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 valid plan to generate the underlying plan rationale. Although there are several successful previous contributions to this rationale-extraction problem, we realize that 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 that link all the literals, including the conditional effects, 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 introduce the concept and details of needs analysis, present the complete algorithm, and provide illustrative example. We discuss the challenges we faced in handling the rationale and conditional effects.

Introduction

Much of the work on plan reuse, plan recognition and agent modelling has been founded on the analysis of example plans and executions. One of the most common approaches to plan analysis has been to create an *annotated ordering* of the example plan (Fikes, Hart, & Nilsson 1972; Regnier & Fade 1991; Kambhampati & Hendler 1992; Kambhampati & Kedar 1994; Veloso 1994), in which the plan is supplemented with a rationale for (some of) the ordering constraints. Annotated orderings allow systems not only to more flexibly reuse 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). Despite the success of the annotated ordering approach, it has not been applied to domains with conditional effects. In this paper, we introduce the SPRAWL algorithm for finding *minimal annotated consistent partial orderings* of observed totally ordered plans.

We assume that we are given or observe a plan that is valid, i.e., the steps in the plan are *needed* in the sense that they have a reason to be part of the plan. We aim at linking 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 to be easily matched, removed, and used independently.

Conditional effects difficult the task of annotating a plan because they cause the effects of a given step to change depending on what steps come before it, thus making the effects of a step difficult to predict. In fact, an 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. The preconditions (pre) are listed and the effects are all represented as conditional effects $\{a\} \rightarrow b$, i.e., if a then add b. A "regular" effect for adding a literal b is then represented as $\{\} \rightarrow b$. Delete effects are represented as negated terms.

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 to the plan. Figures 2 and 3 show the partial orders representing the *use* and *ignore* cases, respectively. Although the totally ordered plans for these two cases are composed of the same steps in the same order, the partial orderings are very different. Treating any conditional effect

[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.

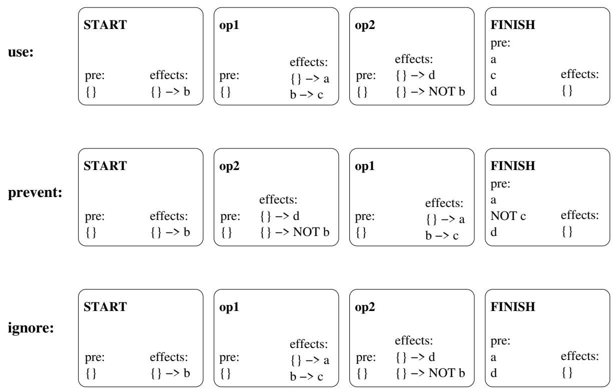

Figure 1: Three totally ordered plans that illustrate the three possible ways of treating a conditional effect in an ordering: using it to achieve a goal, preventing it in order to achieve a goal, or ignoring its effect.

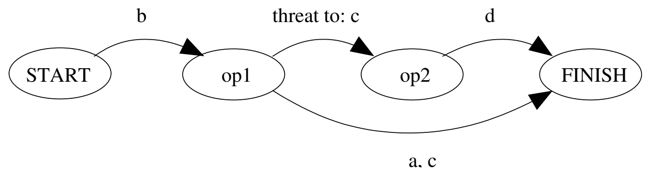

Figure 2: The annotated partially ordered plan which uses the conditional effect of `op1` to achieve a goal.

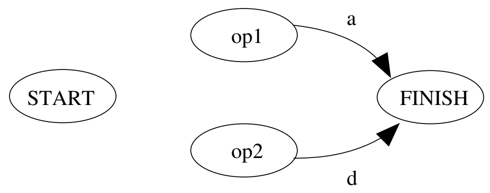

Figure 3: The annotated partially ordered plan which ignores the conditional effect of `op1`.

in a different way will result in a different partial ordering. One way to deal with this 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.

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).

There are cases in which 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 4 and 5. 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.

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 which affect the fulfillment of the goal terms—those which provide for or prevent relevant 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, present formal definitions for the concepts we use and introduce, and finally present our conclusions.

Related Work

Triangle tables: (Fikes, Hart, & Nilsson 1972; Regnier & Fade 1991) FOR REUSE/MACRO LEARNING: store generalized plans in a table that shows which add-effects of each op remain after each subsequent op, & therefore which ops provide precs to subsequent ops–helps to know how to use subplans–can identify which ops are irrelevant for partial reuse (how?) mention the need to be able to "identify the role of each operator in the overall plan: what its important effedcts are (as opposed to side effects) and why these effects are needed in the plan."

FOR PARALLELISM: reg & fade use triangle table to build po from a to. remove "artificial" (vs. "genuine") ordering constraints. find "complete set of actions which can be executed in parallel". change computation of triangle tables, slightly. no ce handling.

Validation structures: (Kambhampati & Hendler 1992; Kambhampati & Kedar 1994) FOR REUSE/MOD: given a p.o., constructs list of validations: 4-tuple (provided effect, providing op, relying condition, relying op). no ces. no validations for threats, but some computation over validations finds threats for you. validation structure acts as proof of correctness. modification = fixing inconsistencies in proof

Annotated "decision-making" rationale: (Veloso 1994)

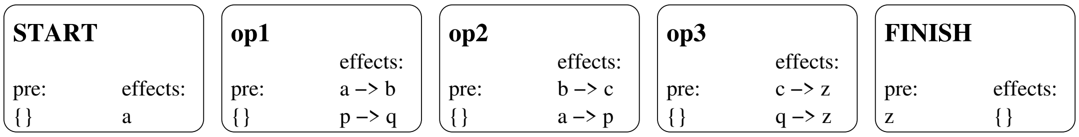

Figure 4: One possible totally ordered plan. In this plan, the goal z is achieved via the first conditional effect of each operator.

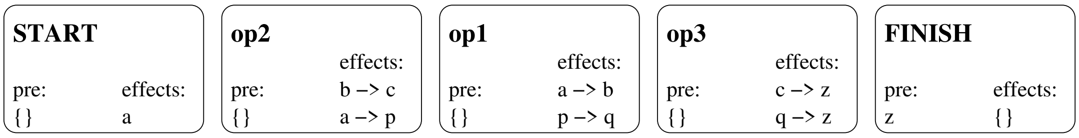

Figure 5: Another possible totally ordered plan achieving the same goals. In this plan, the goal z is achieved via the second conditional effect of each operator.

FOR REUSE/MOD: analogical reasoning—stores cases supplemented with "decision-making rationale" in order to be able to reuse rationale, not just old plan.

Operator graphs & their various-&-sundry uses: (Smith & Peot 1993; 1996) capture interaction between ops by chaining back from goal in very needs-analysis-y sort of way. one node for each operator. used for threat analysis, threat postponing, analyze/identify/avoid "recursions" (a, a-1, a, a-1...).

Goal agendas: (Koehler & Hoffmann 2000) use various methods (including planning graphs) to find "goal agendas"—an ordering for in which order to attack goals. still exponential time, since it doesn't remember the plans, but less exponential, since fewer threat difficulties.

previous partial ordering work: (Veloso, Pérez, & Carbonell 1990) found a po from a non-ce to; no annotations (Bäckström 1993) found that it's np-complete to find a best po, given a to. (Kambhampati 1996) delay threat resolution by using disjunctive orderings—we use disjunctions, too! is this too tenuous a link??

various po-ish planning methods: (Weld 1994) given problem, finds PO. can handle CEs, but does not annotate.

One of the most popular and efficient partial-order planners, Graphplan (Blum & Furst 1997), produces 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 $op\_b\_1 \ldots op\_b\_m$. Graphplan would find the partial ordering shown in Figure 6, which forces *opp* to be one of the first two steps of the plan. The ordering constraint between *opp* and *op*2 does not help achieve the goal, so it would not have been included in a partial ordering created by SPRAWL. SPRAWL would find the partial ordering shown in Figure 7, which allows *opp* to run in parallel with any of the other steps. POINT OUT IRRELEVANT LINKS IN GRAPHPLAN VERSION. NO POSSIBLE REASON FOR THEM.

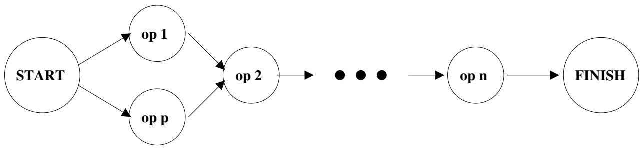

Figure 6: The partial ordering found by Graphplan.

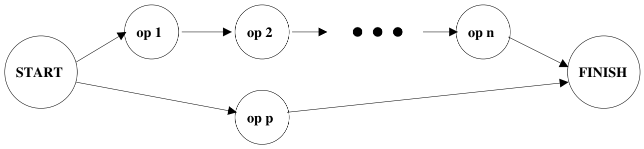

Figure 7: The partial ordering found by SPRAWL.

Needs Analysis

Our first step in finding a partial ordering is to do needs analysis on the totally ordered plan. Needs analysis begins by creating a goal step called FINISH, as in (Smith & Peot 1993), with the terms of the goal state as preconditions. Then it 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, and then which need to be true before the second-to-last plan step in order for *those* terms to be true. We continue this calculation all the way backwards to the initial state, building up a tree of "needs." This needs tree allows us to identify easily the relevant effects of a given step and most of the dependencies in the plan. However, threats not active in the totally ordered plan are not identified by needs analysis, and must be found afterwards.

Needs Tree Structure

In this section, we will discuss the needs that compose the needs tree as well as the structure of the tree.

There are three kinds of needs in the needs tree:

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;
2. Creation Needs terms which must be true before step n in order for step n to create a particular term (or maintain a previously existing term) are called *creation needs* of the term;

3. Protection Needs terms which must be true before step n in order for step n not to delete a particular term are called *protection needs* of the term.

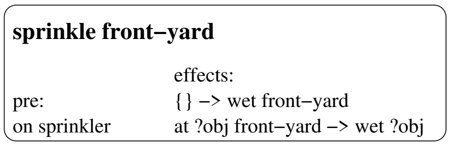

Figure 8: The step sprinkle front-yard.

We will use the plan step shown in Figure 8 to illustrate the three kinds of needs. 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 by ensuring that at shoe front-yard was true before sprinkle front-yard executed or by ensuring that wet shoe was already true before sprinkle front-yard executed, as shown in Figure 9. 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. 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.

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 9. 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.

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 10. We call such needs *accomplished*.

The description of needs must include logical operators. In the example shown in Figure 9,the needs of wet shoe are wet shoe OR at shoe front-yard. Only one of the two needs to be true to satisfy wet shoe. ANDs and NOTs are also necessary.

Needs Analysis Algorithm

The needs analysis algorithm is shown in Table 1, and Figure 11 illustrates in detail how it generates the needs of an individual term. The complexity of needs analysis is

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

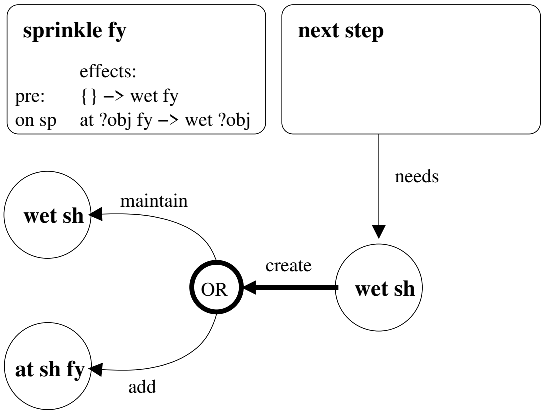

Figure 9: Expanding the need wet shoe in the step sprinkle front-yard. The term wet shoe may be satisfied in either of two ways; this is represented by an OR operator.

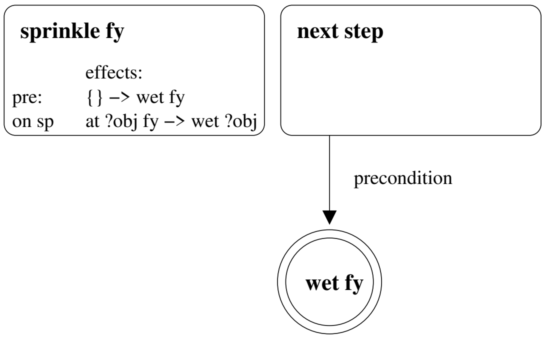

Figure 10: A term may be true after a particular step if a non-conditional effect of the previous step accomplishes it. We indicate this with a double circle around the term.

$O(mP(EC)^n)$, 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: $O(mP)$.

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(T, S_0, S_n + 1):
  for c ← n+1 down-to 1 do
    for each precond of S_c do
      Expand_Term(c, precond)

procedure Expand_Term(c, term):
  Find_Creation(c, term)
  Find_Prevention(c, term)

procedure Find_Creation(c, term):
  for each conditional effect of S_c do
    if effect adds term then
      term.accomplished ← true
    otherwise
      Add_Conditions_To_Creation_Needs(effect, term)
      for each condition of effect do
        Expand_Term(c-1, condition)

procedure Find_Prevention(c, term):
  for each conditional effect of S_c do
    if effect deletes term then
      term.impossible ← true
      return
    otherwise
      Add_Conditions_To_Prevention_Needs(effect, term)
      for each condition of effect do
        Expand_Term(c-1, condition)
```

Table 1: Needs Analysis algorithm.

We will use the totally ordered plan from the sprinkler domain shown in Figure 12 to illustrate the behavior of the needs analysis algorithm. First, the algorithm will create the FINISH step that has, as its precondition needs, the goal terms. Then it will move to the last plan step (sprinkle front-yard), which has one precondition need, to determine how to satisfy the needs of the subsequent step (FINISH). 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 *add* wet front-yard, the other precondition need of the FINISH step, it is accomplished by the step sprinkle front-yard since it is a non-conditional effect of the step.

Next, the algorithm moves back to the previous plan step, move shoe back-yard front-yard, which has one precondition need. The needs carried over from previous steps are *maintain* wet shoe OR *add* at shoe front-yard, the creation need of wet shoe 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. The term wet shoe cannot be prevented or created by this step, so it is satisfied by a maintain creation need: *maintain* wet shoe. The term on sprinkler also cannot be prevented or created by this step, so it, too, is satisfied by a maintain creation need: *maintain* on sprinkler.

Finally, the algorithm reaches the initial state, or START step, and it is able to determine which branches of the needs tree can be accomplished and which can not. The remaining branches of the tree are *add* at shoe back-yard, *maintain* on sprinkler, and *maintain* wet shoe. Two of the needs, *add* at shoe back-yard and maintain on sprinkler are true in the initial state (accomplished by the START step). However, *maintain* wet shoe is cannot be accomplished by the START step, so we call its branch of the tree *unsatisfiable*.

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

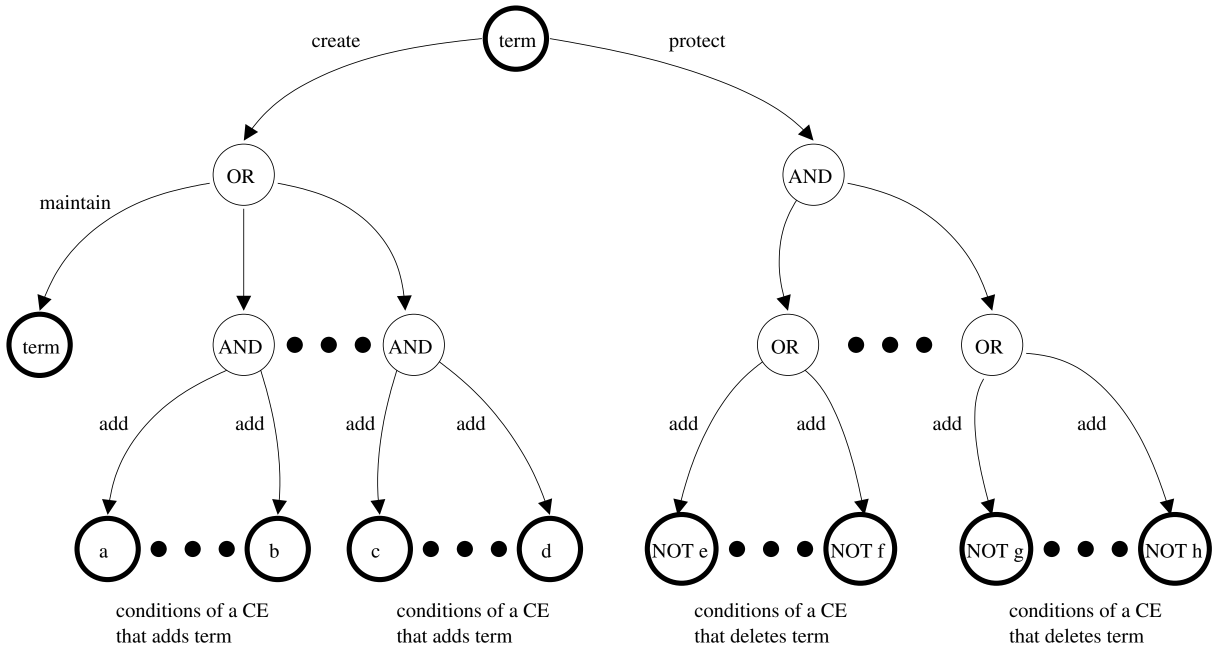

Figure 11: The creation needs of a need at a particular step are calculated by finding all possible ways it can be generated in the previous step and ensuring that at least one of these occurs. The protection needs are calculated by finding all possible ways it can be deleted in the previous step and ensuring that none of these occurs.

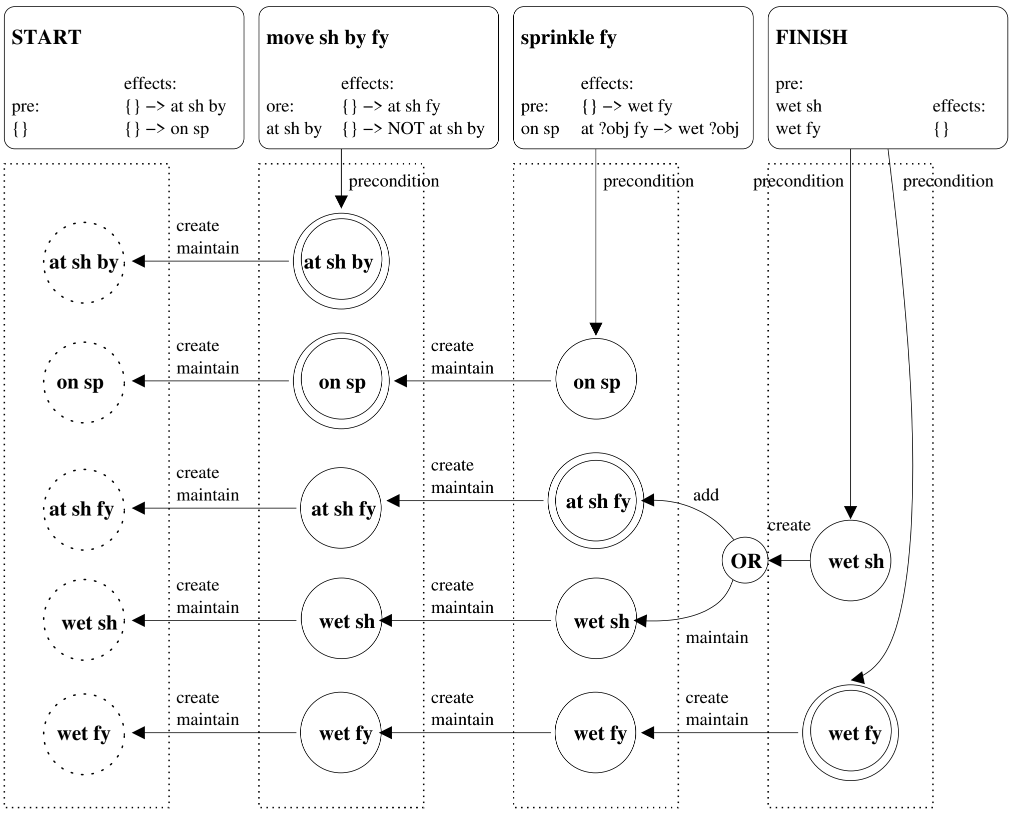

Figure 12: A totally ordered plan in the sprinkler domain and its complete needs tree.

```
Input: A totally ordered plan T = S_1, S_2, ..., 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 P.

procedure Find_Partial_Order(T, S_0, S_n + 1):
  tree ← Needs_Analysis(T, S_0, S_n + 1)
  tree ← Trim_Unaccomplished_Need_Tree_Branches(tree)
  for c ← n+1 down-to 1 do
      for each precondition of S_c do
          Recurse_Need(c, precondition, P)
  Handle_Threats(tree, P)
  Remove_Transitive_Edges(P)

procedure Recurse_Need(c, term, P):
  Add_Causal_Link(choose one way to create term, S_c, P)
  Recurse_Need(c-1, term.create, P)
  Recurse_Need(c-1, term.protect, P)

procedure Handle_Threats(tree, P):
  for each causal link S_i → S_j do
      for c ← 1 up-to i − 1 do
          if Threatens(S_c, S_i → S_j) then
              DEMOTE: Add_Causal_Link(S_c, S_i, P)
      for c ← j + 1 up-to n
          if Threatens(S_c, S_i → S_j) then
              PROMOTE: Add_Causal_Link(S_j, S_c, P)
```

Table 2: The SPRAWL algorithm.

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 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 11 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.

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 3, 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 which achieve the same aim. Bäckström presented an example that neatly illustrates this. The totally ordered plan is shown with its needs tree in Figure 13. 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 14 and 15.

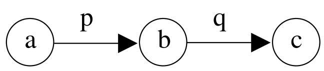

Figure 14: The only partial ordering of Bäckström's example plan permitted by the presented version of the needs analysis algorithm

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).

When an OR logical operator is encountered in the needs tree, SPRAWL arbitrarily chooses which of its branches to

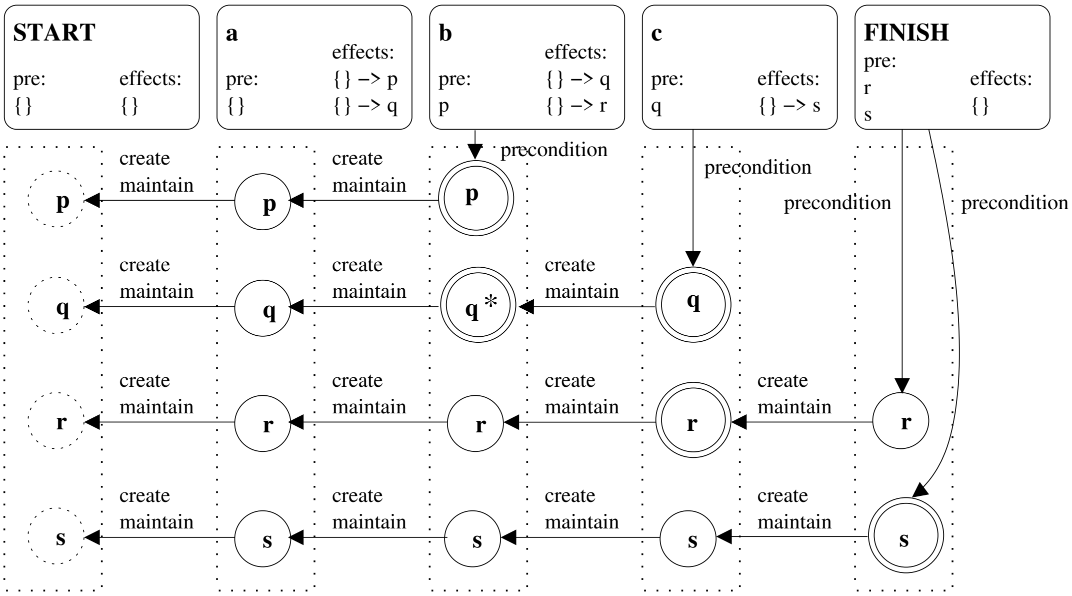

Figure 13: Bäckström's example plan, and the needs tree created if the algorithm does not terminate branches when they are accomplished. Note that the term q is accomplished by two different steps: a and b. This means that two partial orderings are possible: one in which step a provides q to step c, 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.

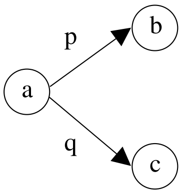

Figure 15: Another partial ordering of Bäckström's example plan. If we make the discussed modifications to the needs analysis algorithm, both this partial ordering and the one shown in Figure 14 would be represented in the needs tree, as shown in Figure 13.

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 13. 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.

Definitions

Totally ordered plan $\mathcal{T}$ *consists of an initial state,* $\mathcal{I}$*, which is a list of terms that are true before the plan begins; a goal state,* $\mathcal{G}$*, which is a list of terms that must be accomplished by the plan; and a list of steps,* $S_1 \ldots S_n$*. Each step has a list of preconditions, or terms that must be true before the step is executable; and a list of conditional effects, which describe the effects of the step. Each conditional effect has a list of conditions and a list of effects, which become true after the plan step executes if the conditions of the effect were satisfied before the plan step executed. The preconditions of the first step in the plan,* S_1*, must be true in the initial state* $\mathcal{I}$*; the preconditions of each subsequent step* S_i *must be true after steps* $S_1 \ldots S_{i-1}$ *execute in order; and the terms of the goal state* $\mathcal{G}$ *must be true after steps* $S_1 \ldots S_n$ *execute in order.*

Partial ordering $\mathcal{P}$ *A partial ordering* $\mathcal{P}$ *of the totally ordered plan* $\mathcal{T}$ *also includes a list of* ordering constraints*. Each ordering constraint specifies that a given step* S_i *must come before another step* S_j*. The preconditions of the first step in the plan,* S_1*, must still be true in the initial state* $\mathcal{I}$*. However, we now demand that the preconditions of each subsequent step* S_i *must be true after any possible ordering of the plan steps that ends at* S_i *that is consistent with the ordering constraints; and that the terms of the goal state* $\mathcal{G}$ *must be true after steps* $S_1 \ldots S_n$ *execute in any order consistent with the ordering constraints.*

Annotated Ordering *an ordering of plan steps supplemented with a rationale for (some of) the ordering constraints.*

Relevant Effect *an effect which affects the fulfillment of a goal term. MORE RIGOROUS DEFINITION IS COMING*

Incidental Effect *an effect which does not affect the fulfillment of a goal term. MORE RIGOROUS DEFINITION IS COMING*

Consistent *a partial ordering $\mathcal{P}$ is consistent with the totally ordered plan $\mathcal{T}$ if all relevant effects active in $\mathcal{P}$ are also active in $\mathcal{T}$.*

Minimal Annotated Consistent Partial Ordering *a partial ordering consistent with the totally ordered plan in which each ordering constraint either provides a term which a relevant effect depends upon or prevents a threat to such a term, and in which each ordering constraint is annotated with which term it provides or protects.*

Conclusions

THIS WAS GARBAGE AND NEEDS TO BE REWRITTEN. BASIC MESSAGE, WE HAVE PRESENTED, YAP YAP YAP, AND WE HOPE IT OPENS THE FIELD FOR PLAN REUSE/ADAPTATION/RECOGNITION AND AGENT MODELLING TO RICHER DOMAIN LANGUAGES WITH CEs.

References

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. 1996. Using disjunctive orderings instead of conflict resolution in partial order planning. Technical Report ASU CSE TR 96-002, Arizona State University, Tempe, Arizona.

Koehler, J., and Hoffmann, J. 2000. On reasonable and forced goal orderings and their use in an agenda-driven planning algorithm. *Journal of Artificial Intelligence Research* 12:338–386.

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.

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 *Topics on Case-Based Reasoning*. Springer Verlag. 33–50.

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