University of Washington

Abstract

A Value-directed Approach to Planning

by Mikel R. Williamson

Chairperson of Supervisory Committee: Associate Professor Steven J. Hanks
Department of Computer Science
and Engineering

The traditional definition of planning in the field of artificial intelligence provides only a very narrow notion of plan quality, namely that a plan is good just in case it achieves a specified goal. For many applications, this black-or-white notion of quality is insufficiently expressive. This dissertations extends the definition of planning to provide a much richer conception of plan quality.

We propose that a planning problem should be posed in terms of a decision-theoretic value function, and that planning should be seen as an optimization process. We present PYRRHUS, a planning system which uses a branch-and-bound technique to find optimal plans for a class of goal-directed value functions which is strictly more expressive than the kind of goal formulas used by classical planning. Such value functions allow one to express not just one's goals, but also how important those goals are, ways in which they may be partially achieved, and the worth of resources that may be consumed by a plan. We empirically explore issues of heuristic control that arise during the planning process, and compare the difficulty of value-directed planning to comparable classical planning problems.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

It is a bad plan that admits of no modification.

—Publius Syrus, 42 B.C.

1.1 Background and motivation

The ability to pursue goals has long been recognized as a fundamental aspect of intelligent behavior. Looking around us, we see that a good deal of human activity seems to be goal-directed. It is thus not surprising that much research in artificial intelligence has been dedicated to the task of *planning*, the cognitive activity of devising schemes to achieve goals.

The aim of planning research is to develop a *planner*, a sub-system or module that acts in support of a larger intelligent system. A planner is given a *planning problem*, and its duty is to find a *plan* which solves that problem. Planning could play a role in many different kinds of systems. For example, a decision support system might receive its planning problems from a user, and simply provide plans as a final product. In an autonomous agent, by contrast, the planning problem might be generated by another component of the agent architecture, and the resulting plan used to control behavior.[1]

As in other areas of A.I., the first problem faced by planning researchers was to

[1] We will use the term *agent* generically to refer to the entity posing the planning problem.

Given:

- a description of the *current world state*,
- a description of the *goal* to be achieved, and
- a set of possible *actions* that can be performed.

Find: a *sequence of actions* satisfying the goal if performed in the current world state.

Figure 1.1: The classical formulation of the planning problem

define the problem itself. Early researchers developed a particular formulation of the planning problem that because of its generality endures to this day as the *classical planning problem* (Figure 1.1).

Most work on the classical planning problem has proceeded along three interrelated lines:

- **Representational** The planning problem as shown in Figure 1.1 is abstract. Specific instantiations of the problem require a commitment to particular representations of world states, goals, and actions. A wide range of such representations have been developed differing in their *expressive power*, i.e. in the kinds of states, goals, and actions that can be modeled.

- **Theoretical** The computational complexity of planning depends on the specific representational framework chosen. As one might expect, more expressive frameworks make the problem more difficult. However even under very strong limitations on the problem representation, planning is still quite difficult (e.g. PSPACE-complete) [Erol *et al.* 92].

- **Algorithmic** Given the theoretical results, we cannot expect planning algorithms to be able to tractably solve all possible problems. But as with many other theoretically "hard" problems in computer science (e.g. the Traveling Salesman problem) , the worst case results may be largely avoidable in practical applications. Planning researchers have developed heuristic algorithms that are able to effectively solve many classes of planning problems.

Despite the considerable progress to date, classical planning suffers from a significant limitation. This limitation stems from the very definition of planning as the process of *satisfying goals*. In the classical framework, planning is a black-or-white affair: either a plan satisfies the goal, or it does not. But in many realistic would-be applications of planning, the situation is more complex: candidate plans vary more widely in their *quality*. The task of planning in such domains is not to find just any plan, but to find a *good* plan.

Classical planning is unable to draw the necessary distinction between alternative plans because it is lacking crucial information. The agent posing the planning problem has *preferences* between the outcomes that would arise from the execution of various plans. The quality of a plan can only meaningfully be defined with respect to these preferences. Classical planning provides only a meager channel for expression of preferences, equivalent to statements of the form, "I prefer plans that achieve goal G to plans that do not." But agents are likely to have more sophisticated preferences than this, as we will illustrate with the following examples.

1.1.1 Example: the package delivery problem

Consider a simple transportation domain (Figure 1.2) where the agent's goal is to make a delivery from one location to another location by noon. Two roads are available, a direct road, for which a toll is charged, and a longer road, which may be traveled for free. The delivery truck can drive at various speeds, consuming more or less fuel.

What is the "right" plan in this scenario? Obviously, the answer depends on

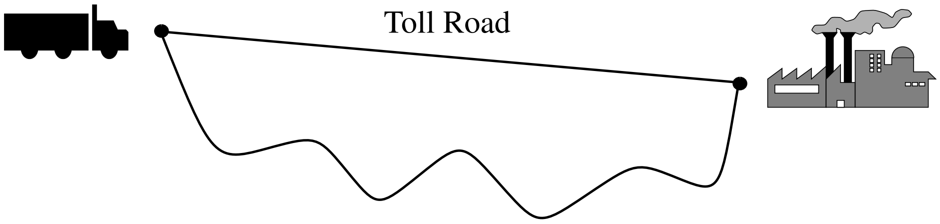

Figure 1.2: A simple transportation planning scenario

parameters as yet unspecified. Some of these are just facts about the domain, such as the actual lengths of the roads, the cost of the toll, the rate of fuel consumption at different speeds, and (if fuel can be purchased) the cost of fuel. But even this information is not enough.

In order to decide the best course of action, a planner would need to know how *important* the delivery is with respect to the value of resources—fuel and money—that might be consumed, and also how valuable those resources are with respect to each other.[2] If the delivery could be made on time regardless of speed, it would certainly be better to drive slowly and conserve fuel. Moreover, if fuel is particularly precious and the delivery is not really so important, the best plan might be to forget about the delivery and do nothing at all.

A related issue is the importance of meeting the 12:00 deadline. What if the delivery were made one minute late? Disaster, or only minor inconvenience? What about an hour late? If the agent still sees some (albeit reduced) value in making a late delivery, the deadline is *soft*. In this case, the planner may need to consider the tradeoff between the expenditure of resources and the timeliness of the delivery. Perhaps it's better to take the longer road and be a bit late than pay an exorbitant toll to be on time.

This example shows two important points. First, although the agent prefers to

[2] Note that the *cost* of fuel is not at all the same as the *value* of fuel relative to money. It might well be that the agent could buy fuel at a cost of $1.00 per gallon, but would strongly prefer having a dollar to having a gallon of gas.

achieve the goal, the agent may also have preferences about other aspects of the world apart from the goal, such as the consumption of resources. Second, the soft deadline is an example of how a goal might be *partially satisfiable*, by admitting more than two degrees of satisfaction. Making the delivery late is not as good as making it on time, but is better than not making it at all.

There is another way in which a goal could be partially satisfied, independent of temporal considerations. Imagine that the agent wishes to deliver two packages to separate locations, but there is not enough fuel to make both deliveries. Should the agent give up entirely? Perhaps making just one of the deliveries would be better than doing nothing at all. In fact, even if sufficient fuel is available, it might still be better to only make one of the deliveries, if making the second delivery would consume more fuel than it was worth.

Thus, a conjunctive goal could be partially satisfied by satisfying just some of its conjuncts. A complete description of the agent's preferences in such a case would need to allow for a description of the importance of achieving each subset of goal conjucts, in relation to each other and in relation to other aspects of the world that the agent might care about.

Transportation logistics problems are good at exhibiting the shortcomings of classical planning. Much of our research has been focused on a domain model of the Truckworld simulated transportation planning testbed [Hanks *et al.* 93], and many of the examples in this dissertation will be drawn from that domain. But similar difficulties also arise in very different domains, as we shall see in the next section.

1.1.2 Example: an information gathering softbot

Imagine a software agent (or "softbot" [Etzioni 93]) with the goal of finding a particular person's phone number and social security number. A planner for this agent might have three alternative plans to consider: to use an online staff directory that can be accessed for free but provides only the phone number, to purchase a credit report containing both pieces of information but costing thirty-five dollars, or to send email to the person asking them for the information. Should the agent go to the expense of obtaining the credit report? Should the agent trouble the person to provide the information? Or should the agent be satisfied with only the phone number?

Again, the "right" plan depends on the specific preferences of the agent. This is another example of a conjunctive goal that could be partially satisfied. To find the best course of action, the planner needs to know the relative importance of knowing just the phone number. If knowing the phone number alone is almost as good as knowing both numbers, then it might be better to find just the phone number than to incur the expense of obtaining the credit report or to bother the person. Also as above, the planner needs to know the importance of the goal in relation to resources that might be consumed. But in this case, there is a very different kind of "resource": the time of the person with whom the software agent is interacting. Since the purpose of software agents is to ease people's lives rather than to complicate them, it seems very likely that we would like to characterize plans in terms of how much they bother people. We also need to relate such effects to other factors. If the person were an important customer, we might prefer spending the money to bothering him, whereas if there person were just a graduate student...

The kinds of temporal considerations raised in the transportation example also apply to software agents. How soon do we need the information about the person? Money aside, purchasing a credit report might take too long to be worthwhile. When we give a software agent virtually any task, we would prefer that it be done reasonably quickly, and most probably there would be some "expiration date" beyond which performing the task is no longer useful.

1.1.3 Goals alone are inadequate

A classical planner couldn't even begin to solve the problems posed in these two examples. The correct solution in both cases is not adequately modeled by just the satisfaction of a goal, but depends on more subtle preferences of the agent about ways the goal could be partially satisfied, and about other aspects of the world besides the goal. The formulation of a classical planning problem is not sufficiently expressive to allow the agent to convey these preferences to the planner. Hence, an adequate solution to these problems must begin with a richer representation for the formulation of planning problems.

1.2 Value-directed planning

Decision theory provides a well-developed mechanism for representing an agent's preferences. If an agent's preferences constitute a partial order over outcomes and satisfy certain rationality criteria (e.g. transitivity), the central theorem of decision theory states that these preferences can be represented by a *value function* V, a real-valued function over outcomes such that for every pair of outcomes o_1 and o_2, $V(o_1) > V(o_2)$ if and only if outcome o_1 is preferable to outcome o_2. Value functions provide a concise and often intuitively appealing method of encoding preferences. The study of multi-attribute value theory investigates the relationship between the structure of the preference ordering and the form of the value function. The related field of decision analysis has developed effective techniques for assessing an agent's preferences and constructing a corresponding value function.

Thus, decision-theoretic value functions would seem to perfectly suit our representational needs; they are guaranteed to be sufficiently expressive, and it is reasonable to expect that we can actually provide them. Based on this foundation, we formulate *value-directed planning* as shown in Figure 1.3. Value-directed planning differs only in the substitution of a value function for classical planning's goal formula.[3] We continue to make the standard simplifying assumptions of classical planning: com-

[3] Note that value-directed planning is strictly more expressive than classical planning. Any classical planning problem may be represented by a value function assigning 1.0 to outcomes in which the goal is satisfied, and 0.0 to other outcomes.

Given:

- a description of the *current world state*,
- a value function V, and
- a set of possible *actions* that can be performed.

Find: a *sequence of actions* maximizing V if performed in the current world state.

Figure 1.3: The value-directed planning problem

plete and correct knowledge of the initial state, complete and correct descriptions of preconditions and effects of actions, deterministic actions, and no exogenous change. However, value-directed planning still constitutes a significant extension to classical planning. In particular, it allows us to represent problems such as the delivery problem described above. The agent can more thoroughly express its preferences to the planner, which can in turn draw finer distinctions between alternative plans. In short, the value function provides a meaningful basis for determining the *quality* of plans.

Note that decision theory does not itself provide a solution to the value-directed planning problem. Although decision theory provides a rich, well-grounded representation for preferences, and provides a clear-cut procedure for deciding between given alternatives, it does not effectively address the *generation* of alternatives.[4] The synthesis of candidate plans (from among infinite possibilities) has been the hallmark of classical planning.

1.2.1 A question of tractability

The proposal to use decision-theoretic techniques is not new to A.I. (e.g. [Feldman& Sproull 77, Haddawy&Hanks 90, Wellman 93]), but the planning comunity has generally been slow to adopt such representations. The common belief has been that

[4] Although Keeney [1992] begins to move the decision theory camp in this direction.

classical planning is hard enough already, and that a decision-theoretic approach such as value-directed planning could only be much harder. The complexity results support this belief; since value-directed planning subsumes classical planning, it has to be at least has hard *in the worst case.* The widely felt intuition, though, is that even in the general case, finding the *best* plan (with respect to a value function) must be harder than finding just *any* plan (that satisfies a goal). But, as we shall see, this is not necessarily true.

The value-directed planning problem shown in Figure 1.3—like the earlier definition of the classical planning problem—is an abstract framework, a paradigm within which one could instantiate specific planning problems. Actual planners would vary in their representations for the various components of the planning problem. The difficulty of value-directed planning depends strongly on the particular representation used for the value function.

Consider classical planning as a degenerate special case of value-directed planning. The algorithmic techniques used by classical planners derive computational leverage by exploiting the restricted structure of the value function. Goal regression, for example, depends on the fact that *only* plans satisfying the goal might maximize value. The use of satisficing search is justified by the fact that *any* plan satisfying the goal maximizes value. These techniques, in conjunction with domain-independent and domain-specific heuristics, allow classical planning systems to solve broad classes of problems.

At the opposite extreme, consider attempting to find the optimal plan for an arbitrary value function, which might assign any value to any outcome. With no knowledge of the function's structure, the planner could do no better than to measure the value of every possible plan. Since most interesting domains admit an infinite number of possible plans, finding an optimal plan would be impossible.

Hence, an effective value-directed planning system must use a representation for the value function that strikes a balance between expressive power and computational tractability. The representation must be sufficiently rich to capture the kinds of preference models that arise in those domains to which the planning system will applied. At the same time, it must retain sufficient structure to allow the creation of heuristically effective planning algorithms.

1.3 The PYRRHUS *value-directed planning system*

PYRRHUS[5] is a fully-implemented value-directed planning system. An overview of the major components of PYRRHUS, and their relationship, is shown in Figure 1.4. (The numbers in brackets are the chapters in which those modules are described.)

PYRRHUS supports two different representations for value functions based on Haddawy and Hanks' theory of goal-directed utility models [Haddawy&Hanks 93]. These value functions allow the representation of partially-satisfiable deadline goals and partially-satisfiable conjunctive goals. They also allow the expression of non-goal preferences in terms of resources that a plan might consume or produce. The value functions supported by PYRRHUS are sufficient to represent problems typified be the two examples above, but at the same time they have important structural characteristics that allow the construction of effective planning algorithms.

PYRRHUS uses R-STRIPS, an extension of the classical STRIPS schematic action language. R-STRIPS schemas support metric (real-valued) resources such as *time*, *money*, or *fuel-level* in the preconditions and effects of actions. (We do not consider the resource reasoning techniques used by PYRRHUS to be a significant research contribution *per se*; rather they are the minimal extentions to STRIPS that provide the requirements of our value functions.)

---

[5] Named for the father of value theory, Pyrrhus of Epirus.

> Pyrrhus, when his friends congratulated to him his victory over the Romans under Fabricius, but with great slaughter of his own side, said to them, "Yes; but if we have such another victory, we are undone."
>
> —Francis Bacon, Apothegms No. 193.

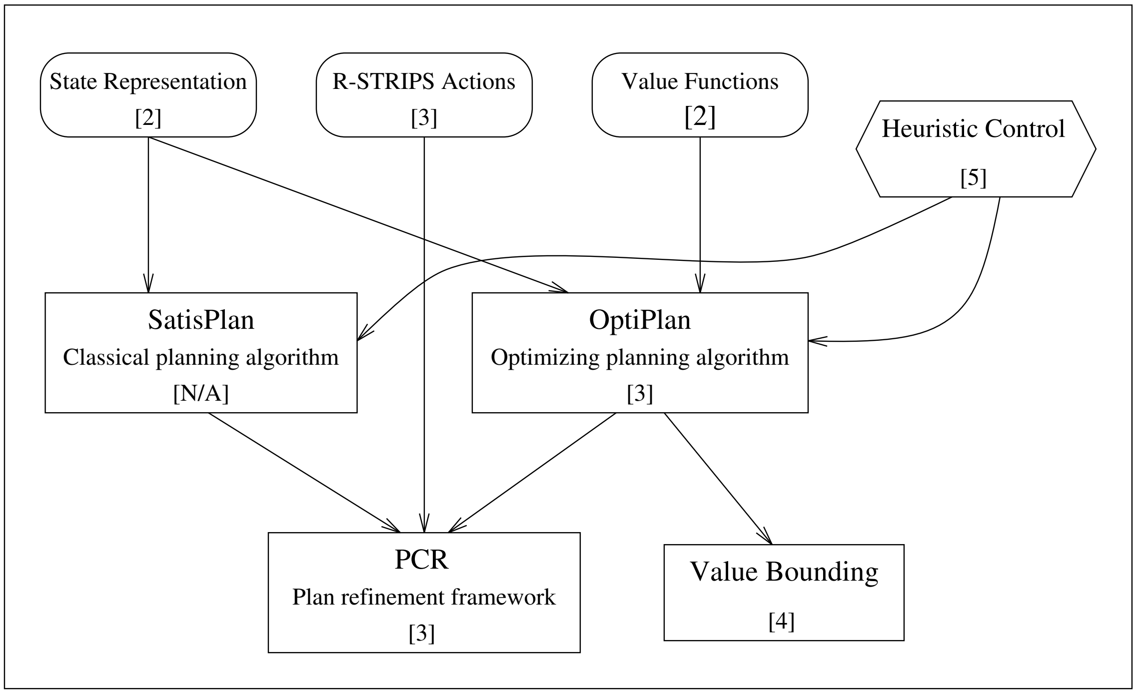

Figure 1.4: An overview of the PYRRHUS planning system

PYRRHUS performs value-directed planning by combining branch-and-bound optimization with techniques developed in the field of A.I. planning. The application of branch-and-bound to any problem requires a method for representing partial solutions and "branching" them; effective methods for doing just that have been developed in the context of classical planning. The planning algorithm used by PYRRHUS is decomposed into several parts. OPTIPLAN is an abstract, branch-and-bound planning algorithm. It assumes the existence of an independent underlying plan-refinement framework which can generate a complete space of partial plans for a given problem. Many different approaches to plan refinement are possible, but PYRRHUS uses PCR, a framework based on the partial-order, causal-link planning algorithms found in classical planning. OPTIPLAN also assumes the existence of a module which can return bounds on the value of a partial plan.

PYRRHUS allows heuristic control of the optimization algorithm by both domain independent and domain specific means. Many such techniques have been developed for classical planning systems. PYRRHUS includes several novel techniques which take advantage of the additional information available in the value function.

For purposes of empirical comparison, PYRRHUS also includes SATISPLAN, a classical, goal-satisfying planning algorithm for R-STRIPS. SATISPLAN performs a heuristic, best-first search of the plan space provided by PCR. SATISPLAN and PCR taken together could be thought of as an extension of SNLP [McAllester&Rosenblitt 91] to the R-STRIPS action language.

1.4 What's in this thesis?

This thesis describes an investigation of some representational, algorithmic, and empirical aspects of the paradigm of value-directed planning as embodied by the PYRRHUS planning system.

Chapter 2 presents two different representations for value functions supported by PYRRHUS. Chapters 3, 4, and 5 describe PYRRHUS, a planning system that supports the value functions described in Chapter 2. Chapter 3 defines the R-STRIPS action language, and describes the central planning algorithms used by PYRRHUS: OPTIPLAN, the abstract optimization algorithm, and PCR, a particular partial-order, causal-link refinement framework. Chapter 5 describes the heuristic control subsystem. It presents some domain-independent strategies for making certain heuristic decisions and evaluates them empirically. It also describes a mechanism for the incorporation of domain-specific heuristic knowledge.

Chapter 6 relates an empirical investigation of the relative difficulty of value-directed and corresponding classical planning problems. It shows how the difficulty of a value-directed planning problem depends on various characteristics of the value function. Chapter 7 concludes by suggesting avenues of future research, pointing out related work, and summarizing the contributions of this thesis.

Chapter 2

GOAL-DIRECTED VALUE FUNCTIONS

2.1 Introduction

In Chapter 1 we saw examples of how the goal formulas heretofore used by A.I. planning systems are insufficient to capture certain important aspects of realistic planning problems. Our proposed solution is to represent a planning problem as a *value function* to be optimized, rather than just a goal formula to be satisfied. Value functions provide a strictly richer representation; they can capture the symbolic goals that have been so useful to planning, while providing further essential information about:

- the relative importance of satisfying the goals;
- the temporal scope of the goals;
- ways in which the goals can be partially satisfied; and
- preferences about other effects the plan might have, such as the consumption of resources.

This chapter will describe two different kinds of value functions that PYRRHUS supports.

Value functions[1] have a long history in the economics literature. A good primer and historical perspective can be found in [Fishburn 78]. Essentially, a value function

[1] Value functions are also known variously as "utility functions," "preference functions," "worth functions," etc. Following the convention of [Keeney&Raiffa 76], we will use the term "value function" when describing preferences over deterministic outcomes, and reserve the term "utility

is a real-valued function $\mathcal{V}$ over *outcomes* describing an agent's preferences over those outcomes. That is, $\mathcal{V}(o_1) > \mathcal{V}(o_2)$ if and only if the agent would prefer outcome o_1 to outcome o_2. It can be shown that as long as the agent's preferences satisfy certain rationality criteria (e.g. transitivity within the preference structure), such a value function will exist [Fishburn 70]. Further, there is a rich literature within the decision theory community on techniques for assessing preferences and constructing value (or utility) functions (e.g. [von Neumann&Morgenstern 47, Krantz *et al.* 71, Keeney&Raiffa 76, Zhong *et al.* 90, Moon&O'Connor 93]). Thus, we can suppose both that value functions will be sufficiently expressive for our needs, and that it is realistic to expect that value functions can actually be constructed for our planning problems.

Haddawy and Hanks [1993] identify a class of *goal-directed utility functions* that provide the additional expressive power mentioned above, while at the same time—in theory—retaining sufficient structure to support effective planning algorithms. They go on to describe several subclasses of the general class of goal directed utility functions that represent various specific kinds of goals.

Pyrrhus implements two different value functions based on utility functions described by Haddawy and Hanks. These value functions should be seen as an *extension* rather than an *alternative* to the notion of a goal as traditionally used in planning. Each value function will contain as part of its specification a particular goal formula, as well as additional information about the importance of the goal, ways in which the goal can be partially satisfied, preferences about non-goal aspects of the domain, etc.

Pyrrhus's representations of value constitute a tradeoff between expressive power and computational tractability. The value functions place various restrictions on the kinds of preference structures that can be modeled. From the perspective of representational power alone, such restrictions can only be seen as detrimental. But in the following chapters, which describes the planning algorithm, we will see how

function" to describe preferences over nondeterministic outcomes (i.e. preferences over lotteries over outcomes).

these restrictions support efficient planning. In as much as these restrictions are much less severe than those implicit in classical planning (i.e. they allow representation of a large group of interesting and realistic problems), we feel that they are acceptable.

The remainder of this chapter will formally describe the the general family of goal-directed value functions, and then the two different specific members of this family supported by PYRRHUS.

2.2 Terminology: Attributes, states, outcomes

Before describing the value functions themselves, we begin with the introduction of some supporting terminology. Most importantly, we define *outcomes*, the entities over which value functions are defined. An outcome is a sequence of *world states.* (From the planning perspective an outcome is the natural result of executing a plan, but we will defer the precise definition of "plans" and their relationship to outcomes until Chapter 3. This chapter will deal with outcomes as abstract entities.) World states are described in terms of *attributes*, which may be *propositional* or *metric.*

The attributes making up a state description are composed from a set of *object symbols*, a set of *relation symbols*, a set of *resource symbols*, and the set of real numbers. The sets of symbols employed will depend on the particular domain being modeled.

Definition 1 (Propositional attribute) *A* propositional attribute *is a formula*

$$\pi(\alpha_1, \alpha_2, \ldots)$$

where the α_i *are symbols denoting objects in the domain, and* π *is a symbol denoting some relation over those objects.*

Propositional attributes are used to describe logical relationships between objects in the world. Typical propositional attributes in a transportation domain might be *at(Truck, Depot)* or *loaded-in(Truck, Carton-7)*. A software domain might have propositional attributes such as *login-name(Person-12, Mikew)*.

Definition 2 (Metric attribute) *A* metric attribute *is a formula*

$$\langle \rho, l \rangle$$

where ρ is a symbol denoting some resource in the domain, and l is a real number denoting the amount *or* level *of that resource.*

Metric attributes are used to describe numerically-valued, time-varying characteristics of the domain.[2] Examples from a transportation domain are $\langle$*fuel-level*$, 10\rangle$, or $\langle$*bank-balance*$, 50000\rangle$. In a software domain, we might want to model the amount of interaction time with a human being required by a plan, which could be represented by a metric attribute like $\langle$*human-time*$, 15\rangle$. The units in which metric attributes are measured are implicit in the domain descriptions; any appropriate unit could be used (e.g. liters, US dollars, seconds).

Definition 3 (World state) *A* world state *is an ordered pair $S = \langle S_p, S_m \rangle$ where:*

- *S_p is a set of propositional attributes, and*
- *S_m is a set of metric attributes, one of which is for the distinguished resource* clock-time*.*

World states are assumed to constitute a complete description of the world (i.e. we make the traditional closed-world assumption [Reiter 78]). The *clock-time* attribute represents the point in time (in some fixed frame of reference) described by that world state.

Definition 4 (Outcome) *An* outcome *is a sequence of world states $(S_0, S_1, \ldots, S_n)$.*[3] *S_0 is the* initial state *of the outcome, and S_n is the* final state*. The value of the* clock-time *attribute of each state must be strictly increasing over the sequence.*

---

[2] The term *fluent* has also been used in this context.[McDermott 82]

[3] The term "chronicle" has also been used to refer to such sequences of world states [McDermott 82].

Note that an outcome only describes the state of the world at some finite number of discrete points in time. The state of the world at times between those specified in an outcome is undefined.

2.3 Goal-directed value functions

It is undeniable that intelligent agents act in a goal-directed manner. But this does not mean that slavish pursuit of goals is by itself a satisfactory model of intelligent agency. In other words, goal-directedness is necessary, but not sufficient, for intelligence. Agents have preference about the world apart from their goals, and their goals may have relatively greater or lesser importance in relation to these other preferences. A planning system that was informed only of one's goals—but not of any of one's other preferences—would be quite unlikely to produce an acceptable plan.

We can remedy this problem by formulating a planning problem in terms of a *goal-directed value function* instead of merely a symbolic description of the goal. Goal-directed value functions augment symbolic goal descriptions with additional information about an agent's preference structure. Following the theoretical framework of Haddawy and Hanks [1993] we define this class as follows:

Definition 5 (Goal-directed value function) *A* goal-directed value function *is a real-valued function over outcomes of the form:*

$$\mathcal{V}(o) = k_g V_G(o) + k_r V_R(o)$$

This value function consists of two additive components. The goal value component consists of a numeric *goal value coefficient*, k_g, and a *degree-of-satisfaction* function, V_G. The residual value component similarly consists of a numeric *residual value coefficient*, k_r, and a *residual value function*, V_R. The degree-of-satisfaction function represents the degree to which the goals are satisfied in a particular outcome. It's range varies from zero (in the case of a goal that is not satisfied at all) to one (in the case of a completely satisfied goal). The residual value function represents the intrinsic worth of an outcome with respect to an agent's background (non-goal) preferences. The structure of the residual value function, and the attributes upon which it depends, will vary depending on the particular domain.

This class of value functions provides several desirable features lacking in classical goal formulas:

- **Partial satisfiability and temporal scope** The degree-of-satisfaction function encodes goals in a way that allows them to be partially satisfied. PYRRHUS supports two different kinds of degree-of-satisfaction functions, described in detail in the following sections. One particular way that a goal can be partially satisfied is for the goal's value to depend on the *time* at which the goal is achieved. Thus, the degree-of-satisfaction function can represent goals with temporal scope.

- **Regard for other preferences** The residual value function allows encoding preferences about other effects that the plan may have apart from satisfying the goal. Section 2.6 will describe in detail the class of residual value function that PYRRHUS supports.

- **Relative value of goals** The ratio of the two numeric coefficients (k_g and k_r) determines the relative importance of the goal with respect to other factors, such as the consumption of resources.

PYRRHUS supports two different specific instances of the class of goal-directed value functions by providing two different degree-of-satisfaction functions, either of which can be combined additively with a specific residual value function to create a particular goal-directed value function. The following sections will describe the two degree-of-satisfaction functions and the residual value function that PYRRHUS employs.

2.4 Partially satisfiable deadlines goals

Partially satisfiable deadline goals are an extension of simple propositional goal formulas that allow the representation of additional information about temporal scope. In particular, they allow the specification of soft deadlines, that is, deadlines after which the value of achieving the goal begins to decay. Hard deadlines, after which there is *no* value in achieving the goal, are a degenerate special case of partially satisfiable deadline goals.

A partially satisfiable deadline goal is specified by providing a propositional goal formula, and a soft deadline function. The degree-of-satisfaction function for a partially satisfiable deadline goal is given in the following definition.

Definition 6 (Partially satisfiable deadlines goals) *Given:*

- *a propositional goal formula G*
- *a* soft deadline D_G*, which is a monotonically non-increasing function from clock-time onto the interval* $[0 \ldots 1]$

A partially satisfiable deadline goal *is represented by a degree-of-satisfaction function of the form:*

$$V_G(o) = G_s(o) \cdot D_G(G_t(o))$$

where $G_s(o)$ is a function returning 1 if G is satisfied in the final state of o and 0 otherwise, and $G_t(o)$ is a function returning the clock time of the earliest state in o such that G is satisfied in that and all subsequent states.

The soft deadline function D_G may be any real-valued, monotonically non-increasing function from clock time to the unit interval. It implicitly defines a distinguished time

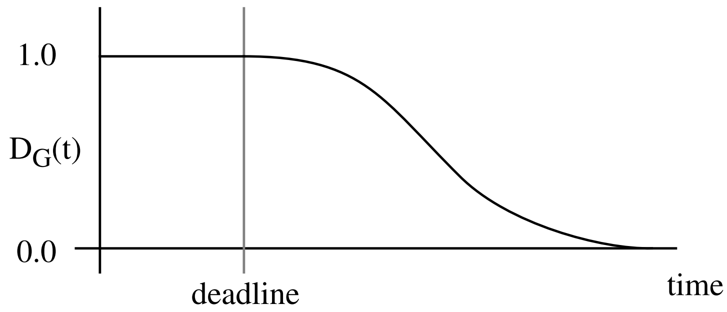

Figure 2.1: An example of a soft deadline function

point, the *deadline*, which is the latest point in time at which D_G takes on the value 1.0.[4]

In a transportation domain, for example, we might have a partially satisfiable deadline goal specified by the goal formula

$$G = \mathit{at(Apples,\ Warehouse)} \wedge \mathit{at(Pears,\ Warehouse)}$$

and the soft deadline function D_G as shown in Figure 2.1. A hard deadline, on the other hand, could be specified by

$$D_G(t) = \begin{cases} 1.0 & \text{if } t \leq \mathit{deadline} \\ 0.0 & \text{otherwise} \end{cases}$$

2.5 Partially satisfiable conjunctive goals

A symbolic goal that is conjunctive presents the possibility of partial satisfaction by the satisfaction of some (but not all) of its conjuncts. Given the goal $G = P \wedge Q \wedge R$,

[4] Our soft deadline functions differ from the temporal weighting coefficients of Haddawy and Hanks because we adopt the persistence assumptions from classical planning: that anything achieved by the agent will hold indefinitely. In this case, the "earliest time at which satisfying the goal has value to the agent" (Haddawy and Hanks' semantics for "deadline") becomes vacuous. We consequently use the term "deadline" in the more conventionally intuitive manner to refer to the *latest* time at which full value is obtained. Terminology aside, our representations are equivalent (given the persistence assumption).

for example, there might be some lesser value in achieving just P, or Q, or $P \wedge R$, or any other conjunction of some subset of the conjuncts of G.

A goal with k conjuncts will have 2^k different possible truth assignments to the conjuncts, and so in general could have 2^k different degrees of satisfaction. But often, some truth assignments to the conjuncts of G will not be differentiable with respect to the value function. In the above example, it might be that if P is not satisfied, then the state of Q and R do not matter. Such indifferentiable truth assignments can be lumped together into equivalence classes, and described collectively by formulas including only some of the conjuncts. For example, "$\neg P$" describes the abovementioned set of all truth assignments in which P is not true (which includes half of all possible truth assignments to the conjuncts of G). We will refer to such equivalence classes as *fractions* of G. Each fraction is defined by a *fractional goal formula*, and the fractions of a goal must be mutually exclusive and exhaustive.

Definition 7 (Partially satisfiable conjunctive goal) *Given:*

- *a propositional goal formula* $G = g_1 \wedge g_2 \wedge \ldots \wedge g_k$
- *a set of* fractional goal formulas, $G^* = \{G_1, G_2, \ldots, G_n\}$, *where the elements of* G^* *mutually exclusively and exhaustively partition the possible truth assignments of the* g_i
- *a* fractional value function $F_G : G^* \rightarrow [0, 1]$

A partially satisfiable conjunctive goal *is represented by a degree-of-satisfaction function of the form:*

$$V_G(o) = F_G(W_{G^*}(o))$$

where $W_{G^*}(o)$ *is a function returning the element of* G^* *that holds for the final state of the outcome* o.

Table 2.1: An example fractional value function

Members of G^*	Value of F_G
know(Person-9, Phone-number) $\wedge$ know(Person-9, SSN)	1.0
know(Person-9, Phone-number) $\wedge$ $\neg$know(Person-9, SSN)	0.8
$\neg$know(Person-9, Phone-number)	0.0

In our software domain, for example, we might have the goal formula

$$G = \text{know(Person-9, Phone-number)} \wedge \text{know(Person-9, SSN)}$$

(meaning that we want to know the phone number and social security number of a given person[5]) with the fractional goal formulas and fractional value function as shown in Table 2.1. This would specify a value function in which knowing just the person's phone number is nearly as good as knowing both the phone number and the social security number.

2.6 Residual value

One of the main limitations of goal formulas as a means of expressing planning problems is that they don't allow a distinction to be made between various outcomes in which the goal is satisfied but which differ in regard to other attributes. Following the suggestion of Haddawy and Hanks [1993], PYRRHUS allows the expression of preference between such differing outcomes through the *residual value* component of the value function.

Residual value is used to encompass whatever is important to the agent aside from the satisfaction of the particular goal. For example, while an agent may have the goal of delivering a package to a particular destination by a certain time, it would prefer

[5] This is a simplistic approach to representing "state of knowledge" in a propositional logic. A more principled approach would be to use a modal logic of knowledge (e.g. [Moore 85, Morgenstern 88, Etzioni *et al.* 92]), but bear with us for the course of this example.

to do so while consuming as little money and fuel as possible. Such attributes, which we will call *assets*, would thus constitute the basis for the residual value function.

From a theoretical standpoint, one might ask whence assets derive their value. There are two possible semantic interpretations of residual value. The first is that assets have *inherent* value, that is, they simply arise from the underlying preference structure in just the same way that, for example, goals do. (See [Wellman&Doyle 91] for a discussion of how goals can arise from a preference structure.) A second interpretation is that the residual value of assets is *derived* from the expected ability to satisfy future goals. Under this interpretation, attributes such as money have no value *per se*, they are only valuable in as much as one expects to have certain goals in the future which the possession of money will allow one to satisfy.

One might wonder whether all preferences between outcomes couldn't simply be represented as part of a symbolic goal formula. Indeed they could, but to do so would require a goal language sufficiently expressive to represent—at the very least—minimization and maximization (e.g. "Deliver the package and use as little fuel as possible."). A numeric value function, as employed by PYRRHUS, is probably the most perspicuous method for representing such preferences.

We now formalize several underlying concepts, beginning with the notion of *assets* and *liabilities*, the fundamental elements upon which residual value will ultimately be defined.

Definition 8 (Assets and liabilities) Assets *and* liabilities *are a subset of the metric attributes in a domain, the levels of which enter into the agent's preference structure such that, all other things being equal:*

- *the agent never prefers to have less than more of any individual asset, and*
- *the agent never prefers to have more than less of any individual liability.*

Assets, then, are things that the agent values having, such as *bank-balance* or *fuel*, while liabilities are things that the agent values *not* having, such as *credit-card-balance* or *nuclear-waste*. We will often simply use the term "assets" to refer to both assets and liabilities, unless the distinction is important.[6]

Definition 9 (Asset position) *If there are N assets, then an* asset position *is a numerically-valued N-vector whose elements represent the quantities of each of the assets (and liabilities) in the domain (in some canonical order).*

In a software domain, one might designate the metric resource *money* to be an asset and *human-time* to be a liability. An asset position in this domain would be any vector (m, h) representing m units of money and h units of human time.

An agent describes its background, non-goal related preferences by providing a real-valued function over asset positions.

Definition 10 (Asset position value function) *An* asset position value function *is a real-valued function defined over asset positions that is monotonically nondecreasing with respect to the amount of each individual asset and monotonically nonincreasing with respect to the amount of each individual liability.*

Conceptually, the asset position value function is similar to the soft deadline function, D_G, except that whereas D_G is defined over a one-dimensional space (i.e. time), an asset position value function is defined over an N dimensional space (where N is the number of assets).

Figure 2.2 shows an example of an asset position value function defined over the assets *money* and *fuel-level*. Although this example shows a linear combination of the levels of the assets, this need not be the case. The asset position value function need

[6] In fact, any preference structure over a domain with both assets and liabilities could be syntactically converted to an equivalent preference structure over a domain with only assets by replacing each liability with an "anti-liability", an asset that is the arithmetic inverse of the original liability.

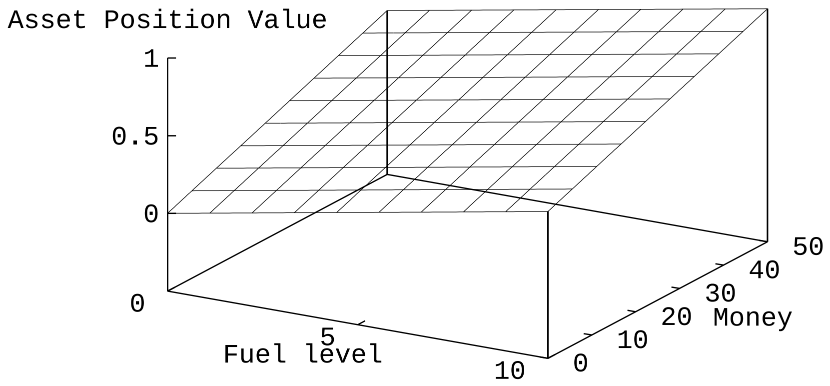

Figure 2.2: An example of an asset position value function

not even be continuous. The only restriction is that of monotonicity with respect to each individual asset.

One could assess an asset position with respect to any particular world state. An outcome, being a sequence of world states, defines a corresponding sequence of asset positions through time. As a basis for the representation of residual value, however, we will use only the latest state in the outcome:

Definition 11 (Asset position of an outcome) *Given an outcome o, the asset position of that outcome (denoted $\vec{A}(o)$) is the asset position made up of the values of each of the assets in the final world state of o.*

It might seem to be a limitation that the asset position of an outcome is defined just by the level of the assets in the final state rather than their levels throughout the sequence of states that make up the outcome. But we feel this restrictive definition to be quite reasonable in light of the interpretation of residual value being derived from an estimation of the ability to satisfy future goals. In this context, only the final asset position has any meaningful value, and intermediate asset levels are irrelevant.

Pyrrhus constructs its measure of residual value based on an agent's preferences, as specified by an asset position value function.

Definition 12 (Residual Value Function) *Given an asset position value function $\widehat{V_R}$, a* residual value function *is a function over outcomes of the form:*

$$V_R(o) = \widehat{V_R}(\vec{A}(o))$$

where $\vec{A}$ is a function returning the asset position of an outcome.

It is important to note that, unlike goal value, residual value is not necessarily bounded either below or above. Many assets that we would like to model can grow to arbitrarily high levels, and in many of those (e.g. money) residual value continues rise. Given such a value function, there will not necessarily *be* an "optimal outcome", that is, an outcome (with a finite number of states) which maximizes value. This issue will be dealt with at length in Section 4.3.

2.7 Summary

This section presented the representation for two different goal-directed value functions used by PYRRHUS. Goal-directed value functions overcome a significant limitation of goal formulas as used by classical planning. They allow representation of partially satisfiable goals and temporally scoped goals. They also allow the representation of other (non-goal) preferences, and the relative importance of the goals compared to these other preferences.

Value functions are defined over outcomes, which are sequences of world states. Each world state is described by sets of propositional attributes (describing logical relations between objects in the domain) and metric attributes (describing the numeric levels of time-varying resources in the domain).

The value functions used by PYRRHUS are each the sum of a goal value term and a residual value term. PYRRHUS supports two different kinds of goal value functions, one representing partially satisfiable deadline goals, and the other representing partially satisfiable conjunctive goals. A partially satisfiable deadline goal is parameterized by a propositional goal formula and a soft deadline function. The parameters for a partially satisfiable conjunctive goal are a propositional goal formula, a set of fractional goal formulas, and a fractional value function. Each of these two goal functions can be combined with a common residual value function, which models background, non-goal preferences in terms of *assets*.

Chapter 3

PYRRHUS: A VALUE-DIRECTED PLANNING SYSTEM

3.1 Introduction

As outlined in Chapter 1, we are proposing *value-directed planning* as a solution to some serious shortcomings of the traditional approach to planning in A.I. In the previous chapter we defined two specific kinds of goal-directed value functions that improve upon the classical notion of symbolic goals. But these new representations are of little use without accompanying algorithms. What good is the ability to *pose* a value-directed planning problem unless one can go on to *solve* it?

PYRRHUS is a fully-implemented planning system which finds optimal plans for the two kinds of value functions described in Chapter 2.[1] PYRRHUS supports a STRIPS-like action representation, extended to include metric attributes as defined in the last chapter. In this chapter we describe the central algorithm used by the PYRRHUS, which uses a branch-and-bound technique to find optimal plans for goal-directed value functions. The algorithm is similar to existing least-commitment planning algorithms: it uses regression from the "goal" component of the goal-directed value functions to define a space of partial plans which must contain all feasible plans of greater value than the status quo.

The description of the planning algorithm is broken up into two parts. OPTIPLAN is an abstract optimization procedure. It assumes the existence of an independent, underlying plan refinement framework which can generate a complete space of partial plans. PCR is the particular plan refinement framework used by PYRRHUS. It draws

[1] Subject to some limitations described in Chapter 4.

on existing work in partial-order, causal-link, least-commitment planning.

This chapter first describes "inputs" (action representation) and "outputs" (plan representation) used by PYRRHUS and formally defines a *planning problem.* It then describes the OPTIPLAN optimal planning algorithm and the PCR plan refinement framework.

3.2 Actions, plans, and the value-directed planning problem

Before an algorithm can be presented as a solution to a problem, the problem itself must be well defined. This section introduces some essential terminology and notation, and then formally defines the specific *value-directed planning problem* addressed by PYRRHUS.

3.2.1 Action representation

Part of the formulation of a planning problem is the specification of the possible actions from which plans can be composed. Traditionally, the possible actions have been described in terms of:

- **Preconditions** — the conditions which must entail in the world when the action is executed in order for it to have its intended effects.

- **Effects** — how the world will change upon the execution of this action.

Most planning systems support a *schematic* (or "lifted" [McAllester&Rosenblitt 91, Robinson 65]) representation for actions, in which parts of the description are replaced with variable parameters. Such action schemas can concisely represent large sets of possible actions. An action schema is instantiated into a ground action by binding each variable to an object from the world; each possible set of bindings represents a different possible action. (We will adopt a common syntactic convention of denoting variables with a leading question mark, e.g. *?object.*)

The STRIPS action representation [Fikes&Nilsson 71] has been widely studied over the past several decades. In STRIPS, preconditions are represented by a list of propositional attributes, denoting that the conjunction of those propositional attributes must hold for the action to work.[2] The effects of an action are represented by an "add list" and a "delete list," which are lists of propositional attributes made true and false (respectively) by execution of the action. Many modern incarnations of the STRIPS representation combine the add and delete lists together by marking the falsified propositional attributes with a negation symbol (e.g. $\neg$*at(Truck, Depot)*).

PYRRHUS uses a variant of the STRIPS action representation. Our representation of propositional preconditions and effects is relatively conventional; they are both represented by conjunctions formed of propositional attributes.

Definition 13 (Propositional conjunctions) *A* propositional conjunction *is a conjunction of possibly negated propositional attributes. Given a propositional conjunction c, $Pos(c)$ denotes the set of non-negated propositional attributes of c, while $Neg(c)$ denotes the set of propositional attributes that appear negated in c.*

Since PYRRHUS supports metric attributes (as described in Chapter 2), the action representation must be extended to support them as well. We treat metric attributes much like propositional attributes in the way they enter into an action's preconditions and effects. The difference, of course, is that while propositional preconditions are logical conjunctions, metric preconditions are sets of numeric inequalities (e.g. $0 \leq$ *fuel-level* ≤ 10). And while propositional effects are logical assertions, metric effects describe the *change* in the levels of attributes.[3] Since an action schema represents

[2] Disjunctive preconditions can be accomplished by duplicating the action schema for each disjunct.

[3] There has been a significant body of work dedicated to the support of metric resources in planning systems (e.g. [Wilkins 88, Currie&Tate 91, Penberthy 93, Lansky&Philpot 94]. PYRRHUS does not constitute a significant advance in this area; many existing systems provide much richer representations for the production and consumption of metric resources. But, since metric attributes are central to the notion of plan quality, we deem them necessary. Our aim has been to provide the minimal extension to a classical (propositional) representation that allows us to capture the kinds of problems that motivate our main research interest.

many possible actions which may have differing metric preconditions and effects, our representation must allow the preconditions and effects to depend on the bindings of variables in the action schema. We accomplish this by using functions over the space of possible bindings for the variable parameters.

Definition 14 (Metric precondition) *Given a list of variable parameters* $(p_1, p_2, \ldots, p_n)$, *a* metric precondition *is a formula*

$$\langle \beta, F_p \rangle$$

where β *is a resource, and* F_p *is a* metric precondition function *defined over all possible bindings of* $(p_1, p_2, \ldots, p_n)$ *and returning an interval over the real numbers.*

Semantically, such a precondition is meant to encode that the specified resource is *within* the real-valued interval returned by the given function. For example, a *drive* action schema might have the parameter list (*?from*, *?to*), and the metric precondition:

$$\langle \textit{fuel-level}, \textit{Fuel-needed-fn} \rangle$$

Fuel-needed-fn is a function over possible bindings of *?from* and *?to*, and returns an interval specifying the required *fuel-level* for driving between any pair of locations. If driving from the depot to the warehouse requires 6 gallons of fuel, and the maximum capacity of the fuel tank is 15 gallons, then

$$\textit{Fuel-needed-fn}(\textit{Depot}, \textit{Warehouse}) = [6, 15]$$

Note that this interval does *not* indicate an uncertainty about the amount of fuel consumed, which is deterministic. Rather, the interval denotes the range of legal values of *fuel-level* for which the particular drive action can be performed.

Definition 15 (Metric effect) *Given a list of variable parameters* $(p_1, p_2, \ldots, p_n)$, *a* metric effect *is a formula*

$$\langle \beta, F_e \rangle$$

where β is a resource, and F_e is a metric effect function *defined over all possible bindings of* $(p_1, p_2, \ldots, p_n)$ *and returning a real number.*

Metric effects are similar in nature to metric preconditions, except that the associated function returns a specific real number rather than an interval. This number indicates the change to the level (or quantity) of the specified resource. In the above example, the *drive* action would have a metric effect like

$$\langle \textit{fuel-level}, \textit{Fuel-used-fn} \rangle$$

where *Fuel-used-fn* is a metric effect function defined over the possible bindings of the parameters. For the case above,

$$\textit{Fuel-used-fn}(\textit{Depot}, \textit{Warehouse}) = -6$$

indicating that six gallons of fuel would be consumed.

During the course of planning, it is sometimes necessary to assess the metric preconditions and effects of an action for which the variable parameters are not all bound to constants. We extend the domain of the metric precondition and effect functions to include unbound variable parameters as follows. If B is the set of variable parameters of some action (some of which are unbound), then any metric precondition or effect function *MF* of that action will return an interval:

$$MF(B) = \left[min_{B' \in Gr(B)}[MF(B')], max_{B' \in Gr(B)}[MF(B')]\right]$$

where $Gr(B)$ returns the set of all possible instantiations ("groundings") of the variables in B.

Putting these components together, we arrive at PYRRHUS's representation for action schemas, which we call "R-STRIPS".

Definition 16 (R-STRIPS Action schema) *An* R-STRIPS action schema α *is a sextuple*

$$\alpha = \langle \alpha_n, \alpha_v, \alpha_{pp}, \alpha_{mp}, \alpha_{pe}, \alpha_{me} \rangle$$

```
(operator drive (?from ?to ?speed)
    :propositional-precondition (:and (truck-at ?from)
                                      (road-connecting ?from ?to)
                                      (speed ?speed))
    :metric-preconditions (fuel-level #'fuel-required-fn)
    :propositional-effect (:and (:not (truck-at ?from))
                                (truck-at ?to))
    :metric-effects (:and (fuel-level #'fuel-consumed-fn)
                          (clock-time #'drive-duration-fn)))
```

Figure 3.1: The `drive` action schema from the Truckworld domain.

where

- α_n *is a symbolic* name*,*
- α_v *is a list of* variable parameters*,*
- α_{pp} *is a* propositional precondition *(a propositional conjunction),*
- α_{mp} *is a set of* metric preconditions*,*
- α_{pe} *is a* propositional effect *(a propositional conjunction), and*
- α_{me} *is a set of* metric effects.

The propositional conjunctions in the preconditions and effects may include elements of α_v *as well as object symbols.*

As an example, Figure 3.1 shows an R-STRIPS action schema from the Truckworld domain in the LISP-like syntax used by PYRRHUS. In this schema, *fuel-required-fn*, *fuel-consumed-fn*, and *drive-duration-fn* are functions that are defined over the possible binding of *?from*, *?to*, and *?speed.*

Definition 17 (Ground action) *A* ground action *is an* R-STRIPS *action schema in which all variables have been bound to object symbols in the domain. The instantiated* *parameters of a ground action α are denoted as α_b.*

A ground action represents a mapping between world states. However, this mapping is only defined over those states in which the action is *viable*, i.e. the preconditions of the action are satisfied. In addition to the preconditions, if an action has a metric effect on some resource, it will only be viable in states which have *some* attribute for that resource.[4]

Definition 18 (Viability of an action) *An action α is* viable *in a state S just in case:*

$$Pos(\alpha_{pp}) \subset S_p$$

$$Neg(\alpha_{pp}) \cap S_p = \emptyset$$

$$\forall(\langle r, F_p\rangle \in \alpha_{me})\exists(\langle r', level\rangle \in S_m) : r = r'$$

and

$$\forall(\langle r, F_p\rangle \in \alpha_{mp})\exists(\langle r', level\rangle \in S_m) : r = r' \wedge level \in F_p(\alpha_b)$$

Given world state and a viable ground action, we can describe the effect of executing that action as a transformation to another state.

Definition 19 (Action Execution) *The* execution *of a viable action α in a world state S (written $\Omega_a(\alpha, S)$) is a new world state $S' = \langle S'_p, S'_m\rangle$ such that*

$$S'_p = Pos(\alpha_{pe}) \cup S_p \setminus Neg(\alpha_{pe})$$

and

$$S'_m = \{\langle r, l + \Delta(r, \alpha)\rangle | \langle r, l\rangle \in S_m\}$$

where

$$\Delta(r, \alpha) = \begin{cases} F_e(\alpha_b) & \text{if } \exists(\langle r', F_e\rangle \in \alpha_{me}) : r = r' \\ 0 & \text{otherwise} \end{cases}$$

[4] Hence, for a given domain, any state will generally include a metric attribute for *every* resource in the domain.

Informally, the execution of an action is just what one would expect: the non-negated propositional attribute effects are added to the state, the negated propositional attribute effects are deleted from the state, and the resources listed in the metric effects have their levels changed as specified by the metric effect function.

3.2.2 Plan representation

With our action representation in hand, in this section we turn to the definition of *plans*, the end product of the PYRRHUS planning system, We also describe the relationship between plans and outcomes, the foundation of value defined in the previous chapter.

Definition 20 (Plan) *A* plan *is an ordered sequence of zero or more ground action schemas.*

Just as ground actions cannot necessarily be executed in every state, plans too are limited. We extend the concept of viability, defined for actions in the previous section, to apply to plans as well. Essentially, a plan will be viable if each of its actions is viable in the state resulting from execution of the previous actions.[5]

Definition 21 (Viability of a plan) *A plan P is* viable *in state S_0 just in case:*

- *P has no actions, or*
- *$P = (\alpha) \circ P'$, the action α is viable in S_0, and P' is viable in $\Omega_a(\alpha, S_0)$.*

We can finally formalize the relationship between plans and outcomes (over which value functions are defined). An outcome arises from the execution of a viable plan. Informally, the outcome that arises from the execution of a (viable) plan is just the sequence of states that the world goes through after the execution of each individual action.

[5] We will use the notation "$A \circ B$" to indicate concatenation of sequences. E.g., if $S = (s_3, s_4)$, then $(s_1, s_2) \circ S = (s_1, s_2, s_3, s_4)$.

Definition 22 (Plan execution) *The* execution *of a viable plan P in a state S_0 (written $\Omega(P, S_0)$) is an outcome O such that*

- *if P has no actions, then $O = (S_0)$;*
- *if $P = (\alpha) \circ P'$, then $O = (\Omega_a(\alpha, S_0)) \circ \Omega(P', \Omega_a(\alpha, S_0)))$*

Another important concept derived from viability is that an initial state and a set of action schemas together define a space of possible plans, representing all feasible courses of action from the initial state.

Definition 23 (Viable Plan Space) *Given an initial state $\mathcal{I}$, and a set of* R-STRIPS *action schemas $\mathcal{A}$, the* viable plan space $P_{\mathcal{I},\mathcal{A}}$ *is the set of all plans composed from instantiations of members of $\mathcal{A}$ that are viable in $\mathcal{I}$.*

3.2.3 The value-directed planning problem

In Chapter 1 we proposed value-directed planning as a general paradigm in which to develop planning systems. In Chapter 2 we presented a representation for world states based on propositional and metric attributes, and representations for two different kind of goal-directed value functions. This chapter has introduced a representation for actions using STRIPS-like action schemas extended to support metric attributes. We now formalize the definition of a specific value-directed planning problem in terms of the representations developed in this and the previous chapter.

Definition 24 (R-STRIPS Value-directed Planning Problem) *An* R-STRIPS value-directed planning problem *is a triple $\langle \mathcal{I}, \mathcal{A}, \mathcal{V} \rangle$ consisting of an initial state, a set of* R-STRIPS *action schemas, and a goal-directed value function. Its* solution *is a plan $P \in P_{\mathcal{I},\mathcal{A}}$ maximizing $\mathcal{V}(\Omega(P, \mathcal{I}))$.*

3.3 Planning by branch-and-bound optimization

Value-directed planning is an optimization problem. It requires finding a plan of greatest possible value from a potentially infinite space of viable candidates. In this regard, value-directed planning is very different from classical planning. A classical planning system is faced with the same space of viable plans, but it need only seek until it finds a satisfactory plan. By contrast, a value-directed planning system must in general consider—at least implicitly—*every* viable plan in the space.[6] The only hope for a practical value-directed planning system, then, is to avoid explicit consideration of most of the plan space. A general technique that has long been used for the effective solution of other difficult combinatorial optimization problems is *branch-and-bound* optimization (e.g. [Agin 66, Garfinkel 79, Balas 79]).[7]

3.3.1 Branch-and-bound basics

Branch-and-bound works by exploring a space of *candidate sets*. Each node in this space corresponds to a (perhaps infinite) set of possible feasible solutions to the problem. A candidate set is explicitly represented by *partial solution*, a set of constraints on features of the eventual solution, which implicitly represents the candidate set of all complete solutions that possess those features. For example, when applied to the infamous Traveling Salesman Problem (TSP) [Garey&Johnson 79], a partial solution might be encoded as a prefix of a tour, representing the candidate set of all tours that start with that prefix.

[6] An exception is when the value function has a known bound over the plan space, in which case a value-directed planner can halt when it finds a plan attaining maximal value. Although this might occur in degenerate cases of value-directed planning (e.g. classical planning), in general the value function will not be provably bounded. Moreover, even if the value function is provably bounded, the existence of such "perfect" plans would likely be the exception rather than the norm.

[7] This technique has also known as "implicit enumeration".

Branching A branch-and-bound algorithm begins with a single vacuous partial solution, which has no constraints and thereby embodies all possible candidate solutions. This initial partial solution is the root of a tree-structured space. The algorithm "branches" on a partial solution by extending it in some fashion; a new offspring is created for each possible extension of the partial solution. When solving the TSP, for example, extension of a partial solution would create a set of new offspring by appending each of the as-yet-unvisited cities to the end of the existing tour prefix. Extending a partial solution results in a partitioning of the candidate set among the offspring.[8] Depending on the representation employed, there might be multiple points at which a partial solution could be extended, and thus multiple ways of partitioning the candidate set. In the case of TSP, for example, a partial solution might instead be represented by both a prefix and a postfix of a tour. Such a partial solution could be extended either by adding a city to the prefix, or by adding a city to the postfix. Each if these two extension points would give rise to a different partitioning of candidate set.

Eventually, each partial solution will be extended until the candidate set either contains a single complete solution, or is empty (that is, the constraints that make up the partial solution are inconsistent). No further extension is possible.

Bounding The other major facet of a branch-and-bound algorithm, of course, is "bounding." For each partial solution, upper and lower bounds on solution quality are computed over the complete solutions in the candidate set. In general, these bounds will not be exact. In the TSP example, we know the exact cost of the prefix, but must make optimistic and pessimistic estimates of the cost of the unfinished part of the tour. For the algorithm to be correct, the bounds must be conservative, that is, the quality of all candidates must actually lie between the bounds. When a partial

[8] Indeed, we will sometimes speak of "extending the solution" and at other say "partitioning the candidate set". These two terms refer to precisely the same process, seen from different perspectives.

solution has been extended to completion, its bounds converge to the exact quality of that solution. A partial solution is *dominated* if the upper bound on its quality is less than (or equal to) the greatest lower bound among the other partial solutions, allowing it to be eliminated from consideration (or "pruned").

The best complete solution generated at any point is referred to as the *incumbent*. Eventually (if all goes well) the *incumbent* will dominate all other partial solutions, at which point it is established as the optimal solution. Branch-and-bound will not necessarily halt if there is an infinite candidate set that can be partitioned indefinitely without sufficient improvement of quality bounds. In some cases, this can be avoided by using a different representation for partial solutions, or perhaps by using a different strategy for selecting among alternative extension points. In other cases, though, the problem simply may not be amenable to branch-and-bound.

3.3.2 Branch-and-bound applied to the planning problem

PYRRHUS uses a variant of the branch-and-bound technique to solve the R-STRIPS value-directed planning problem. Construction of a branch-and-bound algorithm requires:

- a representation for candidate sets,
- a technique for partitioning candidate sets, and
- a technique for computing bounds on the value of candidate sets.

In the case of planning, we represent a candidate set by a *partial plan*, and partition a candidate set by *refining* the partial plan. This is very similar to the plan-space search technique employed by many classical planning algorithms. Kambhampati *et al.* [1994] present a generalized classical planning algorithm based on "refinement search," which closely resembles the "branching" half of branch-and-bound. They describe several alternatives that arise in the representations for partial plans, such as whether plan steps are partially or totally ordered, or whether dependencies are recorded by causal links or rechecked on the fly using the modal truth criterion. In this section, though, we will describe an *abstract* branch-and-bound planning algorithm, which does not depend on a particular representation for partial plans. In Section 3.4 we will describe the specific partial-plan representation and refinement framework used by PYRRHUS. In Chapter 4, we will describe how bounds on value are computed for partial plans in that representation.

Partial plans, refinement, and candidate sets

One potential difficulty is that the spaces of partial plans explored by most classical planning algorithms do not include all possible viable plans. The candidate set represented by a partial plan depends not only on the constraints encoded by the partial plan, but also on the refinement technique used by the planner. In essence, a partial plan represents all complete plans that can be generated through further refinement. Classical planners gain efficiency by using specific refinement techniques, such as goal regression, that restrict the candidate set to just the relevant plans, namely those that satisfy the goal. The refinement process is generally constrained by focusing on *flaws*, i.e. deficiencies of the partial plan with respect to its ability to satisfy a particular goal.[9]

If we are to use partial plans to represent candidate sets for a branch-and-bound-based value-optimizing planner, we must ensure that we begin with a candidate set that includes every possibly optimal plan within the space of all viable plans. At the same time, we would still like to avoid truly irrelevant parts of the plan space by using the techniques developed for classical planning algorithms. In PYRRHUS we narrow

[9] Notable exceptions are the forward-chaining planners (e.g. TEMPLAN [Bacchus&Kabanza 95]). In such planners, the initial partial plan does in fact represent the entire viable plan space, and the single refinement technique (adding an action to the plan prefix), does not take the goal into account (although the goal may be used to guide the selection of *which* plan to refine).

the search by exploiting the structure of the value functions. For a partially satisfiable deadline goal, for example, we know *a priori* that an optimal plan will either be one that achieves the goal (at some point in time), or will be one that improves residual value. We can therefore limit the initial candidate set to those plans that achieve the goal and those plans that improve upon the final asset position. Details about initial partial plan generation are given in Section 3.4.1.

The OPTIPLAN *algorithm*

Figure 3.2 shows OPTIPLAN, the planning algorithm used by PYRRHUS. OPTIPLAN assumes the existence of an underlying planning framework providing:

1. A specific representation for partial plans.

2. A function *MakeInitialPlans* which returns a set of initial partial plans for a given value function. The union of the candidate sets represented by these partial plans must contain every possibly optimal plan in the plan space.

3. A function *Flaws* that returns a set of different possible extension points of a partial plan. Each flaw corresponds to a different partitioning of the candidate set represented by the plan. A partial plan with no flaws represents a single complete viable plan.

4. A function *Refinements* that, given a partial plan and a flaw in that partial plan, returns a set of offspring comprising all possible extensions for that flaw. That is, the *Refinements* function performs the actual partitioning of the candidate set.

5. A function for bounding the value of a partial plan. In general, one can only find meaningful upper bounds on the value of a partial plan, since further

Algorithm OptiPlan ($\mathcal{I}$, $\mathcal{A}$, $\mathcal{V}$)

Let *incumbent* = StatusQuo ($\mathcal{I}$)

Let *Horizon* = MakeInitialPlans ($\mathcal{I}$, $\mathcal{V}$)

While *Horizon* is not empty do:

  Choose a partial plan P from *Horizon* (and remove it)

  If $UB(\mathcal{V}, P) \leq \mathcal{V}(incumbent)$ then discard P

  Else

    Let F = Flaws(P)

    If $F = \emptyset$ and $\mathcal{V}(P) > \mathcal{V}(incumbent)$ then

      Let $incumbent = P$

    Else **Choose** a flaw $f \in F$

      Add Refinements(P, f, $\mathcal{A}$) to *Horizon*

Return *incumbent*

Figure 3.2: An abstract description of Pyrrhus's optimal planning algorithm

refinements to a plan might make it arbitrarily bad. Lower bounds can only be found on complete plans, which have an exact value.

The actual planning framework used to implement OptiPlan in Pyrrhus is described in the next section, but in principle, any of a variety of classical planning techniques could be adapted to support OptiPlan.

OptiPlan begins by initializing the incumbent to the *status quo*, the plan of doing nothing. The variable *Horizon* contains the fringe of the search space—the set of partial plans that have been generated but not yet explored. OptiPlan differs from typical branch-and-bound algorithms in that *Horizon* is initialized to a set of initial partial plans (which together represent all possibly optimal solutions) rather than a single initial partial solution. That is, OptiPlan explores a forest of partial plans rather than a three. Each of the individual initial partial plans represents a subset of the viable plan space that takes a particular approach to increasing value. (See Sections 3.3.2 and 3.4.1 for more details.)

OptiPlan repeatedly selects a non-dominated partial plan from the horizon, refines it, and adds its offspring back into the horizon. The incumbent is updated whenever a more highly valued complete plan is found. When the horizon has been exhausted the algorithm terminates, returning the incumbent as the optimal plan.

The choices of which partial plan in the horizon to refine, and which flaw in that plan to address, are crucial to the performance of the algorithm. In OptiPlan these decisions are made heuristically, as described in detail in Chapter 5.

3.4 pcr*: A partial-order, causal-link* r-strips *refinement framework*

In this section, we describe pcr, the plan refinement framework used by Pyrrhus. This framework provides a space of partial plans which includes every possibly optimal complete plan for a given value function. The partial plan space is made available to a search algorithm (in our case, OptiPlan) through the provision of three functions: one that generates the roots of the trees that make up a forest of partial plans, one that returns a set of flaws for a given partial plan, and one that returns a set of refinements that correct a given flaw in a given plan. The representation of partial plans and these three functions will be described in turn.

pcr is based on techniques developed for "least-commitment" classical planning systems [Sacerdoti 75, Chapman 87, McAllester&Rosenblitt 91, Penberthy&Weld 92]. (Also see [Weld 94] for a brief introduction to the topic, and [Kambhampati *et al.* 94] for a unifying survey.). The philosophy behind least-commitment planning is to generate partial plans that are as general as possible for as long as possible during the planning process. The hope is that this will allow the planner to avoid working on plans that cannot achieve the goal because of "inessential" commitments, which arise as artifacts of the partial plan representation but are not necessary features of a solution. In particular, the main approach taken by many recent planners to reduce commitment is to represent partial plans as *partially-ordered* sequences of actions.

Another technique common to many partial-order planners is the use of *causal links*[10] to record dependencies between the effects and preconditions of various steps in the plan. Causal links allow the planner to tradeoff some bookkeeping effort and some increase in the size of the search space in exchange for a reduction in the effort required to determine whether a partial plan is viable.

PCR is closely modeled on the partial-order, causal-link planning framework used by SNLP (and its popular successor, UCPOP). The key difference is that the partial plan representation and refinement techniques have been augmented to provide support for metric attributes as they appear in R-STRIPS action schemas.

Definition 25 (PCR partial plan) *A* PCR partial plan *is a sextuple* $\langle S, O, B, C, R, F\rangle$ *where:*

- S *is a set of* steps*. Each step is a copy of an* R-STRIPS *action schema with fresh variables.*

- O *is a set of* ordering constraints*. Each ordering constraint is a pair* $\langle s_b, s_a\rangle$ *where* s_b *and* s_a *are steps in* S*.*

- B *is a set of variable* binding constraints *on the variables that appear in steps in* S*.*

- C *is a set of* causal links*. Each causal link is a triple* $\langle s_p, a, s_c\rangle$ *where* s_p *(the "producer") and* s_c *(the "consumer") are steps in* S *and* a *is a propositional precondition of* s_c *that unifies with a propositional effect of* s_p *given the variable bindings* B*.*

[10] Sometimes known as "protection intervals."

- *R is a set of* resource links. *Each resource link is a triple $\langle s_p, r, s_c \rangle$ where s_p and s_c are steps in S and r is a resource symbol mentioned in a metric precondition or effect of both s_p and s_c.*

- *F is a set of* flaws. *Each flaw is of one of the following four types:*

  1. *An* open propositional precondition *(OPP) flaw is a pair $\langle s, p \rangle$ where s is a step in S and p is a propositional precondition of s.*
  2. *An* open resource requirement *(ORR) flaw is a pair $\langle s, r \rangle$ where s is a step in S and r is a resource mentioned in either a metric precondition or a metric effect of s.*
  3. *A* threatened causal link *(TCL) flaw is a pair $\langle l, s \rangle$ where l is a causal link and s is a step, possibly between the producer and the consumer of l given the ordering constraints O, with a propositional effect that negates the propositional attribute of l given the binding constrains B.*
  4. *A* threatened resource link *(TRL) flaw is a pair $\langle l, s \rangle$ where l is a resource link and s is a step, possibly between the producer and the consumer of l given the ordering constraints O, with either a metric precondition or a metric effect mentioning the resource symbol in l.*

3.4.1 Creating the initial partial plan set

Like SNLP, PCR encodes the initial state and the goal conditions into a partial plan by using dummy steps. But there are two key differences. First, in order to find an optimal plan, PCR must generate a space of partial plans that includes all possibly optimal plans for a given value function. It is not sufficient to consider only those plans which satisfy a single goal formula. PCR surmounts this problem by exploiting the known structure of the value functions; it creates a set of partial plans, each of which represents some region of the space of all viable plans that might improve upon value over the status quo.

Second, since PCR supports metric resources, the notion of a goal must be generalized to include them as well. In particular, we must produce initial plans that will lead to the consideration of actions that might increase the final asset position. Just as propositional goals are equivalent to propositional preconditions, a reasonable representation for metric goals would be the same as that used for metric preconditions. PCR explores plans to improve the final asset position by adding a vacuous resource preconditions (e.g. $\langle \textit{fuel-level}, [-\infty, \infty] \rangle$) to the goal step for each asset. So doing will lead the refinement procedure to consider all plans which have any effect on asset position.

In Figure 3.3 we see the *MakeInitialPlans* function. This function creates the appropriate set of initial plans for the given value function by using an auxiliary function that creates an initial plan for a given goal formula and set of resources. In the case of a partially satisfiable deadline goal, two initial plans are created. One will generate plans in which the goal is satisfied and assets are improved, while the other will only generate plans in which assets are improved. For partially satisfiable conjunctive goals, a separate initial plan is created for each goal fragment, as well as for no goal at all. In each case, the improvement of assets is also part of the initial plan.

3.4.2 Refining partial plans

The refinement of a partial plan in PCR is driven by the extant flaws, and occurs in two stages. The *Flaws* function simply returns the set of flaws in a given partial plan. The *Refinements* function returns a set of offspring which represent all possible resolutions of the given flaw. That is, the *Refinements* function partitions the candidate set represented by a given partial plan by adding commitments, either to achieving a particular precondition of one step with a particular effect of another step, or to

Function MakeInitialPlans ($\mathcal{I}$, $\mathcal{V}$)

Let A be the set of assets of $\mathcal{V}$

Let $M = \{\langle a, [-\infty, \infty]\rangle | a \in A\}$

If $\mathcal{V}$ is a partially satisfiable deadline goal then:

Let G be the goal formula of $\mathcal{V}$

Return $\{\text{MakeIP}(\mathcal{I}, G, M), \text{MakeIP}(\mathcal{I}, \emptyset, M)\}$

Else ($\mathcal{V}$ *is a partially satisfiable conjunctive goal)*

Let G^* be the set of fractional goal formulas of $\mathcal{V}$

Let $Inits = \{\text{MakeIP}(\mathcal{I}, \emptyset, M)\}$

For all $g \in G^*$ do:

Let $Inits = Inits \cup \text{MakeIP}(\mathcal{I}, g, M)$

Return $Inits$

End MakeInitialPlans.

Function MakeIP ($\mathcal{I}$, *Prop-goal*, *Metric-goal*)

Let I_p be the propositional attributes of $\mathcal{I}$

Let I_m be the metric attributes of $\mathcal{I}$

Let S_0 be a step with propositional effect I_p and metric effect I_m

Let S_g be a step with propositional precondition *Prop-goal*

and metric precondition *Metric-goal*

Return a partial plan with steps S_0 and S_g,

an OPP flaw for each propositional precondition of S_g,

and an ORR flaw for each metric precondition of S_g

End MakeIP.

Figure 3.3: The *MakeInitialPlans* Function

avoiding the violation of an earlier such commitment by ordering steps or binding variables in a certain way.

The decoupling of flaw selection from refinement gives the search procedure (e.g. OptiPlan) essential control over the generation of the search space. Research on classical planning has clearly established the impact that flaw selection can have on planning performance. As we will see in Chapter 5, flaw selection is equally crucial to the performance of a value-optimizing algorithm like OptiPlan.

Figure 3.4 shows the PCR *Refinements* function. Its treatment of propositional attributes through open propositional preconditions and threatened causal links is similar to that of classical partial-order, causal-link planners [McAllester&Rosenblitt 91, Penberthy&Weld 92]. It differs in its support for metric resources.

Reasoning about metric attributes

PCR has two requirements for its reasoning about resources. First, to determine that a plan is viable, it must confirm that the metric preconditions of each step are met. Second, to determine the value of a plan, it must calculate the levels of the assets at the conclusion of the plan. PCR reasons about metric resources by using resource links. Each step that uses a resource, either as a precondition or an effect, must be supported by a resource link from the previous step in the partial plan that uses the resource. In a complete plan, the resource links for a given resource will form a *resource chain* of all the steps in the plan that use the resource. Every resource chain in a complete plan must begin at the initial step, and ends at the final step using that resource (which may or may not be the goal step).

In a complete plan, a resource chain can be checked for *resource usage consistency* by starting with the level of the resource in the initial step, progressing the level through the effect of each successive step in the chain, and comparing the current level to the precondition of each step in the chain. If the level falls within the interval specified by every precondition then the chain is consistent, otherwise it is not. Every

Function Refinements (P, $flaw$, $\mathcal{A}$)

If $flaw$ is an *open propositional precondition* $\langle s_c, prop\rangle$, then:

For each existing step s_p in P possibly before s_c with a propositional effect matching $prop$:

Create an offspring including the causal link $\langle s_p, prop, s_c\rangle$

For each action schema a in $\mathcal{A}$ with a propositional effect matching $prop$:

Create an offspring with:

A new step s_p copied from a with fresh variables

A causal link $\langle s_p, prop, s_c\rangle$

An ordering constraint placing s_p before s_c

If $flaw$ is an *open resource requirement* $\langle s_c, resource\rangle$, then:

For each existing step s_p in P possibly before s_c with a metric precondition or effect matching $resource$:

Create an offspring including the resource link $\langle s_p, resource, s_c\rangle$

For each action schema a in $\mathcal{A}$ with a metric precondition or effect matching $resource$:

Create an offspring with:

A new step s_p copied from a with fresh variables

A resource link $\langle s_p, resource, s_c\rangle$

An ordering constraint placing s_p before s_c

If $flaw$ is a *threatened causal link* $\langle\langle s_p, prop, s_c\rangle, s_t\rangle$ then:

If consistent with current constraints:

Create an offspring with a constraint ordering s_t before s_p

Create an offspring with a constraint ordering s_c before s_t

Create an offspring with additional variable binding constraints such that no effect of s_t matches $prop$

If $flaw$ is a *threatened resource link* $\langle\langle s_p, resource, s_c\rangle, s_t\rangle$ then:

If consistent with current constraints:

Create an offspring with a constraint ordering s_t before s_p

Create an offspring with a constraint ordering s_c before s_t

Remove $flaw$ from each offspring

Return all offspring that are *resource usage consistent.*

Figure 3.4: The *Refinements* Function

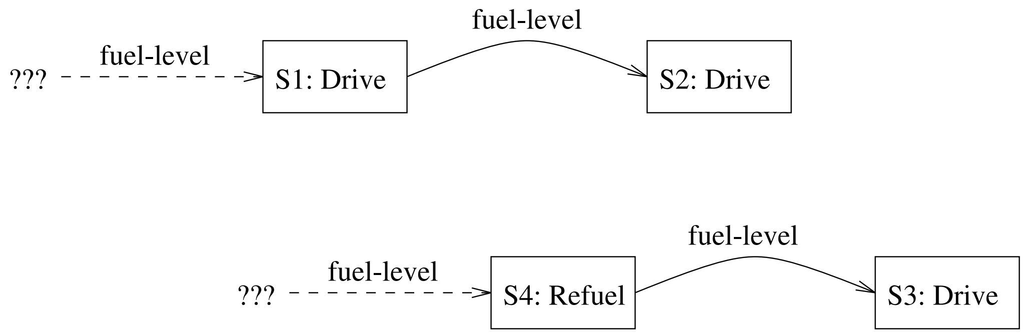

Figure 3.5: Part of a partial plan with two segments of resource chain

resource chain must be consistent in order for a plan to be viable.

During the planning process (before a partial plan is complete) there can be multiple, disconnected *segments* of resource chains for a single resource. These segments of resource chains are created, extended, or joined together by the refinement of open resource requirement flaws. For example, Figure 3.5 shows part of a partial plan with two segments of resource chain for the *fuel-level* resource (and two open resource requirement flaws). If the open resource requirement flaw of step *S4* were resolved by adding a resource link from step *S2*, the two segments would be joined into a single chain.

A segment of a resource chain in an incomplete partial plan can also be checked for resource usage consistency, using a generalization of the method given above for complete plans. Instead of progressing an exact current level down the chain, we progress an *interval* representing all possible current levels of the resource. The current level is initialized to the vacuous interval $[-\infty, \infty]$. The current level is progressed through the precondition of each step (if any) by intersecting it with the interval representing the precondition. If the result of this intersection is empty the segment is inconsistent, otherwise the intersection is taken to be the new current level. Since some of the variable parameters of a step may not be bound to constants, the metric effects of some of the steps may be intervals as well. The current level is progressed through such effects using interval addition.

A segment of resource chain must be checked for consistency whenever it is extended or joined with another segment. Such a check is straightforward to implement. However, a segment must also be rechecked for consistency whenever there are changes to the bindings of any of the variable parameters of any of the steps in the segment (since the new bindings may narrow the precondition or effect intervals). Since this check could be quite computationally expensive, an alternative approach would be to delay the check and continue to refine a possibly inconsistent partial plan. If this approach were adopted, all resource chains would need to be rechecked for consistency at some future point—perhaps when the plan is complete.

3.5 Summary

This chapter has formally described a particular formulation of value-directed planning. Based on the R-STRIPS action representation (an extension to the classical STRIPS language with support for metric resources), an R-STRIPS *value-directed planning problem* is posed with the provision of an initial state, a set of action schemas, and a goal-directed value function of the form given in Chapter 2. The solution to such a problem is a plan constructed from the given action schemas that produces a state of optimal value.

PYRRHUS solves the R-STRIPS value-directed planning problem using OPTIPLAN, an abstract branch-and-bound optimizing planning algorithm. OPTIPLAN requires an underlying plan-refinement framework capable of generating a space of partial plans that is complete with respect to the value function. While such a framework could be constructed using any of a number of classical planning techniques, PYRRHUS uses PCR, a plan-refinement framework based on "least-commitment" planning as exemplified by SNLP.

Chapter 4

BOUNDING VALUE

4.1 Introduction

Recall that a value-directed planning problem in PYRRHUS is posed as an initial state, $\mathcal{I}$, a set of R-STRIPS action schemas, $\mathcal{A}$, and a goal-directed value function, $\mathcal{V}$. The initial state is defined in terms of domain attributes, which may be propositional or metric. The action schemas are defined by preconditions and effects in terms of these attributes and represent transformations between world states. $\mathcal{V}$ is a function over *outcomes* (i.e. sequences of world states) of the form

$$\mathcal{V}(o) = k_g V_G(o) + k_r V_R(o)$$

$V_G(o)$ is the degree to which the agent's goal(s) are satisfied in o, while $V_R(o)$ defines the residual value of outcome o.

We have assumed that actions are deterministic and that no exogenous changes to the state occur, so execution of any viable plan, p, in a given initial state, $\mathcal{I}$, gives rise to a unique outcome, denoted $\Omega(p, \mathcal{I})$. The value function can therefore be mapped onto the domain of plans. Let $P_{\mathcal{I},\mathcal{A}}$ be the set of all possible legal plans constructed from $\mathcal{A}$ beginning in $\mathcal{I}$. In the PCR plan refinement framework, a *partial plan* represents a candidate set, a subset of $P_{\mathcal{I},\mathcal{A}}$ consisting of all plans that can be generated by further refining that partial plan. PYRRHUS finds plans that are optimal with respect to $\mathcal{V}$ by performing branch-and-bound optimization over its space of partial plans. This procedure requires the ability to calculate an upper bound on $\mathcal{V}$ over the candidate set of any given partial plan. The exact (least) upper bound on value of a partial plan is defined as

$$LUB(\mathcal{V}, P) = \max_{p \in \ cs(P)} \mathcal{V}(\Omega(p, \mathcal{I}))$$

where $cs(P)$ is the candidate set P. Because it is infeasible to enumerate the candidate set, we can't directly compute the least upper bound on value of a partial plan. Instead, we must find a conservative approximation

$$UB(P) \geq LUB(P)$$

that can be computed in reasonable time. The better our approximation (i.e. the closer UB is to LUB), the more efficient our planning algorithm's performance.

Since $\mathcal{V}$ is an additive function, the upper bounds on the goal component (V_G) and the residual value component (V_R) can be calculated independently for any partial plan. The bound for $\mathcal{V}$ on that partial plan is then the sum of these two individual bounds weighted by their respective coefficients:

$$UB(\mathcal{V}, P) = k_g \cdot UB(V_G, P) + k_r \cdot UB(V_R, P)$$

This chapter will describe how the bounds on V_G may be calculated for the partially satisfiable deadline goals and partially satisfiable conjunctive goals described in Chapter 2. It will also describe how bounds on V_R may be calculated depending on particular characteristics of the set of actions.

A further consideration is that optimization can only be performed if an optimal plan does, in fact, exist. $P_{\mathcal{I},\mathcal{A}}$ may contain an infinite number of plans, so $\mathcal{V}$ might not be bounded over this space. Since V_G *is* bounded over $P_{\mathcal{I},\mathcal{A}}$ (it takes on a finite maximum value if and only if the goal(s) that it encodes are completely satisfied), $\mathcal{V}$ is bounded just in case V_R is bounded. But, since V_R is generally unbounded with respect to increasing amounts of at least some assets (e.g. money), such assets must themselves be bounded over $P_{\mathcal{I},\mathcal{A}}$ if a bound on V_R is to exist.[1]

---

[1] If $\mathcal{V}$ is *not* bounded over $P_{\mathcal{I},\mathcal{A}}$, then goal regression planning is probably no longer appropriate.

What features must $\mathcal{I}$ and $\mathcal{A}$ have in order to provide such bounds? This chapter will define *asset bounding characteristics*, i.e. restrictions on $\mathcal{I}$ and $\mathcal{A}$ that are sufficient to assure that V_R (and thus $\mathcal{V}$) will be bounded over $P_{\mathcal{I},\mathcal{A}}$. Asset bounding characteristics vary in their *strength*, that is, in how restrictive they are of the actions in $\mathcal{A}$. The methods used for calculation of upper bounds on V_R depend on which asset bounding characteristics hold over $\mathcal{I}$ and $\mathcal{A}$, and on how strong they are. These methods will be described in detail in this chapter.

4.2 Bounding goal value

This section describes how to calculate an upper bound on the goal component of the value function for a given partial plan. As described in Chapter 2, the goal component is a function V_G mapping an outcome into the interval $[0, 1]$. This function represents the degree of satisfaction of the agent's goal(s) in the given outcome. The calculation of $UB(V_G, P)$ depends on the way in which the goals can be partially satisfied. The two cases, partially satisfiable deadline goals and partially satisfiable conjunctive goals, will be examined in turn.

A key feature in both cases, though, is that the space of partial plans that PYRRHUS explores is generated through goal regression (see Chapter 3). This means that any given partial plan P will have an associated symbolic goal formula P_O (the precondition of the goal step), known as the *objective* of that partial plan. Any individual planning problem might require the generation of multiple, separate spaces of partial plans for a variety of objectives (see Section 3.4.1).

First, the problem really needs to be reformulated with an even richer notion of plan quality than just "maximum value". For example, one might seek to maximize average value per time unit, or total discounted value, over an infinitely long plan (policy). The resulting problem can be described as an infinite-horizon Markov decision process, for which a number of solution techniques are available. A complete survey is beyond the scope of this thesis, but see for example [Monahan 82, Cassandra *et al.* 94].

4.2.1 Partially satisfiable deadline goals

In the case of partially satisfiable deadline goals, the degree of satisfaction depends on whether or not a particular symbolic goal formula G is achieved, and if so, the earliest time at which it is achieved. The goal component of the value function can be decomposed as:

$$V_G(o) = G_s(o) \cdot D_G(G_t(o))$$

G_s is the *goal satisfaction* term, which takes on the value 1.0 if the goal G is satisfied in o (irrespective of time) and 0.0 otherwise. G_t is a function which returns the first time in o at which G becomes satisfied. D_G, the *deadline* term, is a monotonically non-increasing function from time into the interval $[0,1]$ describing the nature of the deadline.

Any given partial plan P will have an associated objective P_O. P_O will either entail G (in the case of plans generated in pursuit of the goal) or not (in the case of plans generated just in pursuit of residual value). In the latter case, the partial plan has no goal value, so $G_s = 0$ and the upper bound on V_G can be taken to be zero as well.

When P_O *does* entail G, G_s is taken as 1.0, and the bound on V_G is then just the bound on D_G. Based on the monotonicity of D_G, and on the fact that additional refinements to a partial plan can only increase the minimum amount of time it takes to achieve the goal, we can calculate an upper bound on value of the goal component of a partial plan P as:

$$UB(V_G, P) = D_G\left(\sum_{s \in steps_{P_O}(P)} duration(s)\right)$$

where $steps_{P_O}(P)$ is the set of steps in the partial plan P that necessarily occur before the achievement of P_O, and $duration(s)$ is the amount of the *clock-time* resource consumed by a step s. If any of the variable parameters of a step are unbound, then the metric effect function for *clock-time* may return a real-valued interval; in that case, *duration* should return the maximum extent of the interval.

4.2.2 Partially satisfiable conjunctive goals

As defined in Chapter 2 a partially satisfiable conjunctive goal $G = g_1 \wedge g_2 \wedge \ldots \wedge g_k$ is represented by a set of mutually exclusive and exhaustive *fractional goal formulas* $G^* = \{G_1, G_2, \ldots, G_n\}$, with each G_i constructed from the conjuncts of G. An associated *fractional value* function, $F_G : G^* \rightarrow [0, 1]$ describes the degree of satisfaction associated with achieving each fractional goal.

When PYRRHUS plans for G, each partial plan that it constructs will have one of the fractional goals as its objective. That is, for any partial plan P, $P_O \in G^*$. The upper bound on the goal component for this partial plan, then, is just the degree of satisfaction of its objective:

$$UB(V_G, P) = F_G(P_O)$$

One might question the completeness of this approach, since the actual outcome of a plan might serendipitously achieve a fractional goal with a higher degree of satisfaction than the objective of the partial plan from which the plan was itself derived. But, if this were the case, there would be *another* partial plan, with the higher valued fractional goal as its objective, also under consideration. This is because PYRRHUS constructs a complete partial plan space for each fractional goal.

4.3 Asset bounding characteristics

Before we address the issue of computing bounds on the residual value component of the value function, we must confront a deeper problem: residual value might not, in fact, *be* bounded. If our agent, like Rumpelstiltskin, could spin straw into gold, there might be plans that exceed any fixed finite value. Although PYRRHUS cannot be applied to such situations, this limitation is not necessarily problematic, since there is a wide variety of reasonable problems to which Pyrrhus *can* be applied. (We take "reasonable" here to mean that the value function and set of action schemas are such that there is an optimal plan of finite length.)

This section describes various conditions that are sufficient to ensure that an optimal plan exists. These conditions will also support the calculation of upper bounds on the residual value of a partial plan. It is important, though, that these restrictions not be too restrictive, lest they unduly limit the scope of problems to which Pyrrhus could be applied.

As mentioned above, an optimal plan exists only if the residual value component of the value function, V_R, is bounded over the set of possible plans, $P_{\mathcal{I},\mathcal{A}}$. More precisely, as defined in Chapter 2, the residual value function depends only on the final quantities of a particular subset of domain attributes, which we will call *assets*. That is, although nominally defined over outcomes, the residual value function can be decomposed as:

$$V_R(o) = \widehat{V_R}(\vec{A}(o))$$

where $\vec{A}(o)$ is a function returning the *asset position*, a vector of the quantities of the assets in the final state of outcome o, and $\widehat{V_R}$ is the asset position value function. We require that $\widehat{V_R}$ be monotonically non-decreasing with respect to an increased amount of each individual asset. In a transportation domain, for example, assets might be *fuel-level* and *money*, and the function $\vec{A}$ would return a pair (f, m) representing the amount of fuel and money remaining at the conclusion of a plan. $\widehat{V_R}$ would be defined over such fuel/money pairs.

Asset boundedness We now define the fundamental feature we require of a planning scenario in order for Pyrrhus to be applicable.

Definition 26 *An initial state and set of action schemas, $\langle \mathcal{I}, \mathcal{A} \rangle$, is* **asset bounded** *with respect to some $\widehat{V_R}$ just in case there exists some asset position $\vec{a}$ such that for* *every* $p \in P_{\mathcal{I},\mathcal{A}}$

$$\widehat{V_R}(\vec{A}(\Omega(p,\mathcal{I}))) \leq \widehat{V_R}(\vec{a})$$

Asset boundedness is necessary and sufficient to assure that an optimal plan exists. Note that if $\widehat{V_R}$ is itself inherently bounded over its entire domain, then every $\langle \mathcal{I}, \mathcal{A} \rangle$ is trivially asset bounded. However, as mentioned above, this condition does not generally hold. There will typically be some asset, such as money, in increasing amounts of which (all other things being equal) residual value grows unboundedly.[2] If there is some asset in which value grows unboundedly, a planning problem will only be asset bounded (and thus admit a solution) if some restrictions are placed on the initial state and/or the actions themselves that limit the attainable asset positions.

Definition 27 *For a given* $\widehat{V_R}$, **asset bounding characteristics** *are a set of constraints on initial states and actions such that for all* $\mathcal{I}$ *and* $\mathcal{A}$ *that satisfy those constraints,* $\langle \mathcal{I}, \mathcal{A} \rangle$ *is asset bounded.*

We will now define a number of constraints on $\mathcal{I}$ and/or $\mathcal{A}$, which will generally be parameterized by some indeterminate asset X. These constraints will be said to be "satisfied for X" when they hold for that specific asset. Asset bounding characteristics will be composed from these constraints by quantifying over all assets. That is, given some constraint C on $\mathcal{I}$ and $\mathcal{A}$ for a given X, we might have the asset bounding characteristic, "C is satisfied for every X."

Asset bounding characteristics are not hard to find. For example, consider the following constraints:

Constraint 1 (No Production of X (NPX)) *No action in* $\mathcal{A}$ *may produce the asset* X*.*

---

[2] Even when $\widehat{V_R}$ does have an inherent bound, such a bound is probably so great as to be of no computational value. In the case of money, for example, although there might be some amount beyond which its marginal value would be zero, this amount would probably be very large. If we bounded the residual value of a partial plan just by assuming that it might generate up to this amount of money, such a bound would be uselessly loose.

Constraint 2 (No Asset Production (NAP)) *Constraint 1 (NPX) is satisfied for every asset.*

Constraint 2 (NAP) is clearly an asset bounding characteristic, since if no action can produce any asset, then $\widehat{V_R}$ can never take on a higher value than it does in the initial state. In fact, this is a very strong constraint that lends great support to the upper bound calculation for partial plans. But this constraint is really *too* strong; it unacceptably restricts the class of actions that can be represented. In the transportation domain, for example, if fuel in the truck's tank were an asset, this constraint would preclude any kind of refueling action which increased the amount of fuel in the tank. Such a strong constraint is clearly not generally satisfiable.

We would like to find weaker sets of asset bounding constraints — that is, conditions that are sufficient to bound residual value, but still allow expression of a wide variety of domains and problems. In those cases where stronger constraints *are* satisfied, though, the additional support that they lend to the calculation of bounds on partial plans may be advantageous. We will proceed in the remainder of this section to describe several different kinds of asset bounding characteristics, and in the next section we will describe how these characteristics can be exploited to compute bounds on the residual value of partial plans.

4.3.1 Structural Resource Limits

Some assets have maximum absolute values imposed by the physics of the domain. For example, the level of fuel in a tank can never exceed the capacity of that tank. The level of such assets is clearly bounded over $P_{\mathcal{I},\mathcal{A}}$, so actions that produce such assets are limited in their impact on residual value.

Constraint 3 (Structural Limitation of X (SLX)) *There exists a real number B_X (known as a* universal bound *on X) such that for every action $A \in \mathcal{A}$ which produces X, if A produces δ_X units of X, then A has as a precondition that the prior* *amount of X be no greater than $B_X - \delta_X$. Furthermore, the amount of X in the initial state is no greater than B_X.*

Constraint 4 (General Structural Limitation (GSL)) *Constraint 3 (SLX) is satisfied for every asset.*

Constraint 4 (GSL) is an asset bounding characteristic. Let $\vec{U}$ denote the *maximum possible asset position*, that is, the asset position made up of the universal bounds on each asset. Since no action can increase the amount of any asset beyond its universal bound, no outcome in $P_{\mathcal{I},\mathcal{A}}$ can have a greater amount of any asset than the amount of that asset in $\vec{U}$. Therefore $\widehat{V_R}$ is bounded over $P_{\mathcal{I},\mathcal{A}}$ by $\widehat{V_R}(\vec{U})$.

Now, it is clear that while the constraint SLX is satisfied quite naturally by many kinds of domain attributes (such as fuel), there are other kinds of domain attributes (such as money), for which it will not be satisfied. Thus GSL, like the constraint NAP, above, cannot be expected to hold generally. However, the individual asset bounding constraints SLX and NPX might hold for particular assets, which is useful in itself (see Section 4.3.4).

4.3.2 Strong Asset Monotonicity

Heretofore, we have only considered asset bounding characteristics that bound residual value over $P_{\mathcal{I},\mathcal{A}}$ by bounding the amount of each individual asset over $P_{\mathcal{I},\mathcal{A}}$. We have found these constraints to be overly restrictive, precisely because a realistic description of many domains requires that some assets be allowed to grow without bounded. In such domains, residual value might still be bounded if any action that produces some assets consumes other assets in doing so.

For example, imagine that money and postage stamps are both assets in a domain, and that there is an action, *buy-stamps*, that consumes money and produces stamps. Such an action would violate the NPX and SLX constraints for postage stamps, but residual value might still be bounded over plans in this domain as long as the stamps produced by this action are not of greater marginal value than the money consumed.

This section will formalize this notion of actions which exchange assets without increasing residual value. First we formalize the the impact of changes in asset position on residual value:

Definition 28 (MRVI of an asset change) *Let $\vec{A}$ be the space of all possible asset positions (i.e. the domain of $\widehat{V_R}$), and $\vec{a}$ be a change in asset position. The* **maximum residual value increase** *(MRVI) of asset change $\vec{a}$ is:*

$$\mathrm{MRVI}(\vec{a}) = \max_{\vec{b} \in \vec{A}} [\widehat{V_R}(\vec{b} + \vec{a}) - \widehat{V_R}(\vec{b})]$$

In other words, the MRVI of some change in asset position is the maximum amount (over all possible asset positions) by which that change in asset position could increase residual value. If an asset position change does not decrease the amount of any asset, then its MRVI will not be less than zero. Conversely, if an asset position change does not increase the amount of any asset, then its MRVI will not be greater than zero.

Note that MRVI does not exist unless the marginal residual value of each asset (i.e. the partial derivative of $\widehat{V_R}$ with respect to each asset) is bounded. That is, assets must not indefinitely continue to grow in marginal value as their quantity increases. This seems like a reasonable restriction for most domains; in fact it is far more common for assets to have *decreasing* marginal value as their quantity increases.

The definition of MRVI can be extended from asset changes to action schemas by maximizing over all possible instantiations of the action schema:

Definition 29 (MRVIA of an action) *Let $\alpha \in \mathcal{A}$ be an action schema, with $I(\alpha)$ representing the set of all instantiations of α, and $\vec{\Delta}(\alpha')$ representing the asset change of an action instantiation α'. The* **maximum residual value increase of an action** *α is:*

$$\mathrm{MRVIA}(\alpha) = \max_{\alpha' \in I(\alpha)} \mathrm{MRVI}(\vec{\Delta}(\alpha'))$$

We can use this definition to construct the following constraint on actions which produce any particular asset:

Constraint 5 (Strong Asset Monotonicity of X (SAMX)) *For every action $a \in \mathcal{A}$ that produces X,* $\mathrm{MRVIA}(a) \leq 0$.

This constraint allows actions which convert one asset into another, such as the *buy-stamps* action in the example above, as long as such conversions do not increase residual value. It does allow actions which convert assets at a loss, such as a *buy-stamps-from-machine* action which provides three 32 cent stamps in exchange for a dollar.

As in the earlier cases, we can quantify this constraint over all assets.

Constraint 6 (Strong Asset Monotonicity (SAM)) *Constraint 5 (SAMX) is satisfied for every asset.*

Constraint 6 (SAM) is an asset bounding characteristic. SAM holds trivially for every action that does not produce any asset. If an action does produce an asset, then to satisfy SAM it must consume other assets of sufficient value that the action could not result in a net increase in residual value, *regardless of the absolute asset position.*

Limitations of strong asset monotonicity

SAM is clearly less restrictive than NAP. A problem with SAM, though, is that it introduces an unfortunate coupling between the set of action schemas $\mathcal{A}$ and the value function $\mathcal{V}$. One would like to be able to construct $\mathcal{A}$ and $\mathcal{V}$ separately, but to ensure satisfaction of SAM one may have introduce new, artificial assets into both the actions and value function.

To illustrate the difficulty here, consider a transportation domain in which the assets are money and the truck's fuel level, and in which refueling is performed in a two-step process. The action *buy-fuel-drum* consumes money and produces a full fuel drum. The action *refuel* consumes the full fuel drum, and increases the level of fuel in the tank. In this case, if the only assets are money and the fuel level in the truck's tank, the *refuel* action violates the SAM constraint, since it produces the fuel level asset without *directly* consuming money. This situation could be coerced into satisfying SAM by making full fuel drums an asset, but doing so seems rather contrived.

The recycling problem Further imagine that, in addition to increasing the fuel level in the tank, the *refuel* action leaves an empty fuel drum, and that there is an additional action *recycle-fuel-drum* which can turn the empty fuel drum back into some amount of money. Again, this action violates the SAM constraint. But in this case, even the introduction of extra assets doesn't provide a satisfactory solution: if we make empty fuel drums an asset—including them in the residual value function so that *recycle-fuel-drum* can produce no net increase in value—then there would be no reason to add recycling actions to the plan. Note that residual value can still be bounded in this domain: although recycling actions increase residual value, they can't do so indefinitely. Clearly, we would like to be able to model domains in which actions like recycling occur, so some weaker restrictions than SAM are needed.

4.3.3 Weak Asset Monotonicity

We would like a set of asset bounding characteristics that are less restrictive than SAM, which would allow actions to be formulated naturally in terms of their obvious effects on the world, and which would allow the value function to be defined just in terms of those attributes of intrinsic value. Moreover, these constraints need to allow *local* increases in residual value (as occur in recycling actions), while still guaranteeing that residual value is globally bounded.

The basic intuition is that if any action produces an asset, the action must consume some some other attribute which, though not an asset itself, is somehow limited in quantity. This limitation could occur in two ways:

1. *Non-renewable attributes* are provided to some degree in the initial state, but cannot be produced by any other action.

2. *Renewable-at-cost attributes* can be can be produced by other actions, but only at an eventual cost in assets equal to or greater than the value of the asset produced by the step consuming the attribute.

We will call such attributes *limiting attributes.*

We recursively define the *production cost* of an attribute as follows:

Definition 30 (Production cost (PC)) *Let $\vec{a}$ be a unit of some attribute.*

1. *If $\vec{a}$ is an asset, then*
$$\mathrm{PC}(\vec{a}) = \mathrm{MRVI}(-\vec{a})$$

2. *Otherwise, if no action produces $\vec{a}$, then*
$$\mathrm{PC}(\vec{a}) = -\infty$$

3. *Otherwise, let S be the set of actions producing $\vec{a}$. For each $s \in S$, let $N_{\vec{a}}(s)$ be the number of units of $\vec{a}$ produced, and $C(s)$ be the set of attributes consumed by s. Then,*
$$\mathrm{PC}(\vec{a}) = \max_{s \in S} \frac{\mathrm{MRVIA}(s) + \sum_{c \in C(s)} PC(c)}{N_{\vec{a}}(s)}$$

Note that we use the term "cost" here in a narrow technical sense; it refers specifically to the minimum decrease in residual value incurred to produce arbitrary amounts of the attribute in question. This is a theoretical, steady-state lower bound, and does not imply that attributes can generally be produced at this cost. Note that the semantics are reversed from the normal colloquial usage of the term "cost" in that *negative* production cost indicates a penalty in residual value to produce the attribute, while attributes with a non-negative production cost can be had "for free".

This idea of production cost can be used to generalize and weaken the strong asset monotonicity constraint.

Constraint 7 (Weak Asset Monotonicity of X (WAMX)) *Every action* $a \in \mathcal{A}$ *which produces asset* X *must consume some limiting attribute* l *such that* $\mathrm{MRVIA}(a) + \mathrm{PC}(l) \leq 0$.

As before, the constraint with respect to individual assets can be extended by quantifying over all assets.

Constraint 8 (Weak Asset Monotonicity (WAM)) *Constraint 7 (WAMX) is satisfied for every asset.*

Constraint 8 (WAM) is an asset bounding characteristic. This constraint constitutes a significant relaxation of SAM. It allows certain kinds of actions that produce assets (such as the recycling actions mentioned above), while still guaranteeing asset boundedness.

While WAM is sufficient for bounding residual value, it does not necessarily hold for every $\langle \mathcal{I}, \mathcal{A}, \mathcal{V} \rangle$that is asset bounded. Since production cost is a conservative lower bound, there may be domains for which WAM is apparently violated, but in which some attribute cannot actually be obtained for its production cost, so that residual value is in fact bounded over $P_{\mathcal{I},\mathcal{A}}$.

The advantage of WAM is that it is both readily computable and sufficiently weak to allow expression of a wide class of problems. While there might be even weaker sets of asset bounding constraints, it is likely that testing their satisfaction would be intractable.

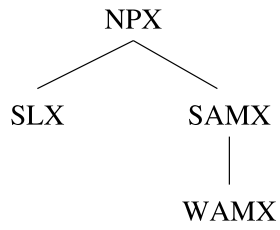

Figure 4.1: A taxonomy of individual asset bounding characteristics

4.3.4 Combining asset bounding characteristics

The four individual asset bounding characteristics which we have described (NPX, SLX, SAMX, and WAMX) can be classified according to their strength as shown in Figure 4.1. If a stronger constraint (higher on the taxonomy) holds for an asset, each weaker (lower) constraint must hold as well. From each of these individual constraints, a general asset bounding characteristic was defined by quantifying over all assets (NAP, GSL, SAM, and WAM, respectively).

But what if no individual constraint is satisfied for all assets? In the case of NPX, SAMX, and WAMX, since the latter is strictly more widely satisfied, we could just ignore NAP and SAM, and ask whether WAM is satisfied. SLX, though, presents an orthogonal weakening of NAP. This suggests the following disjunctive constraint:

Constraint 9 (General Asset Bounding Characteristic (GABC)) *Every asset satisfies Constraint 3 (SLX) or Constraint 7 (WAMX), or both.*

Constraint 9 (GABC) is an asset bounding characteristic that is strictly weaker than either GSL or WAM.

In practice, though, the distinction between strong and weak asset monotonicity is very important. As will be described in Section 4.4, if a domain is only weakly monotonic (i.e. WAMX holds for some asset that SAMX does not), the calculation of upper bounds on a partial plan is more difficult, and the resulting bound is looser, resulting in a generally more difficult planning problem.

4.3.5 A special case: Time as an asset

As we have seen, the basic requirement for a planning problem to be asset bounded is to have sufficient assets being consumed in order to offset the value of any assets that can be produced. A special case of particular note is the attribute *clock-time*. A distinctive feature of *clock-time* is that it is consumed by all (or at least most) actions, and it is never produced by any action. Having *clock-time* as an asset could serve to bound residual value in the presence of actions that would otherwise not satisfy even the weak asset monotonicity constraints. For example, the problematic *spin-straw-into-gold* action could be admitted while still guaranteeing a finite optimal plan if the marginal time cost of performing this action outweighed the marginal benefit of having additional gold.

It is quite plausible that time should be an asset. The intuitive meaning of the residual value function is as a measure of expected ability to satisfy future goals. Just as we value money or fuel because they leave us generally more able to solve unspecified future problems, so too might we value the *availability* of the agent. Time spent in pursuit of the current problem is time that cannot be spent otherwise. This is a completely separate notion from any role that time might play in the (partial) satisfaction of a deadline goal. If we asked our agent to mail a letter by 5:00 PM, and our agent mailed the letter by that time but then went carousing until 4:00 AM, we would probably be dissatisfied. The proper venue for representing this dissatisfaction is in the residual value function.

We could, then, treat time like any other asset, but doing so might not give very good bounds on residual value. We would often expect the marginal impact on residual value of spending more time to *increase* with time. For example, we might be willing to allow our agent to work on a problem for a certain amount of time before residual value begins to decay (as shown in the one-dimensional residual value function in Figure 4.2). The problem here is that our definition of production cost for

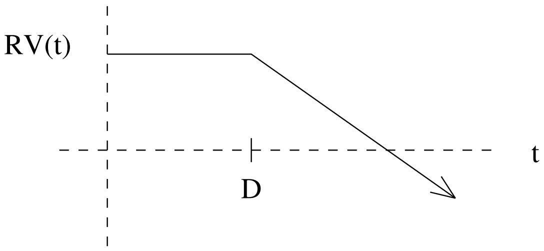

Figure 4.2: An example of residual value as a function of time

an asset, based on the MRVI function, takes its maximum marginal increase (which is to say, minimum marginal decrease) over the entire domain of the asset. Thus, for the residual value function shown in Figure 4.2 the production cost of a unit of time would be zero, despite the fact that for all plans beyond a certain length the marginal cost of time is distinctly negative.

We can get a better definition of production cost by exploiting the fact that time is never produced, and redefining production cost as follows:[3]

Definition 31 (Production cost of time ($\mathrm{PC}(\vec{t})$)) *Let $\vec{t}$ be a unit of time. Let δ be the smallest value for which there exists some time D such that for every asset position $\vec{A}$ in which time is greater than D:*

$$\widehat{V_R}(\vec{A}+\vec{t}) - \widehat{V_R}(\vec{A}) \leq \delta$$

If no such δ exists, then

$$\mathrm{PC}(\vec{t}) = -\infty$$

otherwise

$$\mathrm{PC}(\vec{t}) = \delta$$

According to this definition, if there is some point in time beyond which residual value (as a function of time) is decreasing at (or greater than) some constant rate, that

[3] This redefinition could also be applied to any other asset which is only consumed.

rate will be taken as the production cost of time. For the example shown in Figure 4.2, this new definition would yield the slope of the residual value function beyond the point D rather than zero. This higher cost can guarantee asset boundedness in a wider variety of situations than the previous definition.

4.4 Bounding residual value

We now turn to the question of computing upper bounds on the residual value of a partial plan. The difficulty of doing so depends on the weakness of the asset bounding constraints for the planning problem. If $\langle \mathcal{I}, \mathcal{A}, \mathcal{V} \rangle$ satisfies the stronger constraints NAP or SAM, then an upper bound is readily computed: one just determines the residual value of the asset position entailed by the partial plan, since these constraints assure that no addition of steps to the plan can lead to an asset position of higher residual value.

But if $\langle \mathcal{I}, \mathcal{A}, \mathcal{V} \rangle$ only satisfies a weaker constraint, i.e. GSL, WAM, or GABC, then finding the upper bound is more difficult. Additional steps can indeed increase residual value of the asset position by some amount. We must be able to determine (at least an upper bound on) *how much.* This calculation consists of two phases:

1. Before planning, the action schemas and residual value function are analyzed to determine:

   - the *maximum potential value* (MPV) of producible attributes which are not assets (in the case of WAM and GABC); and

   - the *asset upper bound* (AUB) of assets which are structurally limited (in the case of GSL and GABC).

2. During planning, these maximum potential values and asset upper bounds are used to adjust the upper bound on residual value of a partial plan.

The work done in the first phase need only be done once for each set of actions and residual value function. Our expectation is that this effort can be amortized over multiple planning episodes which differ only in the goal component of the value function, but in any case these computations are generally dominated by the cost of the planning process itself.

In the following sections, we will describe how to determine maximum potential value and asset upper bounds, how to compute a baseline upper bound on a partial plan, and how to use MPV and AUB values to relax that baseline bound.

4.4.1 Determining maximum potential value

In Section 4.3.3 we introduced the notion of *limiting attributes*, those attributes whose production cost offsets the value of assets that are being produced. Such attributes are said to have *potential value* since, although they are not themselves assets, they can potentially enable the production of assets. Further, just as production cost was recursively defined, so too can attributes have potential value just because they enable the production of other (non-asset) attributes with potential value.

In order to find an upper bound on residual value of a partial plan when some assets only satisfy WAMX, we need to know an upper bound on the potential value of all attributes in the domain. Figure 4.3 shows an algorithm which, given a set of action descriptions and a residual value function, returns a maximum potential value for each (non-asset) attribute in the domain.

This algorithm works backwards from each action that produces assets without consuming other assets of equal or greater value (i.e. those actions that produce assets satisfying only WAMX). For each such action, the algorithm assigns the maximum residual value increase (MRVIA) of the action to the MPV of each non-asset attribute consumed by that action. The algorithm then recursively regresses this MPV through each action that produces each such attribute. As it regresses through each action, the algorithm deducts the MRVIA of the action from the MPV being propagated.

Algorithm MPV ($\mathcal{A}$, $\mathcal{V}$)

Let E be the set of domain attributes in $\mathcal{A}$ which are not assets in $\mathcal{V}$.

For all $e \in E$, let $PV[e] = 0$.

Let $A = \{a \in \mathcal{A} | \text{MaxRVI}(a) > 0\}$.

For each a in A, call $RegressMPV(a, 0)$.

Return PV.

Procedure **RegressMPV** (p, v)

Let $I = v + \text{MaxRVI}(p)$.

If $I > 0$ then:

Let C be the set of domain attributes consumed by p.

For each c in $C \cap E$ do:

Let c_{amt} be the quantity of c consumed by p.

Let $PV[c] := \max(\frac{I}{c_{amt}}, PV[c])$.

For each a in $\mathcal{A}$ that produces c do:

Let p_{amt} be the quantity of c produced by a.

call $RegressMVP(a, I\frac{p_{amt}}{c_{amt}})$.

Figure 4.3: The Maximum Potential Value Algorithm

Table 4.1: Example MPV values

Attribute	MPV (in dollars)
clean-fuel-drum	10
dirty-fuel-drum	5
full-fuel-drum	3
ration-coupon	0

The algorithm terminates when the regressed MPV is less than zero.

An MPV example For explanatory purposes, consider a domain in which there is only one asset, money, and that residual value is linear in the amount money. This allows us to describe MRVIA and MPV more clearly in monetary terms. Imagine that you have the following actions:

- *buy-fuel-drum* consumes 50 dollars and a *ration-coupon* and produces a *full-fuel-drum.*

- *refuel* consumes 2 dollars and a *full-fuel-drum* and produces a *dirty-fuel-drum.*

- *wash-fuel-drum* consumes 5 dollars and a *dirty-fuel-drum* and produces a *clean-fuel-drum.*

- *recycle* consumes a *clean-fuel-drum* and produces 10 dollars.

Table 4.1 shows the MPV values that would be calculated for each of the attributes in this example. The algorithm would begin with *recycle* and, working through the actions in the reverse of their listed order, produce the maximum potential values in the order shown in the table. Note that recursion stops when reaching *buy-fuel-drum* action because the regressed MPV value would be -47, so *ration-coupons* have no potential value.

The MPV algorithm will necessarily halt as long as the actions in the domain satisfy WAM. If WAM is violated, the algorithm will be infinitely recursive, since some attributes will have unbounded potential value. The MPV algorithm can be employed to check for such circularities, so providing a test of weak asset monotonicity.

4.4.2 Determining asset upper bounds

Section 4.3.1 described the constraint SLX, which bounded residual value by limiting the maximum amount of an asset. Before we begin generating plans, we need to determine, for each asset satisfying only SLX, its maximum attainable amount. This value, known as the *asset upper bound* (AUB), will be used during the planning process to calculate bounds on the residual value of partial plans.

Figure 4.4 shows an algorithm for calculating asset upper bounds on assets satisfying only SLX. This algorithm can also be used to test for satisfaction of SLX: if SLX does not hold for some asset, the precondition $\langle e, [l, u]\rangle$ will not exist for some action that produces that asset.

4.4.3 Calculating residual value bounds

We finally address the crux of this chapter—the calculation of the upper bound on residual value of a partial plan. We assume that the action schemas and residual value function have been analyzed as described in the preceding two sections to determine whether any assets satisfy only SLX or WAMX, and if so, to calculate the corresponding MPV's and AUB's.

The basic idea is to first calculate the *final asset position* entailed by the partial plan, based on the initial asset position and the effects of those actions already in the partial plan. (Recall that an asset position is a vector of the amounts of various

Algorithm AUB ($\mathcal{A}$, $\mathcal{V}$)

Let E be the set of assets not satisfying WAMX.

For each $e \in E$ do:

Let A be the set of actions producing e.

Let $UB[e] = \max_{a \in A} AssetLimit(e, a)$.

Return UB.

Function **AssetLimit** (e, a)

Let $\langle e, [l, u] \rangle$ be the resource precondition on e in a.

Let p be the amount of e produced by a.

Return $p + u$.

Figure 4.4: The Asset Upper Bound Algorithm

assets.) Next, this asset position has to be relaxed to allow for possible increases in assets which are only bounded by SLX, as described by the set of AUB's. The asset position value function $\widehat{V_R}$ is applied to the resulting asset position to determine a *baseline* bound on residual value. Finally, this bound is further relaxed to allow for possible improvement based on the conversion of limiting attributes, described by the MPV's. The subcomponents of this calculation will be described individually, and then composed together at the conclusion of the section.

Final asset position The final asset position of a partial plan is just the sum of the changes to assets by the steps in the plan applied to the asset position in the initial state:

Definition 32 (Final asset position of a partial plan (FAP)) *The* final asset position *of partial plan P is:*

$$\mathrm{FAP}(P) = \vec{A_{\mathcal{I}}} + \sum_{s \in steps(P)} \vec{\Delta}(s)$$

where $\vec{A_{\mathcal{I}}}$ is the asset position in the initial state and $\vec{\Delta}(s)$ is the effect on asset position due to action s.

If an action schema s is not fully instantiated, then the effect on any particular asset might be uncertain, and so elements of the $\vec{\Delta}(s)$ vector might be interval-valued. In general, FAP will return a vector of intervals representing the possible final amounts of each asset.

Upper bound relaxation If any of the assets in the domain satisfy only SLX, then the final asset position obtained from a partial plan must be relaxed to account for possible future increase in the amounts of these assets. This relaxation is performed using the asset upper bounds as follows:

Definition 33 (Upper bound relaxation (UBR)) *The* upper bound relaxation *of an asset position $\vec{A}$ with respect to a set of asset upper bounds AUB, denoted* $\mathrm{UBR}(\vec{A}, \mathrm{AUB})$*, is a vector $\vec{A'}$ such that for each asset i:*

$$\vec{A'_i} = \begin{cases} max(\mathrm{AUB}[i], \vec{A_i}) & \text{if AUB is defined for } i \\ \vec{A_i} & \text{otherwise} \end{cases}$$

where $\vec{A_i}$ is the element of $\vec{A}$ representing the amount of asset i.

Potential value relaxation Finally, we need to determine the final amounts of other attributes in the domain (those limiting the production of assets satisfying only WAMX) which have the potential further to increase residual value. We identify such attributes using the results of the MPV algorithm:

Definition 34 (Potentially valuable attributes) *Let* PVA *be the set of potentially valuable attributes, that is,* $\mathrm{PVA} = \{a | \mathrm{MPV}[a] > 0\}$.

The effect of a partial plan on the quantities of such attributes can be determined by a method similar to the calculation (above) of final asset position:

Definition 35 (Attribute final amount) *The* attribute final amount *for an attribute a and a partial plan P is:*

$$\mathrm{AFA}(a, P) = \mathcal{I}_a + \sum_{s \in steps(P)} \Delta_a(s)$$

where $\mathcal{I}_a$ is the amount of a in the initial state, and $\Delta_a(s)$ is the amount of a produced (or, if negative, consumed) by action s.

As before, if an action schema is only partially instantiated, the amount of an attribute produced or consumed might be uncertain, so the function Δ_a may return an interval value. Thus AFA is also in general interval-valued.

Putting it all together

The functions just defined can be used to calculate the upper bound on residual value of a partial plan. The final asset position is calculated, and relaxed according to the asset upper bounds. A baseline bound is calculated by applying the residual value function, and further relaxed by determining the possible increase due to each potentially valuable attribute:

$$UB(\widehat{V_R}, P) = \widehat{V_R}(\mathrm{UBR}(\mathrm{FAP}(P), \mathrm{AUB})) + \sum_{a \in \mathrm{PVA}} \mathrm{AFA}(a, P) \cdot \mathrm{MPV}[a]$$

In case there are any partially instantiated action schemas in the partial plan, FAP and/or AFA might return interval values. A sound upper bound is obtained by having such intervals take on their maximum value.

4.5 Summary

This chapter has addressed two issues central to the value-directed planning process: the conditions under which value will be bounded, and the determination of upper bounds on the value of a partial plan.

When posing a planning problem, the set of actions must satisfy certain constraints if a finite optimal plan is to exist. There are a variety of constraints that are sufficient to bound value, and the difficulty of calculating the upper bounds on a partial plan depends on the strength of the constraints that hold. We have presented a taxonomy of such constraints, called *asset bounding characteristics*, that vary in restrictiveness. The weaker constraints allow representation of a rich class of planning problems, but make the calculation of value bounds on a partial plan more difficult.

This chapter has also provided algorithmic techniques for the calculation of upper bounds on the value of partial plans. This calculation can be decomposed into the sum of bounds on the goal component and the residual value component. The former depends on the kind of goal being represented, while the latter depends on which asset bounding characteristics hold for the problem. We have presented algorithms that preprocess a domain to determine which asset bounding characteristics hold and derive the information necessary to calculate the upper bounds on residual value.

BIBLIOGRAPHY

[Agin 66] Agin, N. Optimum seeking with branch-and-bound. *Management Science*, 13:B176–185, 1966.

[Bacchus&Kabanza 95] Bacchus, F. and Kabanza, F. Using temporal logic to control search in a forward chaining planner. In Ghallab, M., editor, *Proceedings of the Third European Workshop on Planning (EWSP-95)*, 1995.

[Balas 79] Balas, E. Branch and bound/implicit enumeration. *Annals of Discrete Mathematics*, 5:185–192, 1979.

[Cassandra *et al.* 94] Cassandra, A. R., Kaebling, L. P., and Littman, M. L. Algorithms for partially observable markov decision processes. Technical report 94-14, Brown University, Providence, Rhode Island, 1994.

[Chapman 87] Chapman, D. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[Currie&Tate 91] Currie, K. and Tate, A. O-plan: the open planning architecture. *Artificial Intelligence*, 52(1):49–86, November 1991.

[Erol *et al.* 92] Erol, K., Nau, D., and Subrahmanian, V. On the complexity of domain-independent planning. In *Proc. of the Tenth National Conference on A.I.*, pages 381–386, 1992.

[Etzioni *et al.* 92] Etzioni, O., Hanks, S., Weld, D., Draper, D., Lesh, N., and Williamson, M. An Approach to Planning with Incomplete Information. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, San Francisco, CA, October 1992. Morgan Kaufmann. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[Etzioni 93] Etzioni, O. Intelligence without robots (a reply to brooks). *AI Magazine*, 14(4), December 1993. Available via anonymous FTP from `pub/etzioni/softbots/` at `cs.washington.edu`.

[Feldman&Sproull 77] Feldman, J. and Sproull, R. Decision theory and artificial intelligence II: The hungry monkey. *Cognitive Science*, 1:158–192, 1977.

[Fikes&Nilsson 71] Fikes, R. and Nilsson, N. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3/4), 1971.

[Fishburn 70] Fishburn, P. C. *Utility Theory for Decision Making.* John Wiley and Sons, New York, 1970.

[Fishburn 78] Fishburn, P. C. Value theory. In Moder, J. J. and Elmaghraby, S. E., editors, *Handbook of Operations Research: Foundations and Fundamentals*, pages 398–422. Van Nostrand Reinhold, 1978.

[Garey&Johnson 79] Garey, M. R. and Johnson, D. S. *Computers and Intractability.* W. H. Freeman and Co., San Francisco, 1979.

[Garfinkel 79] Garfinkel, R. S. Branch and bound methods for integer programming. In Christofides, N., Mingozzi, A., Toth, P., and Sandi, C., editors, *Combinatorial Optimization*, pages 1–20. John Wiley and Sons, New York, 1979.

[Ginsberg 87] Ginsberg, M., editor. *Readings in Nonmonotonic Reasoning.* Morgan Kaufmann, San Mateo, CA, 1987.

[Haddawy&Hanks 90] Haddawy, P. and Hanks, S. Issues in Decision-Theoretic Planning: Symbolic Goals and Numeric Utilities. In *DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control.* Morgan Kaufman, November 1990.

[Haddawy&Hanks 93] Haddawy, P. and Hanks, S. Utility Models for Goal-Directed Decision-Theoretic Planners. Technical Report 93–06–04, Univ. of Washington, Dept. of Computer Science and Engineering, September 1993. Submitted to *Artificial Intelligence.* Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[Hanks *et al.* 93] Hanks, S., Nguyen, D., and Thomas, C. The truckworld planning testbed, August 1993. Submitted to *IEEE Expert.*

[Kambhampati *et al.* 94] Kambhampati, S., Knoblock, C., and Yang, Q. Planning as refinement search: A unified framework for evaluating design tradeoffs in partial order planning. Department of Computer Science and Engineering TR-94-002, Arizona State University, 1994. To appear in Artificial Intelligence Special Issue on Planning and Scheduling.

[Keeney 92] Keeney, R. L. *Value-focused Thinking.* Harvard University Press, Cambridge, Mass., 1992.

[Keeney&Raiffa 76] Keeney, R. L. and Raiffa, H. *Decisions with Multiple Objectives: Preferences and Value Tradeoffs.* John Wiley and Sons, 1976. Republished in 1993 by Cambridge University Press.

[Krantz *et al.* 71] Krantz, D. H., D., L. R., Suppes, P., and Tversky, A. *Foundations of Measurement*, volume 1. Academic Press, New York, 1971.

[Lansky&Philpot 94] Lansky, A. L. and Philpot, A. G. Ai-based planning for data analysis. *IEEE Expert*, 9(1):21–7, 1994.

[McAllester&Rosenblitt 91] McAllester, D. and Rosenblitt, D. Systematic nonlinear planning. In *Proc. 9th Nat. Conf. on A.I.*, pages 634–639, July 1991.

[McDermott 82] McDermott, D. A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155, April 1982.

[Monahan 82] Monahan, G. E. A survey of partially observable markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.

[Moon&O'Connor 93] Moon, S. and O'Connor, J. T. Estimate of utility values using fuzzy set theory. In *Proceedings of the Second International Symposium on Uncertainty Modeling and Analysis*, pages 393–400. IEEE Computer Society Press, April 1993.

[Moore 85] Moore, R. A Formal Theory of Knowledge and Action. In Hobbs, J. and Moore, R., editors, *Formal Theories of the Commonsense World.* Ablex, Norwood, NJ, 1985.

[Morgenstern 88] Morgenstern, L. *Foundations of a Logic of Knowledge, Action, and Communication.* PhD thesis, New York University, 1988.

[Penberthy 93] Penberthy, J. *Planning with Continuous Change.* PhD thesis, University of Washington, 1993. Available as UW CSE Tech Report 93-12-01.

[Penberthy&Weld 92] Penberthy, J. and Weld, D. UCPOP: A sound, complete, partial order planner for ADL. In *Proc. 3rd Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 103–114, October 1992. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[Reiter 78] Reiter, R. On closed world databases. In Gallaire, H. and Minker, J., editors, *Logic and Data Bases*, pages 55–76. Plenum Press, 1978. Reprinted in [Ginsberg 87].

[Robinson 65] Robinson, J. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1), January 1965.

[Sacerdoti 75] Sacerdoti, E. The nonlinear nature of plans. In *Proceedings of IJCAI-75*, pages 206–214, 1975.

[von Neumann&Morgenstern 47] von Neumann, J. and Morgenstern, O. *Theory of Games and Economic Behavior*. Princeton University Press, 1947.

[Weld 94] Weld, D. An introduction to least-commitment planning. *AI Magazine*, pages 27–61, Winter 1994. Available via FTP from `pub/ai/` at `ftp.cs.washington.edu`.

[Wellman 93] Wellman, M. Challenges for decision-theoretic planning. In *Proceedings of the AAAI 1993 Symposium on Foundations of Automatic Planning: The Classical Approach and Beyond*, March 1993.

[Wellman&Doyle 91] Wellman, M. and Doyle, J. Preferential semantics for goals. In *Proc. 9th Nat. Conf. on A.I.*, pages 698–703, 1991.

[Wilkins 88] Wilkins, D. E. *Practical Planning*. Morgan Kaufmann, San Mateo, CA, 1988.

[Zhong *et al.* 90] Zhong, J. Z., Jian, B. Y., and Dong, L. X. A hierarchical analysis model for multiobjective decision making. In Baosheng, H., editor, *Analysis, Design and Evaluation of Man-Machine Systems*. Pergamon, Oxford, U.K., 1990.