Analyzing Plans with Conditional Effects

Elly Winner and **Manuela Veloso**
Computer Science Department
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
{elly,veloso}@cs.cmu.edu
fax: (412) 268-4801

Abstract

Several tasks, such as plan reuse and agent modelling, need to interpret a given or observed plan to generate the underlying plan rationale. Although there are several previous methods that successfully extract plan rationales, they do not apply to complex plans, in particular to plans with actions that have conditional effects. In this paper, we introduce SPRAWL, an algorithm to find a minimal annotated partially ordered structure in an observed totally ordered plan with conditional effects. The algorithm proceeds in a two-phased approach, first preprocessing the given plan using a novel *needs analysis* technique that builds a *needs tree* to identify the dependencies between operators in the totally ordered plan. The needs tree is then processed to construct a partial ordering that captures the complete rationale of the given plan. We provide illustrative examples and discuss the challenges we faced.

Introduction

Analyzing example plans and executions is crucial for plan adaptation and reuse, e.g., (Fikes, Hart, & Nilsson 1972), and could be useful for plan recognition and agent modelling, e.g., (Kautz & Allen 1986). One of the most common approaches to plan analysis has been to create an *annotated ordering* of the example plan, e.g., (Fikes, Hart, & Nilsson 1972; Regnier & Fade 1991; Kambhampati 1989; Kambhampati & Hendler 1992; Veloso 1994), in which an ordered plan is supplemented with a rationale for the ordering constraints. Annotated orderings allow systems not only to reuse more flexibly portions of the plans they have observed, but also to reuse the reasoning that created those plans in order to solve new problems.

In recent years, the focus of the planning and agent modelling community has shifted from the simple STRIPS domain-specification language (Fikes & Nilsson 1971) towards richer languages like ADL (Pednault 1986) that capture the nondeterminism in the effects of real-world actions. Despite the success of the annotated ordering approach for simple domain-specification languages, it has not been applied to plans with conditional effects.

In this paper, we introduce the SPRAWL algorithm for finding *minimal annotated consistent partial orderings* of observed totally ordered plans with conditional effects. A consistent partial ordering $\mathcal{P}$ of a totally ordered plan $\mathcal{T}$ is one in which all *relevant* effects (those which affect the fulfillment of the goal) active in $\mathcal{P}$ are also active in $\mathcal{T}$. We call the partial orderings found by SPRAWL *minimal* because they do not include extraneous ordering constraints; each constraint either:

- provides a term upon which a relevant effect depends, or
- prevents a threat to such a term.

SPRAWL annotates each ordering constraint with the term the constraint provides or protects.

We assume that we are given or that we observe a plan that is valid, i.e., all preconditions of the steps are satisfied, and, when executed, the plan produces the goal state. SPRAWL links the steps of the plan through the literals or terms that they support. Partial orderings are capable of representing these dependencies. [1] In addition, partial orderings can isolate independent subplans that can be reused or recognized separately, and they also identify potential parallelism.

We assume that observed example plans are totally ordered as plans of single executors. The annotations on the ordering constraints should *explain* the rationale behind the plans and allow portions of them easily to be matched, removed, and used independently.

Conditional effects make the task much more difficult because they cause the effects of a given step to change depending on what steps come before it, thus making step behavior difficult to predict. In fact, any ordering must treat each conditional effect in the plan in one of three ways:

- **Use:** make sure the effect occurs;
- **Prevent:** make sure the effect does not occur;
- **Ignore:** don't care whether the effect occurs or not.

Figure 1 shows totally ordered plans that demonstrate these three cases. Note that all three plans have the same initial state and the same operators. We are able to demonstrate all three cases by changing only the goals. The preconditions

[1] A partial order is a precedence relation $\preceq$ with the following three properties 1) reflexivity: $a \preceq a$; 2) non-symmetric (no cycles): if $a \preceq b$ then not $b \preceq a$, unless $a = b$; and 3) transitivity: if $a \preceq b$ and $b \preceq c$, then $a \preceq c$. The relation is a "partial" order because there may be uncomparable elements: i.e., elements a, b such that neither $a \preceq b$ nor $b \preceq a$. Note that a DAG is a partial order if we define $a \preceq b$ as a path from a to b.

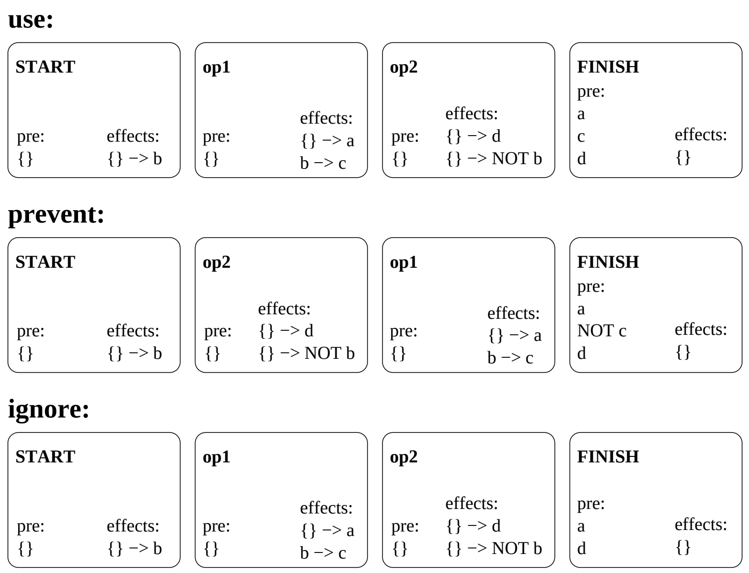


Figure 1: Three totally ordered plans that illustrate the three possible ways of treating a conditional effect in an ordering: using it to achieve a goal, preventing it in order to achieve a goal, or ignoring its effect.

(pre) are listed, as are the effects, which are represented as conditional effects $\{a\} \rightarrow b$, i.e., if a then add b. A non-conditional effect that adds a literal b is then represented as $\{\} \rightarrow b$. Delete effects are represented as negated terms (e.g., $\{a\} \rightarrow NOTb$. In the first plan, the conditional effect of op1 is *used* to generate the goal term c. In the second plan, it is *prevented* from generating the term c, and in the third plan, the effect is irrelevant, so it is *ignored*.

Figure 2 shows the annotated partial orderings generated by SPRAWL for each of these cases. The ordering constraints are annotated with a rationale explaining why they are necessary. Although the plans for these three cases are composed of the same steps, SPRAWL is able to reveal that the partial orderings are very different. In the "use" case, SPRAWL identifies that op2 threatens the goal term c, which is created by op1, and enforces the ordering op1 $\rightarrow$ op2 to protect c. In the "prevent" case, SPRAWL is able to surmise that the step op1 must not be able to execute the conditional effect that adds the term c, and so ensures that the condition of this effect, the term b, is not true before the step executes. In this way, SPRAWL discovers the ordering constraint op2 $\overset{NOTb}{\rightarrow}$ op1. It also notes that the START step, since it adds b, is a threat to this link, and must therefore come before op2. Finally, SPRAWL is able to identify that, in the "ignore" case, the conditional effect is irrelevant, so op1 and op2 may run in parallel.

Treating *any* conditional effect in a plan in a different way will result in a different partial ordering, creating exponentially (in the number of conditional effects) many partial orders, many of which may be invalid. One way to deal with this difficulty is to insist that exactly the same conditional effects must be active in the partial ordering as are active in the totally ordered plan, but this will result in an overly restrictive partial ordering in which some ordering constraints may not contribute to goal achievement. Instead, we perform needs analysis on the totally ordered plan to discover which conditional effects are relevant. Needs analysis allows us to ignore incidental conditional effects in the totally ordered plan.

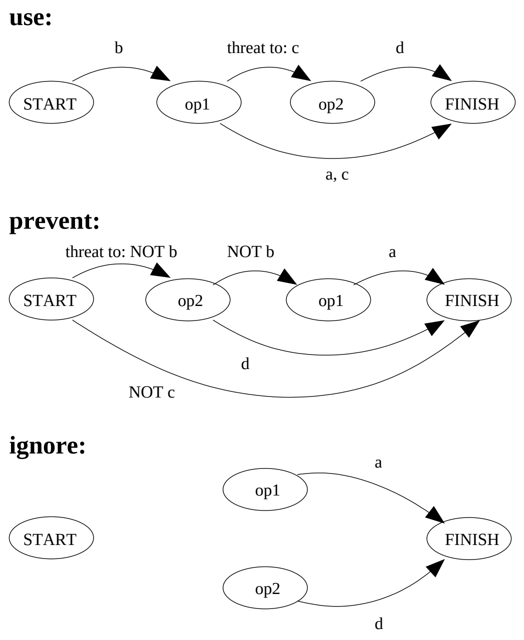


Figure 2: The annotated partially orderings generated by SPRAWL for the three totally ordered plans shown in Figure 1.

Instead of looking for the optimal (according to some metric) partially ordered plan to solve a problem, we chose to focus on finding partial orderings *consistent* with the given totally ordered plan, or those in which all relevant effects were also active in the total ordering. There are two reasons for this. The first is that the totally ordered plan contains a wealth of valuable information about how to solve the problem, including which operators to use and which conditional effects are relevant. The second is that for many applications, including plan modification and reuse and agent modelling, it is important to be able to analyze an observed or previously generated plan (for example, to find characteristic patterns of behavior or to identify unnecessary steps).

However, since our purpose is to reveal underlying structure, we do have some requirements on the form of the resulting partial ordering; we allow only ordering constraints that affect the fulfillment of the goal terms—those that provide for or protect relevant effects. SPRAWL achieves this via a two-phased approach. It first uses needs analysis to identify the relevant effects of each operator and the needs of each operator (which terms must be true before the operator executes in order to ensure that the relevant effects occur). SPRAWL is able to use this to find the annotated partial order by treating the operators almost as though they have no conditional effects—using the needs as preconditions and the relevant effects as non-conditional effects.

The remainder of this paper is organized as follows. We first discuss related work in plan analysis. Then we introduce the needs analysis technique, illustrate its behavior and discuss its complexity. Next, we explain how the SPRAWL algorithm uses needs analysis to find a partial ordering and discuss the complexity of the entire algorithm. We then discuss the limitations and capabilities of the algorithm and present our conclusions.

Related Work

Many researchers have addressed the problems of annotating orderings and of finding partially ordered plans. We discuss a selection of the research investigating annotation and partial ordering.

Triangle tables are one of the earliest forms of annotation (Fikes, Hart, & Nilsson 1972). In this approach, totally ordered plans are expanded into triangle tables that display which add-effects of each operator remain after the execution of each subsequent operator. From this, it is easy to compute which operators supply preconditions to other operators, and thus to identify the relevant effects of each operator and why they are needed in the plan. Fikes, Hart, and Nilsson used triangle tables for plan reuse and modification. The annotations help to identify which subplans are useful for solving the new problem and which operators in these subplans are not relevant or applicable in the new situation.

Regnier and Fade alter the calculation of the triangle table by finding which add-effects of each operator are *needed* by subsequent operators (instead of which add-effects remain after the execution of subsequent operators) (Regnier & Fade 1991). They use the dependencies computed in this modified triangle table to create a partial ordering of the totally ordered plan.

The triangle table approach has been applied only to plans without conditional effects. When conditional effects are introduced, it is no longer obvious what conditions each operator "needs" in order for the plan to work correctly. Although we do not use the triangle table structure, our needs analysis approach can be seen as an extension of the triangle table approach to handle conditional effects.

Another powerful approach to annotation is the validation structure (Kambhampati 1989; Kambhampati & Hendler 1992; Kambhampati & Kedar 1994). This structure is an annotated partial order created during the planning process. Each partial order link is a 4-tuple called a validation: $< e, t', c, t >$, where the effect e of step t' satisfies the condition C of the step t. The validation structure acts as a proof of correctness of the plan, and allows plan modification to be cast as fixing inconsistencies in the proof. This approach is shown to be effective for plan reuse and modification (Kambhampati & Hendler 1992) and for explanation-based generalization of partially ordered and partially instantiated plans (Kambhampati & Kedar 1994). The approach has not been applied to plans with conditional effects. Although (Kambhampati 1989) presents an algorithm for using the validation structures of plans with conditional effects to enable modification and reuse, no method is presented for finding these structures. And since the structures are created during the planning process, no method is presented for finding validation structures of any observed plans, even those without conditional effects.

Derivational analogy (Veloso 1994) is another interesting approach to and use of annotation. In this approach, decisions made during the planning process are explicitly recorded along with the justifications for making them and unexplored alternate decisions. This approach has been shown to be effective for reusing not only previous plans, but also previous lines of reasoning. The approach can handle conditional effects, but, like the validation structure approach, is applicable only to plans that have been created and annotated by the underlying planner.

The final approach to annotation that we will discuss is the operator graph (Smith & Peot 1993; 1996). This approach does not analyze plans, but rather interactions between operators relevant to a problem. The operator graph includes one node per operator, and one node per precondition of each operator. A link is made between each node representing a preconditions of an operator and the operator node, and between the node of each operator which satisfies a particular precondition and the node representing that precondition. A threat link is also added between the node of each operator which deletes a particular precondition and the node representing that precondition. Smith and Peot use these operator graphs before the planning process to discover when threat resolution may be postponed (Smith & Peot 1993) and to analyze potential recursion (Smith & Peot 1996). Operator graphs do not apply to domains with conditional effects, and are less applicable to plan reuse and behavior modelling than other approaches, since they analyze operator interactions, not plans.

There has been some previous work on finding partial orderings of totally ordered plans. As previously mentioned, Regnier and Fade (Regnier & Fade 1991) used triangle tables to do this for plans without conditional effects. Veloso et al also presented a polynomial-time algorithm for finding a partial ordering of a totally ordered plan without conditional effects (Veloso, Pérez, & Carbonell 1990). The algorithm adds links between each operator precondition and the most recent previous operator to add the condition. It then resolves threats and eliminates transitive edges. However, Bäckström shows that this method is not guaranteed to find the most parallel partial ordering, and that, in fact, finding the *optimal* partial ordering according to any metric is NP-complete (Bäckström 1993).

There has been a great deal of research into generating partially ordered plans from scratch. UCPOP (Penberthy & Weld 1992) is one of the most prominent partial-order planners that can handle conditional effects. One of the strengths of UCPOP is its nondeterminism; it is able to find all partially ordered plans that solve a particular problem. However, it is difficult to use the same technique to partially order a given totally ordered plan. The total order contains valuable information about dependencies and orderings, but the UCPOP method would discard this information and analyze the orderings from scratch. Not only is this inefficient, but it may result in a partial ordering of the totally ordered steps that is not consistent with the total order.

Graphplan (Blum & Furst 1997), another well-known partial-order planner, is also able to find partially ordered plans in domains with conditional effects (Anderson, Smith, & Weld 1998). However, it produces non-minimal (overconstrainted) partial orderings, which does not suit our purpose. Consider the plan in which the steps $op\_a\_1 \ldots op\_a\_n$ may run in parallel with the steps $op\_b\_1 \ldots op\_b\_n$. Graphplan would find the partial ordering shown in Figure 3 because it only finds parallelism within an individual time step. In the first time step, op_a_1 and op_b_1 may run in parallel, but there is no other operator that may run in parallel with them, so Graphplan moves to the second time step (in which op_a_2 and op_b_2 may run in parallel). Graphplan constrains the ordering so that no operators from one time step may run in parallel with operators from another. None of the ordering constraints between op_a steps and op_b steps help achieve the goal, so they are not included in the partial ordering created by SPRAWL, shown in Figure 4. SPRAWL reveals the independence of the two sets of operators.

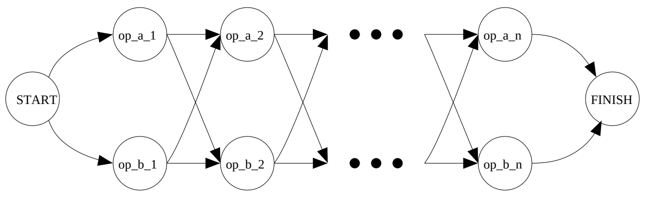


Figure 3: This partial ordering, found by Graphplan, contains many irrelevant ordering constraints.

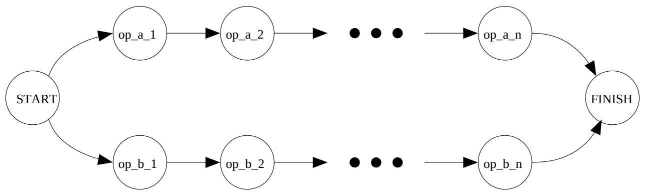


Figure 4: This partial ordering, found by SPRAWL, contains only necessary ordering constraints.

Needs Analysis

Needs analysis, the first step of the SPRAWL algorithm, computes a tree of needs for the totally ordered plan. We first create a goal step called FINISH with the terms of the goal state as preconditions. Needs analysis calculates which terms need to be true before the last step in the plan in order for the preconditions of FINISH to be true afterwards. Then it calculates which need to be true before the second-to-last plan step in order for *those* terms to be true. This calculation is executed for each step of the plan, starting from the last step and finishing at the START step, creating a tree of "needs." This needs tree allows us to identify the relevant effects of a given step and most of the dependencies in the plan. However, not all threats are identified in Needs Analysis; SPRAWL uses the needs tree to calculate the remaining threats.

Needs Tree Structure

In this section, we will discuss the needs that compose the needs tree as well as the structure of the tree. The needs tree consists of three kinds of needs:

1. **Precondition Needs:** the preconditions of a step are called *precondition needs* of the step—they must be true for the step to be executable. For example, the precondition needs of the FINISH step are the goals of the plan.
2. **Creation Needs:** terms that must be true before a step n in order for n to create a particular term or to maintain a previously existing term are called *creation needs* of the term at the step n. In the "use" example in Figure 2, one creation need of the term c at the step op1 is b, since op1 will generate c if b is true before it executes.
3. **Protection Needs:** terms that must be true before step n in order for n not to delete a particular term are called *protection needs* of the term at the step n. In the "prevent" example in Figure 2, one protection need of the term NOT c at the step op1 is NOT b, since if NOT b is not true before step op1, then op1 will add c (thereby deleting NOT c).

For the sake of simplicity, instead of abstract plan steps, we will illustrate the three kinds of needs using plan steps from a domain in which we have a sprinkler that, if on, can wet the yard as well as any object that may be in the yard. Figure 5 shows the operator sprinkle front-yard. The term on sprinkler is a *precondition need* of the step sprinkle front-yard.

To illustrate *creation needs*, let us assume that, after executing the step sprinkle front-yard, wet shoe must be true. This could be accomplished in two ways:

- by ensuring that at shoe front-yard was true before sprinkle front-yard executed, or

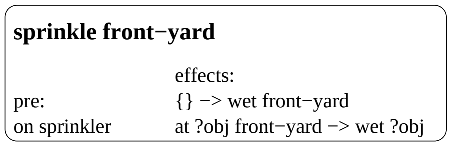


Figure 5: The step sprinkle front-yard.

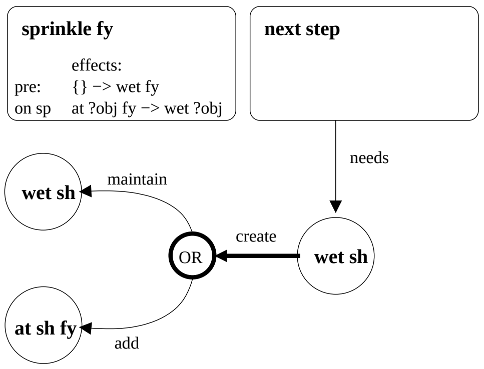


Figure 6: Expanding the need wet shoe in the step sprinkle front-yard. The term wet shoe may be satisfied in either of two ways; this is represented by an OR operator.

- by ensuring that wet shoe was already true before sprinkle front-yard executed, as shown in Figure 6.

These two terms are called *creation needs* of wet shoe at the step sprinkle front-yard, since they provide ways for the term wet shoe to be true after the step sprinkle front-yard.

We must also make a distinction between *maintain* creation needs and *add* creation needs. [2] As mentioned above, there are two ways to ensure that wet shoe is true after the execution of the step sprinkle front-yard, both illustrated in Figure 6. [3] One way is for wet shoe to have been true previously. We call this a *maintain* creation need since the step does not generate the term, but simply maintains a term that was previously true. However, the step sprinkle front-yard could generate the term wet shoe if at shoe front-yard were true before the step executed. We call this an *add* creation need, since we have introduced a new need in order to satisfy another.

Note that, because there may be multiple ways to create a term, the description of needs must include the OR logical operator, as shown in Figure 6. It must also include the AND logical operator, since we allow a conditional effect to have multiple conditions, and in order to guarantee that the effect occurs, we must be able to specify that all must be true.

To illustrate *protection needs*, assume that, after executing the step sprinkle front-yard, the term NOT wet shoe must be true. In order to protect the term NOT wet shoe, we must ensure that NOT at shoe front-yard is true before sprinkle front-yard executes. This is called a *protection need* because it protects the term from being deleted (i.e., prevents wet shoe from being added).

It is not always necessary to generate new needs to satisfy a need term; it may also be satisfied if a non-conditional effect of the step satisfies it, as illustrated in Figure 7. We call such needs *accomplished*, and indicate this in our diagrams with a double circle.

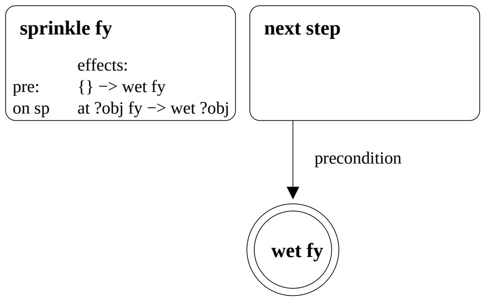


Figure 7: A term may be true after a particular step if a non-conditional effect of the previous step accomplishes it. We indicate this with a double circle around the term.

Needs Analysis Algorithm

The needs analysis algorithm is shown in Table 1. We now describe in detail how needs analysis generates the needs of an individual term. Each needed term t must be created and protected from deletion; we represent this as two branches of needs: creation needs and protection needs. As explained previously, t's creation needs at a particular step n are terms which must be true before step n to ensure that t is true after step n. There are two possibilities for creation needs: either t may have been true before step n, or a conditional effect of step n may generate t [4]. The protection needs of t at step n are terms which must be true before step n to ensure that step n does not delete t. Prevention needs are therefore negated conditions of any conditional effects of step n that delete t. [5] Figure 8 illustrates the needs created to satisfy each needed term.

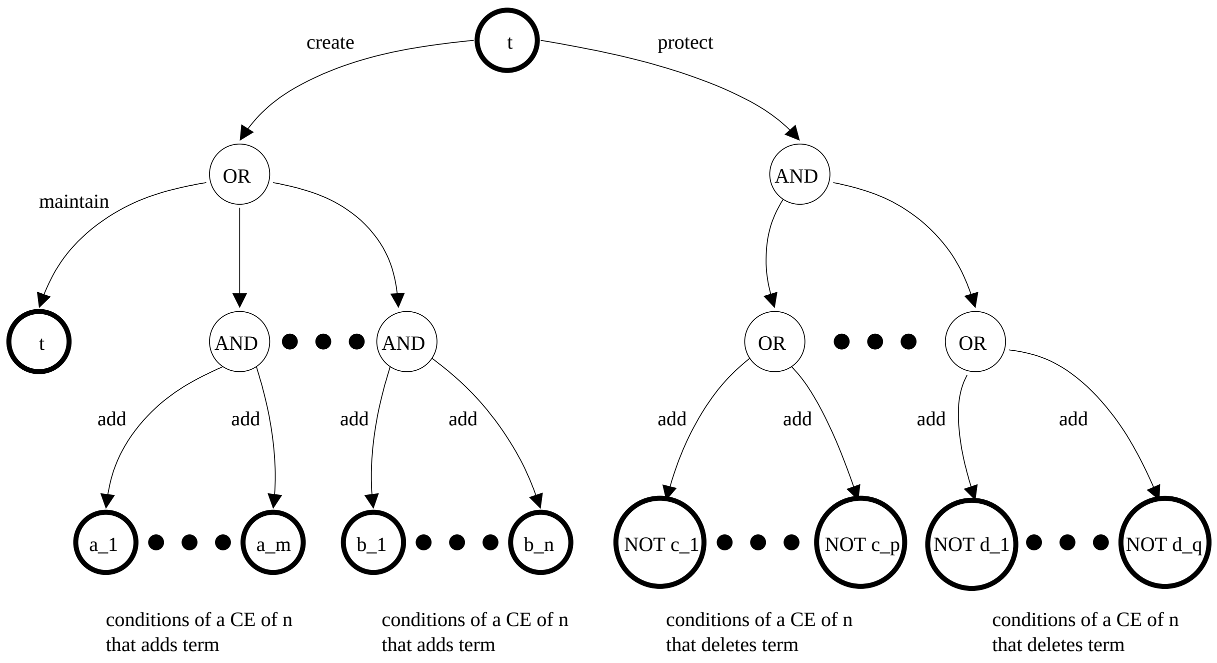
We will use the totally ordered plan from the sprinkler domain shown in Figure 9 to illustrate the behavior of the needs analysis algorithm. First, the algorithm will analyze the last plan step (sprinkle front-yard), which has one precondition need (on sprinkler), to determine how to satisfy the needs of the subsequent step FINISH (wet shoe and wet front-yard). As previously discussed, there are two ways for the step sprinkle front-yard to satisfy wet shoe: either wet shoe could be true before this step executes, or at shoe front-yard must be true before this step executes. So the needs of the term wet shoe are *maintain* wet shoe OR *add* at shoe front-yard. As for wet front-yard, the other precondition need of the FINISH step, it is accomplished by the

[2] Precondition needs and protection needs are always *add* needs.

[3] In the remainder of the sprinkler examples, we abbreviate the literals sprinkler as sp, front-yard as fy, back-yard as by, and shoe as sh.

[4] Non-conditional effects of step n that add t do not add needs—nothing needs to be true before step n in order for them to occur

[5] If t is deleted by a non-conditional effect of step n, then we call it *unsatisfiable* and end its branch of the needs tree.



t
create
protect
OR
AND
maintain
t
AND
AND
OR
OR
add
add
add
add
add
add
add
add
a_1
a_m
b_1
b_n
NOT c_1
NOT c_p
NOT d_1
NOT d_q
conditions of a CE of n that adds term
conditions of a CE of n that adds term
conditions of a CE of n that deletes term
conditions of a CE of n that deletes term

Figure 8: The creation needs of a need at a particular step are calculated by finding all possible ways it can be generated in the previous step and ensuring that at least one of these occurs. The protection needs are calculated by finding all possible ways it can be deleted in the previous step and ensuring that none of these occurs.

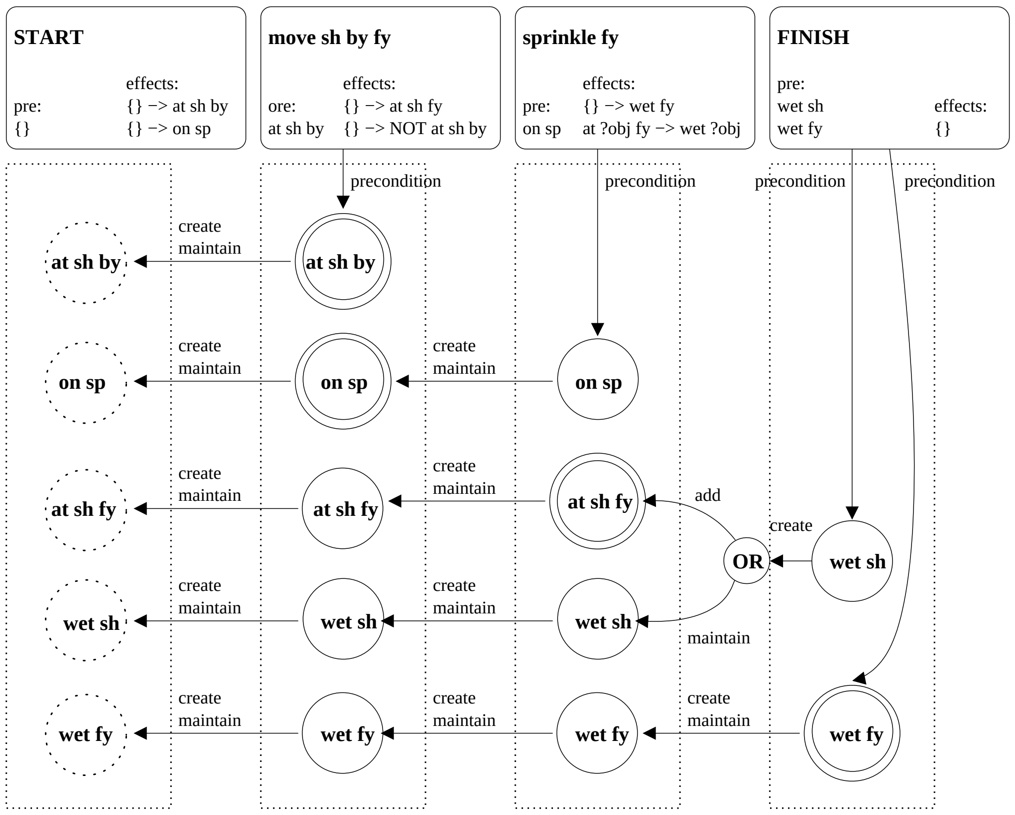


Figure 9: A totally ordered plan in the sprinkler domain and its complete needs tree.

```
Input: A totally ordered plan $\mathcal{T} = S_1, S_2, \ldots, S_n$,
       the START operator $S_0$ with add effects set to the
       initial state, and the FINISH operator $S_n + 1$ with
       preconditions set to the goal state.
Output: A needs tree $N$.

procedure Needs_Analysis($\mathcal{T}$, $S_0$, $S_n + 1$):
1.  for $c \leftarrow$ n+1 down-to $1$ do
2.      for each precond of $S_c$ do
3.          Expand_Term(c, precond)

procedure Expand_Term(c, term):
4.  Find_Creation(c, term)
5.  Find_Protection(c, term)

procedure Find_Creation(c, term):
6.  for each conditional effect of $S_c$ do
7.      if effect adds term then
8.          term.accomplished $\leftarrow$ true
9.      otherwise
10.         Add_Conditions_To_Creation_Needs(effect, term)
11.         for each condition of effect do
12.             Expand_Term(c-1, condition)

procedure Find_Protection(c, term):
13. for each conditional effect of $S_c$ do
14.     if effect deletes term then
15.         term.impossible $\leftarrow$ true
16.         return
17.     otherwise
18.         Add_Conditions_To_Protection_Needs(effect, term)
19.         for each condition of effect do
20.             Expand_Term(c-1, condition)
```

Table 1: Needs Analysis algorithm.

step sprinkle front-yard since it is a non-conditional effect of the step. However, the algorithm continues to look for other ways to accomplish the term. Since there are no conditional effects of sprinkle front-yard that either generate or delete wet front-yard, the algorithm just adds the maintain creation need, *maintain* wet fy.

Next, the algorithm moves back to the previous plan step, move shoe back-yard front-yard, which has the precondition need at shoe back-yard. The needs carried over from previous steps are *maintain* wet shoe OR *add* at shoe front-yard, the creation needs of wet shoe from the FINISH step; *maintain* wet front-yard, the creation need of wet front-yard from the FINISH step; and on sprinkler, the precondition need of the step sprinkle front-yard. The term at shoe front-yard is a non-conditional effect of this step, so it is accomplished, but, as with wet fy in the previous step, the algorithm adds a maintain creation need (*maintain* at shoe front-yard) in order to find other ways to accomplish the term. The terms *maintain* wet shoe, *maintain* wet front-yard, and on sprinkler cannot be prevented or created by this step, so each is satisfied by a maintain creation need (*maintain* wet shoe, *maintain* wet front-yard, and *maintain* on sprinkler).

Finally, the algorithm reaches the initial state, or START step, and is able to determine which branches of the needs tree can be accomplished and which can not. The remaining branches of the tree are at shoe back-yard, *maintain* at shoe front-yard, *maintain* wet shoe, *maintain* wet front-yard, and *maintain* on sprinkler. Two of the needs, at shoe back-yard and *maintain* on sprinkler are accomplished by the START step. However, all of the other remaining needs are not accomplished by the START step. We call these needs *unsatisfiable* and indicate this in our diagrams with a dashed circle.

The complexity of needs analysis is $O(mP(EC)^n)$, where m is the number of steps without conditional effects, n is the number of steps with conditional effects, P is the bound on the number of preconditions, E is the bound on the number of conditional effects in each step, and C is the bound on the number of conditions per conditional effect. Note that the complexity of needs analysis on a plan with no conditional effects is linear: $O(mP)$.

The SPRAWL Algorithm

Table 2 shows the SPRAWL partial ordering algorithm. SPRAWL performs needs analysis, then walks backwards along the needs tree and adds causal links in the partial ordering between steps that need terms and the steps that generate them. The complexity of the SPRAWL algorithm is $O(mP(EC)^n + A * (m + n + 2)^3)$, where m is the number of steps without conditional effects, n is the number of steps with conditional effects, P is the bound on the number of preconditions, E is the bound on the number of conditional effects in each step, and C is the bound on the number of conditions per conditional effect.

Resolving Threats

We rely heavily on the totally ordered plan to help us resolve threats. There are three ways to resolve threats in a plan with conditional effects, as described in (Weld 1994):

1. **Promotion** moves the threatened operators before the threatening operator;
2. **Demotion** moves the threatened operator after the threatening operator;
3. **Confrontation** may take place when the threatening effect is conditional. It adds preconditions to the threatening operator to prevent the effect causing the threat from occurring.

To find all possible partial orderings, all these possibilities should be explored. However, since we are provided the totally ordered plan, we do not need to search at all to find a feasible way to resolve the threat; we can simply resolve it in the same way it was resolved in the totally ordered plan. In fact, if threats are resolved in a different way, then the resulting partial ordering would not be consistent with the totally ordered plan.

If, in the totally ordered plan, the threatening operator occurs before the threatened operators, then promotion should be used to resolve the threat in the partial ordering. Similarly, if it occurs after the threatened operators, demotion should be used to resolve the threat in the partial ordering. If

```
Input: A totally ordered plan $\mathcal{T} = S_1, S_2, \ldots, S_n$,
        the START operator $S_0$ with add effects set to the
        initial state, and the FINISH operator $S_n + 1$ with
        preconditions set to the goal state.
Output: A partially ordered plan shown as a directed graph $\mathcal{P}$.

procedure Find_Partial_Order($\mathcal{T}$, $S_0$, $S_n + 1$):
1.  tree ← Needs_Analysis($\mathcal{T}$, $S_0$, $S_n + 1$)
2.  tree ← Trim_Unaccomplished_Need_Tree_Branches(tree)
3.  for $c$ ← n+1 down-to 1 do
4.      for each precondition of $S_c$ do
5.          Recurse_Need(c, precondition, $\mathcal{P}$)
6.  Handle_Threats(tree, $\mathcal{P}$)
7.  Remove_Transitive_Edges($\mathcal{P}$)

procedure Recurse_Need(c, term, $\mathcal{P}$):
8.  Add_Causal_Link(choose one way to create term, $S_c$, $\mathcal{P}$)
9.  Recurse_Need(c-1, term.create, $\mathcal{P}$)
10. Recurse_Need(c-1, term.protect, $\mathcal{P}$)

procedure Handle_Threats(tree, $\mathcal{P}$):
11. for each causal link $S_i \rightarrow S_j$ do
12.     for $c$ ← 1 up-to $i - 1$ do
13.         if Threatens($S_c$, $S_i \rightarrow S_j$) then
14.             DEMOTE: Add_Causal_Link($S_c$, $S_i$, $\mathcal{P}$)
15.     for $c$ ← $j + 1$ up-to $n$
16.         if Threatens($S_c$, $S_i \rightarrow S_j$) then
17.             PROMOTE: Add_Causal_Link($S_j$, $S_c$, $\mathcal{P}$)
```

Table 2: The SPRAWL algorithm.

the threatening operator occurs between the threatened operators in the totally ordered plan, then we know that confrontation must have been used in the totally ordered plan to prevent the threatening conditional effect from occurring. Needs analysis takes care of confrontation with *protection needs*, shown in Figure 8, which ensure that steps that occur between a needed term's creation and use in the totally ordered plan do not delete the term.

Discussion

The SPRAWL algorithm does not create a partially ordered plan from scratch; its purpose is to partially order the steps of a given totally ordered plan to aid in our understanding of the structure of the plan. Because of this, SPRAWL is restricted to partial orderings consistent with the totally ordered plan.

However, frequently there are many partial orderings consistent with the totally ordered plan. Here, we discuss the space of possibilities explored by SPRAWL as we have described it, and how that space can be extended to include all possible partial orderings consistent with the totally ordered plan.

Different Total Orderings of the Same Steps May Produce Different Partial Orderings

In some cases, a different total ordering of the same plan steps would produce a different partial ordering, but these are cases in which the relevant effects differ. Consider the two totally ordered plans shown in Figures 10 and 11. Although they consist of the same operators, in the first totally ordered plan, the sequence of relevant effects that produces the goal term z is different than the sequence that produces z in the second totally ordered plan. We consider these two plans to be non-equivalent, though they solve the same problem. SPRAWL would never produce the same partial ordering for both of them; the partial orderings would each preserve the same relevant effects as are active in the respective totally ordered plans.

Active Conditional Effects May Differ from Those in Totally Ordered Plan

Though SPRAWL is restricted to partial orderings consistent with the totally ordered plan it is given, this does not mean that all conditional effects active in the totally ordered plan must be active in the partial ordering, or vice versa. There are sometimes irrelevant conditional effects in the totally ordered plan or in the partial ordering, and SPRAWL does not seek to maintain or prevent these irrelevant effects. The ignore case shown as a totally ordered plan in Figure 1 demonstrates this. In this problem, one of the active effects in the totally ordered plan is wet shoe. However, this effect does not affect the fulfillment of the goal state, and so is not a relevant effect. In fact, as is shown in Figure 2, SPRAWL would enforce no ordering constraints between the two steps in its partial ordering. Though the different orderings produce different final states, the goal terms are true in each of these final states, so it doesn't matter which occurs.

Partial Ordering May Not Include All Relevant Effects in Total Ordering

Although, as we discussed, SPRAWL is restricted to partial orderings with no relevant effects not active in the given totally ordered plan, this does not mean that all relevant effects in the totally ordered plan must be relevant effects in the partial ordering. Sometimes, there are several relevant effects in the totally ordered plan that achieve the same aim. Bäckström presented an example that neatly illustrates this. The totally ordered plan is shown with its needs tree in Figure 12. In this plan, two different relevant effects provide the term q to step c—both step a and step b generate q. Choosing a different relevant effect to generate q creates a different partial order. The two partial orders representing each of the two relevant effect choices are shown in Figures 13 and 14.

Finding Multiple Partial Orderings

In the interest of speed, SPRAWL finds exactly one partial ordering and does not search through different partial orderings to find a "better" one according to any measure. The needs analysis algorithm shown in Table 1 produces a needs tree that encompasses all possible partial orderinsg consistent with the totally ordered plan, but the version of SPRAWL shown in Table 2 arbitrarily chooses one possible partial ordering from those represented by the needs tree. SPRAWL can be modified to search through more possible partial orderings, however, finding the best partial ordering according to any measure is NP-complete (Bäckström 1993).

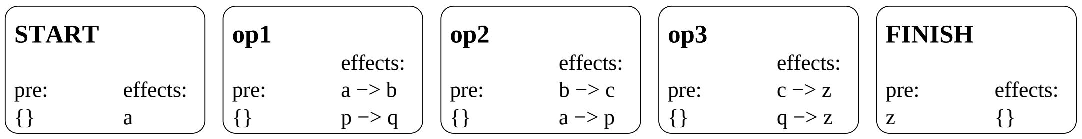


Figure 10: One possible totally ordered plan. In this plan, the goal z is achieved via the first conditional effect of each operator.

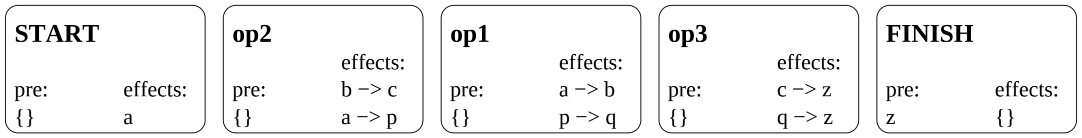


Figure 11: Another possible totally ordered plan achieving the same goals. In this plan, the goal z is achieved via the second conditional effect of each operator.

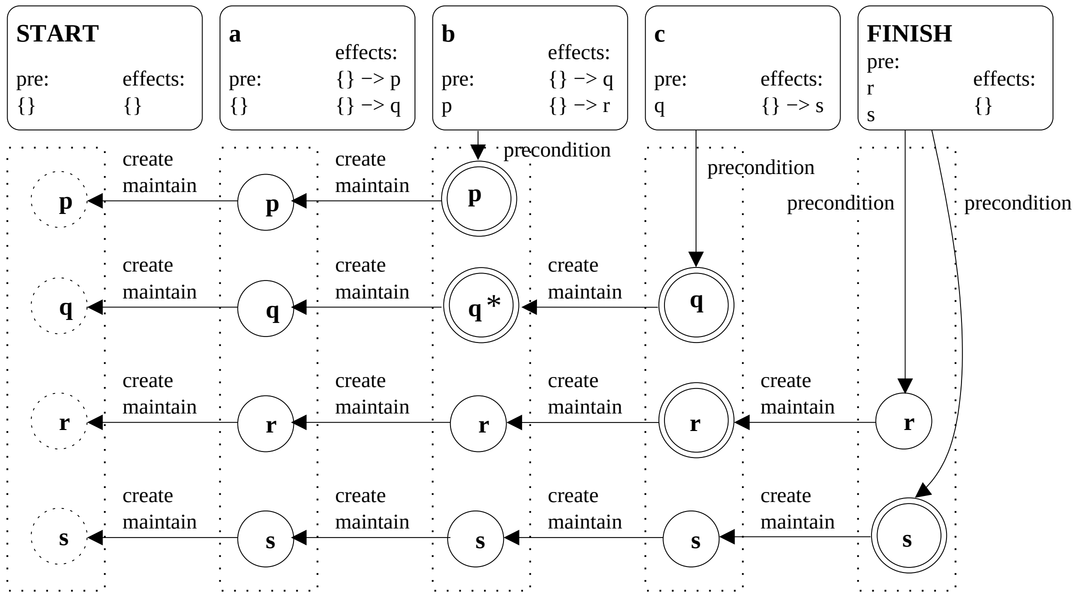
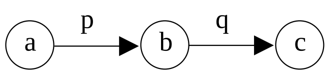
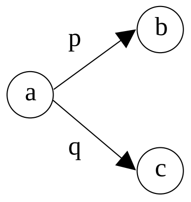


Figure 12: Bäckström's example plan, and the needs tree created if the algorithm does not terminate branches when they are accomplished. Note that the term q is accomplished by two different steps: a and b. This means that two partial orderings are possible: one in which step a provides q to step c, and one in which b does. If branches are terminated as they are accomplished, the accomplished need marked q*, which represents step a providing q to step c, would not be found.



$$\begin{tikzcd} a \arrow[r, "p"] & b \arrow[r, "q"] & c \end{tikzcd}$$

Figure 13: The only partial ordering of Bäckström's example plan permitted by the presented version of the needs analysis algorithm



$$\begin{tikzcd} & b \\ a \arrow[ur, "p"] \arrow[dr, "q"'] & \\ & c \end{tikzcd}$$

Figure 14: Another partial ordering of Bäckström's example plan. If we make the discussed modifications to the needs analysis algorithm, both this partial ordering and the one shown in Figure 13 would be represented in the needs tree, as shown in Figure 12.

When an OR logical operator is encountered in the needs tree, SPRAWL arbitrarily chooses which of its branches to follow and ignores the others (Table 2, step 8). Instead, we could search through the possibilities to find the branch that contributes to the best partial ordering.

If we modify the needs analysis algorithm as discussed above, there is sometimes more than one way to accomplish a need, as with the need q in Figure 12. SPRAWL arbitrarily chooses one of these ways to be the need's creator in the partial ordering (Table 2, step 8). Again, we could search through all possibilities instead, and choose the one that contributes to the best partial ordering.

SPRAWL resolves threats in the same way they were resolved in the totally ordered plan. It is possible instead to search over all three ways (promotion, demotion and confrontation) to resolve each. However, the partial ordering will only be consistent with the totally ordered plan if threats are resolved in the same way.

Conclusions

In this paper, we have described our SPRAWL algorithm for finding minimal annotated consistent partial orderings of observed totally ordered plans. We first described some of the previous work in plan analysis. We then described our novel needs analysis approach to finding the relevant effects and needs of each operator, presented the needs analysis algorithm in detail, illustrated its behavior with an example, and discussed its complexity. We then presented and explained the complete SPRAWL algorithm for finding partial orderings. Finally, we discussed the limitations of the algorithm and some techniques which can extend its capabilities.

Acknowledgements

This research is sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under agreement number No. F30602-00-2-0549 and by a fellowship from the National Science Foundation (NSF). The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of DARPA, AFRL, or NSF.

References

Anderson, C. R.; Smith, D. E.; and Weld, D. S. 1998. Conditional effects in graphplan. In Simmons, R.; Veloso, M.; and Smith, S., eds., *Proceedings of the fourth international conference on Artificial Intelligence Planning Systems*, 44–53. Pittsburgh, PA: AAAI Press.

Bäckström, C. 1993. Finding least constrained plans and optimal parallel executions is harder than we thought. In Bäckström, C., and Sandewall, E., eds., *Current Trends in AI Planning: EWSP'93—2nd European Workshop on Planning*, Frontiers in AI and Applications, 46–59. Vadstena, Sweden: IOS Press.

Blum, A., and Furst, M. 1997. Fast planning through planning graph analysis. *Artificial Intelligence* 90:281–300.

Fikes, R., and Nilsson, N. J. 1971. Strips: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and executing generalized robot plans. *Artificial Intelligence* 3(4):251–288.

Kambhampati, S., and Hendler, J. A. 1992. A validation-structure-based theory of plan modification and reuse. *Artificial Intelligence* 55(2-3):193–258.

Kambhampati, S., and Kedar, S. 1994. A unified framework for explanation-based generalization of partially ordered and partially instantiated plans. *Artificial Intelligence* 67(1):29–70.

Kambhampati, S. 1989. *Flexible Reuse and Modification in Hierarchical Planning: A Validation Structure Based Approach*. Ph.D. Dissertation, University of Maryland, College Park, MD.

Kautz, H. A., and Allen, J. F. 1986. Generalized plan recognition. In *Proceedings of the fifth National Conference on Artificial Intelligence*, 32–37. Philadelphia, PA: AAAI press, Menlo Park, CA.

Pednault, E. 1986. Formulating multiagent, dynamic-world problems in the classical planning framework. In Georgeoff, M., and Lansky, A., eds., *Reasoning about actions and plans: Proceedings of the 1986 workshop*, 47–82. Los Altos, California: Morgan Kaufmann.

Penberthy, J. S., and Weld, D. 1992. UCPOP: A sound, complete, partial-order planner for adl. In Nebel, B.; Rich, C.; and Swartout, W., eds., *proceedings of the third international conference on knowledge representation and reasoning*, 103–114. Cambridge, MA: Morgan Kaufmann.

Regnier, P., and Fade, B. 1991. Complete determination of parallel actions and temporal optimization in linear plans of action. In Hertzberg, J., ed., *European Workshop on Planning*, volume 522 of *Lecture Notes in Artificial Intelligence*. Sankt Augustin, Germany: Springer-Verlag. 100–111.

Smith, D. E., and Peot, M. A. 1993. Postponing threats in partial-order planning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93)*, 500–507. Washington, D.C.: AAAI Press/MIT Press.

Smith, D. E., and Peot, M. A. 1996. Suspending recursion in causal-link planning. In Drabble, B., ed., *Proceedings of the third international conference on Artificial Intelligence Planning Systems*, 182–190.

Veloso, M.; Pérez, A.; and Carbonell, J. 1990. Nonlinear planning with parallel resource allocation. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control*, 207–212. San Diego, CA: Morgan Kaufmann.

Veloso, M. M. 1994. Prodigy/analogy: Analogical reasoning in general problem solving. In Wess, S.; Althoff, K.-D.; and Richter, M., eds., *Topics on Case-Based Reasoning*. Springer Verlag. 33–50.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.