Dominance and Subsumption in Constraint-Posting Planning

Michael P. Wellman[1]
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Abstract

By integrating a dominance prover into the plan search process, the traditional constraint-posting planning paradigm can be generalised to permit partially satisfiable goals. In this approach, the view of planning as theorem proving is retained, although the emphasis is on proving theorems about dominance relations on classes of candidate plans. Plan classes are generated by posting constraints at various levels of abstraction, then classified within a plan lattice that manages inheritance of properties and dominance characteristics. An analysis of TWEAK demonstrates how to recast existing planning ideas in this framework, providing insight into the planner's capabilities with a dominance-proving interpretation. The plan lattice representation highlights the role of plan subsumption in optimizing search.

I. Introduction

One glaring limitation of the traditional AI robot planning paradigm is its view of goals as logical predicates on world states. Such a view precludes the consideration of uncertainty except in rare cases where either uncertainty is adequately captured in tolerances or execution monitoring with replanning is viable. Even in the absence of uncertainty, planning to satisfy goal predicates provides no mechanism to choose among the plans that achieve the goal nor any guidance about what to do when no such plan exists. The literature has offered ad hoc solutions to overcome some of these limitations in specific cases, for example, simple resource allocation mechanisms or domain-specific meta-planning rules.

Alternate views, such as Bayesian decision theory, offer more comprehensive choice criteria but do not address the planning task of *constructing* strategies from more primitive actions.

In this paper I describe a formal framework for planning that permits the broader range of choice criteria necessary in an environment of partially satisfiable goals with multiple objectives. At the same time it can incorporate many of the principles, representations, and techniques of existing research on planning. Central to the framework is the notion of planning as the derivation and propagation of dominance relations among classes of candidate plans.

Definitions and explanations of the structures constituting the planning framework are provided in the sections below. The theoretical apparatus is illustrated with an investigation of TWEAK, a nonlinear constraint-posting planner [Chapman, 1985a]. Although TWEAK is defined in terms of logical goal predicates, an interpretation in terms of the dominance relation provides insight into its capabilities and efficiency. An analysis of TWEAK's search procedure highlights the role of plan subsumption computations in this approach to planning.

II. The Plan Lattice

Let C be the planning language, or equivalently, the set of all syntactically valid plans. For example, if $A = \{a_1, \ldots, a_n\}$ is an alphabet of primitive actions, then $L = A^*$ is the language of linear plans. The language of nonlinear plans is similar, extended by an encoding for partial orders. A *plan class* is any set of plans, $P \subset L$. P is also called a *partial plan* when it represents a set of constraints that incompletely specifies the plan we are concerned about.

We can view the planning process as one of incrementally adding constraints to candidate plan classes until some problem is solved regarding the plan to be executed. In traditional planning the problem is to identify a plan that satisfies the given goal. With a more versatile evaluation criterion, the problem is to find the best plan. Except for some special cases where convenient optimization techniques are applicable, it is not possible to determine whether a given plan is optimal by examining it in isolation. It may be more reasonable to answer questions about the optimal plan without necessarily constructing a complete description.

The partial plans generated during the planning process can be organized in a specialization lattice according to the subset relation. An example of a plan specialization lattice appears in Figure 1. The node in the graph marked $^{u}A^*a_1A^*{}''$ denotes the set of all plans with at least one instance of action a_1. The set of plans *beginning* with a_1 forms a subclass, as does the set of plans where a_1 is followed by a_2.

The plan lattice representation of a search space

[1] Supported by National Institutes of Health Grant No. R01 LM04493 from the National Library of Medicine.

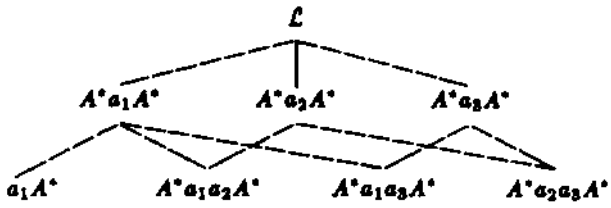

Figure 1: A plan specialization lattice.

supports a *constraint-posting* approach to planning. A constraint-potting planner, illustrated best by MOLGEN [Stefik, 1981], can be more efficient than a planner that evaluates only complete plans. Flexibility is gained by allowing many forms of constraints rather than, for instance, just adding actions to a sequence or specifying exact bindings for variables. However, these advantages depend on having some justification for the constraints based on properties of the partial plan. For example, MOLGEN knows that for a *screen* operation to be useful, it must select the appropriate bacteria. Thus, when adding a *screen* step to a plan, MOLGEN is justified in posting a constraint of the form (resists *antibiotic-1 bacterium-4*); constraining the antibiotic to a particular chemical agent would be unjustifiably specific at this stage.

By adding only the constraints that have the best justifications, a planner implements a *least commitment* strategy. An extreme form of least commitment propagates only provable properties of admissible plans. In practice, however, real planners, like MOLGEN, have to make guesses when no provable constraints are available. The least commitment heuristic tends to minimize both the likelihood of wrong guesses and the extent of backtracking required to recover from such mistakes. A policy of working on plan classes at high levels of abstraction is simply a particular form of least commitment strategy. Because it avoids backtracking, a search procedure that preserves the entire lattice of partial plans corresponds to "no commitment" planning.

III. Dominance in the Plan Lattice

To speak meaningfully of dominance among plan classes, we need to introduce a *preference relation,* $\succ$, over plans. In categorical planning, for example, one plan is preferred to another if it achieves the goal and the other does not. To state this in terms of the *situation calculus* [McCarthy and Hayes, 1969], we write:

$$\pi_1 \succ \pi_2 \;\Leftrightarrow\; G(result(robot, \pi_1, s_i)) \wedge \neg G(result(robot, \pi_2, s_i)). \quad (1)$$

G is the goal predicate, defined on situations resulting from the robot performing a plan in a given situation. Here s_i denotes the initial situation. Two plans that both achieve or do not achieve the goal are equally preferred, or *indifferent,* denoted by $\sim$. The expression $\pi_1 \succeq \pi_2$ means that π_1 is preferred or indifferent to π_2.

The preference relation characterises the choice criterion employed by the planner. A planner based on expected utility takes

$$\pi_1 \succ \pi_2 \Leftrightarrow E[u(\pi_1)] > E[u(\pi_2)]. \quad (2)$$

The discussion of dominance that follows does not depend on any particular criterion for plan choice. Although we assume that $\succeq$ is a total order on plans, we do not insist that the planner be given a complete or even an explicit description of the preference relation.

We say that a class of plans *dominates* another if, for any plan in the second class, some plan in the first class is preferred or indifferent. It should be emphasised that it is possible to prove dominance without identifying the superior plan—otherwise this approach provides no advantage over branch-and-bound search. The dominance relation, D, is defined as follows:

$$D(P_1, P_2) \stackrel{def}{=} \forall \pi_2 \in P_2 \; \exists \pi_1 \in P_1 \; \pi_1 \succeq \pi_2. \quad (3)$$

The strict version of dominance, D' is defined similarly, except that here a particular plan in the first class is preferred to any in the second.[2]

$$D'(P_1, P_2) \stackrel{def}{=} \exists \pi_1 \in P_1 \; \forall \pi_2 \in P_2 \; \pi_1 \succ \pi_2. \quad (4)$$

Clearly, strict dominance implies dominance. In addition, the properties below follow easily from the definitions:

D is reflexive, transitive, and complete. (5)

D' is anti-reflexive, transitive, and asymmetric. (6)

$$D(P_1, P_2) \Leftrightarrow \neg D'(P_2, P_1) \quad (7)$$

$$P_2 \subseteq P_1 \Rightarrow D(P_1, P_2) \quad (8)$$

$$D(P_1, P_2) \wedge D(P_3, P_4) \Rightarrow D(P_1 \cup P_3, P_2 \cup P_4) \quad (9)$$

$$D(P_1, P_3) \vee D(P_2, P_3) \Leftrightarrow D(P_1 \cup P_2, P_3) \quad (10)$$

$$D'(P_1, P_2) \wedge D(P_2, P_3) \Rightarrow D'(P_1, P_3) \quad (")$$

These properties serve as dominance propagation rules within the plan lattice. By (8) and the transitivity of D_y dominance by a particular class is inherited in the plan lattice. Strict dominance is also inherited, by (11) and weak dominance inheritance. Thus, markers or links indicating dominance relations need be stored only at the upper envelope of classes to which they apply. Dominance

[2] This difference is required because of the possibility of infinite plan classes with no maximal elements. If (4) were exactly a strict version of (3), then such a class would strictly dominate itself. For the same reason, a definition of weak dominance that merely substituted $\succeq$ for $\succ$ in (4) would not entail the reflexive property. Assuming every plan class has a maximal plan is unreasonable, even if it is appropriate to require that t does.

is propagated upwards in the lattice by application of the union properties (9 and 10), which also hold for D'.

A plan class is *restricted* by asserting that it is weakly dominated by one of its subsets. In the MOLGEN example given above, if P_1 is the class of plans that include the *screen* operation, and P_2 is the subclass that includes the resists relation as well, then $D(P_2, P_1)$ asserts that (resists *antibiotic-1 bacterium-4)* is a *valid* constraint. The new dominance assertion represents progress because it lets us focus our attention on a smaller set of plans. Thus, deriving these restrictions is an important task of the dominance prover.

Constraints might be posted to explore the search space even though the dominance relation does not provably hold. Often, such constraints are justified by identifiable *assumptions* that imply dominance. We can express this case by asserting the *conditional dominance relation,* D_S, for S an assumption proposition

$$D_S(P_1, P_2) \stackrel{\text{def}}{=} S \Rightarrow D(P_1, P_2). \qquad (12)$$

Normal dominance is just dtrue. As an example of conditional dominance, suppose that we are uncertain about the identity of the organism of interest; it could be either *bacterium-£ or bacterium-S.* For i = 2 and 3 let s_i be the proposition "Bacterium-i is the organism of interest" and P_i the plan class that restricts p_1 to those plans in which the resists relation holds between *antibiotic-1* and *bactcrium-i.* Then we have $D_{S_2}(P_2, P_1)$, $D_{S_3}(P_3, P_1)$, and $S_2 \vee S_3$. By the definition of conditional dominance (12), we g $D(P_2, P_1) \vee D(P_3, P_1)$.a t i o n of (10) yields the result $D(P_2 \cup P_3, P_1)$.

Of particular value are conditional dominance relations where S itself contains dominance assertions. For example, if $S' \equiv D(P_1, \mathcal{L})$, then $D_{S'}(P_2, P_1)$ asserts that given the optimal plan is in ?\ we can further confine attention to P_2.[3] This is one way to derive the restrictions mentioned above. In fact, this is precisely the strategy employed by both Pednault [Pednault, 1985] and Chapman [Chapman, 1985b) to limit the search space of their planners. We will see how this works for the latter example in Section V below.

Reasoning about conditional dominance can be implemented straightforwardly via any mechanism for reason maintenance [de Kleer, 1986a, McAllester, 1982]. The interesting task for the dominance prover is to come up with meaningful conditions that imply useful dominance relations.

IV. Planning

A program manipulates the plan lattice by repeatedly performing the following steps (not in any particular order):

- Generate new plan classes by adding constraints to undominated classes.
- Construct and refine the prediction/evaluation part of the world model.
- Derive and propagate dominance relations among plan classes. Strengthen conditional dominance relations.

We will say that a program performing these tasks is *planning.* Note that in this view planning is not a search for a single plan to execute, but an exploration of properties of admissible plans. A planner performs useful work by refining the plan lattice, even if the lowest-level classes are never reduced to singleton sets. If, after much computation, the planner has narrowed the admissible plans to a set that contains 10^{400} or possibly an uncountable infinity of plans, this may seem like little progress. But if we can determine that all plans contain, for example, an appendectomy, we solve a significant problem.[4]

The prevailing view of planning as construction of a completely specified course of action is never totally accurate. Planners devote their resources to isolated *decisions,* as in whether or not to perform an appendectomy, without specifying all other features of the plan. A plan to obtain some bananas is complete only with respect to a *have-bananas* goal; in the larger context of satisfying all physical and emotional needs forever, the agent never stops planning. Figuring out how to get the bananas is a small act of refinement on THE BIG PLAN.

The framework presented so far should be regarded as an abstract model of planning with partially satisfiable objectives. It generalizes the case of goal predicates and applies to uncertain situations. Rather than prove that a plan necessarily achieves a goal, as in traditional AI planning, the planner tries to prove properties of the optimal plan. These properties define the class of admissible plans.

To instantiate the abstract model to a particular planning mechanism, one needs to specify:

- The plan language, $\mathcal{L}$.
- A constraint language (representation for partial plans).
- A domain modeling language, including a way to describe the effects of actions and a representation for the preference relation, $\succ$.
- A dominance prover.

The structures described earlier—the plan lattice and dominance relations—serve mainly as theoretical machinery for analysis of this class of planners. Specifying the languages and the dominance prover is the real work in designing a planner.

[3] In categorical planning, $D_{S'}$ is the same as D (that is, $D_{S'}(P_1, P_2) \Leftrightarrow D(P_1, P_2)$) due to the binary nature of the preference relation (1).

[4] Plan classet with fuch huge cardinalities should be the norm, not exceptions. If the plan language includes real-valued parameters, then all but the tightest constraints still leave an uncountable set of candidate plans. The plan class "Administer a dose of drug *X* within the next minute* includes individual plans where the drug is given at any point in the 60-second interval.

As a simple illustration, consider mathematical optimization techniques as planners. Optimization is a special case of dominance proving where the program tries to find a singleton dominator, often in one step. For example, if our plan language is 39 and the domain model consists of a linear objective function and a set of linear constraints among the elements of the vector, then our dominance prover should be a linear programming algorithm. In this case there are no partial plans. Branch-and-bound integer programming is an example of an optimization procedure that does make use of partial plans and explicit dominance proving.

In the development of the planning model up to now, we have paid little attention to efficiency. The computational value of planners in this framework depends on a judicious choice of the languages and algorithms that define it. Although it is difficult to characterize efficiency at the present level of generality, there are a few high-level issues that can be identified at this point. First, the addition of constraints during lattice refinement cannot be arbitrary. The planner must generate constraints that relate to the problem at hand and are meaningful to the dominance prover. Unless the prover can establish dominance relations on the lattice, refinement is irrelevant. Second, it is important to consolidate the plan lattice to avoid redundancy and further the propagation of dominance relations. We will examine this topic further in Sections VI and VII below.

V. An Extended Illustration: Tweak

In the introduction I suggested that existing planning work can be recast in this framework. In this section I examine TWEAK [Chapman, 1985a, Chapman, 1985b], a nonlinear planner that captures much of the state-of-the-art in a neat algorithm. Though TWEAK belongs to the mainstream planning tradition in considering only goal predicates, its main ideas can be expressed clearly in terms of the plan lattice and dominance relation.

As described in the previous section, the way to instantiate this planning framework is to define the various representations appearing in and operations performed on the plan lattice. The plan language for TWEAK is particularly simple. A plan is a sequence of steps, each specifying an action applied to some objects. The term "nonlinear" refers not to plans, but to the representation for incomplete plans, or plan classes.[6] A plan class is nonlinear if it specifies only a partial order on its steps. There are two other sources of incompleteness in TWEAK: steps may be missing, and steps may refer to variables rather than constant objects. Thus any partial plan in TWEAK may be specified by the steps it includes, the ordering constraints on the steps, and the constraints on the designations of variables in the steps.

The domain modeling language is also quite simple. The effects of actions are completely described by finite sets of pre- and post-conditions on each step. A world model corresponding to each plan class records the status of goal and condition propositions in a propositional database. Finally, $\succeq$ is defined by the categorical planning preference relation (I). In TWEAK, *G* is simply a conjunction of propositions.

The interesting part of TWEAK is not these representation languages, but the dominance properties that are applied. The power of the planning algorithm derives from the fact that, given a partial plan, we need to consider only a few types of constraints to guarantee that if a satisfactory completion of the partial plan exists, one also exists among its constrained subclasses. Constraints are posted via *plan modification operators*. For our purposes it helps to regard these operators as functions that return constrained plan classes given a starting plan class and possibly some other arguments. We need five plan modification operators:

- $addstep(P, t)$
- $order(P, s, t)$
- $codesignate(P, x, y)$
- $noncodesignate(P, x, y)$
- $ifcodesignate(P, x, y, z, w)$

Each returns the set of elements of *P* that satisfy the indicated constraint. *Addstep* constrains the partial plan to include an instance of step *t*. *Order* confines *P* to those plans in which step *s* is applied before step *t*. Note that if this (or the result of any modification operation) is a contradiction, then the function returns the empty plan class. Analogously, if the constraint is already implied, the operator is the identity function on *P*. The *code si gnat e* (abbreviated *cod)* and *noncodesignate (ncod)* functions add the appropriate constraints to the elements indicated. In general, there may be several ways to implement a codesignation among complex elements (such as predicate instances) in terms of their primitive constituents. Finally, the *ifcodesignate (ifcod)* function constrains *z* and *w* to codesignate if *x* and *y* do.

TWEAK is defined by a nondeterministic procedure that achieves a goal by applying combinations of these operators to a partial plan. Chapman presents the procedure as a simple graph ([Chapman, 1985a, page 1024], [Chapman, 1985b, page 11]) where the paths from start to end exhaust the possible sequences of plan modifications that can restrict a partial plan *P* to achieve a goal proposition *p* at step *s*. A completely analogous description in our functional notation is the expression appearing in Figure 2. This complicated expression, consisting of a combination of plan classes formed by various modifications on P, represents the set of plans nondeterministically explored by Chapman's algorithm.

[8] A planner with a parallel execution capability (for example, a multi-agent planner) could actually have a nonlinear plan language. The constraint language for such a planner would be more complex.

$$\bigcap \bigcup_{t \in E} \bigcap \begin{array}{l} A_t \\ \bigcup_{u \in conseq(t)} B_{t,u} \\ \bigcap_{c \in C} \bigcup \begin{array}{l} C_c \\ \bigcap_{q \in conseq(c)} \bigcup \begin{array}{l} D_q \\ \bigcup_{w \in W} \bigcap \begin{array}{l} \mathcal{E}_{c,w} \\ \mathcal{F}_w \\ \bigcup_{r \in conseq(w)} \mathcal{G}_{q,w,r} \end{array} \end{array} \end{array} \end{array}$$

key:

$A_t = order(addstep(P,t),t,s)$
$B_{t,u} = cod(addstep(P,t),u,p)$
$C_c = order(P,s,c)$
$D_q = ncod(P,p,q)$
$\mathcal{E}_{c,w} = order(addstep(P,w),c,w)$
$\mathcal{F}_w = order(addstep(P,w),w,s)$
$\mathcal{G}_{q,w,r} = ifcod(addstep(P,w),p,q,r,p)$

Figure 2: The TWEAK algorithm.

Let *P'* denote this plan class. In the figure, *E* is the set of *establishers;* that is, actions that possibly assert the goal proposition. *C* and *W* are the potential *clobbercrs* (actions that negate the goal) and *white knights* (actions that re-establish the goal), respectively. The function *conseq* on steps returns the consequent propositions, or postconditions, of the action. That there is a plan in p' that achieves the goal if there is one in *P* can be captured by asserting $D(P', P)$. This is a restriction, because each of the components of the *P'* expression is itself a subset of *P*.

The main theoretical contribution of TWEAK boils down to this dominance condition. TWEAK refines *P* by adding constraints until it finds a partial plan that necessarily achieves the goal, that is, a class provably containing only goal-achieving plans. The theoretical apparatus we have developed can help us understand this process.

Each term in the expression of Figure 2 corresponds to a plan class which is a subset of P. These distinguished plan classes serve as intermediate structure in the plan lattice depicted in Figure 3. This lattice reflects a decomposition of the search space based on the TWEAK algorithm. The plan classes with subscripts denote parameterised classes; a complete diagram of this lattice would include, for instance, an *At* for every $t \in E$. For the directed links, the plan class pointed to is defined by the boxed set operation applied to the others. Thus $K_t \stackrel{def}{=} B_t \cap A_t$ and $I_c \stackrel{def}{=} \bigcap_q I_{c,q}$. All connections indicate specialization of plan classes from higher to lower levels.

Because it would be difficult for the planner to generate a description of $P.^1$ directly from the definition of Figure 2, in practice it is necessary to manipulate representations of these intermediate classes. The TWEAK search procedure nondeterministically chooses intermediate plan classes to instantiate and combine until it finds a subset of P^1 that necessarily achieves p at s. In the process, the algorithm recursively invokes itself, taking one of the intermediate classes as *P* and forming the structure of Figure 3 beneath it.

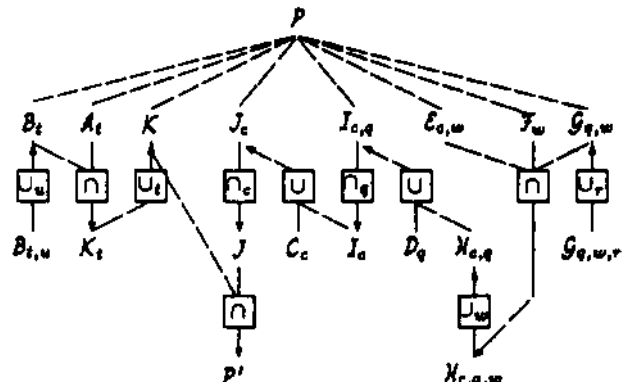

Figure 3: The plan lattice corresponding to the space searched by the TWEAK algorithm.

VI. Classification and Dependency-Directed Search

It is possible that a recursive invocation of the TWEAK dominance relation would reveal that some intermediate plan classes at one level are dominated by more restricted classes at lower levels. More generally, propagation via (8) through (11) may uncover dominance conditions within or between levels which can be exploited to reduce the search space.

These possibilities suggest an addition to the planning steps listed at the beginning of Section IV. When partial plans are generated, they are *classified* by finding their greatest lower and least upper bounds in the plan lattice.[0] Classification serves to consolidate the lattice, resulting in a maximal propagation of the known dominance conditions.

The dominance relation provides us with a class of *nogood* sets to be used in pruning the search space. Planners without a notion of dominance can consider only contradictory plans (that is, empty plan classes) to be nogood; other pruning criteria must be built into the control mechanism. The discussion above illustrates that although TWEAK's search procedure implicitly takes advantage of the fact $D(P', P)$, it cannot exploit the dominance consequences for intermediate plan classes.

Consolidating the plan lattice through classification prevents redundancy in plan space search. Notice, for example, that the plan class D_q depends only on *q,* even

[6]The term *classified is* used here in tht same sense that concepts are clarified in KL-ONE [Schmolse and Lipkis, 1963). Swartout and Neches [Swartout and Neches, 1986) represent plans by the goals they are intended to achieve and employ a classifier to construct a taxonomy of intents. Their decision-making program selects from the taxonomy the most specific plan for the purpose at hand. Because this taxonomy comprises *compiste* plans, their use of classification falls outside of the planning framework developed here.

though its position in the expression for *P'* is within the scope of c. A chronologically backtracking search would generate this class for every *(c,q)* pair encountered during the iteration, resulting in a duplication of effort if clobberers share consequents.

A simple dependency-recording mechanism would most likely avoid this and similar redundancies. A clever enough scheme might even recognize that $\mathcal{A}_t$ and $\mathcal{F}_w$ are the same when $t \simeq w$. However, it is doubtful that any of the standard dependency maintenance mechanisms would catch the more subtle relationships that hold between classes created on successive recursive invocations of the planning algorithm. A precise characterization of avoidable redundancies is rarely offered in descriptions of planners.

Following the terminology used by de Kleer in describing his *assumption-based* truth maintenance system (ATMS) [de Kleer, 1986a], each plan class is an *assumption context* represented by the plan modification operators defining it. The plan specialization lattice corresponds to the context lattice of the ATMS with context subset replaced by plan subsumption. Indeed, an implementation using an ATMS would be just as powerful as the classification scheme presented here, *provided that* we could construct a propositional interface [de Kleer, 1986b] capable of communicating the relevant implications of partial plans. In our case, though, this does not appear feasible. A mapping of partial plans to sets of propositions would force us to create distinctions (for instance, in unique identifiers for steps) that would fail to preserve isomorphism characteristics.

Like the assumption-based approach, the plan lattice structure facilitates the exploration of multiple consistent contexts simultaneously.[7] But contrary to the ATMS view, we are not interested in finding all solutions (the class of all plans that achieve the goal). Therefore, we can restrict the domain of assumption sets that need to be considered to those explicitly created as plan classes by the planning algorithm.

To recap, the scheme presented here offers two sources of benefits with respect to plan search efficiency. First, the dominance relation provides a major new class of nogoods which may potentially shrink the search space. Second, consolidation of the lattice through classification of partial plans takes advantage of dependencies that might be obscured by an interface with a propositional TMS.

VII. The Complexity of Subsumption

As for knowledge representation mechanisms, the key operation in plan classification is the computation of subsumption relations [Brachman and Levesque, 1984]. We saw above that classifying plans as they are generated minimizes the search space.[8] Conversely, a perfect dependency mechanism is in effect computing these subsumptions.

Unfortunately, nonlinear plan subsumption is NP-complete.[9] This is true even for plan classes derived exclusively from *addsttp* and *order* operators, that is, partial orders on steps. The exponential potential of subsumption lies in the combinatorial number of possible mappings between the steps of the two plan classes. If, however, we can specify the correspondences between steps (for example, which *put-on* in $\mathcal{P}_1$ corresponds to which in $\mathcal{P}_2$), then subsumption is at worst quadratic. In practice we will generally not have complete correspondences, but typically the possible mappings between steps will be highly constrained. Actions may map only to others of the same type, therefore the computation will not be prohibitive as long as plan classes do not contain many steps of a single type. Codesignation constraints also help to restrict the possible mappings, as do explicit identifications among steps which may be provided by the planner when introducing steps in several partial plans at once. Finally, we might consider restricting the constraint language so that subsumption is tractable, perhaps to tree-shaped partial orders.

The effect on subsumption complexity of proposed extensions to the constraint language should also be considered. For example, we could allow actions themselves to be expressed at multiple levels of abstraction (as in the sequence *low-dose steroid therapy* is-a *steroid therapy* is-a *drug therapy)* without significant cost in complexity, as long as action subsumption itself is not expensive.[10] Extending the plan modification operators to include union and intersection, as suggested by Figure 3, is not as benign. Intersection (conjunction) presents no problem because the plan modification operators associate and commute, but computing subsumption among classes that are unions (disjunctions) of other classes appears substantially more difficult.

VIII. Summary

In this paper I have introduced and applied some analytical tools for studying and designing constraint-posting planners. The main components of this framework are

[7] This contrasts with the dependency-directed backtracking employed by TWEAK. The design of SCHEMER [Zabih, 1987, Zabih *tt* of., 1M7]—a dependency-directed interpreter for a non-deterministic LISP—illustrates this difference and highlights the issues in constructing a propositional interface for arbitrary dependencies.

[8] For true optimality we must also determine the most general plan class that is dominated. This corresponds to extracting the minimal nogood assumption set, which is not generally feasible. Note that minimality is with respect to a particular constraint language; slight changes may have dramatic effects on the dominance prover's ability to derive nogood sets at high levels of generality. For example, Pednault [Pednault, 1985] achieves stronger dominance results by including protected conditions as constraints on plans.

[9] The proof, by reduction from EXACT COVER BY 3-SETS, was provided by Ronald L. Rivest, personal communication.

[10] Given a static action lattice, action subsumption takes logarithmic time. If actions are described more flexibly—perhaps as dynamically generated KL-ONE concepts—then subsumption is more complex [Brachman and Levesque, 1984].

a specialisation lattice and a dominance relation defined over plan classes. These concepts were motivated with simple examples from MOLGEN and a more detailed account of TWEAK. The power of the TWEAK algorithm resides in a central dominance result which allows the planner to restrict attention to a small subset of the possible completions of a partial plan. Explicit characterisation of the dominance relation allows the planner to recognise that certain plan classes need not be explored, even though they might contain a valid plan. Although standard dependency-directed backtracking methods improve search efficiency, the only way to ensure complete lack of redundancy is to classify the partial plans in the lattice by computing subsumption among plan classes. This problem is NP-complete for nonlinear planning, but constraints commonly arising in practice may render the computation tractable. Recognising the centrality of subsumption suggests a novel approach to analysing the complexity implications of plan constraint languages.

The true test of the scheme presented here will be how well it supports tasks that require planning in the presence of uncertainty and partially satisfiable objectives. I am currently applying this framework to the design of a planner for the task of formulating decision models from a large medical knowledge base. The domain modeling language and dominance prover proposed for this task are based on ongoing work on qualitative influence networks [Wellman, 1987a, Wellman, 1987b].

Acknowledgments

Discussions with Zak Kohane, Ramesh Patil, Ron Rivest, Elisha Sacks, Peter Szolovits, Kate Unrath, Tom Wu, and Ramin Zabih contributed to the style and content of this paper.

References

[Brachman and Levesque, 1984] Ronald J. Brachman and Hector J. Levesque. The treatability of subsumption in frame-based description languages. In *Proceedings of the National Conference on Artificial Intelligence,* pages 34-37, American Association for Artificial Intelligence, 1984.

[Chapman, 1985a] David Chapman. Nonlinear planning: A rigorous reconstruction. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* pages 1022-1024, 1985.

[Chapman, 1985b] David Chapman. *Planning for conjunctive goals.* AI-TR 802, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA, 02139, November 1985. Revised version to appear in *Artificial Intelligence.*

[de Kleer, 1980a] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence,* 28:127-182, 1986.

[de Kleer, 1986b] Johan de Kleer. Problem solving with the ATMS. *Artificial Intelligence,* 28:197-224,1986.

[McAllester, 1982] David Allen McAUester. *Reasoning Utility Package User's Manual.* AIM 667, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA, 02139, 1982.

[McCarthy and Hayes, 1969] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4,* pages 463 - 502, Edinburgh University Press, 1969.

(Pednault, 1985] Edwin P. D. Pednault. *Preliminary report on a theory of plan synthesis.* Technical Note 358, SRI Artificial Intelligence Center, August 1985.

[Schmolse and Lipkis, 1983] James G. Schmolse and Thomas A. Lipkis. Classification in the KL-ONE knowledge representation system. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence,* pages 330-332, 1983.

[Stefik, 1981] Mark Stefik. Planning with constraints (MOLGEN: part 1). *Artificial Intelligence,* 16(2):111-140, 1981.

[Swartout and Neches, 1986] William Swartout and Robert Neches. The shifting terminological space: An impediment to evolvability. In *Proceedings of the National Conference on Artificial Intelligence,* pages 936-941, American Association for Artificial Intelligence, 1986.

[Wellman, 1987a] Michael P. Wellman. Probabilistic semantics for qualitative influences. In *Proceedings of the National Conference on Artificial Intelligence,* American Association for Artificial Intelligence, 1987.

[Wellman, 1987b] Michael P. Wellman. Qualitative probabilistic networks for planning under uncertainty. In John F. Lemmer, editor, *Uncertainty in Artificial Intelligence,* North-Holland, 1987.

[Zabih, 1987] Ramin Zabih. *Dependency-Directed Backtracking in Non-Deterministic Scheme.* Master's thesis, Massachussetts Institute of Technology, Cambridge, MA, January 1987.

[Zabih *et* a/., 1987] Ramin Zabih, David McAUester, and David Chapman. Non-deterministic lisp with dependency-directed backtracking. In *Proceedings of the National Conference on Artificial Intelligence,* American Association for Artificial Intelligence, 1987.