A Unified Framework for Explanation-Based Generalization of Partially Ordered and Partially Instantiated Plans

Subbarao Kambhampati*†
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287

Smadar Kedar
Sterling Federal Systems
AI Research Branch
NASA AMES Research Center
Moffett Field CA 94035

ASU CSE Technical Report 92-008
April, 1992, Revised: February, 1993
To appear in *Artificial Intelligence*

Running head: EBG of POPI Plans

* A preliminary version of this paper has been presented at the 9th National Conference on Artificial Intelligence (AAAI-91), Anaheim, CA, USA.

† The first author was partially supported by the Office of Naval Research under contract N00014-88-K-0620 (to Stanford University), and by NSF under grant IRI-9210997 (to Arizona State University). The authors' *email* addresses are *rao@asuvax.asu.edu* and *kedar@ptolemy.arc.nasa.gov*.

Abstract

Most previous work in explanation based generalization (EBG) of plans dealt with totally ordered plans. These methods cannot be directly applied to generalizing partially ordered partially instantiated plans, a class of plans that have received significant attention in planning. In this paper we present a natural way of extending the explanation-based generalization methods to partially ordered partially instantiated (POPI) plans. Our development is based on Modal Truth Criteria for POPI plans [3]. We develop explanation structures from these truth criteria, and use them as a common basis to derive a variety of generalization algorithms. Specifically we present algorithms for precondition generalization, order generalization, and possible correctness generalization of POPI plans. The systematic derivation of the generalization algorithms from the Modal Truth Criterion obviates the need for carrying out a separate formal proof of correctness of the EBG algorithms. Our development also systematically explicates the tradeoffs among the spectrum of possible generalizations for POPI plans, and provides an empirical demonstration of the relative utility of EBG in partial ordering, as opposed to total ordering, planning frameworks.

1 Introduction

Creating and using generalized plans is a central problem in machine learning and planning. This paper addresses the problem of generalizing a class of plans known as *partially ordered partially instantiated plans*[1] (POPI plans), that have been extensively investigated in the planning literature [31, 35, 38, 3]. The advantages of POPI plans in plan generation -- such as improved efficiency through least commitment and flexible plan representation -- are well known (cf. [24, 1, 21]). In addition to these, POPI plans also promise a variety of attractive tradeoffs in plan generalization and learning to improve planning performance:

- POPI plans provide a compact way of representing and reasoning with an exponential number of totally ordered plans, thereby allowing attractive storage compactions.
- POPI plans can be generalized in multiple ways: precondition generalizations, order generalizations, possible correctness generalizations etc. These generalizations present interesting plan time vs. generalization time tradeoffs.
- During reuse, POPI plans afford greater flexibility in modifying the retrieved plan (*cf* [13]).

In spite of these potential advantages, the problem of POPI plan generalization has received relatively little attention in the machine learning and planning communities. Much of the previous work on explanation based generalization (EBG) of plans has instead been focused on totally ordered plans (see Section 1.1). In this paper, we will show that generalization methods developed for totally ordered plans cannot be directly applied to POPI plans. We will then develop a unified framework for explanation based generalization of POPI plans. Our development is based on the Modal Truth Criteria [3] , which state the necessary and sufficient conditions for ensuring the truth of a proposition at any point in a plan for particular classes of POPI plans. We develop explanation structures from these truth criteria, and use them as a common basis to derive a spectrum of generalizations including precondition generalization, order generalization and generalization for possible correctness. Our development explicates the spectrum of plan time vs. generalization time tradeoffs offered by the various generalizations. We also provide an empirical analysis of the utility of basing EBG based plan reuse in POPI vs. total ordering planning frameworks.

The paper is organized as follows. In the rest of this section, we shall discuss past work on plan generalization, and explain the motivations behind our current work. In Section 2, we present some terminology regarding POPI plans, and formalize the notion of POPI plan generalization. Section 3 reviews modal truth criteria and develops the explanation of correctness of a POPI plan based on a Modal Truth Criterion. Section 4 discusses the basic precondition and order generalization algorithms based on these explanations: Section 4.1 describes precondition generalization methods, Section 4.2 describes the order generalization methods, and Section 4.3

[1] POPI plans are also widely referred to as *nonlinear plans* in the planning literature. We prefer the former term as it avoids confusion with the linearity assumption.

describes ways of combining order and precondition generalization algorithms. Section 5 discusses extensions to the basic generalization algorithms: Section 5.1 describes how the algorithms in Section 4.1 can be modified to compute generalized conditions for possible correctness. Section 5.2 describes the effect of using more general truth criteria, and Section 5.3 contrasts the notion of weakest preconditions with that of the generalizations developed in this paper. Section 6 summarizes the spectrum of generalizations explored in this paper, and discusses the tradeoffs offered by them. Section 7 discusses the relative utility of EBG in total ordering vs. partial ordering planning. Section 8 discusses related work and Section 9 concludes by summarizing the contributions of this paper, and discussing directions for future work.

1.1 Previous Work and Motivation

The problem of generalizing a plan has traditionally been characterized as that of computing the *weakest* (most general) initial conditions of a sequence of operators. The computed conditions are required to describe exactly the set of initial states such that the generalized plan applicable in those states is guaranteed to achieve a state matching the goals.

Goal regression [29], explanation-based generalization (EBG) [26] [7] [28], and macro-operator formation [10], are some previous analytic solutions to the plan generalization problem. These methods were developed for totally ordered plans. They typically compute the weakest conditions of such plans by regressing variablized goals back through the plan operator sequence to ensure appropriate *producer-consumer* dependencies among effects and preconditions in the generalized plan, and to prevent deletions of needed literals.

Another class of plans that have been widely investigated in the planning community are the so called partially ordered partially instantiated plans, or POPI plans. A POPI plan corresponds to a set of totally ordered totally instantiated plans (aka *completions*, see Section 2). The generalization methods developed for totally ordered plans (discussed above) cannot be directly applied to generalizing POPI plans, since they do not capture all interactions among plan operators for all completions such plans. To illustrate this limitation, consider the simple blocks world problem for stacking four blocks (4BSP) and a POPI plan for solving it, shown in Figure 1. Given an initial state where four blocks *A, B, C, D* are on the table and clear, the goal $On(A, B) \wedge On(C, D)$ can be achieved by the POPI plan corresponding to two totally ordered plans, $Puton(A, B) \rightarrow Puton(C, D)$, and $Puton(C, D) \rightarrow Puton(A, B)$ (where the operator template $Puton(x, y)$ is specified as shown in the figure).

For this problem, the generalization algorithms discussed above produce a generalized plan such as the one shown in Figure 2. These generalized preconditions are not sufficient to guarantee that every total order of this partial order will execute successfully. The reason is that the plan was generalized guided by one specific total order, so constraints for other total orders were not accounted for. For example, if a new problem involves stacking the three blocks A, B and C on top of each other, this generalized plan would be applicable, and yet fail for one of the two total orders (as it includes an incorrect total ordering $Puton(A, B) \rightarrow Puton(B, C)$). What is missing

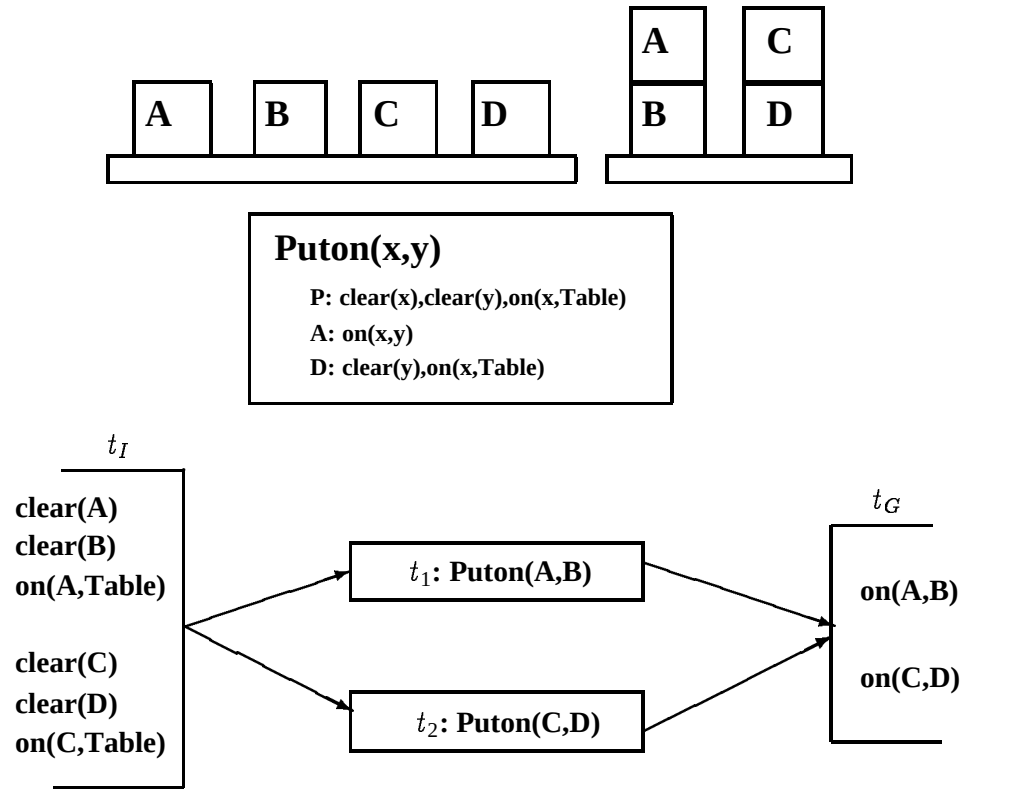


Figure 1: Four Block Stacking Problem (4BSP)

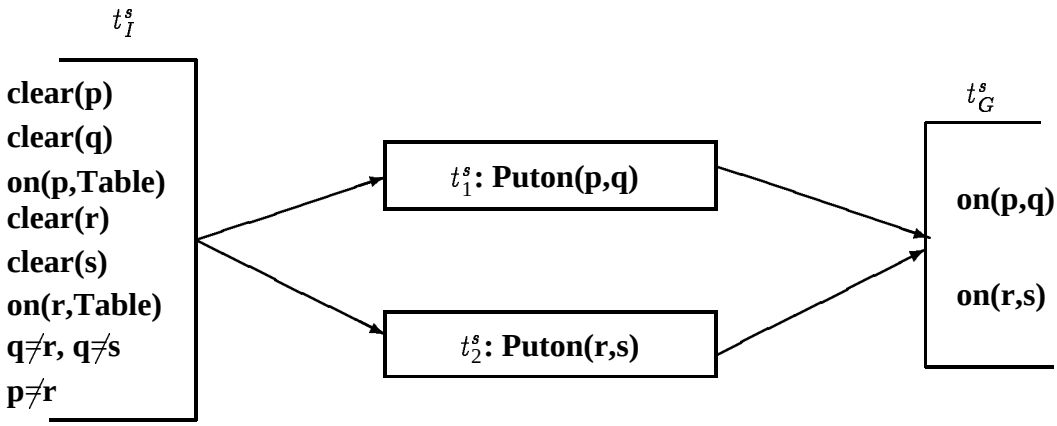


Figure 2: An incorrect generalization of 4BSP

is the constraint that s should not be the same as p (whereas both are codesignating with B in this case).[2]

To avoid this problem, the EBG algorithm needs to be modified to be more systematic in accounting for all possible interactions among operators corresponding to *all* possible linearizations of the plan. There are two potential options for doing this. First is to *modify the algorithm:* For instance, repeatedly compute the generalizations of all total orders of the partial order and then conjoin them together. This approach involves an exponential amount of redundant computation because a POPI plan corresponds to a possibly exponential number of totally ordered totally instantiated plans, while the generalized preconditions of a plan are more directly related to its causal structure.

A better option, explored in this paper, is to *modify the input* to the EBG algorithm: Construct a valid explanation of correctness of the POPI plan, and use that explanation to produce the correct generalized initial conditions for the plan. By modifying the input to EBG, rather than the EBG algorithm, we retain the broad applicability of the algorithm (for different classes of truth criteria, different generalizations can be produced) [3]. We also avoid the redundant computation associated with the first approach.

Our approach is to provide EBG with explanations of correctness of partially ordered plans based on the Modal Truth Criteria [3], which state the necessary and sufficient conditions for ensuring the truth of a proposition at any point in a plan for a class of partially ordered plans. The explanations are represented by a set of dependency links, with well-defined semantics, called *validations*. These explanations are then used as the basis for generalization. The generalizations obtained by this method guarantee successful and interaction-free execution of all total orders of the generalized plan. In addition, the systematic derivation of the generalization algorithms from the Modal Truth Criteria obviates the need for carrying out a separate formal proof of correctness of the EBG algorithms.

More importantly, once we shift to POPI plan representations, we also open up a larger spectrum of possible generalizations. To begin with, in addition to the traditional precondition generalizations, we can also consider order generaliations, or combinations there of. For example, if the planner gave $Puton(A, B) \rightarrow Puton(C, D)$ as the plan for solving the four block stacking problem, we should be able to generalize it by generalizing the preconditions, the order constraints or both. Within the POPI plan representation, the notion of generalization itself can be relaxed to allow for *possible correctness* generalizations, where we attempt to guarantee that at least one instantiation of the POPI generalization is applicable in the new problem situation. We will show that our development provides a unified framework within which all these generalizations can be computed in a principled fashion.

[2]It might be pointed out that in this particular case, adding an implicit universal non-codesignation constraint among all blocks could have allowed even traditional EBG algorithms to get a correct generalization. However, such problem-independent constraints may be unnecessarily strong in some other problems. What we need instead is a way in which the generalization algorithm itself adds the minimum additional constraints while generalizing the plan.

2 Preliminaries and Terminology

Given a planning problem $[\mathcal{I}, \mathcal{G}]$ where $\mathcal{I}$ is a conjunction of literals specifying the initial state and $\mathcal{G}$ is a conjunction of literals specifying the desired goal state, a *partially ordered partially instantiated plan*[3] $\mathcal{P}$ is a 3-tuple $\mathcal{P} : \langle T, O, \rangle$, where T is the set of actions in the plan, and O is a partial ordering relation over T, and is a set of codesignation (binding) and non-codesignation constraints (prohibited bindings) on the variables in $\mathcal{P}$. T contains two distinguished nodes t_I and t_G, where the effects of t_I and the preconditions of t_G correspond to the initial and final states of the plan, respectively. The actions are represented by instantiated STRIPS-type operators with *Add*, *Delete* and *Precondition* lists, all of which are conjunctions of first order literals[4].

O defines a *partial ordering* over T: $O = \{(t_i, t_j) \mid t_i, t_j \in T\}$. We write $t_i \prec t_j$ if either $(t_i, t_j) \in O$, or there exists a sequence of operators $t_1 \cdots t_n \in T$, such that $(t_i, t_1), (t_1, t_2) \cdots (t_n, t_j) \in O$. (Thus, the ''$\prec$'' relation corresponds to the transitive closure of O.) If t_i and t_j are unordered with respect to each other (i.e., $t_i \not\prec t_j$ and $t_j \not\prec t_i$), then we say $t_i \parallel t_j$.

 defines the set of codesignation and non-codesignation constraints on the variables occurring in the plan. We say that a literal p_i can codesignate with another literal p_j (written as $p_i \approx p_j$), if and only if p_i and p_j are unifiable under the variable constraints in . Similarly, we say that p_i *cannot* codesignate with p_j (written as $p_i \not\approx p_j$), if p_i and p_j cannot be unified under the constraints imposed by .[5]

A POPI plan $\mathcal{P} : \langle T, O, \rangle$ corresponds to a set of totally ordered totally instantiated plans called *completions* or *ground linearizations*[6] (denoted by $completions(\mathcal{P})$). Each completion of $\mathcal{P}$ will have the same operators as $\mathcal{P}$, ordered in a *sequence* consistent with O (i.e., corresponds to a topological sort of the operators of $\mathcal{P}$), and with all variables assigned (constant) values that are consistent with .

The modal operators ''$\Box$'' and ''$\Diamond$'' are used to denote the *necessary* and *possible* truth of a statement over all the completions of a plan. The necessary and possible truth of an assertion P are related by the modal equivalence relation: $\Diamond P \equiv \neg\Box\neg P$. Given two literals $On(A, B)$ and $On(x, y)$ in the plan $\mathcal{P}$, we say $\Box[On(A, B) \approx On(x, y)]$ if and only if the codesignation constraints $A \approx x$ and $B \approx y$ belong to . Similarly, we say $\Diamond[On(A, B) \approx On(x, y)]$ if and only

[3]Unless otherwise stated, when we talk about plans in the rest of the paper, we will be referring to partially ordered partially instantiated plans.

[4]We shall use upper case letters for constants and the lower case ones for variables.

[5]Codesignation constraints among literals translate into equalities among variables and constants (domain objects), while the non-codesignation constraints translate into disequalities among variables. For example, $On(A, B) \approx On(x, y)$ if and only if $eq(A, x) \wedge eq(B, y)$, since the most general unifier of the two literals is $\theta = ((A\ x)(B\ y))$. Thus, when we say that $On(A, B)$ can co-designate with $On(x, y)$ in our plan, we mean that θ is consistent with the constraints in (i.e., doesn't constrain x not to co-designate with A or B not to co-designate with y). Similarly, $On(A, B) \not\approx On(x, y)$ if and only if $\neg[eq(A, x) \wedge eq(B, y)]$ (that is $neq(A, x) \vee neq(B, y)$).

[6]Note: There is some confusion in the literature regarding the term **completion**. In this paper, we use it to mean a ground linearization of a POPI plan.

if the non-codesignation constraints $A \not\approx x$ and $B \not\approx y$ *do not* belong to . Similarly, $\Diamond(t_i \prec t_j)$ if and only if t_i can possibly precede t_j in some total ordering of the POPI plan (which means that either $(t_i \prec t_j)$ or $(t_i \parallel t_j)$).[7]

Plan Generalizations: POPI plans, by virtue of their more expressive representation of plans, allow for a variety of generalizations. A POPI plan $\mathcal{P}$ is said to be more general than another plan $\mathcal{P}'$ (written as $\mathcal{P} \sqsupset \mathcal{P}'$) if and only if the set of all completions of $\mathcal{P}'$ is a *subset* of the completions of $\mathcal{P}$:

$$\mathcal{P} \sqsupset \mathcal{P}' \;\Leftrightarrow\; Completions(\mathcal{P}') \subset Completions(\mathcal{P}) \tag{1}$$

We distinguish a generalization of a plan called the ''unconstrained'' (maximal) generalization.

$$unconstrained(\mathcal{P} : \langle T, O, \rangle) \;\equiv\; \langle T, \{t_I \prec t_G\}, \emptyset \rangle$$

It is easy to see that $unconstrained(\mathcal{P})$ is the most general version of $\mathcal{P}$ that still contains the tasks of $\mathcal{P}$.

Next, we define the notion of **correctness preserving generalization** as follows: $\mathcal{P}$ is called a correctness preserving generalization of $\mathcal{P}'$, (written $\mathcal{P} \sqsupset^* \mathcal{P}'$) if and only if $\mathcal{P} \sqsupset \mathcal{P}'$ and both $\mathcal{P}$ and $\mathcal{P}'$ are correct POPI plans (i.e., all their completions are correct). Given two correct POPI plans $\mathcal{P} : \langle T, O, \rangle$ and $\mathcal{P}' : \langle T', O', ' \rangle$, we differentiate between three types of correctness preserving generalizations:

- $\mathcal{P}$ is considered a *precondition generalization* of $\mathcal{P}'$ if and only if $\mathcal{P} \sqsupset^* \mathcal{P}'$ and $T = T'$, $O = O'$ and $\mathtt{TC}(') \supset \mathtt{TC}()$, where TC is the transitive closure operation. In other words, every codesignation and non-codesignation constraint that is consistent with $'$ must also be consistent with .

- $\mathcal{P}$ is considered an *order generalization* of $\mathcal{P}'$ if and only if $\mathcal{P} \sqsupset^* \mathcal{P}'$ and $T = T'$, $= '$ and $\mathtt{TC}(O') \supset \mathtt{TC}(O)$. In other words, every ordering constraint that is consistent with O' must also be consistent with O.

- $\mathcal{P}$ is an *order and precondition generalization of $\mathcal{P}'$ if and only if* $\mathcal{P} \sqsupset^* \mathcal{P}'$ *and* $T = T'$, $\mathtt{TC}(') \supset \mathtt{TC}()$ *and* $\mathtt{TC}(O') \supset \mathtt{TC}(O)$.

Note that all three generalizations discussed above generalize (or relax) the ordering and binding constraints, but leave the steps unchanged. To define the notion of structure generalizations, or generalizations that can also change the plan steps, we have to first define the notion of plan

[7] Note that by this definition, a totally ordered, totally instantiated plan (such as one produced by STRIPS [10]) is just a special case of POPI plans, where O defines a total ordering on the steps of the plan (i.e, $\forall t_i, t_j \in T$, $(t_i \prec t_j) \vee (t_j \prec t_i)$), and there are no variables in the plan (i.e., is such that every variable is constrained to necessarily codesignate with some constant). Thus, the techniques developed here will also be applicable to the former.

refinement. A *refinement* of a POPI plan $\mathcal{P} : \langle T, O, \rangle$ is a totally ordered and totally instantiated plan $\mathcal{P}'$, such that $\mathcal{P}'$ contains all the steps of $\mathcal{P}$, and is consistent with all the ordering and binding constraints of $\mathcal{P}$.(Note that this leaves open the possibility of $\mathcal{P}'$ having additional steps, orderings as well as bindings.) By this definition, every completion of a plan is also a refinement of the plan. Given this definition, we say $\mathcal{P}'$ is a **structural generalization** of $\mathcal{P}$, if and only if every completion of $\mathcal{P}'$ is a *refinement* of $\mathcal{P}$.

3 Explanation of Correctness using Modal Truth Criteria

In [3], Chapman proposed modal truth criteria (MTC), as a formal means of reasoning about (POPI) plans. An MTC provides necessary and sufficient conditions for ensuring the truth of a proposition C before an action t in a POPI plan. In this section, we shall develop the explanation of correctness of a POPI plan in terms of such a truth criterion. For plans containing STRIPS-type operators whose precondition, add and delete lists contain first order literals, a MTC can be stated as follows:[8]

$$
\begin{aligned}
&holds(C, t, \mathcal{P}) \iff \\
&\quad \exists\, t' \text{ s.t. } \Box(t' \prec t) \wedge e \in effects(t') \wedge \Box(e \approx C) \bigwedge \\
&\quad \forall t'' \text{ s.t. } \Diamond(t' \prec t'' \prec t) \\
&\qquad \forall d \in delete(t'') \Box(d \not\approx C)
\end{aligned} \tag{2}
$$

It states that a proposition C holds before an operator t in a POPI plan $\mathcal{P} : \langle T, O, \rangle$ if and only if there exists an operator t' such that an effect e of t' necessarily codesignates with C, and for every operator t'' of the plan that may possibly fall between t' and t, every proposition belonging to the delete list of t'' will necessarily *not* codesignate with C. The truth criterion can usefully be thought of as a completeness/soundness theorem for a version of the situation calculus (*cf.* [3], pp. 340). Alternatively, it can be thought of as a method for doing goal-regression [29] over a class of POPI plans.

In planning, the intended use of the MTC is as a prescription of all possible ways of making a proposition of a POPI plan true during *plan synthesis*. However, the MTC can also be used as the formal basis solely for proving *plan correctness*. In particular, a POPI plan $\mathcal{P} : \langle T, O, \rangle$ is considered *correct* according to the modal truth criterion, if and only if all the goals of the plan, as well as all the preconditions of the individual plan steps can be shown to hold according to the criterion given in equation 2 without extending or modifying $\mathcal{P}$ in anyway.

The explanation of correctness of a plan can then be characterized as a ''proof'' that the plan satisfies this criterion for correctness. The algorithm EXP-MTC shown in Figure 3 constructs the explanation of correctness given a POPI plan, by interpreting equation 2 for each precondition and

[8]For ease of exposition, we will be using a version of Chapman's truth criterion [3] without the *white-knight* clause, in our initial development. Section 5.2 shows how our algorithms can be extended in a straight forward fashion to the more general TWEAK truth criterion.

Algorithm EXP-MTC ($\mathcal{P} : \langle T, O, \rangle$)
  $\mathcal{V} \leftarrow \emptyset$
  foreach $t \in T$ **do**
   foreach $\langle C, t \rangle$ (where $C \in precond(t)$) **do**
    Traverse $\mathcal{P}$ in the reverse topological sorted order and
     find the first operator t' s.t.
          $t' \prec t \wedge \exists e \in effects(t') \wedge \Box(e \approx C) \bigwedge$
          $\forall t''$s.t. $\Diamond(t' \prec t'' \prec t)$,
            $\forall d \in delete(t'')\ \Box(d \not\approx C)$
    if such a t' is found
          then $\mathcal{V} \leftarrow \mathcal{V} \cup \{\langle e, t', C, t \rangle\}$
          else *return failure*
   od od

Figure 3: Explanation Construction Algorithm

goal of the plan. It returns failure if such an explanation structure cannot be found (implying that the plan is incorrect according to the given MTC).

Note that our algorithm represents the computed explanation by a set $\mathcal{V}$ of dependency links; we shall refer to these links as **validations** of the plan [18]. The individual dependency links are of the form $\langle e, t', C, t \rangle$. Intuitively, these represent the interval of operators t' and t over which a literal C needs to hold. C is made true by the effect e of operator t', and is needed at t. It is protected throughout that interval (t', t) from being clobbered, that is, any operator t'' that may *possibly* come between t' and t in some total ordering must not violate the condition C (i.e., t' must not have a delete list literal that can possibly unify with C). The semantics of validations therefore capture both the traditional precondition-effect dependencies and protection violations across all completions. In particular,

$$
\begin{aligned}
v : \langle e, t', C, t \rangle \text{ is a validation of } \mathcal{P} : \langle T, O, \rangle \Longleftrightarrow \\
&\Box(e \approx C) \wedge \Box(t' \prec t) \wedge \\
&\forall t'' \in T \text{ s.t. } \Diamond(t' \prec t'' \prec t) \\
&\qquad \forall d \in delete(t''),\ \Box(d \not\approx C)
\end{aligned}
\tag{3}
$$

For the 4BSP plan shown in Figure 1, the explanation of correctness found by this algorithm would consist of the following validations:

v_1 : $\langle On(A, Table), t_I, On(A, Table), t_1 \rangle$
v_2 : $\langle Clear(A), t_I, Clear(A), t_1 \rangle$
v_3 : $\langle Clear(B), t_I, Clear(B), t_1 \rangle$
v_4 : $\langle On(C, Table), t_I, On(C, Table), t_2 \rangle$
v_5 : $\langle Clear(C), t_I, Clear(C), t_2 \rangle$
v_6 : $\langle Clear(D), t_I, Clear(D), t_2 \rangle$
v_7 : $\langle On(A, B), t_1, On(A, B), t_G \rangle$
v_8 : $\langle On(C, D), t_2, On(C, D), t_G \rangle$

Remarks: The EXP-MTC algorithm shown in Figure 3 finds only *one* out of the possibly many explanations of correctness of the plan. In particular, for each precondition C of an operator t, there might possibly be many operators that can contribute C (according to the criteria stated by MTC). Of these, the algorithm records only the first operator t' encountered in a reverse topological order scan of $\mathcal{P}$ that satisfies the conditions of MTC[9]. It is perfectly reasonable to choose another explanation of correctness (i.e., another set of validation links $\mathcal{V}$) over the one given by this algorithm as long as that explanation also satisfies the MTC. It should however be noted that the generalization phase will be guided by the particular explanation that is chosen at this step (rather than by all possible explanations). This corresponds to a common restriction for EBG termed ''generalizing with respect to the explanation structure'', or ''following the example'' [26].
Complexity: The cost of finding a validation link in the above algorithm is $O(n^2\zeta)$, where ζ is an upper bound on the number of delete literals per operator, and n is the number of operators in the plan[10]. If ξ is the upper bound on the number of preconditions per operator, then there must be $O(\xi n)$ validations links in the explanation. Thus the total cost of explanation construction, using this MTC, is $O(n^3)$.

4 Precondition and Order Generalization methods

4.1 Explanation-Based Precondition Generalization

In this section, we will first use the explanation of correctness developed in the previous section to derive a declarative specification for the generalization phase of EBG for POPI plans. We will then provide an algorithm that interprets this specification.

The generalization process consists of schematizing[11] (variablizing) the plan $\mathcal{P}$ to produce $\mathcal{P}^s$, and determining the weakest co-designation and non-codesignation constraints on the variables under which $\mathcal{P}^s$ will be correct according to the MTC, with respect to the same explanation structure

[9] If no such t' is found, the algorithm returns failure, which means that there exists at least one linearization of $\mathcal{P}$ that will have subgoal interactions.

[10] This assumes that the transitive closure of the partial ordering relations among plan steps is available (for an n step plan, this can be computed in $O(n^3)$ time), thereby allowing the checks on ordering relations during explanation construction to be done in constant time.

[11] We shall use the superscript ''s'' to distinguish entities corresponding to the schematized plan.

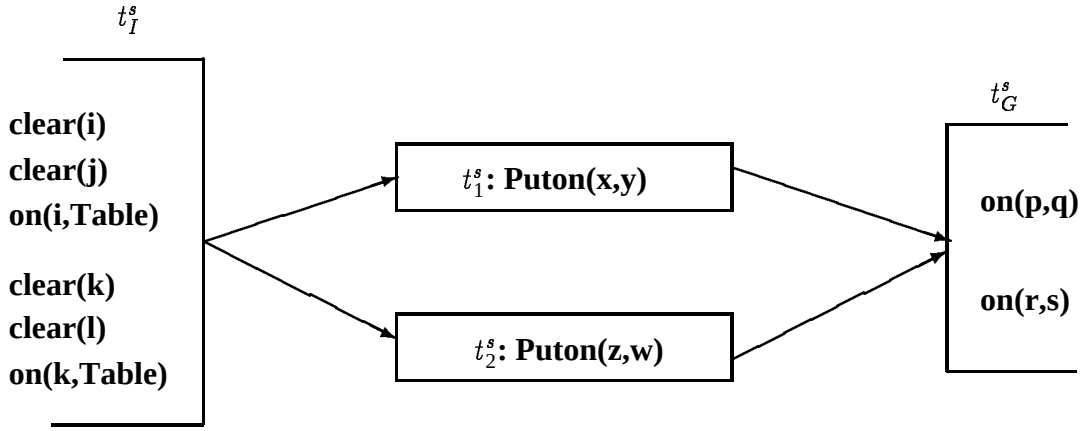


Figure 4: Schematized Plan for 4BSP

as that used to explain the correctness of $\mathcal{P}$.

Given a plan $\mathcal{P} : \langle T, O, \rangle$, we construct its *schematized version*, $\mathcal{P}^s : \langle T^s, O^s, {}^s \rangle$ by replacing each instantiated operator $t \in T$ by the corresponding operator template t^s (with unique variables). (For t_I and t_G, we replace their literals by their variablized versions.) O^s defines a partial ordering on T^s that is isomorphic to O, and ^{s} is initially set to empty set[12] . The objective of the generalization is to compute the weakest ^{s} that is enough to make $\mathcal{P}^s$ correct according to $\mathcal{V}^s$. Figure 4 shows the schematized plan corresponding to the 4BSP plan shown in Figure 1.

The schematization process defines a one-to-one mapping between the add, delete and precondition lists of each step t of $\mathcal{P}$ and those of the corresponding operator template t^s of $\mathcal{P}^s$. For example, the literal $On(A, Table)$ in the preconditions of operator t_1 in the 4BSP plan shown in Figure 1 corresponds to the literal $On(x, Table)$ in the schematized plan. Let LMAP denote this mapping. Given this mapping, a set of explanation links $\mathcal{V}^s$ for $\mathcal{P}^s$ can be constructed such that they are isomorphic to $\mathcal{V}$ of $\mathcal{P}$. For each validation $v : \langle e, t', C, t'' \rangle \in \mathcal{V}$, there will be a validation $v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \mathcal{V}^s$ such that t'^s and t''^s are operator templates in the schematized plan corresponding to t' and t'' respectively, and e^s and C^s are the literals corresponding to e and C according to LMAP defined above. For the 4BSP schematized plan shown in Figure 4, the explanation links in $\mathcal{V}^s$ are:

[12]Note that it is also possible to start with an empty O set and compute just the amount of orderings required by the schematized validation structure, see Sections 4.2 and 4.3.

$$
\begin{array}{|l|}
\hline
v_1^s : \langle On(i, Table), t_I^s, On(x, Table), t_1^s \rangle \\
v_2^s : \langle Clear(i), t_I^s, Clear(x), t_1^s \rangle \\
v_3^s : \langle Clear(j), t_I^s, Clear(y), t_1^s \rangle \\
v_4^s : \langle On(k, Table), t_I^s, On(z, Table), t_2^s \rangle \\
v_5^s : \langle Clear(k), t_I^s, Clear(z), t_2^s \rangle \\
v_6^s : \langle Clear(l), t_I^s, Clear(w), t_2^s \rangle \\
v_7^s : \langle On(x, y), t_1^s, On(p, q), t_G^s \rangle \\
v_8^s : \langle On(z, w), t_2^s, On(r, s), t_G^s \rangle \\
\hline
\end{array}
$$

Notice that after the schematization, $\mathcal{P}^s$ and $\mathcal{V}^s$ are *over general* in that the links in $\mathcal{V}^s$ may no longer constitute an explanation of correctness of $\mathcal{P}^s$ according to the MTC. The objective of the generalization phase is to post constraints (codesignation and non-codesignation) on the variable bindings to specialize this over general schematized plan and validations just enough so that $\mathcal{V}^s$ is an explanation of correctness for $\mathcal{P}^s$ according to MTC. Extracted initial conditions on $\mathcal{P}^s$ are then the weakest (most general) conditions for which $\mathcal{P}^s$ can be executed in any total order consistent with the partial order O^s, according to the same explanation structure

We now develop the declarative specification of the necessary and sufficient conditions under which $\mathcal{V}^s$ will be an explanation of correctness of $\mathcal{P}^s$ according to the MTC. We do this by expressing the conditions under which each element $v^s \in \mathcal{V}^s$ is a validation of $\mathcal{P}^s$. From the semantics of the validations provided in equation 3, these conditions can be stated as the conjunction of codesignation and non-codesignation constraints:[13]

$$
\bigwedge_{\forall v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \mathcal{V}^s} \left[\Box(e^s \approx C^s) \bigwedge \forall t^s \in T^s \, s.t. \; \begin{array}{l} \Diamond(t'^s \prec t^s \prec t''^s), \\ \forall d^s \in delete(t^s) \; \Box(d^s \not\approx C^s) \end{array} \right] \tag{4}
$$

Essentially, the validations offer an ''interval'' view on the explanation -- the intervals in which literals have to hold. For our generalization algorithm to mirror standard EBG algorithms, we regroup the validations to reflect what needs to hold for each operator (the ''operator'' view). The validations grouped for each operator $t^s \in T^s$, describe validations it is required to *support* and *preserve* in the explanation of correctness. The ''interval'' view in expression 4 can thus be re-expressed in an ''operator'' view by grouping the validations at each operator:

$$
\bigwedge_{\forall t^s \in T^s} \left[\begin{array}{l} \underbrace{\forall v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \mathcal{V}^s \; s.t. \, t'^s = t^s}_{e\text{–}conditions(t^s)}, \; \Box(C^s \approx e^s) \wedge \\ \underbrace{\forall v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \mathcal{V}^s \; s.t. \; \Diamond(t'^s \prec t^s \prec t''^s)}_{p\text{–}conditions(t^s)} \; \forall d^s \in delete(t^s) \; \Box(d^s \not\approx C^s) \end{array} \right] \tag{5}
$$

[13] Since there is a direct correspondence between $\mathcal{V}^s$ and $\mathcal{V}$, and O^s is isomorphic to O, for each $v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \mathcal{V}^s$, we already have $t'^s \prec t''^s$ (see equation 3)

Informally, expression 5 states that every operator in the schematized plan should: (i) necessarily support the conditions that its counterpart in the specific plan was required to support according to the explanation and (ii) necessarily preserve all the conditions that its counterpart in the specific plan was required to preserve. In particular, we can define the *e-conditions* (for relevant effect conditions) of an operator as the set of *validations* it is required to *support* in the explanation of correctness:

$$e\text{–}conditions(t^s) \;=\; \{v^s \mid v^s : \langle e^s, t'^s, C^s, t''^s\rangle \in \mathcal{V}^s \; s.t. \; t'^s = t^s\} \tag{6}$$

and *p-conditions* (for preservable conditions) of an operator as the set of *validations* it is required to *protect*:

$$p\text{–}conditions(t^s) \;=\; \{v^s \mid v^s : \langle e^s, t'^s, C^s, t''^s\rangle \in \mathcal{V}^s \; s.t. \; \Diamond(t'^s \prec t^s \prec t''^s)\} \tag{7}$$

Using equations 6 and 7, we can now rewrite expression 5 as:

$$\bigwedge_{\forall t^s \in T^s} \left[\begin{array}{l} \forall v^s : \langle e^s, t'^s, C^s, t''^s\rangle \in e\text{–}conditions(t^s)\,,\; \Box(C^s \approx e^s) \wedge \\ \forall v^s : \langle e^s, t'^s, C^s, t''^s\rangle \in p\text{–}conditions(t^s)\; \forall d^s \in delete(t^s)\; \Box(d^s \not\approx C^s) \end{array} \right] \tag{8}$$

Expression 8 is then the declarative specification of the necessary and sufficient conditions under which the schematized plan $\mathcal{P}^s$ is correct according to MTC, given the same explanation structure $\mathcal{V}^s$. $\mathcal{P}^s$ can be used in any initial state S that satisfies the conditions shown in expression 8. For such states, the plan is guaranteed to succeed in *all* of its total orderings. Furthermore, note that expression 8 computes exactly those conditions that are *required* (by MTC) to make $\mathcal{V}^s$ an explanation of correctness of $\mathcal{P}^s$. In this sense, the computed conditions are the weakest preconditions (modulo the given explanation) for $\mathcal{P}^s$

The algorithm EXP-PREC-GEN shown in Figure 5 implements expression 7 procedurally in a straightforward manner. The algorithm makes one pass through the plan, visiting each operator, computing the codesignation and non-codesignation constraints imposed on the generalized plan. The codesignation constraints are maintained as substitutions in β, and the non-codesignation constraints are maintained as disequality constraints on the variables in γ. At the end of the generalization phase, the substitution list β is applied to all the literals in the schematized plan $\mathcal{P}^s$ and its explanation structure $\mathcal{V}^s$. Finally, the equality and disequality constraints imposed by β and γ respectively are conjoined with the initial state specification[14] of the generalized plan to get the weakest preconditions for the generalized plan.

Complexity: The generalization algorithm runs in polynomial time. In particular, the *e-conditions* and *p-conditions* of all the operators in $\mathcal{P}^s$, as required by the algorithm, can be precomputed in $O(|T^s||\mathcal{V}^s|)$ or $O(n^2)$ time (where n is the number of operators of the plan), and the propositional

[14]the literals in the *e-conditions* of t_I, to be precise

Algorithm EXP-PREC-GEN ($\mathcal{P}^s : \langle T^s, O^s, {}^s \rangle, \mathcal{V}^s$)
Initialize: Let β be a null substitution, ^{s} a null set of constraints, and γ be *True*
foreach $t^s \in T^s$ **do**
foreach $v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in$ *e–conditions*(t^s) **do**
Let β' be the substitution under which $\Box(e^s \approx C^s)$
$\beta \leftarrow \beta \circ \beta'$
foreach $v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in$ *p–conditions*(t^s) **do**
Let γ' be the condition under which
$\forall d^s \in delete(t^s)\, \Box(d^s \not\approx C^s)$
$\gamma \leftarrow \gamma \wedge \gamma'$
${}^s \leftarrow \beta \wedge \gamma$
Substitute β into all the literals of $\mathcal{P}^s$ and $\mathcal{V}^s$
Return $\mathcal{P}^s : \langle T^s, O^s, {}^s \rangle$ as the generalized plan and $effects(t_I^s) \wedge \gamma \circ \beta$
as its generalized preconditions

Figure 5: Precondition Generalization Algorithm

unification required to compute β' and γ' itself can be done in time linear in the length of the propositions[15].

4.1.1 Example

Let us now follow the generalization of the schematized plan for 4BSP by the algorithm EXP-PREC-GEN. Following the definitions in equations 5 and 6, and the schematized validations in Section 4.1, the *e-conditions* and *p-conditions* of the operators in the schematized plan can be computed as:

[15]Note that we do not have any functions in our language, and so the length of literals is bounded by the arity of the predicates.

e-conditions(t_1^s) :	$v_7^s : \langle On(x,y), t_1^s, On(p,q), t_G^s \rangle$
p-conditions(t_1^s) :	$v_8^s : \langle On(z,w), t_2^s, On(r,s), t_G^s \rangle$
	$v_5^s : \langle Clear(k), t_I^s, Clear(z), t_2^s \rangle$
	$v_6^s : \langle Clear(l), t_I^s, Clear(w), t_2^s \rangle$
	$v_4^s : \langle On(k,Table), t_I^s, On(z,Table), t_2^s \rangle$
e-conditions(t_2^s) :	$v_8^s : \langle On(z,w), t_2^s, On(r,s), t_G^s \rangle$
p-conditions(t_2^s) :	$v_7^s : \langle On(x,y), t_1^s, On(p,q), t_G^s \rangle$
	$v_2^s : \langle Clear(i), t_I^s, Clear(x), t_1^s \rangle$
	$v_3^s : \langle Clear(j), t_I^s, Clear(y), t_1^s \rangle$
	$v_1^s : \langle On(i,Table), t_I^s, On(x,Table), t_1^s \rangle$
e-conditions(t_I^s) :	$v_5^s : \langle Clear(k), t_I^s, Clear(z), t_2^s \rangle$
	$v_6^s : \langle Clear(l), t_I^s, Clear(w), t_2^s \rangle$
	$v_4^s : \langle On(k,Table), t_I^s, On(z,Table), t_G^s \rangle$
	$v_2^s : \langle Clear(i), t_I^s, Clear(x), t_1^s \rangle$
	$v_3^s : \langle Clear(j), t_I^s, Clear(y), t_1^s \rangle$
	$v_1^s : \langle On(i,Table), t_I^s, On(x,Table), t_1^s \rangle$

Recall that *e-conditions* of an operator t describe those literals which t supports, and *p-conditions* are those literals it is required to preserve (these would include the preconditions and useful effects of other operators parallel to t; for example, among the four *p-condition* validations of t_1^s, the first one corresponds to preserving the required effect of t_2^s and the other three correspond to preserving the preconditions of t_2^s). Note that by definition, t_G^s will have no *e-conditions* or *p-conditions*, and t_I^s will only have *e-conditions* (since no plan operator can precede t_I or follow t_G).

The EXP-PREC-GEN algorithm computes β_1' for $t_1^s : Puton(x,y)$ by unifying $On(x,y)$ and $On(p,q)$. Thus at this point, β_1' (and therefore β) is $((x\,p)(y\,q))$. Next, γ_1' for $t_1^s : Puton(x,y)$ is computed by ensuring that its delete literals $on(x,Table) \wedge clear(y)$ do not unify with the literals of its *p-conditions*. Thus γ_1' can be computed as: $[neq(x,z) \vee neq(Table,w)] \wedge neq(y,z) \wedge neq(y,w) \wedge neq(x,k)$

Similar processing for $t_2^s : Puton(z,w)$ yields β_2' as $((z\,r)(w\,s))$, and γ_2' as $[neq(z,x) \vee neq(Table,y)] \wedge neq(w,x) \wedge neq(w,y) \wedge neq(z,i)$. Finally, the processing for t_I^s yields β_3' as $((i\,x)(j\,y)(k\,z)(l\,w))$ (there are no *p-conditions* for t_I^s and so γ_3' is trivially *True*).

The resultant global substitution β is thus $\beta_1' \circ \beta_2' \circ \beta_3'$, or:

$$\beta = ((i\,p)(x\,p)(j\,q)(y\,q)(k\,r)(z\,r)(l\,s)(w\,s))$$

Similarly the global non-codesignation constraints on variables γ is computed by conjoining γ_1', γ_2' and γ_3' as:

$$\gamma = [neq(x,z) \vee neq(Table,w)] \wedge neq(y,z) \wedge neq(y,w) \wedge neq(x,k) \wedge [neq(z,x) \vee neq(Table,y)] \wedge neq(w,x) \wedge neq(z,i)$$

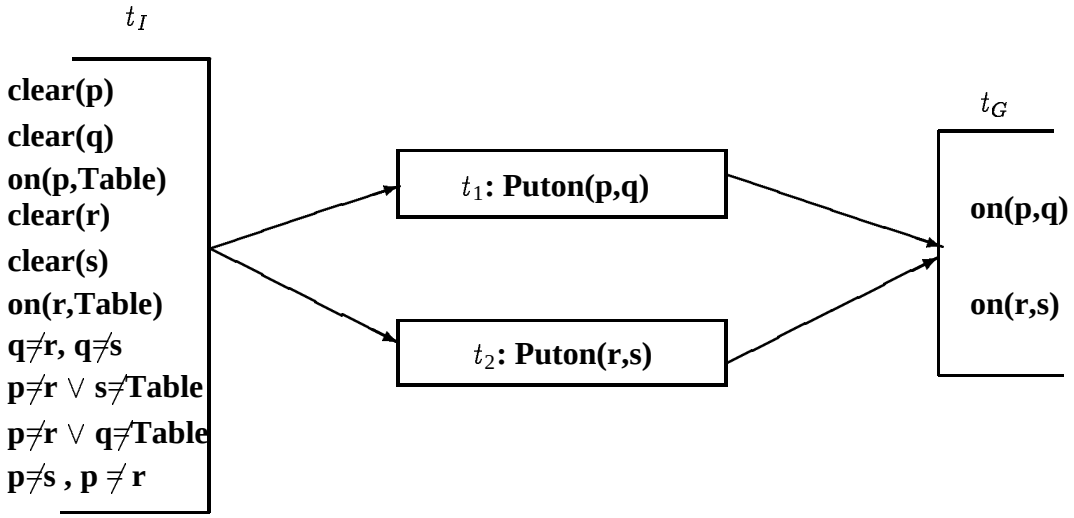


Figure 6: Generalized Plan for 4BSP

Figure 6 shows the generalized plan (computed by substituting β into the literals of schematized 4BSP plan shown in Figure 4), and its weakest preconditions, computed by conjoining $\gamma \circ \beta$ with the effects of t_I^s in the plan. In particular, by substituting β into γ and simplifying, we have:

$$\gamma \circ \beta =$$
$$[neq(p,r) \vee neq(Table,s)] \wedge neq(q,r) \wedge neq(q,s) \wedge [neq(r,p) \vee neq(Table,q)] \wedge neq(s,p) \wedge neq(p,r)$$

Notice that the weakest preconditions rightly prohibit the use of this plan in a situation where the goal is $On(A,B) \wedge On(B,C)$, because they explicitly prohibit codesignation of q and r, and p and s (see $\gamma \circ \beta$). Thus, the algorithm avoids the overgeneralization discussed in Section 1.

4.2 Explanation-based Order Generalization

Given a plan $\mathcal{P} : \langle T, O, \rangle$, the precondition generalization algorithm developed in the previous section uses the validation structure based explanation of correctness of the plan to compute weakest constraints on the variable bindings, , that are required to ensure the correctness of the plan. We can easily extend this methodology to also generalize the ordering relations O on the plan. Just as we can eliminate any bindings that are not required to make the plan correct according to the given validation structure, we can also eliminate any orderings that are redundant. In particular, if $\mathcal{V}$ is the validation structure of the plan $\mathcal{P}$ (see Section 3), then we can justify the ordering relations among the steps of the plan $\mathcal{P}$ with respect to the validation structure. The key insight is that the only ordering constraints that are necessary to ensure correctness of the plan are those that are required to maintain the individual validations (as given by equation 3).

Algorithm EXP-ORDER-GEN ($\mathcal{P} : \langle T, O, \rangle, \mathcal{V}$)
Construct transitive closure of the partial ordering O relations among the steps T of the plan.
Let O^* be the resultant ordering
Initialize: Let O' be the generalized ordering on P. Initialize $O' \leftarrow \emptyset$.
Foreach $o : (t_i , t_j) \in O^*$
If $Justified(o, \mathcal{P}, \mathcal{V})$ (*see text*)
$O' \leftarrow O' + o$
$O'' \leftarrow transitive - closure(O')$
Return($\mathcal{P}' : \langle T, O'', \rangle$)

Figure 7: Order Generalization Algorithm

Let $o : t_i \prec t_j$ be an ordering relation on the plan $\mathcal{P}$. Then, from the semantics of the validation given in equation 3, it can easily be shown that *o* is **justified** with respect to the validation structure $\mathcal{V}$ (or $justified(o : (t_i \prec t_j), \mathcal{P}, \mathcal{V})$) if and only if at least one of the following cases hold:

case 0. Either $t_i = t_I$ or $t_j = t_G$ (all plan steps follow the the start node and precede the end node).

case 1. $\exists v : \langle e, t_i, C, t_j \rangle \in \mathcal{V}$

case 2. $\exists v : \langle e, t', C, t_i \rangle \in \mathcal{V}$ and t_j has a delete list literal d such that $d \approx C$

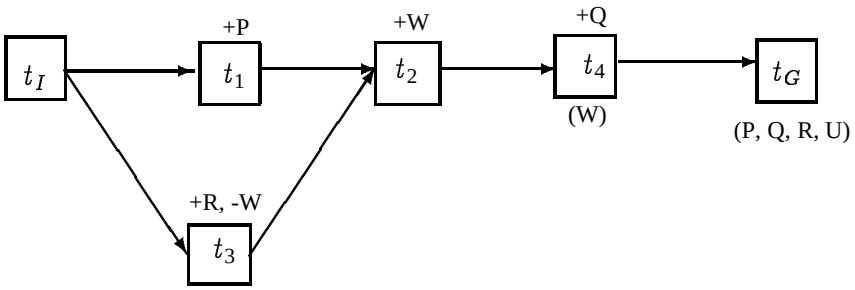
case 3. $\exists v : \langle e, t_j, C, t' \rangle \in \mathcal{V}$ and t_i has a delete list literal d such that $d \approx C$.

Additionally, any ordering relation that belongs to the transitive closure of all the justified ordering relations is also justified.

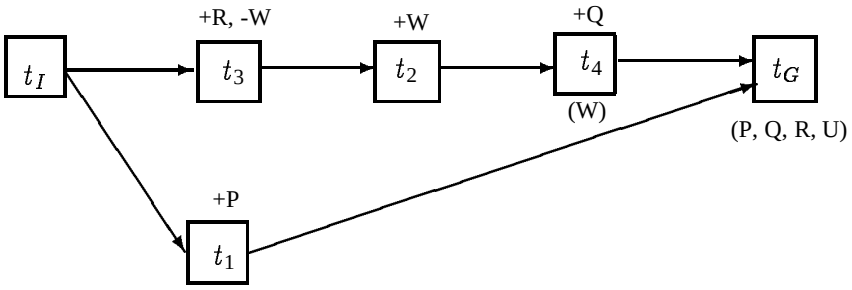
Whenever an ordering relation *o* is not either directly justified or transitively justified, then *o* can be safely removed from $\mathcal{P}$ without affecting the correctness of the plan according the explanation $\mathcal{V}$. Figure 7 provides a straightforward algorithm for doing this type of order generalization. Since there are $O(n^2)$ ordering relations and $O(n)$ validation links in n-step plan, it is easy to see that the order generalization algorithm can be run to completion in $O(n^3)$ time.

Example: Consider the partially ordered propositional plan shown at the top of Figure 4.2. The add and delete list literals (propositions) of each operator are shown above the operator with ''+'' and ''$-$'' signs, while the preconditions are shown under the operator within parentheses. If this plan is given as input to the explanation construction algorithm in Figure 3, it will return the following four validations as the explanation of correctness of the plan:

v_1 : $\langle P, t_1, P, t_G \rangle$
v_2 : $\langle W, t_2, W, t_4 \rangle$
v_3 : $\langle Q, t_4, Q, t_G \rangle$
v_4 : $\langle R, t_3, R, t_G \rangle$



A propositional plan with redundant orderings



The plan after order generalization

Figure 8: An example of order generalization

Using these validations, the algorithm in Figure 7 can generalize the orderings on the plan as follows. Note that the set of initial orderings are:

$$\left\{ \begin{array}{c} t_I \prec t_1,\ t_I \prec t_2,\ t_I \prec t_3,\ t_I \prec t_G, \\ t_1 \prec t_G,\ t_2 \prec t_G,\ t_3 \prec t_4,\ t_4 \prec t_G \\ t_3 \prec t_2, \\ t_3 \prec t_4,\ t_1 \prec t_2,\ t_1 \prec t_4 \end{array} \right\}$$

Of these, the first four are directly justified based on case 0 of the definition, and the next four are directly justified based on case 1 of the definition.[16] The next one, $t_3 \prec t_4$, is directly justified by case 3 of the definition (since $v_2 : \langle W, t_2, W, t_4 \rangle$ is a validation of the plan, and W is a delete list literal of t_3). These are the only orderings that are directly justified. From these, we can see that the ordering $t_3 \prec t_4$ is also transitively justified (since $t_3 \prec t_2$ and $t_2 \prec t_4$ are both justified).

[16] the first two and the last are also justified according to case 0

Algorithm ORD-PREC-GEN ($\mathcal{P} : \langle T, O, \rangle$)
$\mathcal{V} \leftarrow$ EXP-MTC($\mathcal{P} : \langle T, O, \rangle$)
$\mathcal{P}' : \langle T, O', \rangle \leftarrow$ EXP-ORD-GEN($\mathcal{P} : \langle T, O, \rangle, \mathcal{V}$)
$\mathcal{P}'' : \langle T, O', '' \rangle \leftarrow$ EXP-PREC-GEN($\mathcal{P}' : \langle T, O', \rangle, \mathcal{V}$)
Return($\mathcal{P}''$)

Figure 9: An algorithm for generalizing the order and preconditions of a POPI plan

The ordering relations $t_1 \prec t_2$ and $t_1 \prec t_4$ are however are neither directly nor transitively justified according to the set of validations $\{v_1 \cdots v_4\}$. Thus they will be removed from the order generalized plan, giving rise to a more general partially ordered plan shown in the bottom of Figure 4.2. (The latter plan is more general as it has more completions compared to the former; see Section 2).

Note that the order generalization algorithm developed here can also be applied to the plans generated by a totally ordered planner, to remove unnecessary orderings from them and make them more ''parallel.'' In this sense, it is equivalent to the plan order optimization algorithms described in [36] and [9].

4.3 Combining order generalization and precondition generalization

Given the precondition generalization and order generalization algorithms developed above, a natural possibility would be to combine them. Such a combined generalization algorithm can take a totally ordered totally instantiated plan, such as one generated by STRIPS [10] or PRODIGY [23] and generalize it by first removing unnecessary orderings and then generalizing the variable bindings. The algorithm ORD-PREC-GEN in Figure 9 shows a way of doing this. We start by running the explanation construction algorithm (shown in Figure 3) on the plan. Using the validation structure produced by the explanation algorithm, we can then generalize the ordering on the plan (with the help of the algorithm shown in Figure 7). Finally, the order generalized plan and the validation structure can be given as the input to the precondition generalization algorithm (shown in Figure 5).

Example: Suppose we are given the plan $\mathcal{P}_{4BS} : Puton(A, B) \rightarrow Puton(C, D)$ for solving the 4BS problem shown in Figure 1. For this plan, he explanation construction algorithm will give the same eight validations shown in Section 3. Using these validations, the algorithm EXP-ORDER-GEN in Figure 7 will remove the ordering relation between Puton(A,B) and Puton(C,D), from the plan $\mathcal{P}_{4BS}$, thus returning the plan shown in Figure 1. This new plan, along with the validation structure is then provided to the EXP-PREC-GEN algorithm in Figure 5, giving rise to the generalized plan in Figure 6.

4.3.1 Disjunctive generalization

Given a plan $\mathcal{P} : \langle T, O, \rangle$ and a validation structure $\mathcal{V}$, there may typically be more than one way of specializing (adding constraints to) and O to make $\mathcal{V}$ a validation structure of $\mathcal{P}$. In particular, there are essentially two types of ordering and binding constraints that $\mathcal{P}$ has to satisfy so as to maintain $\mathcal{V}^s$ as its validation structure [21]:

Causal constraints: Foreach validation $v : \langle E, t_s, C, t_d \rangle \in \mathcal{V}$, $(t_s \prec t_d) \in O$, and $(E \approx C) \in$.

Safety constraints: Foreach validation $v : \langle E, t_s, C, t_d \rangle \in \mathcal{V}$, foreach node $t' \in T$ such that t' has a delete list literal d that can possibly codesignate with C, either of the following constraints must be satisfied:

- $(t_d \prec t') \in O$ or $(t' \prec t) \in O$ **or**
- $d \not\approx C \in$

Note that although the causal constraints are fixed once a validation structure has been chosen, the safety constraints can be satisfied in more than one way -- either by adding additional ordering constraints or by adding additional codesignation/non-codesignation constraints.

The algorithm ORD-PREC-GEN selects one point along this spectrum -- it uses the strategy of first pegging to one value, and generalizing O with respect to it, and then pegging O to the generalized value and generalizing .

It is also possible to compute all possible generalizations (i.e., all possible ^{s} and O^s combinations) and conjoin them, resulting in a disjunctive generalization of the original plan. This is what is done in Mooney's order generalization algorithm [27]. Computing all the consistent $- O$ combinations for a given plan $\mathcal{P}$ and validation structure $\mathcal{V}$ however involves reinterpreting the truth criterion, thus transferring some of the planning activity into the generalization phase; Mooney's algorithm takes $O(n^5)$ in the length of the plan [27].

Figure 4.3.1 shows the disjunctive generalization of the 4BS plan. Note in particular, that apart from the fully parallel version, the generalized plan also includes one set of orderings to accommodate the scenario where $q \approx r$ and another set of ordering to accommodate the scenario where $p \approx s$.

5 Extending the Generalization Framework

5.1 Computing conditions of possible correctness

Until now, we have looked at generalization algorithms that compute weakest conditions under which *all* completions of a POPI plan can be guaranteed to execute successfully. Failure of these conditions means only that there exists at least one completion that is incorrect. It *does not* preclude the possibility that the plan can be specialized further (by adding further constraints on the

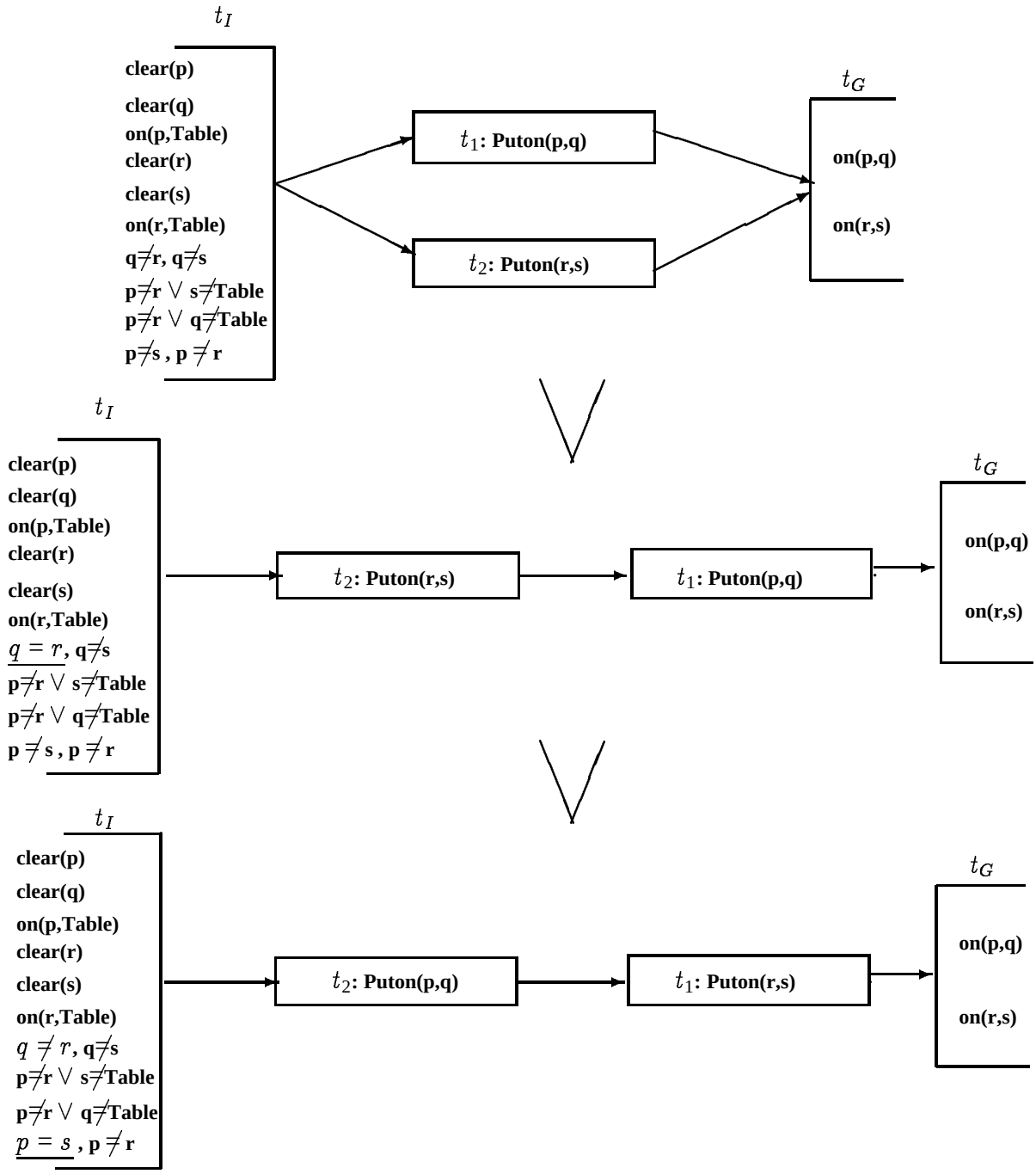


Figure 10: Disjunctive generalization of 4BS problem

orderings and variable bindings). Sometimes, it may be useful to compute weakest preconditions under which at least some completion of the plan can possibly execute. Such preconditions can be used to select a generalized plan that is possibly applicable in a given situation, and make it necessarily applicable by allowing the planner to add only additional ordering and binding constraints *without ever retracting the existing constraints in the plan.* Such a mechanism would, for example, allow us to store the generalized plan for the four-block stacking problem and use it both in a four block stacking scenario and in a three-block stacking scenario (such as the one discussed in Section 1). Since such a mechanism allows a stored plan to be used in a much larger set of situations, it provides an interesting memory vs. planning trade-off.

Consider the schematized plan $\mathcal{P}^s : \langle T^s, O^s, {}^s \rangle$ and its schematized validation structure $\mathcal{V}^s$ described in Section 4.1. Let $\mathcal{P}^s_c$ be a completion of $\mathcal{P}^s$ (see Section 2). We say that a validation $v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \mathcal{V}^s$ holds in $\mathcal{P}^s_c$ ($comp\text{–}holds(v^s, \mathcal{P}^s_c)$) if and only if

1. t'^s comes before t''^s in $\mathcal{P}^s_c$
2. $E^s = C^s$ (note that in a completion corresponds to a totally instantiated plan, all the effects and preconditions will be ground)
3. no step comes between t'^s and t''^s and deletes C^s in $\mathcal{P}^s_c$.

$\mathcal{P}^s$ is possibly correct (under the explanation $\mathcal{V}^s$) if and only if there exists at least one completion of it such that all the validations $v^s \in \mathcal{V}^s$ simultaneously hold in that completion:

$$possibly\text{–}correct(\mathcal{P}^s, \mathcal{V}^s) \equiv \exists \mathcal{P}^s_c \in Completions(\mathcal{P}) \bigwedge_{v^s \in \mathcal{V}^s} comp\text{–}holds(v^s, \mathcal{P}^s_c)$$

Computing conditions of possible correctness as described in the above equation will in general force us to reason with the individual completions, and can be quite costly.[17] Even more importantly, the resulting preconditions would be highly disjunctive and thus the match cost might out-weigh its expected savings [23]. One compromise would be to look for necessary *or* sufficient conditions for possible correctness which will be easier to compute and match against. Since $\Box P \supset \Diamond P$, the preconditions for necessary correctness computed by the algorithm in Figure 5 already provide a set of sufficient conditions for possible correctness.

[17]The relative difficulty of dealing with possible correctness, as compared to necessary correctness (Section 4.1) is due to the asymmetry in the way possible and necessary truths distribute over conjunctions. To compute necessary (possible) correctness of a plan, we need to check that in every (at least one) completion of the plan, all the prerequisites of the plan are true. Since $\Box(P \wedge Q) \equiv \Box P \wedge \Box Q$, necessary correctness can be determined in polynomial time by locally determining the necessary truth of individual propositions using MTC and conjoining them. However such a method cannot be used for possible correctness since $\Diamond(P \wedge Q) \not\equiv \Diamond P \wedge \Diamond Q$. Thus, although local determination of possible truth of individual propositions can be done polynomial time, there is no straightforward way in which they can be *combined* to determine the possible truth of the plan in polynomial time.

A way of computing necessary conditions for possible correctness, without having to reason with individual completions, is to use the modal truth criterion for *possible truth* of a proposition in a POPI plan (see below). The basic idea is to use the possible truth criterion to explain the possible correctness of each precondition and goal of the plan individually, and base the generalization on this explanation. Since $\Diamond P \wedge \Diamond Q$ does not imply $\Diamond(P \wedge Q)$, the conditions that ensure that all preconditions individually possibly hold may not in general guarantee that there exists a completion in which they all hold simultaneously. (In other words, even if its preconditions for possible correctness, as described above, hold in that situation. there may possibly not be any completion of the plan that will work in a given situation) They do however constitute a set of *necessary* conditions for ensuring possible correctness of the plan --- if the generalized preconditions computed by possible correctness *do not* hold in a problem situation, then we know for sure that no specialization of that plan will ever solve that problem. When these conditions do hold, we can use the global constraint satisfaction techniques, such as those described in [39] to check if the plan can be constrained to be applicable to the new situation. The following briefly describes the development of an algorithm for computing such necessary preconditions of possible correctness.

We will start by observing that the truth criterion for guaranteeing the possible truth of a proposition can be obtained by simply reversing the modalities in the equation 2 (substitute ''$\Diamond$'' for ''$\Box$'' and vice versa) [3] as follows:

$$
\begin{aligned}
&possibly\text{–}holds(C, t, \mathcal{P}) \iff \\
&\quad \exists\, t' \text{ s.t. } \Diamond(t' \prec t) \wedge e \in effects(t') \wedge \Diamond(e \approx C) \bigwedge \\
&\quad \forall t'' \text{ s.t. } \Box(t' \prec t'' \prec t) \\
&\qquad \forall d \in delete(t'') \Diamond(d \not\approx C)
\end{aligned}
$$

Thus a condition C is possibly correct at step t as long as there exists some step t' that is possibly preceding t which has an effect that can possibly codesignate with C, and for every step t'' that necessarily comes between t' and t, the effects of t'' will possibly not clobber the condition C. If an individual condition C is possibly correct, then we can make it necessarily correct by adding some additional ordering and binding constraints on the plan, without ever having to retract any constraints from the plan (i.e., without backtracking).

Using this truth criterion, we can develop polynomial time EBG algorithms for necessary conditions for possible correctness (isomorphic to those developed for necessary correctness). In particular, we can represent an explanation that each precondition of the plan is possibly true, in the form of validation links that are similar to those described in Section 2. In keeping with the possible truth criterion, we associate the following weaker semantics to the validation links:

$$
\begin{aligned}
&v : \langle e, t', C, t \rangle \text{ is a validation of } \mathcal{P} : \langle T, O, \rangle \iff \\
&\qquad \Diamond(e \approx C) \wedge \Diamond(t' \prec t) \wedge \\
&\qquad \forall t'' \text{ s.t. } \Box(t' \prec t'' \prec t) \wedge \exists d \in delete(t'') \; s.t. \Diamond(d \not\approx C),
\end{aligned}
$$

Algorithm EXP-PREC-GEN ($\mathcal{P}^s : \langle T^s, O^s, {}^s \rangle, \mathcal{V}^s$)
  Initialize: Let $\beta, \gamma \leftarrow True$
  foreach $t^s \in T^s$ **do**
    foreach $v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in$ *e–conditions'*(t^s) **do**
      Let β' be the conditions under which $\Diamond(e^s \approx C^s)$
      $\beta \leftarrow \beta \wedge \beta'$
    foreach $v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in$ *p–conditions'*(t^s) **do**
      Let γ' be the condition under which
        $\forall d^s \in delete(t^s) \Diamond(d^s \not\approx C^s)$
      $\gamma \leftarrow \gamma \wedge \gamma'$
  Return $\leftarrow effects(t_I^s) \wedge \gamma \wedge \beta$ as the necessary conditions for possible correctness

Figure 11: Algorithm for computing a set of Necessary Conditions for Possible Correctness of a Generalized Plan

Given this, the conditions under which all the validations individually possibly hold are given by

$$\bigwedge_{\forall v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \mathcal{V}^s} \left[\Diamond(e^s \approx C^s) \bigwedge \forall t^s \in T^s \, s.t. \; \begin{array}{l} \Box(t'^s \prec t^s \prec t''^s), \\ \forall d^s \in delete(t^s) \, \Diamond(d^s \not\approx C^s) \end{array} \right]$$

Based on this we can also define the notion of *e–conditions'* and *p–conditions'* for each step in the plan, and develop the declarative specification of the generalization algorithm in terms of those constraints as follows:

$$\textit{e–conditions'}(t^s) = \{v^s \mid v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \mathcal{V}^s \; s.t. \; t'^s = t^s\}$$
$$\textit{p–conditions'}(t^s) = \{v^s \mid v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \mathcal{V}^s \; s.t. \; \Box(t'^s \prec t^s \prec t''^s)\}$$

$$\bigwedge_{\forall t^s \in T^s} \left[\begin{array}{l} \forall v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \textit{e–conditions'}(t^s), \Diamond(C^s \approx e^s) \wedge \\ \forall v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in \textit{p–conditions'}(t^s) \; \forall d^s \in delete(t^s) \; \Diamond(d^s \not\approx C^s) \end{array} \right]$$

Here $\mathcal{V}$ is the set of validation links that constitute the explanation of possible correctness (with the semantics described above). With this alternative definition, we can now use a simple variant of the precondition generalization algorithm described in Figure 5, shown in Figure 11 to compute necessary preconditions for possible correctness. The conditions β and γ returned by the procedure above will be of the form $peq(x, y)$ and $pneq(x, y)$, defined as follows:

$$peq(x, y) \triangleq (x \not\approx y) \notin$$

$$pneq(x, y) \triangleq (x \approx y) \notin$$

These conditions are necessary but not sufficient for ensuring possible correctness. Failure of these conditions in a new problem situation is sufficient to guarantee that the schematized plan cannot be specialized to solve that problem. When they hold, then it *may* be possible (but not necessarily so) to specialize the plan.

Using this algorithm, we can easily compute the conditions under which the schematized 4BS plan (described in Figure 4) will be possibly correct. To begin with, note that the e–*conditions*$'$ on t_1^s, t_2^s and t_I^s are the same as the *e–conditions* on those steps (as shown in Section 4.1.1), but there are no p–*conditions*$'$ on either of these steps (this is because there is no validation $v^s : \langle e^s, t'^s, C^s, t''^s \rangle$ in the validation structure of the schematized 4BS plan such that $\Box(t'^s \prec t_1^s \prec t''^s)$ or $\Box(t'^s \prec t_2^s \prec t''^s)$). With this information, the algorithm in Figure 11 will compute the following necessary conditions for ensuring the possible correctness of the schematized plan:

$$peq(x, p) \wedge peq(y, q) \wedge peq(z, w) \wedge peq(r, s) \wedge peq(k, z) \wedge peq(l, w) \wedge peq(i, x) \wedge peq(j, y)$$

Suppose we want to check if the schematized 4BS plan can be used to solve a three block stacking problem where $i \approx A$, $j \approx B$, $k \approx B$ and $l \approx C$. We can see that the conditions above are not violated in the three blocks situation. Thus, it may be possible to specialize the 4BS plan to solve the three block stacking problem. In this case, we can see that the plan can in fact be specialized to solve the three block stacking problem by adding an additional ordering constraint $t_2^s \prec t_1^s$ to the plan. In [39], Yang provides algorithms based on global constrain satisfaction techniques to efficiently determine whether a given possibly applicable plan can be made necessarily applicable by adding additional ordering and binding constraints.

5.2 Utilizing more general truth criteria

As we pointed out earlier, for exposition purposes, we used a version of Chapman's truth criterion [3] without the *white-knight* clause that would permit deleters to come in between the producer and consumer of a condition, as long as each such deleter is necessarily followed by a so called white-knight step which necessarily precedes the consumer and adds the condition being deleted. Although planners using this truth criterion will be *complete*, in that they will be able to produce correct (POPI) plans for any solvable planning problem (c.f. [21]), the truth criterion itself describes only *sufficient*, but not *necessary* conditions for ensuring the truth of a proposition in a POPI plan. As a result, the generalization algorithm in Figure 5 is *sound*, but not *complete* for POPI plans involving the class of operators described in Section 2. That is, there may be plans, such as the one shown in Figure 12, which are correct, but our generalization algorithm cannot handle them, since they do not have a validation structure with semantics defined by equation 3.

This incompleteness can be avoided by using Chapman's [3] TWEAK truth criterion, which provides both necessary and sufficient (for domains where the operator add/delete/precondition lists have unquantified literals). This criterion can essentially be stated as follows:

$$
\begin{aligned}
&holds(C, t, \mathcal{P}) \iff \\
&\quad \exists\, t' \text{ s.t. } \Box(t' \prec t) \wedge e \in effects(t') \wedge \Box(e \approx C) \bigwedge \\
&\quad \forall t'' \text{ s.t. } \Diamond(t' \prec t'' \prec t) \wedge \exists d \in delete(t'') \; s.t. \; \Diamond(d \approx C), \\
&\qquad \exists w \in \mathcal{P} \; s.t. \Box(t'' \prec w \prec t) \wedge \exists e \in effects(w) \; s.t. \; \Box[(d \approx C) \Rightarrow (e \approx C)]
\end{aligned}
$$

It states that a proposition C holds before an operator t in a POPI plan $\mathcal{P} : \langle T, O, \rangle$ if and only if there exists an operator t' such that an effect e of t' necessarily codesignates with C, and for every operator t'' of the plan that may possibly fall between t' and t, such that it can possibly delete C, there exists another operator w that necessarily follows t' and precedes t and establishes E in each case when t' clobbers C (such a step w is called the *white-knight* step). Obviously, this truth criterion is more general in that it does not summarily prohibit a clobberer possibly coming after the establishing node.

The generalization algorithms of the previous section can be extended in a straightforward fashion to cover this more general truth criterion. In particular, the explanation of correctness of a plan with respect to this truth criterion can be represented in the form of validation links that are similar to those described in Section 3, but have the following more general semantics:

$$
\begin{aligned}
&v : \langle e, t', C, t \rangle \text{ is a validation of } \mathcal{P} : \langle T, O, \rangle \iff \\
&\qquad \Box(e \approx C) \wedge \Box(t' \prec t) \wedge \\
&\qquad \forall t'' \text{ s.t. } \Diamond(t' \prec t'' \prec t) \wedge \exists d \in delete(t'') \; s.t. \; \Diamond(d \approx C), \\
&\qquad\quad \exists w \in \mathcal{P} \; s.t. \Box(t'' \prec w \prec t) \wedge \exists e \in effects(w) \; s.t. \; \Box[(d \approx C) \Rightarrow (e \approx C)]
\end{aligned}
$$

This definition of the validations can be used to develop a generalization algorithm similar to the one shown in Figure 5. In particular, we can use a variant of the algorithm in Figure 3 to construct the validation structure that satisfies the semantics of equation 5.2. Constructing this explanation structure will be costlier --- ($O(n^4)$ for an n-operator plan, instead of $O(n^3)$ for the algorithm in Figure 3. Once such a validation structure is found, a variant of the algorithm in Figure 5, as shown in Figure 13, can be used to compute the generalized preconditions. This algorithm will still be of polynomial time complexity, but will be both sound and complete for any plans representable in the TWEAK operator language (defined in Section 2).

For those plans which can be handled by both EXP-ALG and EXP-ALG-2, the latter returns a slightly more general (disjunctive) preconditions (set of codesignation/non-codesignation constraints among variables). Just as we can use the algorithm in Figure 5 to generalize plans generated by a total ordering planner, we can use the algorithm in Figure 13 to generalize plans generated by a planner using a stronger truth criterions such as the one in equation 2.

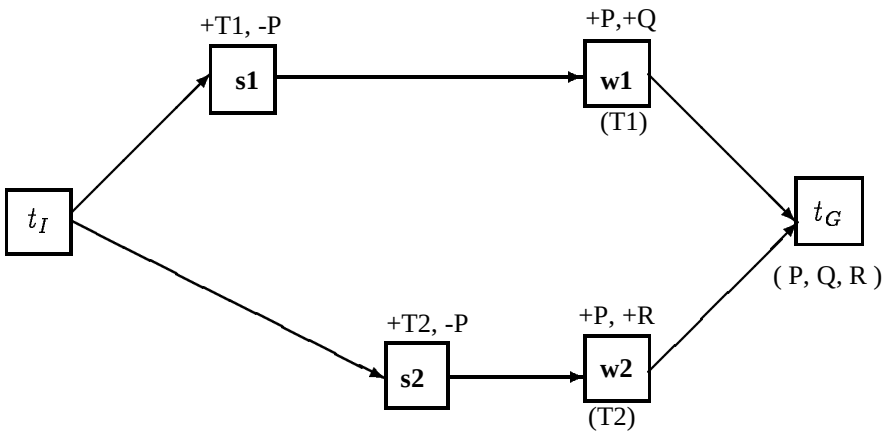


Figure 12: A plan that cannot be handled by the algorithm EXP-PREC-GEN. (the add and delete lists literals of each step are shown above the step with + and − signs, while the prerequisites of the step are in parentheses under the step)

Algorithm EXP-PREC-GEN-2 ($\mathcal{P}^s : \langle T^s, O^s, {}^s \rangle, \mathcal{V}^s$)
  Initialize: Let β be a null substitution and γ be *True*
  foreach $t^s \in T^s$ **do**
    foreach $v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in$ *e–conditions*(t^s) **do**
      Let β' be the substitution under which $\Box(e^s \approx C^s)$
      $\beta \leftarrow \beta \circ \beta'$
    foreach $v^s : \langle e^s, t'^s, C^s, t''^s \rangle \in$ *p–conditions*(t^s) **do**
      foreach $d^s \in delete(t^s)$ **do**
        Let γ' be the conditions under which $\Box(d^s \not\approx C^s)$
        Let β' be *False*
        Foreach $t^s_w \in T^s$ such that $\Box(t^s \prec t^s_w \prec t''^s)$
          Let β'' bethe condition under which $\exists e^s_w \in effects(t^s_w)$ such that $\Box(e^s_w \approx C^s)$
          $\beta' \leftarrow \beta' \vee \beta''$
      $\gamma \leftarrow \gamma \wedge [\gamma' \vee \beta']$
  ${}^s \leftarrow \beta \wedge \gamma$
  Substitute β into all the literals of $\mathcal{P}^s$ and $\mathcal{V}^s$
  Weakest preconditions $\leftarrow effects(t^s_I) \wedge \gamma \circ \beta$

Figure 13: Precondition Generalization Algorithm 2

5.3 Generalization based on Multiple Explanations and ''Weakest Preconditions''

It is instructive to note that all of the algorithms developed in this paper compute generalizations with respect to a particular chosen validation (explanation) structure based on a particular modal truth criterion. Unless the plan has only a single validation structure[18], these conditions will not in general be the weakest such conditions for guaranteeing correct execution of the plan with respect to the modal truth criterion being used.

To compute weakest preconditions with respect to a truth criterion, we have to compute all possible explanation structures of the plan with respect to that truth criterion, and perform generalizations with respect to each of them. (If the truth criterion under consideration is both necessary and sufficient for the class of plans under consideration, then this computation will give the absolute weakest preconditions for the success of the plan). Not surprisingly, finding weakest preconditions is in general costlier than finding an explanation based generalization of the plan. In particular, we note that in an n-action plan, there are at most $O(\xi n^2)$ different validations, where ξ is the upper bound on the number of preconditions per step.[19] Since finding a validation link with respect to TWEAK truth criterion takes $O(n^3)$ (see Section 5.2), the total cost of finding all possible validations is $O(\xi n^5)$. Since computing the codesignation and non-codesignation constraints imposed by a schematized validation can be done in $O(n)$ time (see Section 4.1), the cost of generalization is $O(\xi n^3)$.

By relaxing the notion of ''following the example'' (Section 4.1), we can also systematically develop a spectrum of structural generalizations. Such generalizations involve allowing new planning steps during the generalization phase[20]. This essentially involves using the truth criterion as a way of establishing the truth of a proposition (synthesis) [3] rather than merely as a way of testing for correctness and thus allow additional planning during generalization phase. At one extreme, we can use the modal truth criterion to find out *all possible* plans that will guarantee the goals under consideration. This corresponds to full goal regression based on the truth criterion, and will have the same complexity as generative planning (which, for tweak truth criterion, is NP-hard [3]).

[18]Correct plans with at most one single validation structure are called plans with ''exhaustive validation structures'' (see [15]). Not every plan can have an exhaustive validation structure. It is however possible to take a plan without an exhaustive validation structure and impose additional constraints on the plan so as to produce a plan that will have an exhaustive validation structure.

[19]To visualize this worst case situation, consider an n action, n goal totally ordered plan, with each step of the plan providing one of the goals. Suppose that each step has p preconditions, and they can be supplied by *any* of the steps that precede it. Thus, for the ith step, each of its preconditions can be supported by $(i-1)$ different validations, giving rise to $\xi(i-1)$ validations. Summing this over all the steps, we get $O(\xi n^2)$ different validations.

[20]The structural generalizations discussed in [27] can be seen in this light.

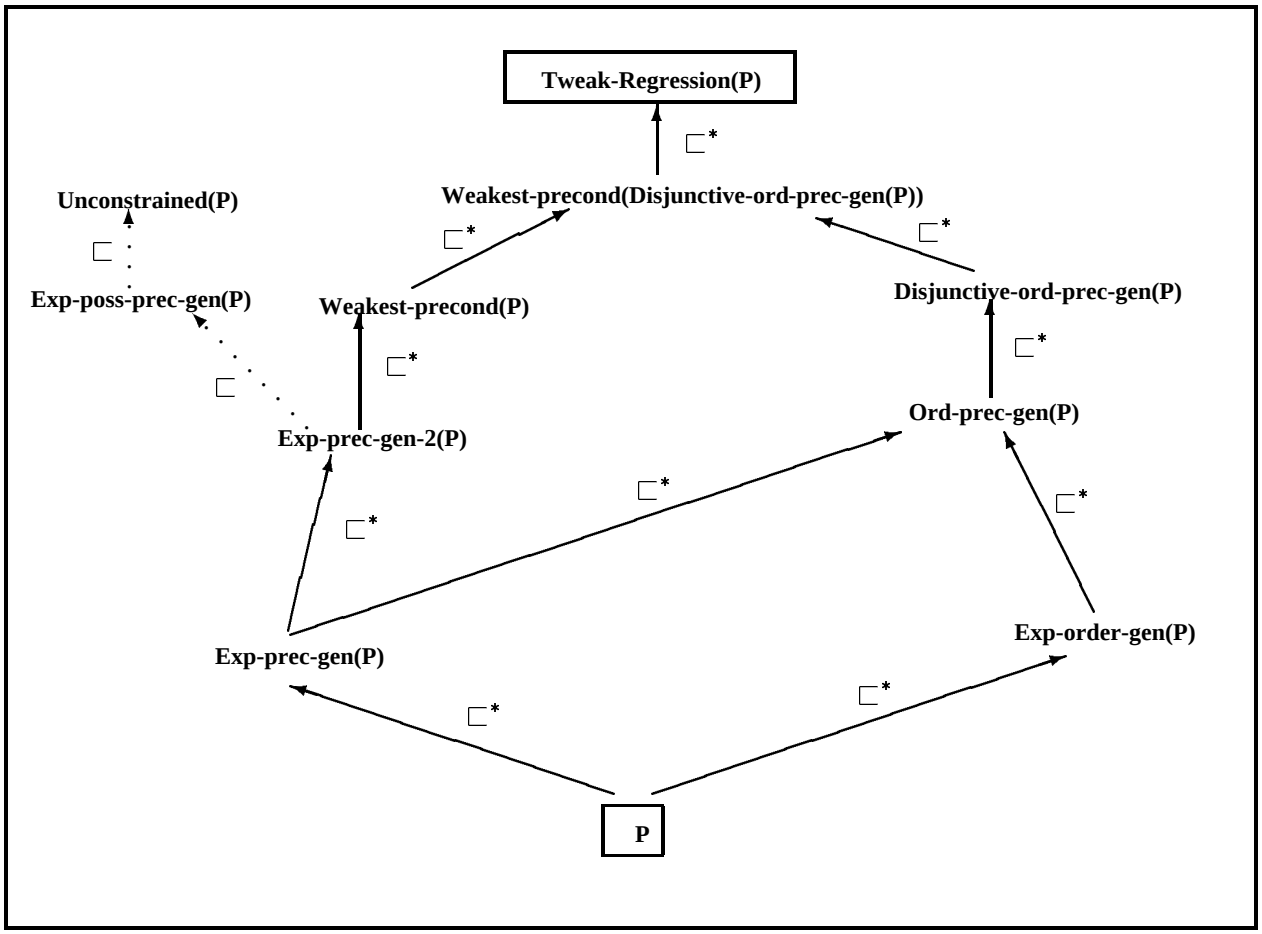


Figure 14: A lattice showing the spectrum of generalizations of POPI plans

6 Tradeoffs between the spectrum of generalizations

We have seen that there are a variety of factors influencing the generalizations of a POPI plan. These include the generality of the truth criterion used as the basis for generalization; the particular constraints on the POPI plan that is being generalized (viz., codesignation constraints, ordering constraints, steps); whether or not the generalization is correctness preserving; and finally whether the generalization is based on a single explanation or multiple explanations.

The various generalization algorithms developed in this paper all fall within a lattice of generalizations delimited by $\mathcal{P}$ as the least general in the lattice and the full goal regression of $\mathcal{P}$ as the most general plan (see Section 5.3). Figure 14 depicts this generalization lattice. In addition to the lattice of correctness preserving generalizations, this figure also shows the of non-correctness preserving generalizations (such as the possible correctness generalization discussed in Section

Type of Generalization	Cost	Usage
Precondition Generalization	$O(n^3)$	Nonlinear macrops/adaptation
Order Generalization	$O(n^3)$	-same-
Precondition generalization with TWEAK MTC	$O(n^4)$	-same-
Disjunctive order generalization [27]	$O(n^5)$	-same-
Precondition Generalization via multiple explanations (Weakest preconditions w.r.t. Tweak MTC)	$O(n^5)$	-same-
Full goal regression based on Tweak MTC [3]	NP-hard	-same-
Possible correctness generalization	NP-hard	Only for adaptation
Necessary cond. for possible correctness	$O(n^3)$	Only for adaptation
Unconstrained generalization	$O(1)$	Only for adaptation

Figure 15: Comparison between generalization algorithms

5.1, and the unconstrained generalization discussed in Section 2). Figure 15 summarizes the comparative costs of producing each of these generalizations.

An important issue brought up by this spectrum is the cost of using each type of generalization during planning. The correctness preserving generalizations of POPI plans developed in this paper can be used during planning either as nonlinear macro-operators [10] or as generalized plans to be reused [18]. In the former case, the stored plans can be instantiated even in cases where only some parts of the plan are applicable. In particular, the validation structure can be used to *edit* the macro operator (c.f. [10]) and find sub-parts of it that are applicable in the current situation [18]. The cost of instantiating a macro-operator depends on the number of preconditions that need to be checked. It is thus typically costlier to instantiate generalizations with disjunctive sets of preconditions (such as the ones described in Sections 4.3.1 and 5.2).

In addition to being used as macro operators, the stored plans can also be instantiated in new situations through plan refinement and modification [18]. This latter process involves using the modal truth criterion to establish additional propositions within the context of the retrieved plan. This option is particularly relevant in the case of non-correctness preserving generalizations, such as the possible correctness generalization, since these generalizations cannot be instantiated as macro operators. The planner will have to specialize them by adding additional ordering and binding constraints.

From the preceding discussion, we see that precondition generalization, order generalization and structure generalization [27] algorithms can be seen as different points on a spectrum of generalizations. They offer varying tradeoffs on plan-time vs. generalization-time analysis, which can be summarized as follows:

- The different truth criteria can be seen as providing differing biases for EBG. In particular, the more general a truth criterion, the more general the generalizations based on it.

- The more general a generalization, the costlier it is to compute.
- The more general a generalization, the costlier it is to instantiate during planning.
- Correctness preserving generalizations are cheaper to instantiate (use) during planning, than non-correctness preserving generalizations. This is because after checking the preconditions of the generalized plan, the planner has to do additional work to see if any completion of the generalized plan can be applied in the given problem situation.
- Non-correctness preserving generalizations provide more compact generalizations (and thus tradeoff instantiation cost to storage cost)

7 Utility of POPI generalizations in improving planning performance

Up to this point, the main focus of this paper has been to provide algorithms to correctly generalize partially ordered partially instantiated plans in a variety of ways. In this section, we shall briefly address the issue of utility of these generalizations in improving planning performance. To begin with, it should be noted that the utility of stored plan generalizations does depend to a large extent on the exact way they are used during planning. There are a variety of ways of reusing stored plans. One obvious way of course is to use the stored plan generalizations as macrops [10]. Another one is to use the stored plan generalizations as schemas and use them in only those situations where they are completely applicable [5, 33]. The former provides a natural ability to combine more than one stored plan to solve a problem at the expense of increased branching factor of the search space, while the latter avoids the utility problems associated with the branching factor increase by going to the stored plan library only once per problem. There are other more sophisticated reuse strategies which improve on both of these: We can use stored plan generalizations in plan reuse, where like schema-based reuse, we retrieve one plan per problem, but unlike schema-based reuse, will facilitate reuse even when the plan is only partially applicable, by allowing flexible modification of the plan (c.f. [18]). We can extend this strategy further by allowing retrieval of multiple plans to cover complementary goal sets of the new problem, and merge and modify the retrieved plans to solve the new problem (see Section 7.1 for a particular strategy along these lines).

The utility tradeoffs between various ways of reusing stored plans has received, and continues to receive, significant attention in the literature (see [16, 12, 22, 20]), and we shall not attempt a general discussion of these issues here. Instead, we will concentrate on the specific tradeoffs and advantages of basing reuse in the context of partially ordered partially instantiated, as opposed to totally ordered, plans.

There are two separate considerations here:

- **Storage Compations:** storing plans as generalized partially ordered and partially instantiated plans
- **Reusing stored plans:** using stored plans in the context of a partial ordering planning vs. total ordering planning framework

We will address these two in turn below.[21] (In the following, the word ''macro'' is used as a short form to any form of stored plan generalization, without committing in any way as to how the generalization will be used as macrops or schemas etc.):

Storage Compactions: The storage compaction provided by the usage of POPI plans has long been acknowledged. As discussed in Section 2, a POPI plan provides a compact representation for the possibly exponential number of its underlying linearizations by specifying just the steps, the partial ordering between steps and the codesignation and non-codesignation constrains between the variables. As we saw in this paper, the flexible plan representation allows for a spectrum of order, precondition and structure generalizations. Storing plans in POPI form also allows for more sophisticated *editing* operations at retrieval time, when the macro is only partly applicable. Specifically, any irrelevant steps and constraints of the plan can be edited out by retracting the corresponding planning decisions. The retraction itself can be facilitated by justifying individual planning decisions in terms of the plan validation structure as discussed earlier.

Consider for example the plan for stacking four blocks x, y, z, w which are all clear and initially on the table, on top of each other (i.e., the goal formula is $On(x, y) \wedge On(y, z) \wedge On(z, w)$):

$$Puton(x, y) \rightarrow Puton(y, z) \rightarrow Puton(z, w)$$

which is being reused in a situation where the goal involves stacking four blocks into two different stacks (i.e., the goal formula $On(A, B) \wedge On(C, D)$). In such a case, with x, y, z and w bound to A, B, C and D respectively, we find that the goal $On(B, C)$ of the macrop is unnecessary in the new problem. This makes the step $Puton(y, z)$ unjustified. When this step is removed from the plan, and the plan is order justified (as described in Section 4.2), we will find that there is no longer any reason to order $Puton(x, y)$ and $Puton(y, z)$ with respect to each other, giving rise to the new edited plan:

$$Puton(x, y) \| Puton(z, w)$$

Clearly, this editing is more flexible than STRIPS triangle table based editing. Once such a justification framework is in place, the retraction of irrelevant constraints can be accomplished with the help of a polynomial time greedy algorithm (see [18, 15]).

However, it must be noted that all the advantages of storage compaction and plan editing will hold whether the underlying planner is a total ordering or a partial ordering planner. For example, as we mentioned several times in this paper, the generalization techniques described here can be used regardless of whether the plan was initially produced by a partial ordering or a total ordering

[21] See [17] for a more complete discussion

ART-IND	$(Action\ A_i\ \underline{precond}:\ I_i\ \underline{add}:\ G_i\ )$
ART-MD	$(Action\ A_i\ \underline{precond}:\ I_i\ \underline{add}:\ G_i\ \underline{delete}:\ \{I_j \mid j < i\})$
ART-MD-NS	$(Defstep\ A_i^1\ \underline{precond}:\ I_i\ \underline{add}:\ P_i\ \underline{delete}:\ \{I_j \mid j < i\})$ $(Defstep\ A_i^2\ \underline{precond}:\ P_i\ \underline{add}:\ G_i\ \underline{delete}:\ \{I_j \mid \forall j\} \cup \{P_j \mid j < i\})$

Figure 16: The specification of Weld et. al.'s Synthetic Domains

planner. In fact, even in reuse frameworks based on total ordering planners (e.g. [37, 34]), order generalization has been used as a way to separate independent parts of the plan and store them separately, thereby containing the proliferation of macrops by reducing the redundancy among them. In other words, although storage considerations motivate the use of POPI plan representation during plan reuse, they do not necessarily argue for the use of POPI planners during planning.

Ability to exploit stored plans during reuse: Normally, when a macro is retrieved to be reused in a new problem situation, it will only be a partial match for the problem under consideration: (i) The macro may contain extraneous goals/constraints that are not relevant to the problem at hand. (ii) There may be some outstanding goals of the problem that the retrieved macro does not match. The first situation can be handled largely through the editing operations described earlier. In the second case, the planner may have to do some further planning work even after the macro is incorporated into the current plan. The way a planner extends the macro during planning critically affects its ability to reuse stored plans in new situations.

Suppose a planner is solving a problem involving a set G of goals, and retrieves a macro M which promises to achieve a subset G' of these goals. Let $g \in (G - G')$ be an outstanding goal of the problem. We will say that g is **sequenceable** with M if and only if there exists a subplan P for achieving g such that $M.P$ or $P.M$ (where ''.'' is the sequencing or concatenation operator) will be a correct plan for achieving the set of goals $G \cup \{g\}$. M is said to be **interleavable** with respect to g, if and only if there exists a sub-plan P for achieving g such that P can be merged with M without retracting any constraints in M or P.

There are many situations when the macros are not sequenceable but only interleavable with respect to the outstanding goals of the planner. Consider the simple artificial domains, ART-IND, ART-MD and ART-MD-NS (originally described in [1]) shown in Figure 16. These domains differ in terms of the serializability of the goals in the domain. ART-IND contains only independent goals (notice that none of the actions have delete lists). The goals in ART-MD are interacting but *serializable* while those in ART-MD-NS are *non-serializable*.[22] In particular, in the latter domain, macros will be interleavable, but not sequenceable with respect to any outstanding goals of the

[22]From the domain descriptions, it can be seen that a conjunctive goal $G_i \wedge G_j$ (where $i < j$) can be achieved in ART-IND domain by achieving the two goals in any order, giving rise to two plans $A_i \rightarrow A_j$ and $A_j \rightarrow A_i$. Only the first of these two plans will be a correct plan in ART-MD domain, since the delete literals in the actions demand that G_i be achieved before G_j. Finally, in ART-MD-NS domain, the subplans for G_i and G_j have to be interleaved to give the plan $A_i^1 \rightarrow A_j^1 \rightarrow A_i^2 \rightarrow A_j^2$.

planner. To illustrate, consider the macro for solving a problem with conjunctive goal $G_1 \wedge G_2$ in ART-MD-NS, which will be: $A_1^1 \rightarrow A_2^1 \rightarrow A_1^2 \rightarrow A_2^2$. Now, if we add G_3 to the goal list, the plan for solving the new conjunctive goal $G_1 \wedge G_2 \wedge G_3$ will be $A_1^1 \rightarrow A_2^1 \rightarrow \underline{A_3^1} \rightarrow A_1^2 \rightarrow A_2^2 \rightarrow \underline{A_3^2}$ (where the underlined actions are the new actions added to the plan to achieve G_3). Clearly, the only way a macro can be reused in this domain is by interleaving it with new operators (unless of course it is an exact match for the problem).

Even when the goals are serializable, as is the case in ART-MD, the distribution of stored macros may be such that the retrieved macro is not sequenceable with respect to the outstanding goals. For example, suppose the planner is trying to solve a problem with goals $G_1 \wedge G_2 \wedge G_3$ from ART-MD domain, and retrieves a macro which solves the goals $G_1 \wedge G_3$: $A_1 \rightarrow A_3$. Clearly, the outstanding goal, G_2 is not sequenceable with respect to this macro, since the only way of achieving $G_1 \wedge G_2 \wedge G_3$ will be by the plan $A_1 \rightarrow \underline{A_2} \rightarrow A_3$, which involves interleaving a new step into the retrieved macro.

The foregoing shows that any planner that is capable of using macros only when they are sequenceable with respect to the outstanding goals is less capable of exploiting its stored plans than a planner that can use macros also in situations where they are only interleavable. Total ordering planners using linearity assumption, such as STRIPS [10], can reuse macros only when they are sequenceable with respect to the outstanding goals. In contrast, POPI planners can reuse macros even when they are only interleavable with respect to the outstanding goals. Although some total ordering planners without linearity assumption do have this capability, the POPI planners' least-commitment strategy, coupled with their flexible plan representation, makes them more flexible and efficient than totally ordered planners in interleaving the new operators. This, we believe is the most important advantage of partial ordering planning during reuse.

7.1 Empirical Evaluation

The discussion above section leads to two plausible hypotheses regarding the utility of POPI planning in plan reuse frameworks. (i) POPI planners are more efficient in exploiting interleavable macros than total ordering planners (with or without linearity assumption) and (ii) This capability significantly enhances their ability to exploit stored macros to improve performance in many situations, especially in domains containing non-serializable sub-goals. We have tested these hypotheses by comparing the performance of three planners -- one a partial ordering planner, and the other two total ordering planners, in conjunction with two different reuse strategies [17].[23] In the following, we describe our experimental setup and discuss the results of the empirical study.

Performance Systems: Our performance systems consisted of three planners implemented by Barrett and Weld [1]: SNLP, TOCL and TOPI. SNLP is a causal-link based partial ordering planner, which can arbitrarily interleave subplans. TOCL is a causal link based total ordering planner, which like SNLP can insert a new step anywhere in the plan, but unlike SNLP, searches

[23] Code and test data for replicating our experiments can be acquired by sending mail to *rao@asuvax.asu.edu*

in the space of totally ordered plans[24]. SNLP, by virtue of its least commitment strategy, is more flexible in its ability to interleave operators than is TOCL. The third planner, TOPI carries out a backward-chaining world-state search. TOPI only adds steps to the beginning of the plan. Thus, unlike SNLP and TOCL, but like planners using linearity assumption, TOPI is unable to interleave new steps into the existing plan.[25] All three planners share many key routines (such as unification, operator selection, and search routines), making it possible to do a fair empirical comparison between them.

Reuse modes: To compare the ability of each planner to exploit the stored plan generalizations in solving new problems, the planners were run in three different modes in the testing phase: **Scratch mode**, **SEBG (or sequenceable EBG) mode** and **IEBG (or interleavable EBG) mode**. In the **scratch** mode, the planner starts with a null plan and refines it by adding steps, orderings and bindings until it becomes a complete plan. In the **SEBG** mode, the planner first retrieves a stored plan generalization that best matches the new problem (see below for the details of the retrieval strategy). The retrieved plan is treated as an opaque macro operator, and is added to the list of operator templates available to the planner. The planner is then called to solve the new problem, with this augmented set of operators. The **IEBG** mode is similar to the SEBG mode, except that it allows new steps and constraints to be introduced between the constituents of the instantiated macro and the rest of the plan, as the planning progresses. To facilitate this, whenever the planner selects a macro to establish a precondition, it splits the macro into its constituent primitive operators and adds them to the existing partial plan. This operation involves updating the steps, orderings and causal links of the current partial plan with those of the macro. In the case of SNLP, the exact ordering of the steps of the macro with respect to the current plan can be left partially specified (e.g., by specifying the predecessor of the first step of the macro, and the successor of the last step of the macro), while in the case of TOCL, partial plans need to be generated for *each* possible totally ordered interleaving of the steps of the macro with respect to the steps of the current partial plan. SNLP is thus more flexible and least committed than TOCL in interleaving macros.

It is easy to see that the SEBG strategy can reuse a macro if and only if it is sequenceable with the other outstanding goals of the plan, while IEBG strategy can reuse a macro whenever it is interleavable with other outstanding goals of the plan. From the description of the three planners above, it should also be clear that only SNLP and TOCL can support IEBG mode.

Storage and Retrieval Strategies: To control for the factors of storage compaction, and flexible plan editing, no specialized storage or editing strategies are employed in either of the planners. The retrieval itself is done by a simple (if unsophisticated) strategy involving matching of the goals of the new problem with those of the macros, and selecting the one matching the maximum number of goals.

Evaluation strategy: As noted earlier, sequenceability and interleavability of the stored macros

[24]Each partially ordered plan produced by SNLP corresponds to a set of totally ordered plans. TOCL generates these totally ordered plans whenever SNLP generates the corresponding partially ordered plans.

[25]If TOPI addresses the goals in a last-in-first-out fashion, then its behavior will be identical to a total ordering state-based planner using linearity assumption, such as STRIPS.

can be varied by varying the type of goals (independent vs. serializable vs. non-serializable domain). The artificial domains described in Figure 16 make ideal testbeds for varying the latter parameter, and were thus used as the test domains in our study. The test domains ART-MD and ART-MD-NS above were somewhat extreme in the sense that the former only has serializable goals while the latter has all non-serializable goals. More typically, we would expect to see a mixture of independent, serializable and non-serializable goals in a problem distribution. To understand how the effectiveness of the various reuse strategies vary for such mixed problem distributions, we also experimented with a mixed domain obtained by combining the actions of ART-IND (the domain with independent subgoals) and ART-MD-NS (the domain with non-serializable subgoals) domains (shown in Figure 16).

Five different training and testing suites, each containing a different (pre-specified) percentage of non-serializable goals in the problem distribution, were generated. We experimented with problem sets containing 0, 25, 50, 75 and 100% non-serializable goals (where 0% corresponding to the problem set having goals drawn solely from ART-IND, and 100% corresponding to the problem set with goals drawn solely from ART-MD-NS).

For each mixture, 50 training problems and 30 testing problems were generated randomly. The training problems all have between 0 and 3 goals. During the training phase, each planner generalizes the learned plans using EBG techniques and stores them. In the testing phase, a set of 30 randomly generated problems, that have between 4 and 7 goals (thus are larger than those used in the training phase) are used to test the extent to which the planners are able to exploit the stored plans in the three different modes. A limit of 1000 cpu sec per problem is enforced on all the planners, and any problem not solved in this time is considered unsolved (This limit includes both the time taken for retrieval and the time taken for planning). To eliminate any bias introduced by the time bound (c.f. [32]), we used the maximally conservative statistical tests for censored data, described by Etzioni and Etzioni in [8], to assess the significance of all speedups. All experiments were performed in interpreted LUCID COMMONLISP running on a Sun Sparc-II.

Experimental Results

The plots in Figure 17 summarize the performance in each problem set as a function of the percentage of the non-serializable goals in the problem set. The plot on the left compares the cumulative time taken by each strategy for solving all the problems in the test suite of each of the 5 problem sets The plot on the right shows the percentage problems successfully solved within the time bound by each strategy for each problem set. We note that SNLP using IEBG shows the best performance in terms of both the cumulative time and the percentage problems solved. IEBG strategy is also the best strategy for TOCL, but turns out to be considerably less effective than the IEBG strategy for SNLP. In all the experiments, TOPI (not shown in the plots) performed considerably worse than SNLP and TOCL. More interestingly, we see that the performance of IEBG strategy compared to the base-level planner improves as the percentage of non-serializable goals in the problem set increases for both SNLP and TOCL. By the same token, we also note that the relative performance of SEBG strategy worsens with increased percentage of non-serializable goals for both SNLP and TOCL. This should not be surprising since as discussed in Section 7.1,

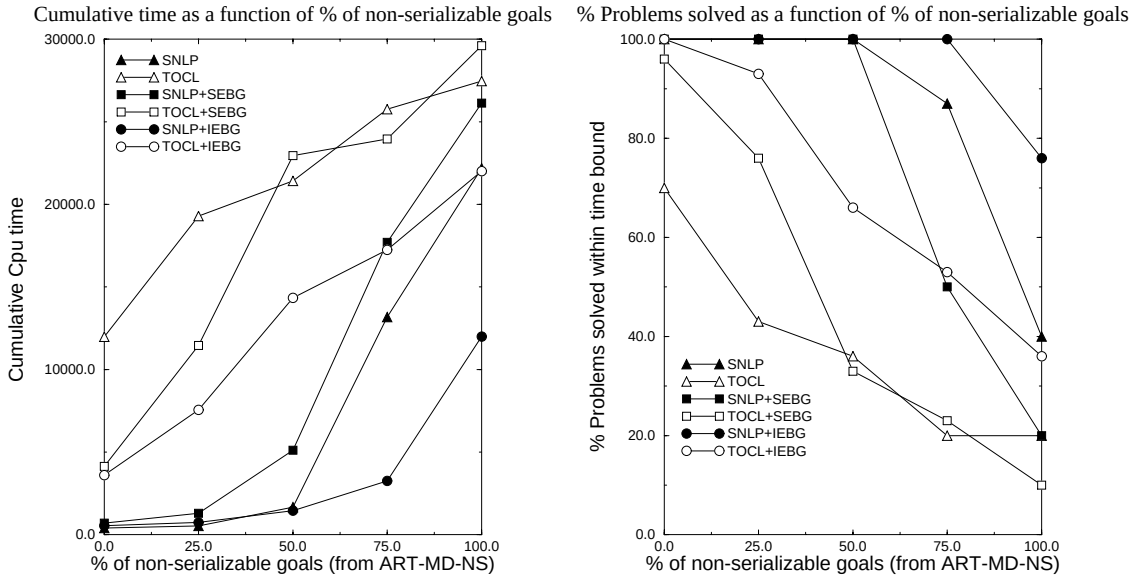


Figure 17: Cumulative performance of various reuse strategies as a function of % of non-serializable sub-goals

as the number of goals from ART-MD-NS increases, the stored plans will not be sequenceable with respect to any remaining outstanding goals of the planner, undermining the ability of SEBG strategies to exploit the stored plans.

The results in our empirical studies are consistent with our hypothesis regarding the superiority of POPI planners in exploiting stored macros. The strategy of using SNLP planner with IEBG reuse strategy significantly outperforms all the other strategies including TOCL+IEBG, both in terms of cumulative time and solvability horizon. The higher cost of TOCL+IEBG strategy can itself be explained by the fact that TOCL generates partial plans corresponding to each possible interleaving of the macro with the new steps, while SNLP can maintain a partially ordered plan and interleave the steps as necessary. Finally, TOPI, which can only add steps at the beginning and end of the plan, is unable to interleave plans, and thus is worse than TOCL in its ability to exploit stored plans.

8 Related Work

In this section, we compare our work to other related efforts that aim to extend Explanation Based Generalization to partial ordering planning.

The research that comes closest to ours is Chien's dissertation work [5] on incremental approximate planning. The main thrust of his work is on failure driven refinement of POPI plans.

His dissertation [5] proposes a method for planning that involves using an approximate model of the domain to *generate* a plan and using a more complex model to *test* that plan. Approximation involves using a simpler modal-truth criterion than is warranted by the operator structure of the domain. This simplifies plan generation at the expense of loss of soundness. To reinstate soundness, every generated plan is checked with respect to the more expressive truth criterion. If a failure is noticed during this testing phase, the plan is ''debugged'' with respect to the latter truth criterion, and is generalized using Explanation-based generalization techniques and the debugged plan is stored in the library of plan schemas. Given a new problem to solve, the planner checks to see if any of its existing schemas directly solves the problem, and resorts to plan generation only when no such plan exists.

It is the generalization phase of Chien's work that is of direct relevance to the work reported in this paper. Both our work and his developed the framework showing that EBG can be performed on partial plans by using the explanation of how a plan satisfies a truth criterion, and generalizing the plan based on that explanation (although the exact structures used for representing the explanations are different). This initial development was done independently and approximately the same time.[26] Our current research extends Chien's and our initial work significantly. We have an extended spectrum of POPI generalizations, and present these in an integrated framework based on validation-structure based representation of plan explanation structures. We discuss the tradeoffs between the spectrum of generalizations, and also provide a comparative analysis of the utility of EBG in POPI and total ordering planning. Chien's research extended the work on generalized POPI plans in a different direction, examining the failure-driven nature of planning with simplified plans.

It is also interesting to note that Chien uses his generalized plans chiefly to guide schema-based planning, where a plan is reused only when it is completely applicable. In contrast, as our empirical studies in Section 7 show, one of the most important advantages of EBG in the context of POPI planning is the ability to *interleave* stored plan generalizations during planning.

Other related work includes methods for generalizing the ordering constraints in total ordering plans (c.f. [27, 4, 36, 9]). These efforts were all directed towards generalizing totally ordered plans produced by total ordering planners to partially ordered plans. The motivation was often to provide storage or execution optimization. Although these efforts are related to our order generalization algorithms, our development is directly based on the partial ordering planning. Consequently our treatment puts order and precondition generalization in an integrated framework which explicates the spectrum of order/precondition generalizations (see Sections 4.2 and 4.3).

Our algorithms for generalizing POPI plans correspond directly to the EBG explanation and generalization steps, but work on specialized explanations of correctness tailored to plans, rather than arbitrary proofs. It is possible to use the standard EBG algorithm [26] itself for this purpose --- by proving (explaining) correctness of a plan directly from first order situation calculus. The advantage of dealing with specialized explanations is that they often can be produced much more efficiently. In particular, we showed that explanations of correctness (validations) based on

[26] Chien's work was brought to our notice after a preliminary version of our work was presented at AAAI-91 ([19])

modal truth criteria, can be generated in polynomial time for plans with STRIPS-type operators without conditional effects (Section 3). In contrast, generating proofs in full situation calculus is undecidable. In addition, by starting with a provably sound and complete truth criterion and deriving the EBG algorithms directly from that, we obviated the need to carry out a separate formal proof of correctness of the algorithms (e.g. [2]).

9 Concluding Remarks

In this paper, we addressed the problem of generalizing partially ordered and partially instantiated (POPI) plans -- a class of plans which have been extensively investigated in the planning literature. Our main contribution is a unified framework for plan generalization based on the modal truth criteria, which gives rise to a spectrum of generalization algorithms. We have developed the formal notion of explanation of correctness for POPI plans based on the modal truth criteria and used this to derive declarative specifications for generalizing them in a variety of ways. These include precondition generalization (for both necessary and possible correctness), as well as order generalization. We have also characterized the spectrum of tradeoffs offered by the various generalizations, including the cost of producing and using them. Finally, we also addressed the utility of these generalizations in improving planning performance, and described an empirical study characterizing the relative advantages of doing EBG based plan reuse in a partial ordering, as opposed to total ordering, planning framework.

The generalization algorithms described in this paper can be easily implemented on top of any partial ordering (nonlinear) planner. We have implemented them on top of a public-domain domain-independent nonlinear planner called SNLP[1], and used the implementation to characterize the relative advantages of basing EBG based plan reuse within partial ordering planning. Even though the development here provided a separate algorithm to compute the explanation of correctness of a POPI plan, often the explanation construction phase can be integrated with the generation of the plan. In particular, most partial-order planners (such as NONLIN [35], SIPE [38], as well as SNLP which we used in our implementation) keep track of the ‘‘validation structure’’ of the plan being developed to aid in plan generation. Thus the generalization can be closely integrated with planning.

9.1 Limitations and Future Directions

The model of planning used in this paper assumes an operator representation that is free of conditional effects and conditional preconditions. We believe that similar methodology can also be used for more expressive operator representations. It must however be noted that increased expressiveness of the operators necessitates truth criteria that are correspondingly expensive to interpret (e.g. [30, 6]), increasing the cost of EBG.

Although we concentrated on the issue of plan generalization, we believe that similar truth criterion based frameworks can also be developed for explanation based learning of other types

of target concepts, such as failure and goal interactions (c.f. [25]). In particular, it seems likely that the truth criterion most directly affects the architecture level axioms of an EBL framework such as the one described in [25]. We are currently working towards verifying this intuition. It should however be noted that the utility of learning particular types of target concepts depends to a large extent on the nature of the underlying problemsolver. In particular, it seems likely that target concepts such as ''goal interactions'' may not be as utile in a POPI planner as they will be in a total ordering planner such as PRODIGY (since the former already deals with step ordering in a more intelligent fashion). On the other hand, loop detection is easier in a total order state based planner[27] than a POPI planner. Thus, this may be a more utile concept to learn in a POPI planner. Understanding such tradeoffs is an important aim of our ongoing work [16].

In Section 7, we showed that POPI planners, by virtue of their ability to intelligently interleave the retrieved plan with new operators, do promise several advantages from the point of view of plan reuse. We can go further along these lines and integrate the generalization process with a plan modification framework such as PRIAR [18] which can flexibly modify any given plan to fit any new problem situation.[28] Such an integrated framework will allow us to reuse generalized plans in a variety of new situations (including those in which they are only partially applicable, and can be made applicable only by retracting some of the previous planning decisions). This presents interesting tradeoffs compared to a MACROP [10] based reuse mechanism in terms of the demands on storage and retrieval. For example, in Section 5.1, we noted the memory-vs-time trade-off that can be achieved by computing and using the conditions for possible correctness of a generalized plan. Further more, given the flexibility of reuse, exact match retrieval may no longer be critical; see [16]). Our future plans include integrating the generalization algorithms on top of PRIAR modification framework [11, 14, 18], and carrying out empirical experimentation to get a clearer understanding of these tradeoffs [16][29].

Acknowledgements: We acknowledge Jengchin Chen's help in implementing and carrying out the empirical comparisons reported in the paper. A remark by Drew McDermott at the 1990 Darpa Planning Workshop, regarding truth criteria and goal regression, led us to investigate truth criteria as the basis for plan generalization. Discussions with Steve Minton, Mark Drummond and John Bresina and Prasad Tadepalli, helped significantly in clarifying our presentation. We also acknowledge helpful suggestions from Peter Cheeseman, Pat Langley and Amy Lansky.

[27] In fact, many of the architectural axioms in [25] depend explicitly on the problem solver's ability to detect ''loops.''

[28] From a theoretical point of view, this essentially involves allowing retraction into the modal truth criterion of the plan.

[29] Such an evaluation should, of necessity, be comparative in the sense that it should (*i*) explain the merits of reuse in comparison to other learning techniques such as abstraction, search control rules and (*ii*) provide methods of fruitfully integrating these different learning techniques. In [16], we provide a preliminary proposal for such a comparative study of tradeoffs.

References

[1] A. Barrett and D. Weld. Partial order planning: Evaluating possible efficiency gains. Technical Report 92-05-01, Department of Computer Science and Engineering, University of Washington, Seattle, WA, June 1992.

[2] N. Bhatnagar. A correctness proof of explanation-based generalization as resolution theorem proving. In *Proceedings of the 1988 AAAI Spring Symposium on Explanation Based Learning*, pages 220--225, 1988.

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

[4] S.A. Chien. Using and refining simplifications: Explanation-based learning of plans in intractable domains. In *Proceedings of IJCAI-89*, pages 590--595, Detroit, MI, 1989.

[5] S.A. Chien. *An Explanation-Based Learning Approach to Incremental Planning*. PhD thesis, TR. UIUCDCS-R-90-1646 (Ph.D. Thesis). Dept. of Computer Science, University of Illinois, Urbana, IL, 1990.

[6] J. Christensen and A. Grove. A formal model for classical planning. In *Proceedings of 12th Intl. Joint Conference on AI (IJCAI-91)*, 1991.

[7] G. DeJong and R. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145 -- 176, 1986.

[8] Oren Etzioni and Ruth Etzioni. Statistical methods for analyzing speedup learning experiments. *Machine Learning*. (To Appear).

[9] B. Fade and P. Regnier. Temporal optimization of linear plans of action: A strategy based on a complete method for the determination of parallelism. Technical report, IRIR-Laboratorie Langages et Systemes Informatiques, Universite Paul Sabatier, France, 1990.

[10] R. Fikes, P. Hart, and N. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3(4):251--288, 1972.

[11] S. Kambhamapti. Supporting plan reuse. In S. Minton and P. Langley, editors, *Learning Methods for Planning*. Morgan Kaufmann, Palo Alto, Stanford, 1991 (in press).

[12] S. Kambhampati. Utility tradeoffs in plan reuse and casebased planning. Submitted to Intl. Conf. on Machine Learning, 1993.

[13] S. Kambhampati. Mapping and retrieval during plan reuse: A validation-structure based approach. In *Proceedings of 8th National Conference on Artificial Intelligence*, August 1990.

[14] S. Kambhampati. A theory of plan modification. In *Proceedings of 8th National Conference on Artificial Intelligence*, August 1990.

[15] S. Kambhampati. Characterizing multi-contributor causal structures for planning. In *Proceedings of 1st Intl. Conf. on AI Planning Systems*, 1992.

[16] S. Kambhampati. Utility tradeoffs in incremental plan modification and reuse. In *Working Notes of the 1992 AAAI Spring Symposium on Computational Considerations in Supporting Incremental Modification and Reuse*, March, 1992.

[17] S. Kambhampati and J. Chen. Relative utility of basing ebg based plan reuse in partial ordering vs. total ordering planning. Submitted to AAAI-93, 1993.

[18] S. Kambhampati and J.A. Hendler. A validation structure based theory of plan modification and reuse. *Artificial Intelligence*, 55(2-3), June 1992.

[19] Subbarao Kambhampati and Smadar Kedar. Explanation based generalization of partially ordered plans. In *Proceedings of 9th AAAI*, 1991.

[20] Smadar Kedar and Kathhleen McKusick. Tradeoffs in the utility of learned knowledge. In *Proc. 1st Intl. Conf. on AI Planning Systems*, pages 281--282, 1992.

[21] D. McAllester and D. Rosenblitt. Systematic nonlinear planning. In *Proceedings of 9th AAAI*, 1991.

[22] S. Minton. Issues in the design of operator composition systems. In *Proceedings of the International conference on Machine Learning*, 1990.

[23] S. Minton. Quantitative results concerning the utility of explanation-based learning. In *Artificial Intelligence*, volume 42, pages 363--392, 1990.

[24] S. Minton, J. Bresina, and M. Drummond. Commitment strategies in planning: A comparative analysis. In *Proceedings of 12th IJCAI*, 1991.

[25] 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.

[26] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based learning: A unifying view. *Machine Learning*, 1(1):47 -- 80, 1986.

[27] R. J. Mooney. Generalizing the order of operators in macro-operators. In *Proceedings of the 5th International Conference on Machine Learning*, pages 270--283, June 1988.

[28] R. J. Mooney and S. W. Bennett. A domain independent explanation-based generalizer. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 551--555, Philadelphia, PA, 1986. Morgan Kaufmann.

[29] N. J. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishers, 1980.

[30] E.P.D. Pednault. Generalizing nonlinear planning to handle complex goals and actions with context-dependent effects. In *Proceedings of 12th Intl. Joint Conference on AI (IJCAI-91)*, pages 240--245, 1991.

[31] E. D. Sacerdoti. *A Structure for Plans and Behaviour*. American Elsevier, New York, NY, 1977.

[32] Alberto Segre, Charles Elkan, and Alexander Russell. A critical look at experimental evaluation of ebl. *Machine Learning*, 6(2), 1991.

[33] J. Shavlik. *Generalizing the structure of Explanations in Explanation Based Learning*. PhD thesis, Dept. of Comp. Sci., University of Illinois, 1988. (Available as technical report UILU-ENG-87-2276).

[34] P. Tadepalli and R. Isukapalli. Learning plan knowledge from simulators. In *Proceedings of the workshop on Knowledge compilation and speedup learning*.

[35] A. Tate. Generating project networks. In *Proceedings of IJCAI-5*, pages 888--893, Boston, MA, 1977.

[36] M. M. Veloso, M. A. Perez, and J. G. Carbonell. Nonlinear planning with parallel resource allocation. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 207--212, November 1990.

[37] Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, Carnegie-Mellon University, 1992. (CMU-CS-92).

[38] D. E. Wilkins. Domain independent planning: Representation and plan generation. *Artificial Intelligence*, 22:269--301, 1984.

[39] Q. Yang. A theory of conflict resolution in planning. *Artificial Intelligence*, 58(1-3):361--392, 1992.

Contents

List of Figures