$ to mean that g has just been added and $\texttt{triggers}(\neg g)$ to mean that g has just been deleted. Using the frame modality $\mathcal{F}$, we can now write

$$\mathcal{F}[\texttt{triggers}(g), \texttt{triggers}(\neg g)] \supset \texttt{holds}(g)$$

This allows us to use the information in $\mathcal{F}$ to find plan sets that achieve g (i.e., plan sets for which g holds).

The rest of the axiomatization defines the extent of `triggers`:

$$\texttt{adds}(a, p) \wedge \texttt{succeeds}(a) \quad \supset \quad \texttt{triggers}(p) \tag{58}$$

$$
\begin{aligned}
\texttt{deletes}(a,p) \wedge \texttt{succeeds}(a) &\supset \texttt{triggers}(\neg p) && (59)\\
\texttt{occurs}(a) \wedge \texttt{precs}(a,l) \wedge \texttt{precs-ok}(l) &\supset \texttt{succeeds}(a) && (60)\\
\texttt{holds}(p) \wedge \texttt{precs-ok}(l) &\supset \texttt{precs-ok}([p|l]) && (61)\\
& \texttt{precs-ok}([\,]) && (62)
\end{aligned}
$$

Axioms (58) and (59) tell us that successful actions add and delete facts as appropriate to the database, and the remaining axioms describe situations under which actions succeed – they succeed if they occur and all of their preconditions are satisfied. A list of preconditions is satisfied if the first one is and all the rest are; the ground case is that an empty list of preconditions is always satisfied. The bracketed expressions in (61) and (62) are lists of preconditions and should not be confused with plans.

Finally, we need to identify situations in which actions occur, so that the sentence $\texttt{occurs}(a)$ is assigned the plan set $[\ldots a]$ and $\texttt{succeeds(init)}$ is assigned the plan set $[\,]$ where `init` is a dummy action that sets up the initial situation. (We could instead say that `init` occurs in the initial situation and has no preconditions, but that would be somewhat less compact.) We can now add specific information about the initial situation by writing, for example, that

$$\texttt{adds}(\texttt{init}, \texttt{on}(C,D))$$

to say that C is on D in the initial situation.

There are other advantages to exploiting the bilattice machinery for planning purposes, since this approach allows us to use a declarative description of action instead of a procedural one. We can add actions that require some variable amount of time to take effect (boiling water comes to mind), have effects without duration (like popping a balloon causing a noise) [8], or have ramifications that need to be computed based on the values of other fluents [3, 13]. All of this work remains in the same overall declarative framework and continues to use this framework to provide the machinery needed for planning. As discussed elsewhere [12], we can also introduce defaults into the declarative language, thereby capturing in our setting the ideas typically associated with hierarchical planners.

5.3 Status

The current implementation is a good – but not perfect – match for the theoretical constructs that we have discussed. Plans and plan sets are both implemented as described. The examples of Section 4 make some specific control assumptions about the nature of the search, and these control decisions are not yet supported by the implementation. (In a bilattice setting, they appear to be restrictions to planning of more general control heuristics, and we are attempting to implement these general control notions as opposed to specializations of them.)

Rather than invoke Theorem 4.7 directly, the planner works by determining at each point whether a particular line of reasoning will have a significant impact on its overall answer. In other words, it decides whether or not its answer would change on a set of measure 0 relative to the current value. This is in keeping with the analysis of Section 4, where we curtailed some portion of the analysis as soon as we could tell that the answer didn't matter. Theorem 4.7 guarantees that there always *will* be a point at which things are clearly irrelevant; the implementation is often able to terminate its reasoning before the conditions of the theorem are satisfied. Most of the time used by the planner is spent in reasoning of just this sort, deciding whether a particular line of reasoning might impact the plan being generated.

6 Conclusion

My overall aim in this paper has been to describe a single idea: that planners should manipulate not specialized plans that are guaranteed to achieve their goals, but more general plans that can only be expected to. We have presented a formalization of this idea of "expecting" a plan to achieve a goal in terms of the plan failing for a set of measure 0 in the set of its possible executions.

Building a planner around this idea introduces additional possibilities that existing planners lack; most important among these is that it is possible to combine approximate plans for each of two subgoals to obtain an approximate plan for their conjunction. The main technical result of the paper is Theorem 4.7, which confirms this observation. An examination of the tower-construction problem indicates that such a planner will have advantages over a conventional one in that it will debug plans constructed using independence assumptions as opposed to catering to all possible plan interactions at the outset.

Finally, we discussed briefly an implementation of our ideas that exploits the fact that plan sets can be viewed as elements of a bilattice. As mentioned in the introduction, some existing planners such as O-Plan appear to make informal use of the ideas that we have discussed, but we know of no planner that explicitly conforms to the notions we have presented. A preliminary implementation of such a planner has been built using the bilattice-based theorem prover MVL [11], but many implementation issues remain to be addressed. Dealing with these is the topic of ongoing research.

Acknowledgement

This work has been supported by the Air Force Office of Scientific Research under contract 92-0693 and by ARPA/Rome Labs under contracts F30602-91-C-0036 and F30602-93-C-00031. I would like to thank David Etherington, Nir Friedman, Will Harvey, Jim Hendler, David McAllester, Drew McDermott, Steve Minton, Dan Weld, David Wilkins, two anonymous reviewers and the members of the PRINCIPIA and CIRL research groups for many useful discussions regarding these ideas.

A Proofs

Lemma 2.4 *If* $(A, \leq, *)$ *is a plan, then for any* $a, b, c \in A$*:*

1. *If $a * b$ and $a * c$, then $b = c$.*

2. *If $a * c$ and $b * c$, then $a = b$.*

Proof. If $a * b$ and $a * c$, then $a * b$ and $a < c$, so that $b \leq c$. Since $a * c$ and $a < b$, $c \leq b$ as well; thus $b = c$. The second claim is similar. ■

Lemma 2.5 *Let $(A, \leq, *)$ be a plan. Then for any $a, b, c \in A$:*

1. *If $c \leq b$ and $c \overline{*} a$, then either $c \overline{*} b$ or $a \leq b$.*

2. *If $b \leq c$ and $a \overline{*} c$, then either $b \overline{*} c$ or $b \leq a$.*

Proof. Suppose that $c \leq b$ and $c \overline{*} a$ but not $c \overline{*} b$. We prove that $a \leq b$ by induction on the length of the chain

$$c = c_0 * \cdots * c_n = a$$

If $n = 0$, then $c = a$ and since $c \leq b$, $a \leq b$. For the inductive case, suppose that the lemma holds for chains of length $n - 1$. Then in the above example, since $c * c_1$ and $c < b$, we must have $c_1 \leq b$. Since $c_1 \overline{*} a$ as well, we can apply the inductive hypothesis to conclude $a \leq b$. ■

Lemma 2.6 *Let $(A, \leq, *)$ be a plan. Then for any $a, b, c \in A$, if $a \overline{*} b$ and $a \leq c \leq b$, then $a \overline{*} c$.*

Proof. Suppose we have

$$a = a_0 * \cdots * a_n = b$$

For each a_i in the chain, if $a_{i-1} < c$, we can conclude that $a_i \leq c$. Thus if $a_j < c$ for each j, we eventually conclude $b \leq c$, so that $b = c$. It follows that c actually appears somewhere in the above chain, so that $a \overline{*} c$. ■

Proposition 2.9 *The instance relation of Definition 2.8 is a partial order.*

Proof. That the relation is reflexive and transitive is clear; to see that it is antisymmetric, if σ_1 is the binding list associated with showing that p_1 is an instance of p_2 and σ_2 is the binding list showing that p_2 is an instance of p_1, then for any action a in p_1, we must have $a|_{\sigma_1}|_{\sigma_2} = a$, so that the actions differ only in the names of the variables. ■

Lemma 2.11 *If $P = (A, \leq, *)$ is a linear plan, then $\leq$ is a total order.*

Proof. Suppose that we have $a \leq b$ and $a \leq c$ with b and c unordered. Then since $\leq \; = \; \overline{*}$, there must be b_i and c_i such that

$$a * b_0 * \cdots * b_m = b$$

and

$$a * c_0 * \cdots * c_n = c$$

If either of the above sequences is contained in the other, then we will have $b \leq c$ or $c \leq b$, so suppose otherwise. Now if i is the first point at which the sequences differ, there is a single point b_{i-1} with $b_{i-1} * b_i$ and $b_{i-1} * c_i$ but $b_i \neq c_i$, in conflict with Lemma 2.4. ■

Proposition 2.12 *$P_1 \subseteq P_2$ if and only if the set of linearizations of P_1 is a subset of the set of linearizations of P_2.*

Proof. One direction is easy: if $P_1 \subseteq P_2$, every instance of P_1 is an instance of P_2 because $\subseteq$ is transitive.

For the other direction, suppose that every linearization of $P_1 = (A_1, \leq_1, *_1)$ is a linearization of $P_2 = (A_2, \leq_2, *_2)$. Specifically, consider the linear plan L in which $\leq_1$ has been extended to a total order and the actions in A_1 have been made ground by binding the variables to unique Skolem constants. Since this is a linearization of P_2, there must be a binding list σ and injection i that embeds P_2 in L. If we modify σ to change the Skolem constants back to the variables they represent in A_1, we get a binding list σ' and injection $i : A_2 \rightarrow A_1$ that injects the actions in A_2 into A_1. It remains only to show that i preserves $\leq$ and $*$.

If we had $a * b$ in P_2 without the corresponding relation holding in P_1, we could add an extra action between a and b in the linear plan L to construct a plan that was a linearization of P_1 but not of P_2, so it follows that $*$ is preserved.

To see that $\leq$ is preserved, suppose that $a \leq b$ in P_2 but not in P_1. Now it follows from the lemmas that in P_1, where $a \not\leq b$, if $a \bar{*} c$ or $c \bar{*} a$, we must not have $c \leq b$ either. Thus b is unordered in P_1 with respect to all the points related to a by $\bar{*}$. We can continue to satisfy Definition 2.2 if we add (in P_1) that $b \leq c$ for all such points; suppose that we call the resulting plan P_1'. It is clear that P_1' has linearizations that P_2 lacks, since $a \geq b$ in P_1 but $a \leq b$ in P_2. But since P_1' is an instance of P_1, this is a contradiction. Thus whenever $a \leq b$ in P_2, $a \leq b$ in P_1 as well. $\leq$ is preserved by i, and the proof is complete. ∎

Proposition 3.4 *Let P be a plan, and P' an instance of P with i the associated injection from A to A'. Then if there is any action in $A' - i(A)$ that is not a variable, P' is of measure 0 in P.*

Proof. Let P'' be a plan that is identical to P' but where some nonvariable action in $A' - i(A)$ has been replaced with a new variable. Now it is clear that P'' is an instance of P, and that P' is of measure 0 in P'' (since it binds the new variable). Thus P' is of measure 0 in P. ∎

Lemma 3.6 *Q is of measure 0 in P if any of the following conditions holds:*

1. *Q is empty.*
2. *Q is a subset of a set of measure 0 in P.*
3. *Q is of measure 0 in a subset of P.*
4. *Q is of measure 0 in a superset S of P with P of measure 1 in S.*

Proof.

1. The empty set is the finite union of no sets, each of which is of measure 0 in P.
2. Take R to be the given set of measure 0 in P, and $S = P$.
3. Take $R = Q$ and S to be the given subset of P. Now $S - P = \emptyset$.

4. Take $R = Q$ and S as given; since P is of measure 1 in S, $S - P$ is of measure 0 in S.

Lemma 3.7 *Q is of measure 0 in P if and only if Q is of measure 0 in $P \cup Q$.*
Proof. Since $P \subseteq P \cup Q$, it is clear that Q is of measure 0 in the union if it is of measure 0 in P. For the converse, take $R = Q$ and $S = P \cup Q$ in the definition; since Q is of measure 0 in S, we must have $S - P$ of measure 0 in S as well. Thus Q is of measure 0 in P. ■

Proposition 3.8 *Approximate equality is an equivalence relation.*
Proof. That approximate equality is reflexive and symmetric is clear; we need only show that it is transitive. To see this, suppose that X is approximately equal to Y and Y is approximately equal to Z. We will show that X is of measure 1 in Z; that Z is of measure 1 in X is similar.

We need to show that $Z - X$ is of measure 0 in Z, but we know that

$$Z - X \subseteq (Z - Y) \cup (Y - X) \tag{63}$$

The first term here is of measure 0 in Z because Y is of measure 1 in Z. The second term is of measure 0 in Y; since Z is of measure 1 in Y, it follows that the second term is of measure 0 in Z as well. Thus the union in (63) is of measure 0 in Z and X is of measure 1 in Z. ■

Proposition 3.9 *Let $P \neq \emptyset$ be a plan set. Then provided that our language includes infinitely many object and action constants, there is no plan set that is both of measure 0 and of measure 1 in P.*
Proof. The proof of this result is essentially unchanged from that of the analogous Proposition 3.3 of [10].

If there were a subset Q of a plan set P that was both of measure 0 and of measure 1 in P, then we could repeatedly apply the secondary clauses of Definition 3.3 to construct a finite collection of plan sets Q_i and a plan set $S = \cup_i Q_i$ such that each $Q_i = S|_{\sigma_i}$ for some nontrivial σ_i.

We can suppose without loss of generality that S contains a single variable `?`; the general case is no harder. For s_j an arbitrary Skolem constant in our language, let S_j be a linearization of S in which `?` has been bound to s_j. Since S_j will be an instance of $Q_i = S|_{\sigma_i}$ only if σ_i binds `?` to s_j, it follows that each Q_i can contain at most one of the various S_j. Since there are an infinite number of Skolem constants available, S cannot be the union of finitely many Q_i. ■

Lemma 3.11 *Suppose we have a sequence P_i of plan sets, where P_{i+1} is of measure 0 in P_i for each i. Then the P_i converge to the empty set.*
Proof. Let P be an arbitrary linear plan that appears in infinitely many of the P_i, and consider only the subsequence of the P_i's whose elements contain P. We can assume without loss of generality that each P_i contains P_{i+1}, since P_{i+1} is of measure 0 in P_i if and only if it is of measure 0 in the union $P_i \cup P_{i+1}$; we can also assume that each P_i is a minimal such set such that P_{i+1} is of measure 0 in P_i.

Given these assumptions, P_i must have been constructed from P_{i+1} by either replacing a subexpression with a variable (for example, replacing `move`$(x,$`block-on(?)`$)$ with `move`$(x,$`?`$)$)

or by introducing a new variable to replace an object constant or variable in the plan. Since if the eventual plan P contains n object or action constants it will contain at most n subexpressions as well, it follows that the maximum length of a chain where every element contains P will be n^2. This is in conflict with the assumption that P appears in infinitely many P_i, and the proof is complete. ■

Lemma 3.12 *Suppose S is of measure 1 in T and of measure 0 in U. Then T is of measure 0 in U.*

Proof. We show instead that T is of measure 0 in $T \cup U$, which is equivalent. We know that $T - S$ is of measure 0 in T, thus of measure 0 in $T \cup U$. S is also of measure 0 in $T \cup U$, so $T \subseteq (T - S) \cup S$ is of measure 0 in $T \cup U$ as well. ■

Proposition 3.13 *Suppose that there is some set D that is of measure 1 in $A \ominus B$ and of measure 0 in A. Then A and B are approximately equal.*

Proof. Suppose that $A - B = F$ and $B - A = E$, so that $A \ominus B = E \cup F$. Now if D is of measure 1 in $E \cup F$ and of measure 0 in A, we know that $E \cup F$ must be of measure 0 in A. Thus F is of measure 0 in A and B is of measure 1 in A.

But we also have that since $E \cup F$ is of measure 0 in A, it is of measure 0 in $A \cup E \cup F = B \cup F$. Thus F is of measure 0 in $B \cup F$, and F is of measure 0 in B. In other words, B is of measure 1 in $B \cup F$. Since E is also of measure 0 in $B \cup F$, it follows that E is of measure 0 in B and A is of measure 1 in B. Thus A and B are approximately equal. ■

Theorem 4.3 *Given a planning system $\mathcal{P}$ and a goal g, the planning sequence generated by $\mathcal{P}$ for g converges to $L(g)$.*

Proof. We begin by showing that the sequence converges, and then argue that it converges to $L(g)$.

To see that it converges, it suffices to show that the symmetric differences between successive elements of the planning sequence converge to the empty set. But at each step of the construction we will have (for example)

$$\mathcal{D}_i(g) = \mathcal{P}(\neg g, \mathcal{P}_{i-1}(g))$$

for i even.

Now we know that $L(\neg g) \cap \mathcal{P}_{i-1}(g)$ is of measure 1 in $\mathcal{D}_i$, but since $\mathcal{P}_{i-1}(g) = \mathcal{P}_{i-2}(g) \cup \mathcal{D}_{i-1}$, we can conclude that

$$L(\neg g) \cap \mathcal{P}_{i-1}(g) = [L(\neg g) \cap \mathcal{P}_{i-2}(g)] \cup [L(\neg g) \cap \mathcal{D}_{i-1}]$$

But we also know that $[L(\neg g) \cap \mathcal{P}_{i-2}(g)]$ is of measure 0 in $\mathcal{D}_{i-2}$, since the $i-2$nd step was supposed to remove all plans that failed to achieve g. Similarly, $[L(\neg g) \cap \mathcal{D}_{i-1}]$ is of measure 0 in $\mathcal{D}_{i-1}$, since the $i-1$st step was supposed to add only plans that *did* achieve g. It follows that

$$[L(\neg g) \cap \mathcal{P}_{i-2}(g)] \cup [L(\neg g) \cap \mathcal{D}_{i-1}]$$

is of measure 0 in $\mathcal{D}_{i-1} \cup \mathcal{D}_{i-2}$, and thus by Lemma 3.12 that $\mathcal{D}_i$ is of measure 0 in

$$\mathcal{D}_{i-1} \cup \mathcal{D}_{i-2}$$

This result holds for odd i as well by virtue of a similar argument.

Now suppose that we construct a new sequence S_i, where $S_i = \mathcal{D}_{2i} \cup \mathcal{D}_{2i+1}$. We can apply Lemma 3.11 to conclude that the S_i converge to $\emptyset$, from which it follows that the $\mathcal{D}_i$ do as well. Thus the planning sequence converges.

The argument that it converges to $L(g)$ is similar; at each step in the construction, we remove a set of measure 1 in the remaining error. Thus the sequence of symmetric differences between $\mathcal{P}_i(g)$ and $L(g)$ also converges to $\emptyset$, and the proof is complete. ■

Proposition 4.5 *There exist plans P_1, P_2, Q_1 and Q_2 with Q_i of measure 0 in P_i but $Q_1 \cap Q_2$ of measure 1 in $P_1 \cap P_2$.*

Proof. An example follows the statement of the proposition in the main text. ■

Theorem 4.7 *Suppose that we have a conjunctive goal $g_1 \wedge g_2$ and a set P. Construct the plan sequences P_i converging to $L(g_1) \cap P$ and Q_i converging to $L(g_2) \cap P$. Now we can always find an i and a j such that both $P_i \ominus P_{i+1}$ and $Q_j \ominus Q_{j+1}$ are of measure 0 in $P_i \cap Q_j$. For any such i and j, $P_i \cap Q_j$ will be approximately equal to $L(g_1 \wedge g_2) \cap P$.*

Proof. The proof rests on the following proposition:

Proposition A.1 *Let $f(P_1, \ldots, P_n)$ be a function on plan sets that distributes with respect to set-theoretic union and such that $f(P_1, \ldots, P_n) \subseteq P_i$ for each i. Now suppose that for each i, we have a sequence*

$$P_{i1}, P_{i2}, \ldots$$

that converges to P_i and such that if we take $\Delta_{ij} = P_{ij} \ominus P_{ij+1}$, each Δ_{ij} is of measure 1 in $P_i \ominus P_{ij}$. Then provided $f(P_1, \ldots, P_n) \neq \emptyset$:

1. *There is a collection of indices j_i such that Δ_{ij_i} is of measure 0 in $f(P_{ij_i})$ for each i, and*

2. *For any such set of indices, $f(P_i)$ and $f(P_{ij_i})$ are approximately equal.*

Proof. We prove the result for $n = 1$ only; the general case is no harder.

Suppose, then, that $f(P)$ is a function on plan sets that distributes with respect to set-theoretic union and such that $f(P) \subseteq P$. Suppose also that we have a sequence P_i that converges to P and such if we take $\Delta_j = P_j \ominus P_{j+1}$, each Δ_j is of measure 1 in $P \ominus P_j$. Then provided $f(P) \neq \emptyset$, we must show that:

1. There is an index j such that Δ_j is of measure 0 in $f(P_j)$, and

2. For any such index, $f(P)$ and $f(P_j)$ are approximately equal.

To see this, fix i and say that $P = (P_i - A) \cup B$, so that $P \cup A = P_i \cup B$. Now

$$f(P) \cup f(A) = f(P_i) \cup f(B)$$

since f distributes with respect to $\cup$. It follows that

$$f(P) \ominus f(P_i) \subseteq f(A) \cup f(B) \subseteq A \cup B = P \ominus P_i$$

and therefore that Δ_i is of measure 1 in $f(P) \ominus f(P_i)$.

Since $f(P) \neq \emptyset$ and the Δ_i's are converging to $\emptyset$, it follows that there is some fixed j such that Δ_j is of measure 0 in $f(P)$. But now we can apply Proposition 3.13 to conclude that $f(P)$ and $f(P_j)$ are approximately equal.

Since $f(P)$ and $f(P_j)$ are approximately equal, Δ_j must be of measure 0 in $f(P_j)$, and the first part of the proposition is proved.

The second part of the proposition is proved as well; if Δ_j is of measure 0 in $f(P_j)$, we know it is of measure 1 in $f(P) \ominus f(P_j)$, so $f(P_j)$ and $f(P)$ must be approximately equal. ■

To prove the original theorem, we can now simply take $f = \cap$; the conditions on the Δ's are guaranteed by the approximate validity of the fashion in which the planning sequences for the subgoals are constructed. ■

Acknowledgement

This work has been supported by the Air Force Office of Scientific Research under contract 92-0693 and by ARPA/Rome Labs under contracts F30602-91-C-0036 and F30602-93-C-00031. I would like to thank David Etherington, Nir Friedman, Will Harvey, Jim Hendler, David McAllester, Drew McDermott, Steve Minton, Dan Weld, David Wilkins, two anonymous reviewers and the members of the PRINCIPIA and CIRL research groups for many useful discussions regarding these ideas.

References

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

[2] K. Currie and A. Tate. O-plan: The open planning architecture. *Artificial Intelligence*, 52:49–86, 1991.

[3] J. J. Finger. *Exploiting Constraints in Design Synthesis.* PhD thesis, Stanford University, Stanford, CA, 1987.

[4] D. E. Foulser, M. Li, and Q. Yang. Theory and algorithms for plan merging. *Artificial Intelligence*, 57:143–181, 1992.

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

[6] M. L. Ginsberg. Formalizing action. Technical report, Stanford University, 1989.

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

[8] M. L. Ginsberg. Computational considerations in reasoning about action. In *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1991.

[9] M. L. Ginsberg. The MVL theorem proving system. *SIGART Bulletin*, 2(3):57–60, 1991.

[10] M. L. Ginsberg. Negative subgoals with free variables. *Journal of Logic Programming*, 11:271–293, 1991.

[11] M. L. Ginsberg. User's guide to the MVL system. Technical report, University of Oregon, 1993.

[12] M. L. Ginsberg and H. W. Holbrook. What defaults can do that hierarchies can't. In *Proceedings 1992 Nonmonotonic Reasoning Workshop*, Plymouth, VT, 1992.

[13] M. L. Ginsberg and D. E. Smith. Reasoning about action I: A possible worlds approach. *Artificial Intelligence*, 35:165–195, 1988.

[14] K. J. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, 45:173–228, 1990.

[15] S. A. Kripke. Semantical considerations on modal logic. In L. Linsky, editor, *Reference and Modality*, pages 63–72. Oxford University Press, London, 1971.

[16] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634–639, 1991.

[17] S. Minton, J. G. Carbonell, C. A. Knoblock, D. R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40:63–118, 1989.

[18] J. S. Penberthy and D. S. Weld. UCPOP: A sound, complete partial order planner for ADL. In *Proceedings of the Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 103–113, Boston, MA, 1992.

[19] G. J. Sussman. *A Computational Model of Skill Acquisition.* American Elsevier, New York, 1975.

[20] A. Tate. Project planning using a hierarchic non-linear planner. Technical Report 25, Depart of Artificial Intelligence, University of Edinburgh, 1976.

[21] D. E. Wilkins. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, San Mateo, CA, 1988.