LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-332

FORMULATION OF TRADEOFFS IN PLANNING UNDER UNCERTAINTY

Michael P. Wellman

June 1987

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Formulation of Tradeoffs in Planning Under Uncertainty

A proposal for doctoral research at the Massachusetts Institute of Technology

Michael P. Wellman

June 1987

Abstract

Planning under uncertainty with multiple, competing objectives is impossible when goals are represented as predicates and the effects of actions are modeled as deterministic functions of situations. Decision-theoretic models, on the other hand, do not address the problem of constructing strategies from more primitive representations of actions. In this proposal, I describe a method for formulating plans from large knowledge bases that can accommodate uncertain and partial satisfaction of goals. At the core of the planner is a dominance prover that derives admissibility properties of plan classes. The representation for the effects of actions is based on a qualitative formalism for asserting influences among variables. The planner makes decisions "up to tradeoffs," an intuitive description that seems to characterize the power of a dominance prover based on the qualitative influence formalism.

Thesis Supervisor: Peter Szolovits

Keywords: planning, decision models, qualitative reasoning, uncertainty, knowledge representation, dominance proving

This work was supported (in part) by National Institutes of Health Grant No. R01 LM04493 from the National Library of Medicine.

Contents

Chapter 1

Introduction

1.1 Overview

This document proposes a program of research to develop theory and methodology to support a new model of planning under uncertainty. The new model generalizes the prevailing planning paradigm by replacing goal satisfaction with expected utility maximization. Under this choice criterion planning resembles the formulation and analysis of decision models.

The proposed planner converges on a strategy of action via a cycle that alternates between the following steps:

1. Construct and refine a domain model to predict and evaluate the effects of candidate plans.

2. Use the model to discover and propagate dominance relations among classes of plans.

The method is developed within a planning model that abstracts from the particular criteria used to choose among plans. By casting previous research in the general framework of searching the plan space with a dominance prover, we see how to salvage some of the existing principles, representations, and techniques even in an environment of uncertainty and partially satisfiable objectives. In particular, theoretical

results constraining the possible completions of partial plans may be incorporated directly into the dominance prover.

Feasibility of the approach is explored by designing a dominance prover and knowledge representations to support formulation of decision models from a large medical knowledge base. The size of the knowledge base places stringent modularity requirements on the representation, because we are not allowed to assume a narrow decision context. The domains of interest may be an order of magnitude wider in scope than today's expert systems. Complete decision models—or even large fragments—tend to be appropriate for only small classes of problems. Therefore, knowledge about the effects of actions and associations among events must be encoded at a fine level of granularity to satisfy these modularity requirements.

Actions and events in the knowledge base (as well as plans being constructed) exist at multiple levels of abstraction. The refinement of plan classes and world models that make up the planning cycle is directed along the axes of specialization defining these levels. Statements about plans at high levels of abstraction correspond to meta-planning rules that have been proposed by AI researchers.

The representation for the effects of actions and probabilistic relations among events is based on continuing work on qualitative probabilistic networks [57,58]. Models that describe only qualitative influences among events and actions provide great modularity advantages because while the precise magnitude of probabilistic relationships varies significantly with context, the direction of influence is often constant. Furthermore, the property of "impossible to resolve via qualitative influences alone" is perhaps the best formal definition available for a tradeoff situation.

This work stops at formulation of tradeoffs because tradeoffs provide a sharp competence boundary for the planner that serves to limit the scope of this project. Nevertheless, the issue of resolving tradeoffs is important for connecting this work to a larger body of complementary research in decision theory and artificial intelligence. The formal device of a "tradeoff oracle" demonstrates the application of resolved tradeoffs in succeeding cycles of plan formulation.

Often, tradeoff resolutions are implicit in knowledge available to the program or follow from heuristic assumptions. For example, assuming the optimality of the

current treatment plan can drastically limit the space of new plans that need to be considered upon an incremental change in information. The fact that a particular course of action was already deemed best implicitly determines some tradeoffs that might not have been directly resolvable by the planner. The role of this and other assumptions (for example, "common practice axioms") in plan formulation need to be explored further.

The remainder of this proposal develops these ideas in greater detail. Sections 1.2 and 1.3 below respectively describe the problem of planning under uncertainty and discuss related work on the problem. Those anxious to get to the "meat" would do well to skip directly to Section 1.4, a guide to the remainder of this proposal.

1.2 Planning Under Uncertainty

AI robot planners solve problems by searching for a plan of action guaranteed to transform the initial situation into one that satisfies some goal. Robots taking this approach in the real world are likely to be defeated for two primary (and innumerable secondary) reasons:

1. Knowledge of the world is imperfect; in practice it is not possible to guarantee much about the result of performing actions in a given situation.

2. Predicates on world states cannot express reasonable goals for real-world agents.

Let us examine each of these in turn.

Categorical planners have the luxury of deductive inference. While it is impossible to derive everything true about the result of applying an action in a situation, many logical consequences of pre- and post-conditions can be easily derived. Valid plans succeed with certainty, though robots do not always find valid plans—whether or not they exist. Figure 1.1 diagrams this planning paradigm. I adopt terminology and notation from the *situation calculus* of McCarthy and Hayes [33] because most work on planning fits into their basic framework.

$$s_i \equiv \text{initial situation}$$

$$s_f = result(robot, \pi, s_i)$$

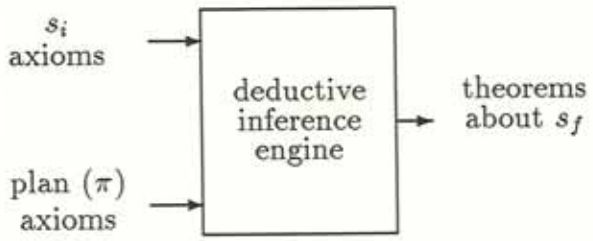


Figure 1.1: Categorical planning. Situations and actions are described by axioms. The task is to find a π that necessarily achieves the goal predicate G in the final situation s_f.

As soon as we admit that the world is uncertain—at least from any robot's perspective—our picture is altered dramatically. Now, characteristics of the situation resulting from executing a plan can be predicted only probabilistically. We see in Figure 1.2 that a *probabilistic model* is required to relate a given plan to the likelihood that the goal is satisfied after its execution.

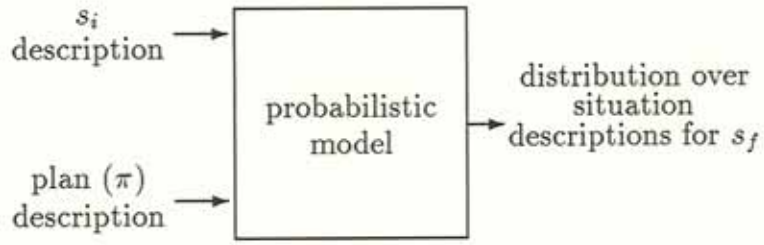


Figure 1.2: Planning under uncertainty. A probabilistic model relates plans to outcomes. There are at least two possible tasks: (1) Find a π such that $\Pr(G(s_f)) \geq p_{threshold}$ (satisficing), or (2) Compute $\pi = \arg\max \Pr(G(s_f))$ (optimizing).

While research on categorical planning has produced representations that associate all knowledge with individual actions, such encodings are more difficult to create for the uncertain case. This is primarily a consequence of non-modularity inherent in probabilistic models [21].

But for the nonce, let us assume that we can overcome these difficulties and provide our problem solver with probabilistic models relating plans to goal satisfaction likelihoods. The planner's task reduces to optimization: finding the plan with the greatest probability of success. This problem has a different flavor from the categorical planning problem and undoubtedly calls for modified techniques. Unfortunately, the real situation is worse. Rarely will the robot limit its cares to the probability of a particular goal predicate being satisfied in the plan result state. Instead, the degree to which a goal is satisfied may vary, or several objectives may be achieved in partial measure. In this sense goal predicates are inadequate for expressing robot desires (reason 2 above).

While it may have appeared possible to salvage much of the basic planning paradigm while admitting uncertainty, the deficiency of goal predicates reaches down to the fundamental structure of the methodology. Some actual planners have tried to patch this hole in the framework by including heuristic rules to handle anticipated planning decisions. McDermott's NASL [34], for example, uses *choice rules* to arbitrate among alternative task reduction paths which arise in planning. He recognizes that such an approach is vulnerable to harmful interactions with other eventualities in the planning environment, as is any scheme that associates actions directly with situations. The only way to cope with unanticipated choice contexts is to explicitly predict the effects of actions and evaluate the outcomes with a more general decision criterion.

At this point we might reconsider whether it makes sense to adopt the planning paradigm at all. Indeed, it may seem that I have been setting up a straw man all along; it should have been obvious from the start that traditional planning methods are not up to the general task of planning under uncertainty with multiple objectives.

A decision criterion of the generality required is provided by Bayesian decision theory [43]. Decision theory replaces the goal predicate with a utility function mapping

outcomes to real numbers (called *utilities*) and prescribes maximization of expected utility for decision making. However, decision theory is silent about forming models from knowledge bases and generating utility functions appropriate for different choices. Charniak and McDermott [8, page 523] say the following about decision theory and planning:

> One might think that an elegant theory of this kind would have been assimilated into robot planners, but this has so far not been the case. The issues addressed by the two approaches are complementary. Planning research has focused on how plans are *constructed*; decision theory has focused on how they are *evaluated*.

That the methodologies are complementary does not imply that they are independent nor that they can be combined straightforwardly. There is no reason to expect that techniques for constructing plans that work well under a goal-satisfaction criterion will enjoy similar success under expected utility maximization. And it is not surprising that decision theory does not answer the construction problem, because any solution must depend entirely on the form and content of the knowledge bases available to the robot. To date, decision-theoretic approaches have only been applied using representations that directly encode probabilistic models and utility functions. In other words, the decision model *is* the knowledge representation.

Unfortunately, decision models do not make good knowledge representations for robot planners that are expected to work over a broad range of problems. The kind of knowledge bases we are most concerned with are significantly wider in scope than those of today's expert systems, perhaps on the order of Lenat's CYC project [28]. A model covering more than a very narrow body of decision contexts is a poor one for any *particular* planning problem because the extraneous features considered tend to entail an unnecessary information-gathering burden and to obscure explanations of the result. General models cannot take advantage of simplifying features that—while present in any given decision problem—vary from case to case. Taking this inadequacy as a premise, our problem is to build a planner that can construct decision models from more reasonable knowledge representations. In the process, it will be necessary

to design reasonable representations capable of expressing the information necessary to build good decision models.

Perhaps surprisingly, the decision-theoretic planner I am proposing borrows much of the high-level structure of traditional categorical planners. Specifically, the view of planning as theorem proving is retained. (This is perhaps the most important single contribution of AI to the theory of robot planning.) The primary difference is that the object of planning under uncertainty is to prove optimality properties of a plan, rather than that it achieves the given goal. Naturally, it will not always be possible to find a unique plan that provably maximizes expected utility. Instead, the planner derives properties of the class of potentially optimal plans.

The theorem-proving architecture regards probabilities and utilities as objects to be reasoned about.[1] The resulting plan classes are derived from partial decision models. Details of the modeling and plan languages are provided in the body of the proposal.

1.3 Related Work

Although much work has been performed on the problem of planning under uncertainty, little of it bears any resemblance to the method described here. In this section, I examine these past efforts to illustrate the motivations for the proposed approach and to provide a basis for comparison of the techniques developed.

1.3.1 Engineering Away Uncertainty

The most common approach for handling uncertainty in planning is to remove explicit reference to likelihood by engineering around the uncertainty. While many sorts of evasive tactics have been tried, two major approaches stand out. First, it is possible to incorporate uncertainty directly into the categorical planning framework by simply weakening the conclusions drawn from actions. For example, in the framework for

[1] For a more general discussion of such an approach, and an elaboration of some of the arguments of this section, see Wellman and Heckerman [59].

robot motion planning advocated by Lozano-Pérez et al. [30], all of the uncertainty is captured in error tolerances added to intended motions. A planner may be able to find a plan for which it can prove that, say, the peg will end up in the hole so long as all motions are within their specified tolerances. Unfortunately, this methodology says nothing about what to do if no such plan exists or how to choose among several satisfactory plans. And, of course, not all uncertainty can be reasonably captured in something like tolerance.

The second important method for finessing the uncertainty problem is *execution monitoring.* Under an extreme version of this approach, the robot plans as though the effect of actions were known with certainty, then replans when the observed effects differ from the expected. This "trial and error" technique is obviously limited to domains with acceptable error costs. While it may be appropriate for some facets of robot assembly tasks, execution monitoring is inadequate when actions such as *transplant-kidney* are available.

1.3.2 Planning with Decision Theory

Feldman and Sproull [15] argue for the use of decision theory in robot planning by demonstrating that a few simple techniques based on numeric utilities and probabilities can enhance the performance of a robot facing an extension of the famous monkey and bananas problem. The methods provided for illustration, however, are specific to unreasonably simple decision models and planning algorithms. For example, their branch-and-bound search pruning strongly depends on utility functions that are additive on steps in the plan and on a planning procedure that assembles plans by sequentially adjoining steps. While it is undoubtedly possible to generalize and extend their methods somewhat, the authors provide little guidance for doing so. The "hungry monkey" serves as evidence of the limitations of ignoring uncertainty and tradeoffs but falls far short of a general framework for incorporating decision-theoretic criteria in robot planners.

An architecture for planning with decision theory is proposed by Langlotz et al. [27] and illustrated by ONYX, a program for planning cancer therapy. ONYX fits the

view of Charniak and McDermott noted above in that it clearly separates the plan construction and evaluation phases. As shown in Figure 1.3, the generation module forms candidate plans from a domain level description of the problem. The candidates are then pumped through first a probabilistic model to predict the effect of the plans, then a utility model to complete the decision-theoretic evaluation.

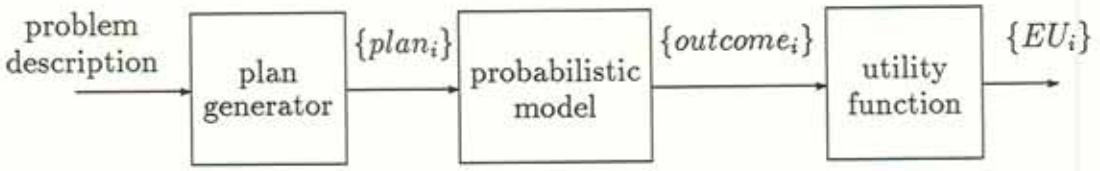


Figure 1.3: The architecture of ONYX.

The major limitation of this architecture lies precisely in its detachment of plan construction and evaluation. Within this framework knowledge used to evaluate strategies can play no role in generating the strategies in the first place. And the model for predicting the outcome of plans cannot be tailored to specific choice sets. These unfortunate separations are necessitated by the incompatibility of the knowledge representations employed in the respective modules. As a consequence, the system embodies considerable redundancy. Further, modifications or extensions to any of the modules generally require corresponding changes to all others. For the program to work at all, the probabilistic model must be able to generate outcome distributions for any plan, and these outcomes must be described in terms of attributes understood by the utility model. For the program to work well, the knowledge implicit in the plan generator must keep pace with the knowledge reflected in the probabilistic model. In this sense, the system contains *two* knowledge acquisition bottlenecks.

As argued elsewhere, a probabilistic model is an unreasonably limited representation for knowledge about the effects of actions. The ONYX architecture may be suitable for problems of the size undertaken by current expert systems but should not be considered a model for planners that have more comprehensive knowledge bases.

1.3.3 Formulating Decision Models

A few projects have specifically addressed themselves to knowledge-based formulation of decision models. In developing specifications for a new formulator, it is instructive to examine the shortcomings of previous attempts.

Rutherford et al. [40] describe a program for Hodgkins disease that is more flexible than typical decision analysis systems because it separates the decision tree interpreter (inference engine) from the specification of diagnostic tests and treatments (knowledge base). A decision model is dynamically constructed and evaluated by the program at consultation time. The flexibility lies in the possibility of modifying the set of available tests and treatments on a case-specific basis.

Hollenberg's Decision Tree Builder (DTB) [22] also generates decision trees using a medical knowledge base of diseases and interventions (tests and treatments). Unlike the Hodgkins program, DTB is intended to handle a broad range of medical decision problems. Consequently, its representations are considerably more general, and its tree generation, correspondingly more flexible. A disease may be parameterized by *attributes*, which in turn may influence the applicability of various interventions as well as probabilities and utilities in the model. Tests and treatments may indicate or modify the values of disease attributes. A simple control structure directs tree construction, employing a model of patient states for bookkeeping purposes.

The problem with this kind of generation approach is that it is extremely difficult to avoid an exhaustive consideration of a combinatorial space of plans and events. The programs must construct strategies that include every action identified as potentially beneficial, and model every event identified as potentially relevant. It is only possible to rule out subsets of the syntactic combinations by identifying avoidable patterns ahead of time or by applying generally unjustified heuristics, such as "do not consider strategies with three or more major surgeries."

In Holtzman's "intelligent decision systems" [23], the domain knowledge is primarily in the form of a general decision model, encoded as an influence diagram with assessment functions for each node of the graph. Constructing a model for a particular decision is largely a matter of reducing the template model that is built

into the program. Note the contrast between this and the tree building approach described above, where models are constructed by combining primitive components in a controlled fashion. The combinatorics of unconstrained generation are avoided at the cost of limiting the applicability of the knowledge base to a well-defined class of decisions.

1.4 Guide to the Proposal

We begin in Chapter 2 with a description of the abstract model of planning with a dominance prover.[2] The planner maintains a lattice of plan classes formed by posting constraints on the set of all plans. A dominance relation is introduced to characterize the admissibility of plan classes. Dominance results are spread throughout the lattice based on some simple propagation rules. The lattice is consolidated by classifying the plans as they are introduced, in the same way that concepts are classified in KL-ONE [44]. This operation highlights the role of subsumption computation in this variety of planning. The structures are illustrated by a recasting of Chapman's TWEAK [7] in this framework.

The rest of the proposal describes the design of SUDO-PLANNER (SUbsumption- and Dominance-Oriented Planner), a particular instance of the planning framework described in Chapter 2. Chapter 3 presents SUDO-PLANNER's languages for plans and plan classes and some features of its knowledge representations for actions and events. A partial algorithm for computing subsumption among plan classes is presented and analyzed. Some central design decisions in the encoding of actions and events are discussed, with special attention to the utility of representations at multiple levels of abstraction.

Chapter 4 provides a more detailed description of the representation for the effects of actions and relations among events, based on qualitative influences. Qualitative influences constitute the basic probabilistic modeling facility of the knowledge representation, providing a decision-theoretic basis for SUDO-PLANNER's dominance-proving

[2]Chapter 2 is a slightly revised version of [56].

component. I show how the definition for qualitative influences follows from basic computational desiderata, and consider other consequences of the representation.

Some loose ends in the specification of SUDO-PLANNER are tied together in Chapter 5. Properties of the dominance prover, input module, and mechanisms for focus of attention are explored.

Chapter 6 illustrates the preceding ideas with a detailed planning scenario for a real medical decision example. A list of outstanding tasks and a set of evaluation criteria make up the research plan of Chapter 7.

Chapter 2

Planning and Dominance Proving

The plan formulation process as envisioned here consists of a successive refinement of the set of candidate plans. In this chapter, I discuss a framework for planning with partially satisfiable objectives that integrates a dominance prover into the formulation process. The framework is presented here in its most abstract form; the particular representations and dominance proving techniques employed by SUDO-PLANNER are developed in subsequent chapters.

2.1 The Plan Lattice

Let $\mathcal{L}$ 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 $\mathcal{L} = A^*$ 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, $\mathcal{P} \subseteq \mathcal{L}$. $\mathcal{P}$ is also called a *partial plan* when it is represented as a set of constraints which incompletely specifies a plan we are concerned about.

We can view the planning process as one of adding constraints to candidate plan classes incrementally 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. But 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 2.1. The node in the graph marked "$A^*a_1A^*$" denotes the set of all plans with at least one instance of action a_1. The set of plans *starting* with a_1 forms a subclass, as does the set of plans with an a_1 followed by an a_2.

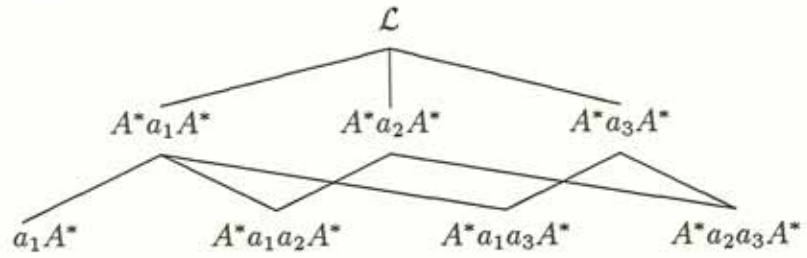


Figure 2.1: A plan specialization lattice.

The plan lattice representation of a search space supports a *constraint-posting* approach to planning. A constraint-posting planner—illustrated best by Stefik's MOLGEN [48]—can be more efficient than a planner that only evaluates 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 justifications 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.

2.2 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* [33], we write:

$$\pi_1 \succ \pi_2 \Leftrightarrow G(result(robot, \pi_1, s_i)) \wedge \neg G(result(robot, \pi_2, s_i)) \tag{2.1}$$

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 characterizes the choice criterion employed by the planner. A planner based on expected utility takes

$$\pi_1 \succ \pi_2 \Leftrightarrow E\,[u(\pi_1)] > E\,[u(\pi_2)] \tag{2.2}$$

The discussion of dominance that follows does not depend on any particular criterion for plan choice. Although we do need to 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 is preferred or indifferent. It should be emphasized 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:

$$D(\mathcal{P}_1, \mathcal{P}_2) \stackrel{\text{def}}{=} \forall \pi_2 \in \mathcal{P}_2 \; \exists \pi_1 \in \mathcal{P}_1 \; \pi_1 \succeq \pi_2 \tag{2.3}$$

The strict version of dominance, D', is defined similarly, except that here a particular plan in the first class is better than any in the second.[1]

$$D'(\mathcal{P}_1, \mathcal{P}_2) \stackrel{\text{def}}{=} \exists \hat{\pi}_1 \in \mathcal{P}_1 \; \forall \pi_2 \in \mathcal{P}_2 \; \hat{\pi}_1 \succ \pi_2 \tag{2.4}$$

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

$$D \text{ is reflexive, transitive, and complete.} \tag{2.5}$$

$$D' \text{ is anti-reflexive, transitive, and asymmetric.} \tag{2.6}$$

$$D(\mathcal{P}_1, \mathcal{P}_2) \Leftrightarrow \neg D'(\mathcal{P}_2, \mathcal{P}_1) \tag{2.7}$$

$$\mathcal{P}_2 \subseteq \mathcal{P}_1 \Rightarrow D(\mathcal{P}_1, \mathcal{P}_2) \tag{2.8}$$

$$D(\mathcal{P}_1, \mathcal{P}_2) \wedge D(\mathcal{P}_3, \mathcal{P}_4) \Rightarrow D(\mathcal{P}_1 \cup \mathcal{P}_3, \mathcal{P}_2 \cup \mathcal{P}_4) \tag{2.9}$$

$$D(\mathcal{P}_1, \mathcal{P}_3) \vee D(\mathcal{P}_2, \mathcal{P}_3) \Leftrightarrow D(\mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{P}_3) \tag{2.10}$$

$$D'(\mathcal{P}_1, \mathcal{P}_2) \wedge D(\mathcal{P}_2, \mathcal{P}_3) \Rightarrow D'(\mathcal{P}_1, \mathcal{P}_3) \tag{2.11}$$

These properties serve as dominance propagation rules within the plan lattice. By (2.8) and the transitivity of D, dominance by a particular class is inherited in the plan lattice. Strict dominance is also inherited, by (2.11) and weak dominance inheritance. Thus, markers or links indicating dominance relations need only be stored at the upper envelope of classes to which they apply. Dominance is propagated upwards in the lattice by application of the union properties (2.9 and 2.10), which also hold for D'.

[1] This difference is required by the possibility of infinite plan classes with no maximal elements. If (2.4) were exactly a strict version of (2.3), then such a class would strictly dominate itself. For the same reason, a definition of weak dominance which merely substituted $\succeq$ for $\succ$ in (2.4) would not entail the reflexive property. Assuming that every plan class has a maximal plan is unreasonable, even if it is appropriate to require that $\mathcal{L}$ does.

A plan class is *restricted* by asserting that it is weakly dominated by one of its subsets. In the MOLGEN example given above, if $\mathcal{P}_1$ is the class of plans that include the **screen** operation, and $\mathcal{P}_2$ is the subclass that includes the **resists** relation as well, then $D(\mathcal{P}_2, \mathcal{P}_1)$ 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

$$D_S(\mathcal{P}_1, \mathcal{P}_2) \stackrel{\text{def}}{=} S \Rightarrow D(\mathcal{P}_1, \mathcal{P}_2). \tag{2.12}$$

Normal dominance is just $D_{\textbf{true}}$. As an example of conditional dominance, suppose that we are uncertain about the identity of the organism of interest: it could be *bacterium-2* or *bacterium-3*. For $i = 2$ and 3, let S_i be the proposition "Bacterium-i is the organism of interest" and $\mathcal{P}_i$ the plan class that restricts $\mathcal{P}_1$ to those plans in which the **resists** relation holds between *antibiotic-1* and *bacterium-i*. Then we have $D_{S_2}(\mathcal{P}_2, \mathcal{P}_1)$, $D_{S_3}(\mathcal{P}_3, \mathcal{P}_1)$, and $S_2 \vee S_3$. By the definition of conditional dominance (2.12), we get $D(\mathcal{P}_2, \mathcal{P}_1) \vee D(\mathcal{P}_3, \mathcal{P}_1)$. Application of property 2.10 yields the result $D(\mathcal{P}_2 \cup \mathcal{P}_3, \mathcal{P}_1)$.

Of particular value are conditional dominance relations where S itself contains dominance assertions. For example, if $S' \equiv D(\mathcal{P}_1, \mathcal{L})$, then $D_{S'}(\mathcal{P}_2, \mathcal{P}_1)$ asserts that given the optimal plan is in $\mathcal{P}_1$ we can further confine attention to $\mathcal{P}_2$.[2] This is one way to derive the restrictions mentioned above. In fact, this is precisely the strategy employed by both Pednault [38] and Chapman [6] to limit the search space of their planners. We will see how this works for the latter example in Section 2.4 below.

Reasoning about conditional dominance can be implemented straightforwardly via any mechanism for reason maintenance [11,32]. The interesting task for the domi-

[2]In categorical planning, $D_{S'}$ is the same as D (that is, $D_{S'}(\mathcal{P}_1, \mathcal{P}_2) \Leftrightarrow D(\mathcal{P}_1, \mathcal{P}_2)$) due to the binary nature of the preference relation (2.1).

nance prover is to come up with meaningful conditions that imply useful dominance relations.

2.3 Planning

How do the plan lattice and dominance relations support planning? A planner operates on these structures 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. Strengthen conditional dominance relations.

We will say that a program performing these tasks is *planning*. It is important to note here 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 even an uncountable infinity of plans, this may seem like little progress. But if we can determine that all of them contain, for instance, an appendectomy, we solve a significant problem.[3]

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

[3]Plan classes with such 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 time the drug is given is any point in the 60-second interval.

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 dominance prover is the real work in designing a planner.

As a simple illustration, consider mathematical optimization techniques as planners from this perspective. 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 $\Re^n$ 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 which 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 2.5 and 2.6 below.

2.4 An Extended Illustration: Tweak

In the Introduction, I suggested that existing planning work can be recast in the framework described here. In this section, I examine Chapman's TWEAK [5,6], 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.[4] 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

[4] 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.

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 (2.1). 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:

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

Each returns the set of elements of $\mathcal{P}$ that satisfy the indicated constraint. *Addstep* constrains the partial plan to include an instance of step t. *Order* confines $\mathcal{P}$ 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 $\mathcal{P}$. The *codesignate* (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 ([5, page 1024], [6, page 11]) where the paths from start to end exhaust the possible sequences of plan modifications that can restrict a partial plan $\mathcal{P}$ to achieve a goal proposition p at step s. A completely analogous description in our functional notation is the expression appearing in Figure 2.2. This complicated expression, consisting of a combination of plan classes formed by various modifications on $\mathcal{P}$, represents the set of plans nondeterministically explored by Chapman's algorithm.

$$
\begin{array}{l}
\bigcap \bigcup_{t \in E} \bigcap order(addstep(\mathcal{P}, t), t, s) \\
\qquad \bigcup_{u \in conseq(t)} cod(addstep(\mathcal{P}, t), u, p) \\
\quad \bigcap_{c \in C} \bigcup order(\mathcal{P}, s, c) \\
\qquad \bigcap_{q \in conseq(c)} \bigcup ncod(\mathcal{P}, p, q) \\
\qquad\qquad \bigcup_{w \in W} \bigcap order(addstep(\mathcal{P}, w), c, w) \\
\qquad\qquad\qquad order(addstep(\mathcal{P}, w), w, s) \\
\qquad\qquad\qquad \bigcup_{r \in conseq(w)} ifcod(addstep(\mathcal{P}, w), p, q, r, p)
\end{array}
$$

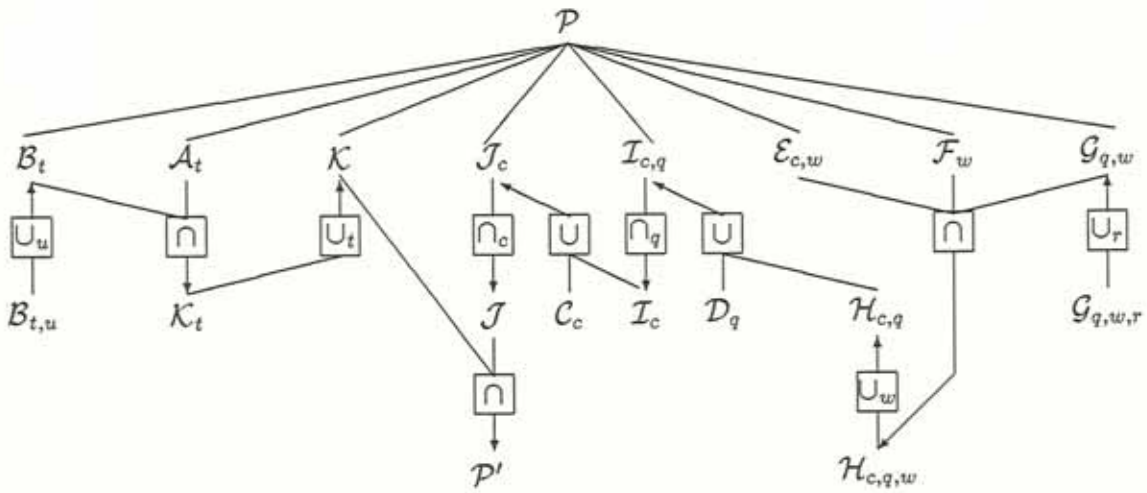
Figure 2.2: The TWEAK algorithm.

Let $\mathcal{P}'$ denote this plan class. In the figure, E is the set of *establishers*; that is, actions that can possibly assert the goal proposition. C and W are the potential *clobberers* (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. The fact that there is a plan in $\mathcal{P}'$ that achieves the goal if there is one in $\mathcal{P}$ can be captured by asserting $D(\mathcal{P}', \mathcal{P})$. This is a restriction, because each of the components of the $\mathcal{P}'$ expression is itself a subset of $\mathcal{P}$.

That's all there is to it, theoretically. TWEAK refines $\mathcal{P}$ by adding constraints until it finds a partial plan that necessarily achieves the goal (equivalently, a class provably containing only goal-achieving plans). How does the theoretical framework we have developed help to understand this process?

Each term in the expression of Figure 2.2 corresponds to a plan class which is a subset of $\mathcal{P}$. These distinguished plan classes serve as intermediate structure in

the plan lattice depicted in Figure 2.3. This lattice reflects a decomposition of the search space based on the TWEAK algorithm. The plan classes with subscripts denote parameterized classes; a complete diagram of this lattice would include, for instance, an $\mathcal{A}_t$ 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, $\mathcal{K}_t \stackrel{\text{def}}{=} \mathcal{B}_t \cap \mathcal{A}_t$ and $\mathcal{I}_c \stackrel{\text{def}}{=} \bigcap_q \mathcal{I}_{c,q}$. All connections indicate specialization of plan classes from higher to lower levels.



key:

$\mathcal{A}_t = order(addstep(\mathcal{P}, t), t, s)$
$\mathcal{B}_{t,u} = cod(addstep(\mathcal{P}, t), u, p)$
$\mathcal{C}_c = order(\mathcal{P}, s, c)$
$\mathcal{D}_q = ncod(\mathcal{P}, p, q)$
$\mathcal{E}_{c,w} = order(addstep(\mathcal{P}, w), c, w)$
$\mathcal{F}_w = order(addstep(\mathcal{P}, w), w, s)$
$\mathcal{G}_{q,w,r} = ifcod(addstep(\mathcal{P}, w), p, q, r, p)$

Figure 2.3: The plan lattice corresponding to the space searched by the TWEAK algorithm.

Because it would be difficult for the planner to generate a description for $\mathcal{P}'$ directly from the definition of Figure 2.2, 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 $\mathcal{P}'$ which necessarily achieves p at s. In the process, the algorithm recursively invokes itself, taking one of the intermediate classes as $\mathcal{P}$ and forming the

structure of Figure 2.3 beneath it.

2.5 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 properties 2.8 through 2.11 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 2.3. When partial plans are generated they are *classified* by finding their greatest lower and least upper bounds in the plan lattice.[5] 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 only consider contradictory plans (that is, empty plan classes) to be nogood; other pruning criteria must be built in to the control mechanism. The discussion above illustrates this by showing that though TWEAK's search procedure implicitly takes advantage of the fact $D(\mathcal{P}', \mathcal{P})$, 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 $\mathcal{D}_q$ only depends on q, even though its position in the expression for $\mathcal{P}'$ 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 $t = w$. However, it is doubtful that any of the standard dependency maintenance mechanisms would catch the more subtle relationships that hold

[5]The term *classified* is used here in the same sense that concepts are classified in KL-ONE [4].

between classes created on successive recursive invocations of the planning algorithm. A precise characterization of the sorts of redundancies that are avoidable is rarely offered in descriptions of planners.

Following the terminology used by de Kleer in describing his *assumption-based* scheme for truth maintenance [11], 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 [12] 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 (in unique identifiers for steps, for instance) that would fail to preserve isomorphism characteristics.

Like the assumption-based approach, the plan lattice structure facilitates the exploration of multiple consistent contexts simultaneously.[6] 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.

[6]This contrasts with the dependency-directed backtracking employed by TWEAK. The design of SCHEMER [63,64]—a dependency-directed interpreter for a non-deterministic LISP—illustrates this difference and highlights the issues in constructing a propositional interface for arbitrary dependencies.

2.6 The Complexity of Subsumption

As for knowledge representation mechanisms, the key operation in plan classification is the computation of subsumption relations [3]. We saw above that classifying plans as they are generated minimizes the search space.[7] Conversely, a perfect dependency mechanism is in effect computing these subsumptions.

Unfortunately, nonlinear plan subsumption is NP-complete.[8] This is true even for plan classes derived exclusively from *addstep* 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 not generally have complete correspondences, but typically the possible mappings between steps will be highly constrained. Actions may only map 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 that 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,

[7] 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 also that minimality is only 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 [38] achieves stronger dominance results by including protected conditions as constraints on plans.

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

as long as action subsumption itself is not expensive.[9] Extending the plan modification operators to include union and intersection, as suggested by Figure 2.3, is not as benign. Intersection (conjunction) presents no problem because the plan modification operators commute, but computing subsumption among classes that are unions (disjunctions) of other classes appears to be substantially more difficult.

2.7 Summary and Prospectus

In this chapter I have introduced and applied some analytical tools for studying and designing constraint-posting planners. The main components of this framework are a specialization 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 was shown to reside 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 characterization of the dominance relation allows the planner to recognize 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. Recognizing the centrality of subsumption suggests a novel approach to analyzing the complexity implications of plan constraint languages.

However, the real test of this point of view will be how well it supports tasks that require planning in the presence of uncertainty and partially satisfiable objectives. The remaining chapters develop a planner for the task of formulating decision models from a large knowledge base. The development follows the instantiation procedure mentioned on page 19: we define plan and constraint languages, knowledge repre-

[9] 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 [3].

sentations for the effect of actions and the preference relation, and a procedure for determining and propagating dominance relations among classes of candidate plans.

Chapter 3

Knowledge Representations

In this section I introduce the representations and mechanisms employed in instantiating the basic planning framework for the formulation of decision models. Remember that this constitutes only a particular instantiation; limitations of the particular structures presented here are not necessarily inherent in the framework of Chapter 2. My goal here is to demonstrate novel capabilities on a class of interesting tasks, not to produce a universally adequate planning engine.

3.1 Plan Language

A plan consists of a collection of actions with conditionals. Actions—more generally, sub-plans—may be conditioned on specially designated predicates called *observables*. The plan language is roughly defined by this simple grammar:

$$\text{PLAN} \longrightarrow \epsilon \mid \text{PLAN ACTION} \mid \text{PLAN (if OBS (PLAN) (PLAN))}$$

Although this production rule produces sequences of actions, plans with the same actions in different orders are considered identical. The only ordering constraint is that actions creating observables must not be conditional on the observables they create (see Section 4.4). Of course, the lack of even simple sequencing makes this an

extremely impoverished plan language—we cannot even hope to stack blocks![1]

It is clear from this description that many of the difficult representational issues are pushed off to the specification for actions. Section 3.2 begins a discussion of these.

3.2 Actions

The knowledge base of actions will be implemented in NIKL [52], a terminological knowledge representation language based on KL-ONE [4]. Classes of actions are NIKL *concepts*, defined by their position in the concept lattice and the structure of their *roles*, or associated features.[2]

The action representation influences two components of the specification for the planner. The mechanisms for abstracting classes of actions partially define the language of constraints on plan classes. And the facilities for representing the effects of actions account for much of the world model part of the planning system.

3.2.1 Levels of Abstraction of Actions and Plans

We deal first with the constraint language. The NIKL lattice of action concepts specifies a multilevel description of actions that may be included in plans. Let $\mathcal{A}$ be the class of all actions, also called the *universal plan class.* In the NIKL concept lattice, $\mathcal{A}$ is a superconcept of every other action concept $\mathcal{A}_i$. Clearly, one way to specialize partial plans in the plan lattice is to specialize individual actions included in that plan class. If $\mathcal{P}_1$ includes action $\mathcal{A}_1 \stackrel{def}{=}$ "give drug," the class of plans $\mathcal{P}_{1D}$ which replace $\mathcal{A}_1$ with $\mathcal{A}_D \stackrel{def}{=}$ "give drug D" is a specialization.

[1] Some would argue that without sequencing we should not even call this activity planning. My view is that the essence of planning is the assembly of separately described actions and whether the result is a sequence or just a collection is secondary. Labels aside, I hope to demonstrate that progress on some other planning issues is orthogonal to temporal representation. Further discussion of this design decision appears in Section 3.5 below.

[2] For alternate plan language representations implemented in NIKL, see Swartout and Neches [50] and Macaisa and Sidner [31]. In both of these, plans as well as actions are represented as NIKL concepts.

The dimension of specialization in the example above corresponded to the drug role of action $\mathcal{A}_1$. Further specializations might specify the drug (perhaps D is a family of drugs) or may include other information, such as dosage or method of administration.[3] In a realistic knowledge base the number of axes of specialization would be enormous.

This abstraction scheme is used in MOLGEN [48]. Although MOLGEN has a fairly simple two-level hierarchy of actions (lab operators), the combination of these operators with the hierarchy of lab objects in their domain forms a rich, multiply hierarchical action structure. In this view the lab objects are merely further axes for specializing the lab operators.

As Tenenberg points out [51], this form of abstraction is orthogonal to the step-components type of hierarchy used in NOAH [42]. In NOAH actions may be decomposed into sequences of lower-level actions, effectively viewing each action as a sub-plan. Actions are associated in this fashion in a "part-of" rather than an "is-a" relation. In terms of the plan specialization lattice, we can mimic this behavior by first creating plan classes that by assertion achieve particular subgoals and then searching for an instantiation of that class.

Another distinct kind of abstraction is the precondition hierarchy of plan spaces introduced by Sacerdoti in ABSTRIPS [41]. Ordering the preconditions imposes structure on the search space rather than on the actions or plans themselves.

Basing the knowledge representation on a terminological reasoner like NIKL has an important advantage. An action lattice based on NIKL definitions does not have to be a static predefined structure. New action classes may be created at planning time by adding constraints to, specifying new features of, or combining existing action classes. These new actions often can be automatically classified; that is, an algorithm can determine the precise place for the new class in the existing lattice. This is important because

1. an action's place in the lattice determines the properties it inherits from and

[3] As in planning, this specialization process never ends. Actions only exist in the knowledge base as classes; "action individuals" can be said to exist only if the planner really has end-effectors and an execution capability.

supplies to other action types, and

2. action subsumption is employed by the constraint language subsumption algorithm of Section 3.3 to generate the plan lattice.

As noted above, planning at abstract levels is a form of least-commitment strategy for improving efficiency. If we can rule out a plan (by proving it is dominated) based on properties of a high-level action class, we save the great deal of work required to explore the more specific variants of that action. One claim that I intend to defend in this thesis is that the advantages attributed to meta-planning rules proposed in the literature (see Wilensky, for example [61]) can be achieved by planning at sufficiently high levels of abstraction. Further discussion of this point here would be premature; I present it now as the motivation for paying so much attention to abstraction.

3.2.2 Effects of Actions

In terms of our planning system components, defining the representation for an action's effects serves to specify part of the world modeling mechanisms of the planner. In the planner, there are two types of effects that we need to capture. First and most obviously, actions may influence events. We would not plan if our actions could not change the world. Second, actions may create new observables. That is, as a result of performing an action the robot may be able to condition future actions on a particular event.

Representing the effect of actions on a world model is usually the hardest task in designing categorical planners. Much of the difficulty arises from the *frame problem* [33]—computational and notational complexity due to the necessity of describing the possible change in status of every proposition for each action. Actual planners circumvent these difficulties by applying some variant of an assumption first applied in STRIPS [16], thereby restricting attention to propositions explicitly mentioned in action specifications. Characterizing such policies in a formal logic has proven to be a difficult task for AI theorists [20].

With uncertainty, the solutions to these problems will be similar in both form

and adequacy. Actions are presumed to affect directly only those propositions explicitly referred to in the specifications. Of course, the uncertain nature of effects require representations quite different from those of categorical planners, and non-effects likewise require a probabilistic interpretation. While add and delete lists that specify the propositions changing truth value as a result of an action are sufficient for categorical planning, actions with uncertain effects must describe changes in probabilistic relations that occur when the action is performed. The corresponding STRIPS assumption for planning under uncertainty is that events not mentioned in an action's representation are assumed to be conditionally independent of that action, given the direct effects and the rest of the plan.

In general, the effect of actions will be conditional on the rest of the plan and some other uncertain events. Placing conditions on effect assertions is analogous to the use of preconditions on actions in traditional AI planning. Rather than saying an action cannot be performed unless the conditions are met, we could specify that it does not have particular effects (usually the desired ones) unless the conditions hold. Under this interpretation, the planner can apply *put-on*(a, b) in any situation, but the result $on(a, b)$ is conditional on *cleartop*(a) and *cleartop*(b). Otherwise, *put-on* is a no-op. By placing the conditions on effects rather than on the action, we can employ actions that may have many contingent effects. In addition, the planner is free to introduce such actions into the plan without guaranteeing that the preconditions are satisfied.

Because the mechanisms for describing and reasoning about probabilistic relations among actions and events is itself a major component of this research effort, I defer further discussion of the subject to Chapter 4. The purpose of this section has been to draw parallels between action representation issues in categorical and probabilistic planning. The important conclusion emerging thus far is simply that, as in categorical planning, an action's effects need to be associated locally with that action.

3.3 Plan Constraint Language

The plan constraint language defines the space of partial plans that can be represented.

3.3.1 Specification

The language for partial plans (P-PLANs) is somewhat more complicated than the plan language.

P-PLAN $\longrightarrow \epsilon$ | [ACTS, ACTS] | CP-PLAN | P-PLAN P-PLAN

CP-PLAN $\longrightarrow$ (if OBS (P-PLAN) (P-PLAN))

ACTS $\longrightarrow$ P-ACTION*

The pair of lists enclosed in brackets denote the actions unconditionally in and out of the plan, respectively. For example the plan class $\mathcal{P}$

$$\mathcal{P} = [\mathcal{A}_1\mathcal{A}_2\mathcal{A}_2\mathcal{A}_4, \mathcal{A}_3\mathcal{A}_4] \tag{3.1}$$

unconditionally contains at least one instance of action $\mathcal{A}_1$, at least two of $\mathcal{A}_2$, none of $\mathcal{A}_3$, and *exactly* one of $\mathcal{A}_4$. The interpretation is that the first string of actions (the IN actions) are asserted to be in the plan without conditions and the second string (of OUT actions) asserts that no further instances of those actions are allowed. Note that while the IN list may contain multiple copies of the same action or actions that subsume each other, such repetition is superfluous for OUT lists.

Partial plans embedded in conditionals specify further constraints on plans given the state of observables. Therefore the plan class expression

$$\mathcal{P} = [\mathcal{A}_1, \epsilon] \text{ (if } o_1 \text{ } ([\mathcal{A}_1\mathcal{A}_2, \epsilon]) \text{ } ([\mathcal{A}_1\mathcal{A}_3, \epsilon])) \tag{3.2}$$

specifies that $\mathcal{P}$ unconditionally contains an instance of $\mathcal{A}_1$ and includes in addition an $\mathcal{A}_2$ or an $\mathcal{A}_3$ depending on the state of o_1. Note that the plan classes specified within the *if* clause must be subclasses of the unconditional plan in the outer scope.

Note that although the P-ACTIONs may be action classes (as described in Section 3.2 above), all observables must be completely specified predicates.

3.3.2 Simplification and Subsumption

Partial plan descriptions can be simplified to facilitate subsumption computation. Rewriting plan classes to reveal common constraints within conditionals, for example,

decreases the amount of case analysis that need be performed by the subsumption algorithm.

Details of the simplification techniques are presented in Appendix A, which also includes the description of a sound but incomplete polynomial algorithm for deriving subsumption relations among simplified plan classes.

3.4 Events

The representational structures denoting events in the knowledge base will be very much like those for actions. Differences are in access to the planner and the types of relationships they may participate in. Unlike actions, the planner does not directly control events. Instead, events are influenced by other events and actions. Like actions, events are arranged in the knowledge base at multiple levels of abstraction. It is probably helpful to view events as structured predicates which can sometimes be ordered by generality.

Many details of these representations, including some central design considerations, are yet to be worked out. Capturing the relevant features of the world in these events may prove to be a monumental knowledge engineering task. We almost certainly will need some special facilities to represent non-propositional parameters, such as quantity variables, with mechanisms for mapping the parameters to the space of events they implicitly define.

A more immediate problem is to develop a mechanism for handling references to events within action and event roles. The reference scheme must handle changes in abstraction appropriately and must ensure that the right equivalences are established among events referenced from several places. This and other problems will be attacked more vigorously in the implementation stage of the project.

3.5 Time

No theory of planning is complete without an adequate treatment of temporal relations among actions and events. The plan representation I have described is atemporal

except for consistency-checking of observable sequencing. Failing to deal with time will be one of the major shortcomings of SUDO-PLANNER.

Most planners handle simple sequencing among actions. For the medical problems I am interested in, such a mechanism is not nearly adequate and barely provides an advantage over no temporal reasoning at all. Instead, I intend to address temporal issues as best as possible within the action and event representation without providing time a special role in the planner. For example, in patients presenting with symptoms of appendicitis, there is a tradeoff between performing an appendectomy immediately or waiting and observing the patient to better establish the diagnosis. Waiting avoids some unnecessary surgeries while increasing the risk of untreated disease. This decision can be represented by including a time-of-surgery role in the specification of the appendectomy action. The filler for that role is a time variable whose effects may be captured via the mechanisms described in Chapter 4.

But even a rich set of representation hacks like that described above is no substitute for an explicit uniform treatment of time within the planner and the semantics for effects of actions. Failing to recognize that the time-of-surgery role of appendectomy is related to other temporal measures in the plan will undoubtedly lead to missed inference opportunities and inability to identify some meaningful distinctions.

An interesting area for study where temporal issues are central is in describing the *outcome* of a plan. By outcome, I mean the result by which plans may be evaluated and compared. Elsewhere I have described a scheme for outcome representation which is organized around a taxonomy of temporal patterns [55]. I believe that this or a similar mechanism can be grounded in a temporal semantics like that of Allen [1]. Ideally, a dominance prover would support utility models that adequately address aspects of the temporal resolution of uncertainty [26]. Conditions corresponding to stochastic dominance of atemporal utility defined by Cox [10] may prove useful in this regard.

Chapter 4

The Effects of Actions

Consider an event space determined by a set of random variables. A *domain model* is a representation of the probabilistic relations among these random variables. With respect to domain models, actions are merely special kinds of events which the planner can directly control. To describe the effects of actions, we need to specify how the choice of particular values for the action variables influences the probability distribution over the remaining event variables. The knowledge representation issues we face are how to encode these specifications to provide for computational tractability, knowledge base integrity, and decision-making power. In this chapter I present a representation that addresses the first two objectives by maximizing modularity and supports the third by providing for decision-making "up to tradeoffs."

4.1 Modularity

In Section 3.2.2 I argued that as in categorical planning, the first requirement of a representation for the effects of actions is locality. Locality is the basis for heuristics for attacking the frame problem and supports modularity by delimiting the implications of modifications to the knowledge base. Modularity is an engineering concern for any knowledge-based system; under uncertainty the problem is magnified by the sensitivity of probabilistic relations to surrounding context [21].

4.1.1 Robustness to Modification

To illustrate one of the modularity problems that arise in probabilistic knowledge representations, I adduce an example from Cooper's NESTOR [9] that was used by Spiegelhalter [46] for another purpose. NESTOR's domain is hypercalcemia, hence the program includes a knowledge base relating a patient's calcium level to other physiological states and associated findings. In Spiegelhalter's model fragment, the unconditional or prior probability of coma is .05. To us non-physicians this value seems high; a much lower fraction of people we know are comatose. Of course, the number may be valid for the population treated by the program: patients identified somehow for a hypercalcemia work-up.

The point is that the knowledge base contains no definition for the population it is applicable to. Elsewhere, Spiegelhalter and Knill-Jones [47] discuss the issue of transportability of statistical knowledge bases, concluding that it remains a serious problem. A greater difficulty, in my view, is the limitation it imposes on the scope of any such knowledge base. Suppose we wished to extend NESTOR's domain by including medical knowledge from another program working in a neighboring or overlapping clinical area. Patil's ABEL [35], for example, models disorders of electrolytes other than calcium, and considers more deeply the acid/base issues relevant to hypercalcemia. In broadening the domain, we have no idea which parts of the hypercalcemia model remain valid and which must be changed in light of the modified population and new interacting variables. Although the same is strictly true of ABEL's knowledge—we cannot be perfectly confident that the electrolyte model is still correct when calcium is considered explicitly—the causal link structure of ABEL is more robust than the precise statistical relationships.

The robustness of causal links rests partly in their imprecision (weaker statements hold in more contexts and are therefore more modular), but also in that they capture a critical aspect of the domain knowledge. A causal model is deserving of the name if its surface representation reflects a theory or set of organizing principles underlying the domain. By this argument, the correspondence to demonstrably robust cognitive structures translates to a computational mechanism that is less sensitive to context

than an arbitrary selection of observed relationships among variables.[1]

4.1.2 Combinatorics of Expansion

Another important modularity problem associated with probabilistic knowledge is a potential for complexity explosion for incremental additions to the knowledge base. In a representation like Pearl's Bayes networks [36], introducing a new event e requires in general a reassessment of the probability distribution over all events not conditionally independent of e [21]. For n such events the complexity of this task is 2^n; adding an event doubles the difficulty.

Cooper's algorithm [9] avoids this problem by deriving weaker conclusions from the network representation. In NESTOR's knowledge base, only marginal influences are specified; interactions among events influencing a common variable are not described. The combination can therefore imply only bounds on the joint probability distribution. The bounds can be strengthened by NESTOR's assumption that multiple influences act at most independently (that is, non-synergistically).

4.2 Qualitative Influences

Modularity has been the prime motivation for restricting attention to qualitative influences. While exact probabilistic statements are invariably context-sensitive, the *direction* of the relationship among variables is often constant. Qualitative influences are abstractions of probabilistic relationships that preserve only this direction information.

The concept of qualitative influences was introduced in a previous paper [58]. In the sections below, I present the basic definitions, extending the formalism to handle multi-valued parameters as well as dichotomous events. The chosen definition for

[1] Ultimately, robustness of any representation is a matter for empirical confirmation. The argument here is based on an assumption that structure imposed on knowledge by humans is motivated by similar computational concerns. See McCarthy and Hayes [33] for an early discussion of computational reasons for some psychological structuring of the world.

qualitative influences is justified by a set of desiderata for the combining operations.

4.2.1 Example: The Digitalis Therapy Advisor

We start our discussion of qualitative influences with a simple causal model example taken from Swartout's program for digitalis therapy [49]. The model, shown in Figure 4.1, is a fragment of the knowledge base that Swartout used to re-implement the Digitalis Therapy Advisor [18] via an automatic programmer.

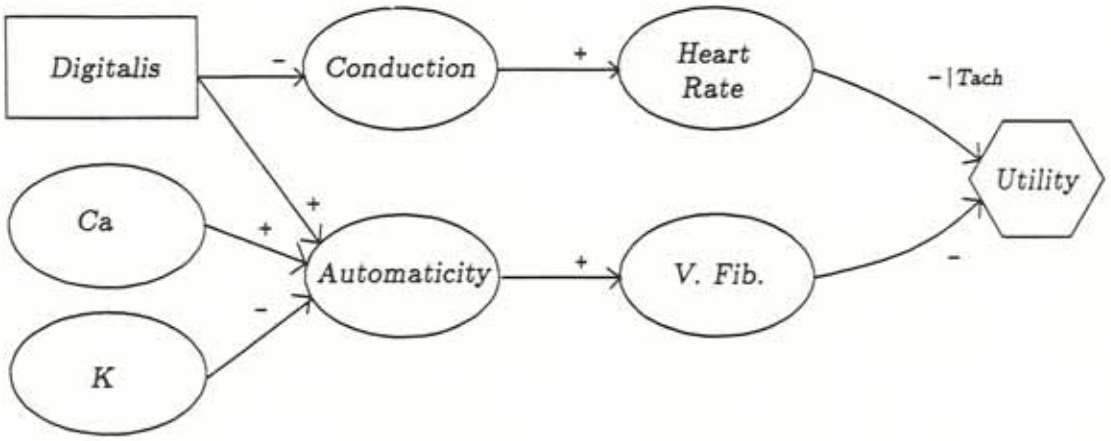


Figure 4.1: Part of the causal model for digitalis therapy. The direction on a link from a to b indicates the effect of an increase in a on b.

In the figure the elliptical nodes represent random variables. The rectangular node is a decision variable, in this case the dosage of digitalis administered to the patient. The hexagonal node is called the value node and represents the utility of the outcome to the patient. This terminology and notation are adapted from *influence diagrams* [45], a probabilistic modeling formalism similar to Bayes networks [36].

Influences among the variables are indicated by dependence links, annotated with a sign denoting the direction of influence. Thus digitalis negatively influences conduction and positively influences automaticity. In the remainder of this section I develop a semantics for these influence links that justifies the kinds of inferences we require from models similar to that of Figure 4.1.

4.2.2 Definitions

Informally, a qualitative influence is a statement of the form "event variable a makes event variable b more (less) likely." When a and b are dichotomous, this statement is easy to capture in a probabilistic assertion. Let A and $\bar{A}$ denote the assertions $a = \textbf{true}$ and $a = \textbf{false}$, respectively, and similarly for B and $\bar{B}$. Then we say "a positively influences b," written $S^+(a, b)$, if and only if

$$\forall x \ \Pr(B|Ax) \geq \Pr(B|\bar{A}x). \tag{4.1}$$

In the equation, x ranges over all assignments to the other event variables consistent with both A and $\bar{A}$. The quantification is necessary to assert that the influence holds in all contexts, not just marginally. Conditions analogous to (4.1) and those following serve to define negative and zero influences, omitted here for brevity.

Formalizing S^+ is not quite so straightforward when a and/or b can take on more than two values. In such cases we want to capture the idea that "higher values of a make higher values of b more likely." An obvious prerequisite for such statements is some interpretation of "higher." Therefore we require that each random variable be associated with a partial order $\geq$ on its values. For numeric variables such as "serum creatinine concentration" this relation has the usual interpretation; for scales like "mild/moderate/severe pain" an interpretation will have to be contrived. Although $\geq$ is a different relation for each variable, in the discussion below the notation is always disambiguated by context.

The more troublesome part of defining positive influences is in specifying what it means to "make higher values of b more likely." Intuitively, we want a statement that the probability distribution for b tends to be shifted toward higher values as a increases. To make such a statement we need an ordering which, given any two cumulative probability distributions G_1 and G_2 over b, determines whether G_1 is "higher" than G_2.[2]

[2]Technically, these cumulative distributions are only well-defined if $\geq$ is a total order. These statements as well as the definitions and conditions below can be extended to partial orders by requiring each to hold for any total order $\geq'$ that is a completion of $\geq$. For example, any statement

However, not all probability distributions can be easily ordered according to size of the random variable. Different rankings are obtained through comparing distributions by median, mean, or mean-log, for example. We require an ordering that is robust to changes of these measures, because the random variables are described by merely *ordinal* scales (see Krantz et al. for a definition [25]). In some cases they may be only partially ordered.

An ordering criterion with the robustness we desire is *first-order stochastic dominance* (FSD) [60]. FSD holds for G_1 over G_2 if and only if the mean of G_1 is greater than the mean of G_2 for *any* monotonic transform of b. That is, for all monotonically increasing functions ϕ,

$$\int \phi(b) dG_1(b) \geq \int \phi(b) dG_2(b). \tag{4.2}$$

A necessary and sufficient condition for (4.2) is

$$\forall b \; G_1(b) \leq G_2(b). \tag{4.3}$$

That is, for any given value of b the probability of obtaining b or less is smaller for G_1 than for G_2. For further discussion and a proof that (4.3) is equivalent to FSD, see Fishburn and Vickson [17].

We are now ready to define qualitative influences. Let $F(b|a_i x)$ be the cumulative density function (CDF) for b given $a = a_i$ and context x.[3] Then $S^+(a, b)$ if and only if

$$\forall a_1, a_2, x \quad a_1 \geq a_2 \Rightarrow F(b|a_1 x) \text{ FSD } F(b|a_2 x). \tag{4.4}$$

Adopting the convention for binary events that **true** > **false**, we see that (4.4) is a generalization of (4.1).

---

involving a cumulative distribution G must hold for all Gs induced by total orders consistent with $\geq$.

[3] As above, x is an assignment to the remaining random variables consistent with the condition $a = a_i$. We need to include x here and in the definitions below because these conditions will be applied in situations where x is partially or totally known. If we had stated the conditions in marginal terms ("on average, a positively influences b") it would not be valid to apply them in specific contexts.

Finally, we need a special definition for the desirability of influences among random variables. The variable a positively influences utility, $U^+(a)$, if and only if

$$\forall a_1, a_2, x \quad a_1 \geq a_2 \Rightarrow u(a_1, x) \geq u(a_2, x), \tag{4.5}$$

where u is a utility function [43] defined over the event space.

4.2.3 Justification for the Definitions

In a forthcoming paper [57], I demonstrate three propositions that provide theoretical support for the S^+ definition above.

Proposition 1 *If a and c are not connected by any direct links, $S^+(a, b)$, and $S^+(b, c)$, then $S^+(a, c)$ holds in the reduced network obtained by removing b from the original and recomputing influences.*

That is, qualitative influences are transitive. Chains of influences can be combined by sign multiplication.

Proposition 2 *Given the following conditions:*

1. *$S^+(a, b)$ is defined by (4.6), where R is some relation on CDFs:*

$$\forall a_1, a_2, x \quad a_1 \geq a_2 \Rightarrow F(b|a_1 x)\ R\ F(b|a_2 x). \tag{4.6}$$

2. *Proposition 1.*

3. *For binary b, $a_1 \geq a_2$, and x,*

$$F(b|a_1 x)\ R\ F(b|a_2 x) \Leftrightarrow \Pr(B|a_1 x) \geq \Pr(B|a_2 x). \tag{4.7}$$

the weakest R is FSD.

In other words, S^+ is the weakest condition on posteriors that generalizes the binary definition and guarantees chaining of influences. To extend the argument for S^+ to other possible (non-posterior) definitions, we must appeal to decision-making.

Proposition 3 *Consider the following conditions:*

1. *a and u are not connected by any direct links*
2. $U^+(b)$
3. $U^+(a)$ *in the network with b removed*

Conditions 1 and 2 imply 3 if and only if $S^+(a,b)$ *as defined by (4.4).*

Proposition 3 demonstrates that while conditions that are weaker than S^+ may be sufficient for propagating influences across chains, they are not adequate to justify decisions across chains.

4.2.4 Conditional Influences

Conditional influences—defined for binary events in a previous paper [58]—simply delimit the range of x in (4.4). The notation $S^{+|C}(a,b)$ means that a positively influences b given that proposition C holds. In terms of the definition above, $S^{+|C}(a,b)$ if and only if

$$\forall a_1, a_2, x \quad a_1 \geq a_2 \Rightarrow F(b|a_1 C x) \text{ FSD } F(b|a_2 C x), \tag{4.8}$$

where x now ranges over assignments to the variables consistent with $a = a_i$ and C.

With conditional influences, the effect of an action or event can be made to depend on other actions or events. As noted in Section 3.2.2, this mechanism corresponds to the use of preconditions in traditional AI planning.

4.2.5 Extensions

The basic definitions above can be extended in a variety of ways. Swartout's XPLAIN knowledge base included the "domain principle" that if a state variable acts synergistically with the drug to induce toxicity, then smaller doses should be given for higher observations of the variable [49]. This fact could be derived by a *domain-independent* inference procedure given a suitable definition for qualitative synergy. We can say that two variables synergistically influence a third if their joint influence is greater (in

the sense of FSD) than separate statistically independent influences.[4] However, in the domain Swartout's program was applied this solution is misleading because digitalis does not really act synergistically with Ca or K deviations in increasing automaticity. Rather, the dosage reduction is warranted by a greater-than-linear increase in toxicity (or disutility) for increases in automaticity. Such a relationship can be captured by an assertion that the utility function is concave in automaticity (in a sense that automaticity is synergistic with itself), given a cardinal scale for automaticity. Cardinality also permits other variations on qualitative influences which are beyond the scope of this proposal.

4.3 Effect Representation and the STRIPS Assumption

The effect of an action is represented by a collection of S^+ and S^- assertions, probabilistic versions of the add and delete lists of STRIPS [16]. Preconditions are captured by conditions on the influences. As stated in Section 3.2.2, the STRIPS-like assumption is that conditional independence holds for variable pairs not explicitly appearing in influences.[5] Conditional independence can be denoted by S^0, defined by substituting "=" for FSD in equation (4.4). We can override the assumption in particular cases without specifying an influence by asserting $S^?(a, b)$.

As Grosof [19] lucidly points out, application of the conditional independence assumption based on the absence of explicit influences is a form of non-monotonic reasoning. In the dynamic view of constructing and manipulating the network of

[4]This is the reverse of the relation assumed by Cooper's NESTOR [9] for combining common influences (discussed in Section 4.1.2).

[5]To define this assumption completely, we need to specify the set of conditioning events for the independence condition. In Pearl's terminology [37, page 34], events a and b are conditionally independent given any set of events that *d-separates* them in the directed graph formed by explicit influences. When a and b are connected by a direct influence there will be no such set. For a definition of d-separation and a more precise characterization of independence properties of probabilistic network formalisms, see Pearl's report.

influences, S^+ may be a non-monotonic predicate. Any formal interpretation of the normative status of the planner as a whole must take into account the process of assembling models from the knowledge base.

4.4 Creating Observables

In addition to influencing the value of event variables, actions can also cause events to become observable and therefore eligible to appear as a condition in a plan. Observable creation places a constraint on plans beyond the specification of Section 3.1; it must be possible to order the actions so that all observables are created before they are used as conditions. This constraint ensures that we are not implicitly conditioning on observation on the value of the observation itself.

I will refer to observable-creating actions as *tests*. This mechanism differs somewhat from the representation of information dependencies in influence diagrams [45]; there the observability of events does not change but the values of the observables are influenced by tests. One way to implement this (Ross Shachter, personal communication) is to make the observable o a deterministic function of an underlying physiologic state s and a binary variable indicating whether the test was performed. The result is $o = s$ if the test is performed, with o a noninformative constant otherwise. In the scheme proposed here, the test action directly causes s to be observable. The result is formally equivalent, but maintaining the distinction in the knowledge representation provides advantages when it comes to plan formulation and dominance proving.

Chapter 5

SUDO-Planner

5.1 Dominance Proving

The dominance prover is the only remaining component needed to instantiate the planning framework of Section 2.3. S^+ provides a basis for the domain modeling language, and U^+ can generate a partial preference order among plans. The ability to prove dominance among plans based on the semantics of action effects is the keystone of the proposed domain modeling language outlined in the two preceding chapters.

Techniques for deriving dominance from a qualitative probabilistic network representation were described and illustrated in a previous paper [58]. In a nutshell, the inference procedure reduces the network by a procedure analogous to Shachter's technique for evaluating quantitative influence diagrams [45]. The S^+ and U^+ relations in the reduced graph induce a partial order on the likelihoods and utilities in the model. Case analysis is then applied to determine that one plan is preferred to another by Savage's "sure-thing principle" [43, page 21]. Other techniques (for example, "hypothetical optimality" introduced in the earlier work [58]) may be applied to improve efficiency and enhance dominance-proving power in particular cases.

The generalization of S^+ to multi-valued parameters creates additional inference opportunities not considered in the previous paper. For example, in some models there exists a landmark value corresponding to the optimal setting of some action parameter, perhaps the ideal dosage of a drug or a threshold value of a test. The

dominance prover can deduce the effect of other actions and events on this optimal value. This is the reasoning process operating in the digitalis example above (Section 4.2.5), where the program determines that the dosage should be reduced in the presence of increased calcium.

A complete and formal specification of the dominance prover is beyond my capabilities in the current state of this research. One feature is clear, however: a planner with only qualitative influences will not be able to make choices involving real tradeoffs. The aim of further work will be to produce a dominance prover that makes "resolvable by qualitative influences" a good operational definition for "tradeoff."[1]

5.2 Input Specification of a Planning Problem

Now that we have all of SUDO-PLANNER's components, we are prepared to discuss the interface between the program and a user with a problem. In the most straightforward application of these planning mechanisms, the user would describe the state of the world via a domain model and set the planner off to construct a plan lattice to determine the set of admissible plans.

This usage mode, however, is bound to prove unsatisfactory for any real problem. A planner with a moderately large medical knowledge base would be obliged to consider the advisability of actions spanning the range from taking blood pressure to prophylactic heart transplants for every patient. We need a method that focuses attention without recklessly overlooking valid therapeutic opportunities.

The criterion I propose is a default assumption about the status quo plan in operation before SUDO-PLANNER is consulted.[2] Specifically, the planner assumes that the status quo plan is optimal given the previous state of information about the world.

[1] That dominance with respect to qualitative influences captures our intuitive notions of tradeoff cannot be demonstrated rigorously. In part to test this claim, I am currently collaborating with Alan Moskowitz, Thomas Wu, and Jerome Kassirer in a cognitive study of therapy planning by physicians. To date, we have collected and are in the process of analyzing protocols of physicians thinking aloud about solutions to the case presented in Chapter 6.

[2] For an autonomous planner, the status quo plan is the one being executed when this deliberation commences.

As input, SUDO-PLANNER takes an encoding of this plan along with a description of the observed changes in the world that prompted the call for re-planning. The planner uses the optimality assumption to restrict its attention to plan modifications that are justified by the information changes.

For example, suppose a patient who is not currently receiving any treatment presents with tooth decay. Given that a heart transplant was not considered a good idea before (it was not part of the status quo plan), it cannot now be recommended because there is no further indication for the surgery in the changed information state. Under the STRIPS-like assumption of Section 4.3, events and actions affect only those variables connected by qualitative influence paths. This should shrink the set of actions we need consider for any particular change in state.

Unfortunately, this scheme does not quite work. In our knowledge base there is an undirected path between any pair of nodes—if only because all eventually connect to the value node. Therefore, without further kinds of assertions or mechanisms our planner cannot always easily shield seemingly irrelevant actions from consideration. The following is a counterexample situation where tooth decay indicates heart transplant.

Suppose our patient is a borderline candidate for transplant that fell just below the threshold in our previous optimization because his life expectancy is too short to justify such a risky and costly operation. Further, the pessimistic estimate of lifetime was partly due to a significant probability that the patient has disease D_1, which is nearly always fatal in short order. Disease D_2—a very benign disorder with similar symptoms to D_1—is considered equally likely, but even a .5 chance of D_1 ruled out the heart transplant. Now the patient complains of tooth decay, which is a common manifestation in D_2 patients but unheard of among D_1 sufferers. Given our revised belief about the unlikely nature of D_1 our patient's life expectancy is substantially increased, thereby qualifying him for a heart transplant.

Although this scenario is contrived, something like it must be admitted possible whenever action and event variables have common effects. Stronger influence assertions (such as synergy) can sometimes help, but will probably not be sufficient in a satisfactory fraction of cases.

SUDO-PLANNER will provide a few facilities for the user to exercise heuristic control over its search space. The most obvious of these is simply to let the user specify (partially or completely) the repertoire of actions for the planner to consider. More sophisticated techniques allow the user to indirectly constrain the action set by restricting attention to particular internal variables. For example, SUDO-PLANNER might be directed to confine itself to plans that treat the present illness or other designated disease states. In the scenario above, the planner could (suboptimally, as it turned out[3]) omit heart transplant from consideration because it does not affect any clinical variables influenced by tooth decay. Instead SUDO-PLANNER would concentrate, presumably, on filling the cavities and recommending good brushing practices.

In the course of implementing SUDO-PLANNER I will explore additional heuristic mechanisms for focusing attention in models such as these. I hope to characterize conditions for their validity. Further work will continue the search for theoretical devices for *justified* focus of attention. A preliminary suggestion is offered in the next section.

5.3 Focus Justifications: Common Practice Cases

In medicine, established practice provides a constraint on acceptable plans beyond what might otherwise be considered reasonable action. We can represent some features of common practice in the structures developed for SUDO-PLANNER.

A common practice axiom describes, for a given case or class of cases, features of the acceptable treatment plans. Often these features can be captured in the plan constraint language of Section 3.3. For a specific patient presentation pattern, common practice might dictate that a particular plan class subsumes the admissible plans, or that others are *nogood*. Alternately, a feature of plans may be associated with necessary or sufficient (rarely both) conditions on cases.

Such constraints, when available, can be directly applied by the dominance prover. Although decisions justified by common practice constraints may be less satisfying

[3]The planner does not wrongly decide that a heart transplant is not in the optimal plan, it merely fails to consider the issue.

than those derived from "first principles" of the domain model, the former is preferred to decisions based on arbitrary heuristics.

5.4 Tradeoff Oracles

The use of common practice axioms described above illustrates the more general principle of using aggregate decisions to constrain lower-level tradeoffs. When common practice dictates a decision not derivable from a domain model, some combinations of tradeoff resolutions in the original model are thereby ruled out.

To study this central process in multi-level reasoning, I will experiment with SUDO-PLANNER augmented with a *tradeoff oracle*: a device for unwedging the dominance prover when it gets stuck. The rationale for this piece of machinery is that in rich reasoning situations, knowledge akin to the common practice constraints above is pervasive. Tradeoff resolutions might arise from a variety of sources—clinical trials, mathematical models, hypothetical reasoning, or leaps of faith, for example—that do not mesh easily with the primary domain model. Analysis of the planner with a tradeoff oracle will shed light on whether and how planners like this can exploit such knowledge without requiring a precise account of the source and generation of the knowledge itself.

In the implementation, the pseudo-oracle will answer either randomly or according to hand-coded specifications. Evaluation of SUDO-PLANNER's use of the oracle is based on the amount of dominance propagation and lower-level tradeoffs resolved for each oracle consultation.

5.5 A Note on Knowledge Engineering

The knowledge representations chosen for SUDO-PLANNER are not really intended for human consumption. While I certainly will have to work with them in constructing some examples for the planner, these structures are really designed as an internal target language to encode planning-relevant objects derived from other forms of information in a large knowledge base.

The other forms of knowledge I am thinking of range from physiological models, like that in Long's program for heart failure [29], to representations of clinical trial studies, as proposed by Rennels [39]. The feasibility of generating qualitative influence assertions from other types of causal models is supported by the existence of programs performing similar tasks—Downing's qualitative sensitivity analysis [13] and Weld's comparative analysis [53], for example.

Chapter 6

An Example: The Hepatoma/AAA Case

The best way to bring together the ideas presented thus far in this proposal is to work through an example plan formulation. The case chosen for illustration was recently analyzed by Wong et al. [62] for a Clinical Decision Consultation at the Tufts-New England Medical Center (TNEMC).

The patient is a 73-year-old-man discovered to have hepatoma (liver cancer) and an abdominal aortic aneurysm (AAA). The cancer severely degrades the patient's life expectancy, as does the threat of aneurysm rupture, which is usually fatal.

6.1 Action Hierarchy

A variety of diagnostic and therapeutic actions are available to the physicians managing this case, shown hierarchically in Figure 6.1. The hierarchy is rooted by $\mathcal{A}$, the set of all actions. The class of *diagnostic tests*, immediately below $\mathcal{A}$ in the figure, is defined informally to include all actions that make observable some clinical state of the patient. One subclass consists of *treatability tests*, where the observable bears upon the efficacy of a potential therapy. Here the only relevant test is the *hepatic workup*, which reveals whether the hepatic mass is surgically removable. We can invent a physiologic state named *resectable* that is made observable by the hepatic

workup.

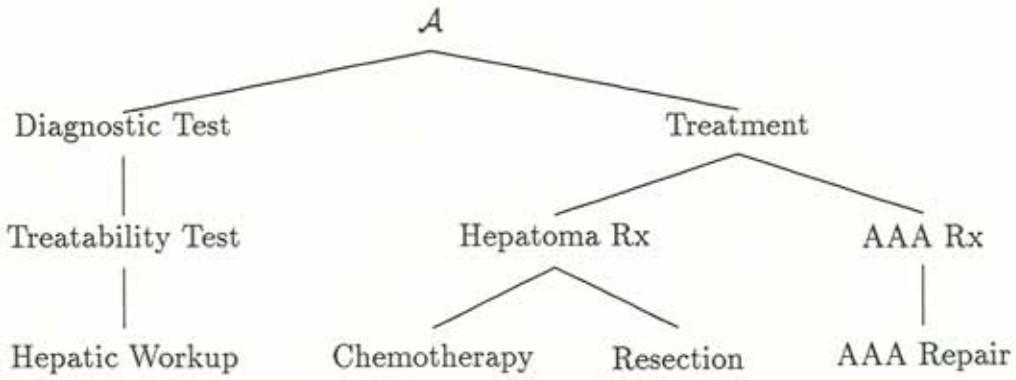


Figure 6.1: A hierarchy of actions for the hepatoma/AAA example.

Treatments are actions that influence clinical variables, usually in a desirable fashion. Subclasses of treatments relevant to this case are those that influence the state of the patient's hepatoma or his AAA. A surgical procedure, *AAA Repair*, is available to treat the latter, while two therapeutic options may be considered for the hepatoma. *Resection* is the surgical removal possible when the mass is *resectable*,[1] and *chemotherapy* is an alternate medical approach.

Given the hierarchy of actions and the specification of observables, we can construct the corresponding plan lattice. The set of all plans is generated by the production rule of Section 3.1 from the primitive action set {*workup, resection, chemotherapy, repair*} and the single observable *resectable* created by *workup*. Under the constraint that each action may be applied at most once,[2] even this simple configuration

[1] As discussed in Section 3.2.2, we model this situation by treating the *resection* as a no-op when the condition *resectable* is false.

[2] This constraint cannot be completely captured within the constraint language. We can designate the plan class $[aa, \epsilon]$ to be *nogood* for a bound to each of the four actions here, but this only limits unconditional repetition of actions. For the remaining discussion, I will assume that a sufficient mechanism for enforcing this restriction exists. Note also that without such a constraint, the set of possible plans is infinite. However, the planner can still derive useful properties about the (also infinite) set of admissible plans. This suggests that cardinality of the admissible plan set is not a perfect measure of progress.

yields 72 distinct syntactically valid plans.[3] Beyond this, we need to rely on our dominance-proving planner to prune the set.

6.2 Case Input

If the knowledge base really contained only these four actions, there would be little need to take advantage of attention-focusing mechanisms in the input module. In a more realistic context, however, the program has specifications for numerous actions in addition to these, most of them unrelated to this case.

Rather than require the user to specify the action set directly, we can achieve the same result in this case by directing the planner to limit attention to actions affecting the hepatoma and AAA disease states. The input consists of

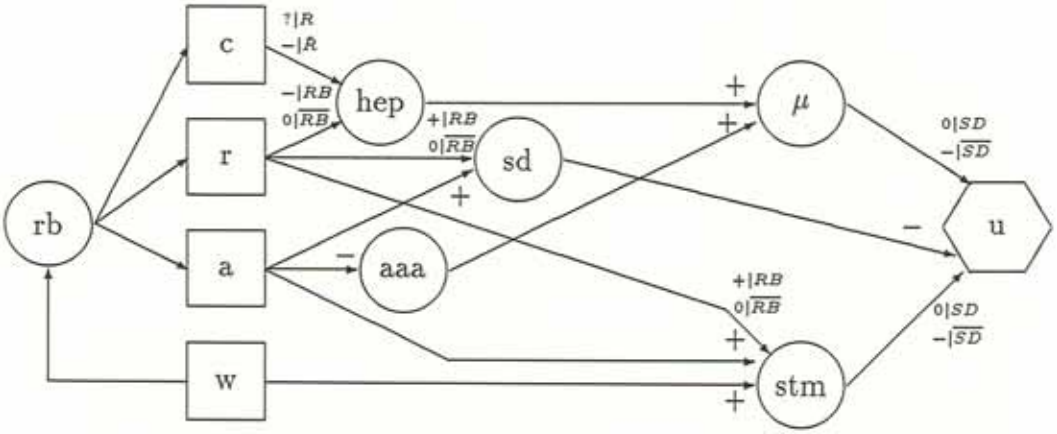
- The previously optimal plan: **nil**
- Changes in patient state: increase (in the FSD sense) in hepatoma state, increase in AAA state
- Focus rule: consider only plans to influence these disease states.

As noted in Section 5.2, this focus rule is not generally valid. However, at present it is the best compromise I have come up with between direct user specification of the plan space and unfocused search.

6.3 Qualitative Influence Diagram

Let us examine the domain model for this problem. The effects of the available actions and relevant events are described by the qualitative influence diagram depicted in Figure 6.2.

[3]Without *workup*, there are $2^3 = 8$ unconditional plans. With workup, plans are conditional on *resectable*. For each value of the condition, there are the same 8 unconditional possibilities, leading to $8 \times 8 = 64$ combinations. The total number of plans is $8 + 64 = 72$.



key:

c:	chemotherapy	hep:	hepatoma disease state severity
r:	resection	aaa:	AAA severity
a:	AAA repair	sd:	death from surgery
w:	workup	stm:	short-term morbidity
rb:	resectability of hepatoma	μ:	mortality "rate"

Figure 6.2: Qualitative influence diagram for the hepatoma/AAA example.

In the diagram, boxes denote action variables and circles event variables. The hexagon is the *value node*, used to distinguish the special event variable u, which denotes expected utility. Directions on the links indicate qualitative influences; a link from a to b annotated with $+|y$, for example, asserts $S^{+|y}(a, b)$. Links without direction are informational. An arrow from action to event indicates observable creation; the reverse orientation signals that the observable is available for conditioning.

The action variables in Figure 6.2 are the primitive actions c, r, a, and w—the leaves of the action hierarchy of Figure 6.1. To reason at higher levels of abstraction we would create corresponding diagrams that substituted intermediate action classes from the hierarchy. Further discussion of this possibility is deferred to Section 6.6.

In the model there are disease states corresponding to the two medical problems, the hepatoma and the AAA. The potential therapies reduce the disease states, except that resection is a no-op when the tumor is not resectable ($S^{0|\overline{RB}}(r, hep)$). Both disease states positively influence mortality "rate"—roughly the likelihood of death in any future time period conditional on surviving that far out.[4] The two surgical procedures, resection and AAA repair, each increase the likelihood of death from surgery. In addition, they contribute to short-term morbidity, as does the hepatic workup to determine resectability. Mortality, short-term morbidity, and surgical death are all undesirable (U^-), although the first two are irrelevant when the latter occurs.

The planner draws conclusions from the model by inference from the qualitative influences. The basic inference rule is influence chaining by sign multiplication as described in Section 4.2. For example, AAA repair has the benefit of reducing mortality due to the AAA, but contributes to short-term morbidity and the risk of mortality from the surgical procedure. In the next section, we will see how inferences such as these can support dominance-proving in the plan lattice.

[4]The quote marks signal that the parameter is not really a rate in that it may not be constant over time.

6.4 Plan Lattice

Part of the plan lattice for this problem is depicted in Figure 6.3. The plan classes in the lattice were selected manually; automatic generation of the lattice will be driven by the dominance prover as described briefly below. Given the plan classes, a simple algorithm can compute all the subsumptions represented in the figure. For the present discussion, we presume that the lattice is set up as presented before the dominance proving and propagation commences.

Specialization links connected by horizontal chords indicate that the included subclasses *cover* the parent class. That is, the parent plan class is equivalent to the union of their plans. From dominance property (2.10) (the $\Leftarrow$ direction), it follows that

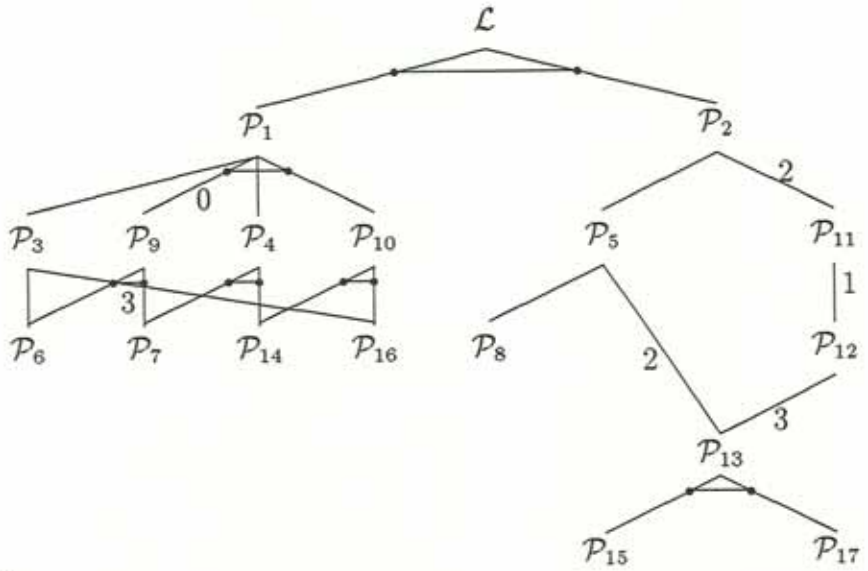
$$\bigcup_{i=1}^{n} \mathcal{P}_i \supseteq \mathcal{P} \Rightarrow \bigvee_{i=1}^{n} D(\mathcal{P}_i, \mathcal{P}), \tag{6.1}$$

at least one of the subclasses must dominate the parent. This fact will be useful below.

Using the qualitative influence diagram of Figure 6.2, the dominance prover can derive dominance conditions (in particular, *restrictions*) that hold among plan classes in the lattice. The following is a plausible sequence of inference steps by which the set of admissible plans is pruned by successive dominance assertions.

Step 0: No resection without workup. This is conceptually more like a domain condition than a dominance condition. It can be specified directly in the knowledge representation by asserting that the partial plan $[R, W]$ is a *nogood*, or alternately $D(\emptyset, \mathcal{P}_{10})$. It is sufficient to constrain unconditional plans in this case because all conditional plans contain W (but see footnote 2). Because the two plan classes $\mathcal{P}_9$ and $\mathcal{P}_{10}$ cover $\mathcal{P}_1$, we can apply property (6.1) to deduce that $\mathcal{P}_1$ is restricted to $\mathcal{P}_9$, or $D(\mathcal{P}_9, \mathcal{P}_1)$. The "0" on the specialization link from $\mathcal{P}_1$ to $\mathcal{P}_9$ in Figure 6.3 indicates that this becomes a restriction link by step 0.

Step 1: Plans with workup must not have resection if not resectable. Our domain model asserts that resection is a no-op if the tumor is not resectable (r is conditionally independent of everything if $\overline{RB}$). No-ops may always be eliminated



key:

$\mathcal{L}$	=	$[\epsilon, \epsilon]$	$\mathcal{P}_9$	=	$[\epsilon, RW]$
$\mathcal{P}_1$	=	$[\epsilon, W]$	$\mathcal{P}_{10}$	=	$[R, W]$
$\mathcal{P}_2$	=	$[W, \epsilon]$ (if RB $(\mathcal{P}_1)$ $(\mathcal{P}_1)$)	$\mathcal{P}_{11}$	=	$[W, \epsilon]$ (if RB $(\mathcal{P}_{10})$ $(\mathcal{P}_1)$)
$\mathcal{P}_3$	=	$[\epsilon, CW]$	$\mathcal{P}_{12}$	=	$[W, \epsilon]$ (if RB $(\mathcal{P}_{10})$ $(\mathcal{P}_9)$)
$\mathcal{P}_4$	=	$[C, W]$	$\mathcal{P}_{13}$	=	$[W, \epsilon]$ (if RB $(\mathcal{P}_{10})$ $(\mathcal{P}_7)$)
$\mathcal{P}_5$	=	$[W, \epsilon]$ (if RB $(\mathcal{P}_1)$ $(\mathcal{P}_4)$)	$\mathcal{P}_{14}$	=	$[CR, W]$
$\mathcal{P}_6$	=	$[\epsilon, CRW]$	$\mathcal{P}_{15}$	=	$[W, \epsilon]$ (if RB $(\mathcal{P}_{14})$ $(\mathcal{P}_7)$)
$\mathcal{P}_7$	=	$[C, RW]$	$\mathcal{P}_{16}$	=	$[R, CW]$
$\mathcal{P}_8$	=	$[W, \epsilon]$ (if RB $(\mathcal{P}_7)$ $(\mathcal{P}_7)$)	$\mathcal{P}_{17}$	=	$[W, \epsilon]$ (if RB $(\mathcal{P}_{16})$ $(\mathcal{P}_7)$)

Figure 6.3: A plan lattice for the example.

from plans, therefore plans with R on the $\overline{RB}$ condition may be ruled out. In this lattice, this enables us to restrict $\mathcal{P}_{11}$ to $\mathcal{P}_{12}$, its counterpart with R on the out-list for the non-resectable condition.

Step 2: Plans with workup must have resection if resectable. This is a special case of the more general principle that it is suboptimal to perform costly tests unless there is some action contingent on the result of the test. Here we derive the conclusion from the following line of reasoning. Suppose the optimal plan includes workup (w). The effects of w are unambiguously non-positive—it increases short-term morbidity which in turn reduces utility. Therefore, the only justification for including w in the plan is that w creates the observable rb. Introduction of an observable can only have benefits if the plan is conditioned on that variable. In this model, the only influences affected by rb are those of resection (r) on the events *hep*, *sd*, and *stm*. Because we already know from step 1 that conditional plans must have $out(R)$ if $\overline{RB}$, they must have the opposite if RB.[5] Thus, $\mathcal{P}_2$ is restricted to $\mathcal{P}_{11}$ and $\mathcal{P}_5$ to $\mathcal{P}_{13}$ because $\mathcal{P}_{10}$ is just $\mathcal{P}_1$ with the constraint that R is included.[6]

Step 3: Plans without resection must include chemotherapy. Chemotherapy has no morbidity cost, and positive benefit on unresected hepatoma, according to the model. Its effect is ambiguous when the tumor has been resected. The constraint that C must be in plans without R restricts $\mathcal{P}_9$ to $\mathcal{P}_7$ and correspondingly $\mathcal{P}_{12}$ to $\mathcal{P}_{13}$.

For the sake of clarity, the pruned plan lattice is reproduced in Figure 6.4. The leaves of this lattice form the set of admissible plans because they are reached from $\mathcal{L}$ via restriction paths. The only branches along the way are for covers, which are in essence disjunctive restrictions by property (6.1).

In our example, the set of admissible plans is $\mathcal{P}_7 \cup \mathcal{P}_{13}$. Resolving the influence of chemotherapy on hepatoma severity given resection would rule out either $\mathcal{P}_{15}$ or $\mathcal{P}_{17}$. If $S^{+|R}(c, hep)$, then adjuvant chemotherapy is warranted ($\mathcal{P}_{15}$).

[5]This derivation is not an artifact of treating resection of an unresectable hepatoma as a no-op. A model that treated it as a surgical procedure with mortality and morbidity but no therapeutic value would support the same inferences.

[6]Note that even though $\mathcal{P}_{10}$ is nogood as a standalone plan (by step 0), it is legitimate as part of a compound plan.

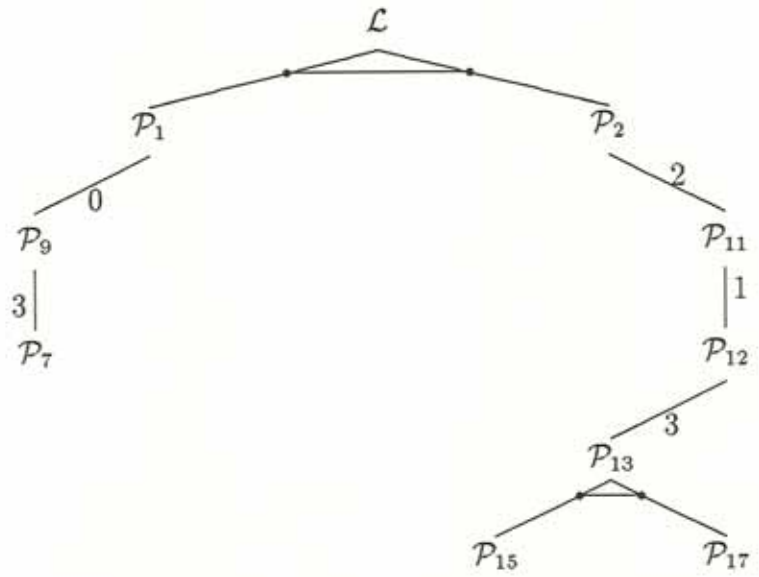


Figure 6.4: The pruned plan lattice.

Assuming that adjuvant chemotherapy is *not* recommended ($\mathcal{P}_{17}$), the admissible plans are:

1. C
2. AC
3. W (if RB (R) (C))
4. W (if RB (R) (AC))
5. W (if RB (AR) (C))
6. W (if RB (AR) (AC))

The plans with adjuvant chemotherapy are the same as 3–6 above with C added to the RB condition. Note that these are complete plans, not plan classes, stated in the plan language of Section 3.1.

6.5 Comparison of Results

It is instructive to compare the admissible plan set derived by the planner with the strategies considered in the original TNEMC analysis. Assuming adjuvant chemotherapy is non-efficacious (an apparent assumption of the TNEMC analysts), the planner produces the six plans listed above. The decision tree for the case, depicted in Figure 6.5, also evaluated six strategies.

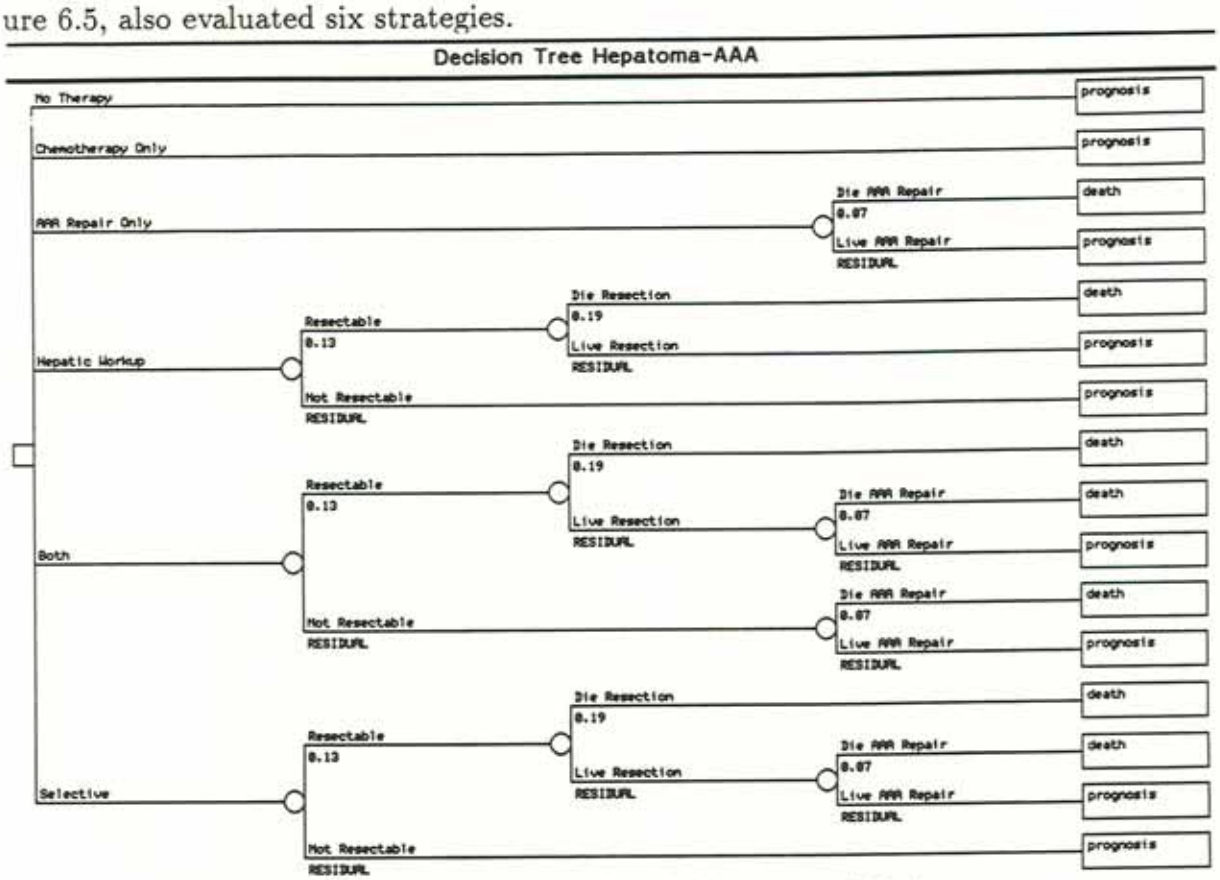


Figure 6.5: The decision tree for the hepatoma/AAA case.

In the decision tree, the strategies *chemotherapy only*, *AAA repair only*, *hepatic workup*, *both*, and *selective* correspond to plans numbered 1, 2, 3, 6, and 5, respectively, in the list at the end of Section 6.4. The remaining strategy, *no therapy*, corresponds to the null plan that was ruled inadmissible by the dominance prover. In fact, *no therapy* is clearly dominated under the model's assumptions about chemotherapy; it was only included in the original analysis for purposes of comparison.

The set of admissible plans included one other that was not considered in the TNEMC analysis.

$$W \text{ (if } RB \text{ } (R) \text{ } (AC)) \tag{6.2}$$

Plan 6.2 is the opposite of the plan labeled *selective* above. According to the plan, AAA repair is performed if and only if the hepatoma is *not* resected. This policy is not at all reasonable because it makes more sense to undergo surgery for the AAA if the tumor is resected and therefore the patient is expected to live longer. The costs of the surgery are up front (possible mortality and short-term morbidity from the procedure), while the benefits are accrued over the patient's lifetime (eliminate risk of rupture). Unfortunately, the qualitative model of Figure 6.2 is too weak to support this argument.

This problem could be fixed via an appeal to qualitative synergy. The hepatoma and AAA have independent effects on the mortality "rate" μ, but this parameter in turn has a greater-than-linear effect on utility.[7] Extensions to the qualitative influence representation and reasoning mechanisms to capture this phenomenon would significantly broaden the powers of this planner.

6.6 Hierarchical Plan Formulation

The preceding analysis proceeded at a single level of abstraction, the lowest. In a simple isolated example such as this, it is difficult to demonstrate the facilities for multi-level reasoning or to make a case for the value of such a capability.

To plan at a higher level of abstraction, we construct a domain model in terms of the higher-level actions in the action taxonomy. For example, we could have constructed a qualitative influence diagram analogous to that of Figure 6.2 with actions like "treatability test" and "hepatoma treatment" substituted for workup and resection. In fact, such an influence diagram for the generic test/treat decision was presented in a previous paper [58]. In this case, most of the qualitative influences

[7]When μ is really a constant rate, life expectancy is $1/\mu$ by the DEALE model [2]. Because this function is convex, decreasing μ by a given amount increases life expectancy more when μ is lower.

specified for the low-level actions are reasonably considered true for their generic categories. That a hepatoma treatment negatively influences the hepatoma disease state when the treatability condition is satisfied is more a definition of these abstract categories than an incidental fact of the world.

Discovering dominance properties at high levels of abstraction can have great advantages in searching a large plan space. Suppose that our knowledge base included twelve different varieties of chemotherapy regimens that could be considered for this patient. Establishing the major relation between chemotherapy actions and admissible plans (step 3 above) at the aggregate level saves a lot of effort compared to deriving the result for each regimen individually.

A plan lattice extensively developed at high levels of abstraction can serve as compiled structure for storing precomputed dominance results that can be applied to particular cases. For example, the results of the generic test/treat constrain just about any more specific planning problem. Starting with the pruned lattice eliminates the need to re-derive those results for the particular instantiations. The conclusions are passed on via plan subsumption and dominance propagation, operations generally simpler than dominance proving.

6.7 Discussion

To put it mildly, this extended example is a bit rough at the edges. The exposition glossed over several issues and leaves a list of unanswered questions including:

- How exactly is the qualitative influence diagram constructed from a primitive knowledge representation? This breaks down into several sub-questions:
  - What does this representation look like in NIKL?
  - How are event *variables* for the influence diagram derived from the representation for events themselves?
  - What is the unification mechanism for creating the right associations between events in influence links? This is especially important for reasoning at multiple levels of abstraction.

- How are nogoods specified outside the dominance process? This was required twice above: consider at most one of each action and no resection without workup.
- What is the complexity of dominance propagation, particularly the use of covers?

These and other unresolved issues will be the primary subjects of attention in the course of this thesis project. I expect that several of the notations and constructs employed in the example will need to be modified in the final implementation of SUDO-PLANNER.

Chapter 7

Research Plan

Work over the coming year will refine this model of planning under uncertainty with partially satisfiable objectives. The major software components will be implemented (plan subsumption and dominance propagation algorithms, influence network generator, dominance prover, and case input interface). Along with the actual design and programming, I will devote effort toward developing mathematical characterizations of the procedure's capabilities. Although interaction with a full-blown knowledge base will not be possible in the short term, substantial knowledge base fragments at an extremely shallow level of specification will be developed to test the planner.

Some specific tasks:

- Collect more examples. In addition to the case of Section 6, I have also closely examined a decision problem involving a patient with a large AAA but at risk for surgery because of coronary artery disease and a history of cerebrovascular problems. This case was analyzed by Dunn [14] and previously explored by me in a study of reasoning about utility models [54].

- Encode actions and events to support examples, at least at the surface level (clinical influences).

- Develop a scenario for deriving qualitative influences from physiological or empirical models (for example, assume availability of the heart failure program [29] or Roundsman [39]). This will not be implemented.

- Invent constructs and reasoning mechanisms to handle qualitative synergy.
- Further study of temporal issues. (However, no extensions to the plan or constraint languages will be incorporated in the implementation.)
- Development of focus mechanisms (input module and constructs).
- Theoretical investigation of valid focus rules. The implementation may include invalid focus heuristics.
- Work out protocol for use of tradeoff oracle.

I expect to get some help in knowledge base construction and example collection from our medical collaborators.

7.1 Current Implementation State

As of June 17, 1987, the following components have been implemented:

- A high-level classifier that assumes the nodes in the subsumption graph handle the predicate `subsumes-p`.
- The algorithms for plan class simplification and subsumption, described in Appendix A.
- Mechanisms for propagating dominance relations within the plan graph, based on the properties of Section 2.2.
- A skeletal NIKL knowledge base of actions and events.
- Development scaffolding, including programmer's interface to the graph structures and a naive planning algorithm based on random mutation of existing plan class descriptions.

7.2 Evaluation Criteria

This project should be considered a success if it achieves the following:

- Reasonable plan formulation performance on several medical examples drawn from the TNEMC case files. The cases should exercise overlapping portions of the knowledge base without confounding each other.
- Demonstration of the facility and advantages of planning at multiple levels of abstraction.
- A dominance prover that seems to capture the intuitive notion of "tradeoff."

7.3 Timetable

Finish implementation of basic planning engine:	1 October 1987
Finish knowledge base construction and example collection and debugging, including tuning of planning engine:	1 March 1988
Circulate near-final thesis draft:	1 April 1988
Thesis delivery date:	29 April 1988

Appendix A

Plan Class Algorithms

A.1 Simplification

Partial plans can be simplified to facilitate subsumption computation. The following rewriting removes superfluous conditionalization by propagating common constraints out of conditionals.

$$\begin{aligned}&(\text{if OBS } ([IN_1, OUT_1]\ \mathcal{CP}_1)\ ([IN_2, OUT_2]\ \mathcal{CP}_2)) \longrightarrow \\ &\qquad [IN_1 \triangle IN_2, OUT_1 \triangledown OUT_2] \\ &\qquad\qquad (\text{if OBS } ([IN_1, OUT_1]\ \mathcal{CP}_1)\ ([IN_2, OUT_2]\ \mathcal{CP}_2))\end{aligned} \tag{A.1}$$

The operations $\triangle$ and $\triangledown$ are variants on intersection that take action subsumption into account. The combination of IN_1 and IN_2 constraints is the collection of actions that are in both or have a specialization in both, taking care to avoid multiple uses of the same constraint. We can compute this combination by finding the maximal match[1] of the bipartite graph formed by connecting the elements of IN_1 and IN_2 that are related by subsumption, then taking the more general element of each edge in the match. For example, suppose we have actions $\mathcal{A}_1, \ldots, \mathcal{A}_n$ such that $\mathcal{A}_i$ subsumes $\mathcal{A}_j$ if and only if $i \geq j$. Some sample values of the $\triangle$ operation for this situation are given in Table A.1.

[1]For this purpose, maximality must take into account preference for edges with stronger (more specific) actions. A more precise statement of the algorithm is deferred to the thesis report.

A_1	A_2	$A_1 \triangle A_2$	$A_1 \triangledown A_2$	$A_1 \triangle' A_2$	$A_1 \triangledown' A_2$
$\mathcal{A}_1$	$\mathcal{A}_2$	$\mathcal{A}_2$	$\mathcal{A}_1$	$\mathcal{A}_1$	$\mathcal{A}_2$
$\mathcal{A}_1\mathcal{A}_2$	$\mathcal{A}_2$	$\mathcal{A}_2$	$\mathcal{A}_2$	$\mathcal{A}_1\mathcal{A}_2$	$\mathcal{A}_2$
$\mathcal{A}_1\mathcal{A}_3$	$\mathcal{A}_2$	$\mathcal{A}_2$	$\mathcal{A}_2$	$\mathcal{A}_1\mathcal{A}_3$	$\mathcal{A}_3$
$\mathcal{A}_2\mathcal{A}_3$	$\mathcal{A}_1\mathcal{A}_4$	$\mathcal{A}_2\mathcal{A}_4$	$\mathcal{A}_3$	$\mathcal{A}_1\mathcal{A}_3$	$\mathcal{A}_4$

Table A.1: Sample values of the $\triangle$, $\triangledown$, $\triangle'$, and $\triangledown'$ operations. In this example, $\mathcal{A}_i$ subsumes $\mathcal{A}_j$ if and only if $i \geq j$.

In combining the OUT lists, we do not have to worry about multiple uses of the constraints. $OUT_1 \triangledown OUT_2$ simply contains the strongest (most general for OUT constraints) actions that are subsumed by actions present in both lists. Some simple examples of $\triangledown$ appear in Table A.1.

Notice that rewrite rule A.1 may sometimes require combination of unconditional action in-out list pairs produced by different conditionals at the same level of nesting. The rule for combining is as follows:

$$[IN_1, OUT_1]\,[IN_2, OUT_2] \longrightarrow [IN_1 \triangle' IN_2, OUT_1 \triangledown' OUT_2], \qquad \text{(A.2)}$$

where $\triangle'$ and $\triangledown'$ are variants of union dual to $\triangle$ and $\triangledown$. Again, the operation on IN lists is the complicated one. To compute $\triangle'$ for two action lists, find the maximal match of the bipartite graph formed by compatibility edges (two action classes are compatible if and only if they are not disjoint), then return the more *specific* element of each edge in the match, along with any unmatched actions. Note that this operation does not produce a unique value if there is more than one maximal match.

The $\triangledown'$ operator produces an out-list containing actions that are in either OUT_1 or OUT_2. If one of the lists contains an action that has a subsuming counterpart in the other, the more general action is included in the combination. Examples of these operations are also included in Table A.1.

We can also rewrite partial plans to achieve any canonical ordering of the nesting

of observables:

$$\begin{array}{l}(\text{if } o_1 \ (\text{if } \ o_2 \ (\mathcal{P}_1) \ (\mathcal{P}_2)) \ (\text{if } o_2 \ (\mathcal{P}_3) \ (\mathcal{P}_4)) \longrightarrow \\ \qquad (\text{if } o_2 \ (\text{if } o_1 \ (\mathcal{P}_1) \ (\mathcal{P}_3)) \ (\text{if } o_1 \ (\mathcal{P}_2) \ (\mathcal{P}_4))\end{array} \tag{A.3}$$

A.2 Computing Plan Subsumption

The analysis of Section 2.6 suggests that an important feature of the constraint language is its support for subsumption computations. The following algorithm derives subsumption relations among partial plans in this language, though it is incomplete in two respects:

1. The algorithm ignores relations among the OBS predicates. It would fail to determine, for example, that $\mathcal{P}_1$ subsumes $\mathcal{P}_2$, where

$$\begin{array}{rcl}\mathcal{P}_1 & = & (\text{if } x > 5 \ (\mathcal{P}_0) \ (\mathcal{L})) \\ \mathcal{P}_2 & = & (\text{if } x > 0 \ (\mathcal{P}_0) \ (\mathcal{L}))\end{array} \tag{A.4}$$

2. Simplification does not catch all condition-independent features of plans that may be buried in *if* clauses. Another example of a missed subsumption of $\mathcal{P}_1$ over $\mathcal{P}_2$ is:

$$\begin{array}{rcl}\mathcal{P}_1 & = & [\mathcal{A}_1, \epsilon] \\ \mathcal{P}_2 & = & (\text{if } o \ ([\mathcal{A}_2, \epsilon]) \ ([\mathcal{A}_3, \epsilon]))\end{array} \tag{A.5}$$

   where $\mathcal{A}_1$ subsumes both $\mathcal{A}_2$ and $\mathcal{A}_3$, but neither $\mathcal{A}_2$ nor $\mathcal{A}_3$ subsumes the other.

The algorithm assumes a subroutine for computing subsumption among action classes (P-ACTIONs). Because actions in SUDO-PLANNER are represented as NIKL concepts, we can determine action subsumption directly from the pre-classified taxonomy. The subsumption algorithm tests the following conditions, sufficient for $subsumes(\mathcal{P}_1, \mathcal{P}_2)$:

1. For every action token $a \in IN_1$ there exists a distinct token $a' \in IN_2$ such that a subsumes a'. Let IN_2' be the set of actions a in IN_2 matched with the actions a from IN_1.

2. For every action token $a \in OUT_1$:

   - there exists no action token a' in $IN_2 - IN_2'$ such that a subsumes a', and
   - there exists a token (not necessarily distinct for different as) $a' \in OUT_2$ such that a' subsumes a.

3. For every top-level conditional in $\mathcal{P}_1$ of the form (if o $(\mathcal{P}_{a1})$ $(\mathcal{P}_{b1})$) there is a corresponding conditional in $\mathcal{P}_2$ of the form[2] (if o $(\mathcal{P}_{a2})$ $(\mathcal{P}_{b2})$) *and* both

   - $\mathcal{P}_{a1}$ subsumes $\mathcal{P}_{a2}$
   - $\mathcal{P}_{b1}$ subsumes $\mathcal{P}_{b2}$.

Condition 1 can be tested with an algorithm for maximal bipartite matching in time $O(n^{5/2})$ [24]. The matching is with respect to the graph formed by connecting a_i to a_j' iff a_i subsumes a_j'. If the cardinality of the matching is equal to the size IN_1 the condition is satisfied.

The test for condition 2 is easier because the mapping from OUT_1 to OUT_2 may be many-to-one. A straightforward verification of the condition is $O(n^2)$.

Notice that condition 3 requires two recursive calls to the subsumption algorithm on partial plans at one deeper level of nesting. Flattened out, condition 3 amounts to a large **and** of a set of subsumption problems whose input sizes sum to that of the original problem. Because the complexity of condition 1 is more than linear, the worst case for subsumption is when the partial plans are described by in-lists only. The overall time complexity of the algorithm is $O(n^{5/2})$ steps, where action subsumption is presumed to take unit time.

I can think of some simple enhancements to the algorithm which would eliminate part of the second type of incompleteness mentioned above without degrading complexity. And there are certainly some simple event predicate subsumptions we can use to strengthen the algorithm. A precise description of the final procedure and characterization of its completeness will appear in the thesis.

[2]It may be necessary to rearrange $\mathcal{P}_2$ via transformations like (A.3) to get this corresponding conditional to top-level. The complexity analysis below ignores this computation phase because it is dominated by verification of these conditions.

Bibliography

[1] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154, 1984.

[2] J. Robert Beck, Jerome P. Kassirer, and Stephen G. Pauker. A convenient approximation of life expectancy (the 'DEALE') I. Validation of the method. *American Journal of Medicine*, 73:883–888, 1982.

[3] Ronald J. Brachman and Hector J. Levesque. The tractability 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.

[4] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216, 1985.

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

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

[7] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, forthcoming, 1987.

[8] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, Reading, MA, 1985.

[9] Gregory Floyd Cooper. NESTOR: *A Computer-Based Medical Diagnostic Aid that Integrates Causal and Probabilistic Knowledge.* PhD thesis, Stanford University, November 1984.

[10] Louis Anthony Cox, Jr. *Mathematical Foundations of Risk Measurement.* PhD thesis, Massachusetts Institute of Technology, May 1986.

[11] Johan de Kleer. An assumption-based TMS. *Artificial Intelligence*, 28:127–162, 1986.

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

[13] Keith L. Downing. Diagnostic improvement through qualitative sensitivity analysis and aggregation. In *Proceedings of the National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, 1987.

[14] Van H. Dunn. Grand rounds, Beth Israel hospital. 1984. Unpublished decision analysis consult report, Division of Clinical Decision Making, Tufts-New England Medical Center.

[15] Jerome A. Feldman and Robert F. Sproull. Decision Theory and Artificial Intelligence II: The hungry monkey. *Cognitive Science*, 1:158–192, 1977.

[16] Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[17] Peter C. Fishburn and Raymond G. Vickson. Theoretical foundations of stochastic dominance. In [60].

[18] G. Anthony Gorry, Howard Silverman, and Stephen G. Pauker. Capturing clinical expertise: A computer program that considers clinical responses to digitalis. *American Journal of Medicine*, 64:452–460, March 1978.

[19] Benjamin N. Grosof. Non-monotonicity in probabilistic reasoning. In John F. Lemmer, editor, *Uncertainty in Artificial Intelligence*, North-Holland, 1987.

[20] Steve Hanks and Drew McDermott. Default reasoning, nonmonotonic logics, and the frame problem. In *Proceedings of the National Conference on Artificial Intelligence*, pages 328–333, American Association for Artificial Intelligence, 1986.

[21] David E. Heckerman and Eric J. Horvitz. The myth of modularity in rule-based systems. In John F. Lemmer, editor, *Uncertainty in Artificial Intelligence*, North-Holland, 1987.

[22] J. P. Hollenberg. The decision tree builder: An expert system to simulate medical prognosis and management. *Medical Decision Making*, 4(4), 1984. Abstract from the Sixth Annual Meeting of the Society for Medical Decision Making.

[23] Samuel Holtzman. *Intelligent Decision Systems.* PhD thesis, Stanford University, March 1985.

[24] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[25] David H. Krantz, R. Duncan Luce, Patrick Suppes, et al. *Foundations of Measurement.* Academic Press, New York, 1971.

[26] David M. Kreps and Evan L. Porteus. Temporal von Neumann-Morganstern and induced preferences. *Journal of Economic Theory*, 20:81–109, 1979.

[27] Curtis P. Langlotz, Lawrence M. Fagan, Samson W. Tu, et al. A therapy planning architecture that combines decision theory and artificial intelligence techniques. *Computers and Biomedical Research*, 1987.

[28] Doug Lenat, Mayank Prakash, and Mary Shepherd. CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Magazine*, 6(4):65–85, 1986.

[29] W. J. Long, S. Naimi, M. G. Criscietello, et al. An aid to physiological reasoning in the management of cardiovascular disease. In *Proceedings of the IEEE Computers in Cardiology Conference*, pages 3–6, September 1984.

[30] Tomás Lozano-Pérez, Matthew T. Mason, and Russell H. Taylor. Automatic synthesis of fine-motion strategies for robots. *International Journal of Robotics Research*, 3(1):3–24, 1984.

[31] Marie B. Macaisa and Candace L. Sidner. Plan representation and plan recognition. 1987. Abstract presented somewhere.

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

[33] 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.

[34] Drew McDermott. Planning and acting. *Cognitive Science*, 2:71–109, 1978.

[35] Ramesh S. Patil. *Causal representation of patient illness for electrolyte and acid-base diagnosis.* TR 267, Massachussetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, October 1981.

[36] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29:241–288, 1986.

[37] Judea Pearl. *Markov and Bayes networks: A comparison of two graphical representations of probabilistic knowledge.* Technical Report R-46, UCLA Computer Science Department, September 1986.

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

[39] Glenn Douglas Rennels. *A Computational Model of Reasoning from the Clinical Literature.* PhD thesis, Stanford University, June 1986.

[40] Cynthia J. Rutherford, Byron Davies, Arnold I. Barnett, et al. *A computer system for decision analysis in Hodgkins Disease.* TR 271, Massachussetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, 1981.

[41] Earl D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5:115–135, 1974.

[42] Earl D. Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier, 1977.

[43] Leonard J. Savage. *The Foundations of Statistics.* Dover Publications, New York, second edition, 1972.

[44] James G. Schmolze 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.

[45] Ross D. Shachter. Evaluating influence diagrams. *Operations Research*, 34:871–882, 1986.

[46] David J. Spiegelhalter. Probabilistic reasoning in predictive expert systems. In Laveen N. Kanal and John F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 47–67, North-Holland, 1986.

[47] David J. Spiegelhalter and Robin P. Knill-Jones. Statistical and knowledge-based approaches to clinical decision-support systems, with an application in gastroenterology. *Journal of the Royal Statistical Society*, 147:35–77, 1984.

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

[49] William R. Swartout. XPLAIN: A system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21:285–325, 1983.

[50] 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.

[51] Josh Tenenberg. Planning with abstraction. In *Proceedings of the National Conference on Artificial Intelligence*, pages 76–80, American Association for Artificial Intelligence, 1986.

[52] Marc B. Vilain. The restricted language architecture of a hybrid representation system. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 547–551, 1985.

[53] Daniel S. Weld. Comparative analysis. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1987.

[54] Michael Paul Wellman. *Reasoning about preference models.* TR 340, Massachussetts Institute of Technology, Laboratory for Computer Science, 545 Technology Square, Cambridge, MA, 02139, May 1985.

[55] Michael P. Wellman. Representing health outcomes for automated decision formulation. In *MEDINFO 86: Proceedings of the Fifth Conference on Medical Informatics*, pages 789–793, October 1986.

[56] Michael P. Wellman. Dominance and subsumption in constraint-posting planning. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, 1987.

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

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

[59] Michael P. Wellman and David E. Heckerman. The role of calculi in uncertain reasoning. In *Proceedings of the Workshop on Uncertainty in Artificial Intelligence*, July 1987.

[60] G. A. Whitmore and M. C. Findlay, editors. *Stochastic Dominance: An Approach to Decision Making Under Risk.* D. C. Heath and Company, Lexington, MA, 1978.

[61] Robert Wilensky. Meta-planning: Representing and using knowledge about planning in problem solving and natural language understanding. *Cognitive Science*, 5(3):197–233, 1981.

[62] John B. Wong, Alan J. Moskowitz, and Stephen G. Pauker. Clinical decision analysis using microcomputers—A case of coexistent hepatocellular carcinoma and abdominal aortic aneurysm. *Western Journal of Medicine*, 145:805–815, 1986.

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

[64] Ramin Zabih, David McAllester, 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.