AI Magazine Volume 9 Number 2 (1988) (© AAAI)

WORKSHOP REPORT

DARPA Santa Cruz Workshop on Planning

William Swartout, *Editor*

This is a summary of the Workshop on Planning that was sponsored by the Defense Advanced Research Project Agency and held in Santa Cruz, California, on October 21-23, 1987. The purpose of this workshop was to identify and explore new directions for research in planning.

The workshop was organized into five sessions. Each session was intended to examine some aspect of planning research or point directions toward future work. The first session, chaired by Thomas Dean of Brown University, examined the basic planning paradigms that have been developed to date and attempted to determine their range of applicability. The second session was led by Ted Linden of Advanced Decision Systems (ADS). It provided a detailed look at the representation of plans and goals. The panel for this session dealt with questions such as the following: What is a plan? How is it different from a conventional computer program? What information needs to be represented in plans and goals that current representations preclude?

The third and fourth sessions were attempts to provide cross-fertilization from other areas of AI. One way of gaining insight into possible directions for future research in planning is to examine related areas of AI in which recent progress has been achieved and attempt to understand the possible utility of such advances for planning research. Accordingly, the third session, chaired by Richard-Fikes of Price Waterhouse, examined the ways that truth maintenance technology has been and might be used in support of the formation, the monitoring, and the repair of plans.

The fourth session, chaired by Peter Cheeseman of NASA/Ames, examined the use of probability and reasoning about uncertainty in planning. Most current planning systems are limited because they assume a complete knowledge of the world and the effects of actions they might take, or they assume that any inaccuracies can be corrected at run time. Such assumptions are limiting: Many situations exist in which it is impossible to know the state of the world exactly, and the effects of actions cannot be predicted with certainty, but it is still necessary to plan because recovery at run time is not feasible. For example, in both medical and military systems, the precise state of the world is usually unknowable, and the exact effect of interventions cannot be predicted; however, one still cannot proceed blindly, hoping to recover from blunders. What is needed is a way to evaluate candidate plans in the face of such uncertainty. This panel was concerned with the ways in which ideas from probability and decision analysis could be applied to planning.

The fifth session was chaired by Drew McDermott of Yale University. This session dealt with planning and execution. Perhaps more than any of the other sessions, this session called into question the relevance of our traditional notions of planning, particularly as they apply to tasks such as directing the execution of a robot. The remainder of this report is a collection of summaries written by the panel chairpersons.

Planning Paradigms

Thomas Dean

Chair: Thomas Dean, Brown University. *Speakers*: Mike Georgeff, SRI International; Matt Ginsberg, Stanford University; Kris Hammond, University of Chicago; and Drew McDermott, Yale University.

For the purposes of this article, a *planner* is a program that controls one or more devices capable of carrying out actions in the real world in order to achieve some definite purpose. A strategic planner is capable of antici-

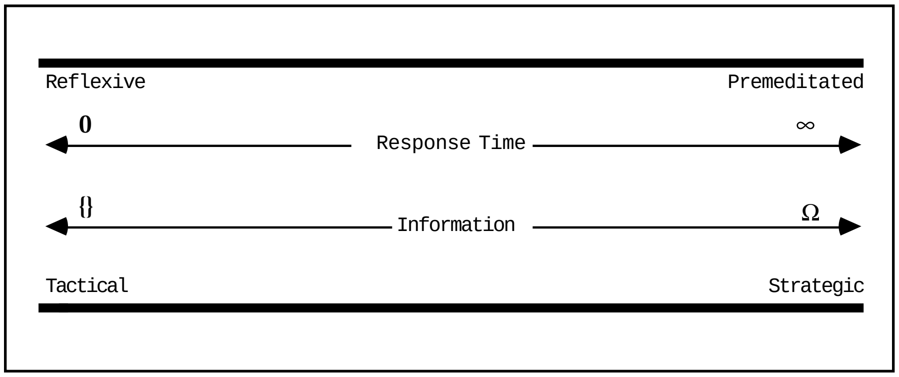

Figure 1. Two Extremes in a Continuum of Problems.

pating (some of) the consequences of its actions and using such anticipated consequences to choose between possible courses of action. Strategic planning techniques have primarily been concerned with representing the world in enough detail to allow prediction and efficiently carrying out predictions in order to support decision making. Three of the approaches explored by the panelists (Variations on STRIPS, Hierarchical Planning, and Case-Based Planning) address issues in strategic planning. Many situations exist in which accurate prediction is either impossible because of a lack of information or useless because of a lack of time. A *tactical planner* is primarily concerned with deciding what to do in situations in which the available information is limited and uncertain. The subsection on situated activity addresses the problem of deciding what to do on the basis of what the robot immediately perceives about its environment. The basic idea is that highly directed behavior can arise out of interaction with a complex environment to satisfy our requirement that planning ultimately achieve some definite purpose. Purely reactive or purely strategic planners can be considered as extremes on a continuum (see figure 1) determined by (1) the amount of time allowed for the system to respond and (2) the availability and volatility of the information potentially useful to the system in deciding how to respond.

The first four subsections summarize the position papers and presentations of the four panelists. It should be noted none of the approaches explored by the panelists constitute a paradigm in that it is a coherent and self-contained approach to solving problems, suggesting research directions, and assessing theories. The subsection entitled "An Emerging Paradigm" describes an attempt to coalesce some ideas that might give rise to a fledgling paradigm. The framework developed in this subsection is as yet incomplete but was felt to be promising by a number of the participants.

Variations on STRIPS

In this subsection[1] and the three that follow, we assume a familiarity with basic techniques. Readers wishing to know more about STRIPS should consult the original papers (Fikes and Nilsson 1971) or an introductory AI text that covers the material (Nilsson 1980).

STRIPS owes a considerable debt to Newell and Simon's General Problem Solver (GPS) (Ernst and Newell 1969). Somewhat simpler, STRIPS is a specialized and precisely formulated version of GPS that directly addresses the issue of reasoning about actions which precipitate change in the world. Perhaps the most useful contribution of STRIPS involves the form and content of action representations. In STRIPS and systems like it, actions are represented as functions from sets of sentences to sets of sentences in some appropriate language. Such a representation allows for a rather simple approach to planning often referred to, somewhat disparagingly, as *linear planning*. Linear planning using the STRIPS action representation utilizes a fairly efficient means for computing the results of an action, which are described in terms of changes to a database. For an action a, we can associate a database update function u(a) described as a function from sets of sentences to sets of sentences:

$$u(a):P(L) \rightarrow P(L),$$

where L is the language being used to describe any particular planning domain, and P(L) is the set of all subsets of L. The particular representation used by STRIPS associates with an action an add list A(a) and a delete list D (a), with u(a) given by

$$u(a)(S) = S + A(a) - D(a).$$

Even though the STRIPS action representation has been studied extensively over the last two decades, the advantages and disadvantages of the approach are still being debated. It is obvious that the STRIPS action representation has limited application; domains exist that cannot adequately be represented using such simple functions. The language L influences the expressivity of STRIPS actions as well as the efficiency with which the results of actions can be computed.

Certainly, more expressive languages exist for reasoning about time and change, but expressivity is not the only consideration. In many domains, efficiency is an important factor, and given the availability and volatility of information in such domains, more expressive languages might not provide an advantage in predictive ability, whereas they might prove a decided disadvantage in terms of speed.

It appears that the idea of representing actions as mappings from sets of sentences to sets of sentences in a suitable language will continue to play a role in planning research. The simplicity of the basic framework makes it amenable to analytic treatment (Lifschitz 1987), and its potential efficiency makes it a promising candidate for addressing complexity issues, such as those involving the frame and qualification problems (Ginsberg and Smith 1986).

Hierarchical Planning

Hierarchical planning[2] is perhaps the best known and most misunderstood of the approaches to planning explored in this article. A number of important ideas seem to be tangled in the current use of the term. This article does not make a concerted effort to unravel the tangle, but we try to identify a couple of important ideas that have come to be associated with hierarchical planning.[3] Hierarchical planning arose out of dissatisfaction with linear planning, as epitomized in programs such as STRIPS (Fikes and Nilsson 1971) and HACKER (Sussman 1975). In the linear planning framework, plans correspond to sequences of actions (usually quite primitive), and planning decisions tend to follow the temporal order of the actions being considered. It seemed that there should be some way of abstracting away from (1) the primitive nature of the actions themselves and (2) commitments to the precise order in which such actions were to be carried out.

In the STRIPS formulation, a partial plan corresponds to a sequence of primitive actions. At any given time, the planner is attempting to extend its current partial plan by inserting additional actions. It would be desirable if a planner's current partial plan could always be extended to a complete plan: a sequence of primitive actions that achieves the goal specification. However, because STRIPS often has to make arbitrary ordering decisions in order to keep the current partial plan linearly ordered, the current partial plan frequently cannot be extended to form a solution. Work by Sacerdoti (1977) using partial plans represented as partially ordered actions appeared to solve this problem by enabling a planner to postpone commitment to the order in which actions are to be executed. The idea of postponing ordering commitments was extended to postponing commitment with regard to other types of choices (Stefik 1981; Wilkins 1984; Tate 1977). However, the natural extension of such postponement strategies, often referred to as *least commitment planning*, has turned out to be problematic.

Planning is generally divided into two stages: (1) *refinement,* making choices (commitments) in the course of extending a partial plan, and (2) *validation,* determining if the commitments made so far are likely to lead to a complete plan. Postponing commitments attempts to simplify the refinement stage by complicating the validation stage. In the worst case, validating a plan corresponding to a partially ordered set of actions can be exponential in the size of the set of actions (Chapman 1987). Alternative incomplete methods for validating partial plans (Dean and Boddy 1987) exist, but it is now clear that least commitment is a heuristic strategy and not an easy solution to a fundamentally hard problem.

In retrospect, perhaps the most important use of hierarchical abstraction is concerned with the relation between tasks and subtasks (McDermott 1977). A planner can decide to perform action A and later decide to do A by doing $A_1 \ldots A_n$. We use *task* to represent an action the planner is committed to and *subtask* for a task it is committed to as a means of carrying out another commitment. Usually, when a researcher states that a planner is hierarchical, the researcher means that system uses this sort of abstraction.

The basic ideas behind hierarchical planning have come to be recognized not so much as an approach to planning but rather as part of the problem statement. In an important sense, planning is hierarchical and abstract. The subsection "An Emerging Paradigm" attempts to extract from the work of the last two decades a framework that might be considered as the basis for a research paradigm.

Case-Based Planning

Case-based planning(CBP)[4] takes as a starting premise that the organization of experience is paramount in formulating new plans and debugging old ones. Competing approaches differ on how experience is stored in memory and how experiences are retrieved and exploited during planning. Given that storage and retrieval are so important, most case-based approaches involve some theory of learning and memory organization.

HACKER (Sussman 1975) is an early example of an approach to planning that took learning into account. HACKER's memory organization techniques were somewhat simplistic, involving libraries of plans, debugging techniques, and domain axioms, and retrieval was done using standard pattern-matching techniques. Nevertheless, two important ideas surfaced from this work. First, planning consists of using and debugging known plans, and second, once a planner has constructed a useful plan, it should store the results for later use. Hammond's (1986) work emphasizes that what you learn from constructing and debugging a plan is much more than just the resultant plan: You learn about how and when to apply the plan; how to debug similar plans; when and where to apply your general knowledge about the world; and finally, how to index your newly found knowledge in such a way that you are likely to find this knowledge the next time you need it. Another memory-based approach draws on the learning research of Newell and Simon; the SOAR program (Laird, Newell, and Rosenbloom 1987) solves problems, generalizes on what was learned during problem solving, and caches the results for subsequent reuse.

Another offshoot from Sussman's work concerns the notion of debugging techniques that serve to transform a buggy plan into an alternative and, hopefully, bug-free plan. The idea of debugging "almost right" programs using transformations has received a fair amount of attention in automatic program synthesis (Green and Westfold 1982), and interest is picking up once more in planning (Simmons and Davis 1987).

During the discussions at the workshop, it became apparent that much of the terminology surrounding CBP was not well defined. In particular, terms such as *rules, plans,* and *cases* appeared to mean different things to different people. At one point, a group of participants attempted to make sense of the following equation:

Cases + Transformations = Plans + Refinement Strategies .

It was finally agreed that functionally the equivalence holds; that is, given enough time, a traditional hierarchical planner using traditional plans and refinement strategies can come up with exactly the same plans that a case-based planner comes up with using cases and transformations. CBP advocates claim, however, that on the average, the case-based planner comes up with a plan faster and that the plan is bug free more often than for the traditional planner. This issue is interesting because it seems to be addressing a basic time-space trade-off in planning. However, the issue cannot be resolved until there is some agreement about the meaning of the basic terms. This impasse points out the pressing need for precise terminology and appropriate analytic tools for comparing competing approaches. The intuitions behind CBP are exciting and have served to breathe new life into various areas of planning and learning research; however, without some agreement on terminology, further progress will be slow.

Situated Activity

For the most part, the planning approaches discussed thus far are concerned with *offline strategic planning:* problem solving with an emphasis on reasoning about change as a result of action.

Strategic planners tend to assume that the situation in which a plan is to be carried out can be predicted at the time planning occurs. If the current situation is completely known, and the planner has an accurate model of how actions serve to change the world, then, perhaps, this assumption is warranted (we haven't said anything about how long it takes to make the requisite predictions).
Situated activity[5] is concerned with deciding what to do based on what is currently known, where what is known is assumed to be significantly less than what is true. A situated planning system is responsible not only for formulating plans but for seeing that its plans are carried out properly.

In situations in which precise models are unavailable, or the system lacks critical information concerning the current situation, offline strategic planners are of limited use. With many interesting domains, much of the information required to decide how best to carry out a given task will have to be acquired by carrying out specific actions to gather this information. Strategic planners attempt to deal with such domains by (1) constructing highly conditional plans with many branches, most of which will never be used, or (2) using primitive operators that serve to invoke highly conditional routines which can themselves react quickly to a changing situation. The first method works well only in restricted domains; generally, far too many contingencies need to be dealt with. The second method simply ignores the problem by relegating it to someone else's module.

To deal with the problem of generating plans in uncertain environments, researchers have turned to systems that interleave plan formation and execution (Chien and Weissman 1975; Durfee and Lesser 1986). In such hybrid systems, it is possible to defer planning until information of a specific sort is available. We have already seen in the subsection on hierarchical planning that decisions about whether to defer commitment are fraught with complications. The trade-offs are complex, and issues such as dynamic replanning are just beginning to be addressed. As Georgeff notes in his position paper from the workshop, most existing systems are far too committed to the plans they formulate and tend to rely heavily on models of the environment and not enough on the environment itself. Such systems tend not to tailor their planning to the situation at hand. Given the same abstract task to achieve, these systems perform the same computations no matter how much time and information is available. They cannot determine when further planning is futile, and they do not have the capability to consider alternative strategies when pressed for time (Dean 1987).

In the last few years, a number of systems have been developed for dealing flexibly with uncertain environments. Brooks (1985) proposes decomposing the overall problem into task-achieving units realizing distinct behaviors. Rosenschein, Kaelbling, and Pack's (1987) approach to describing the behavior of situated planning systems promises to yield interesting formal properties (Rosenschein 1987). Chapman and Agre (1987) propose a scheme whereby a highly reactive system can achieve complex behaviors without the use of traditional abstract plans represented as quantified formulae. Georgeff and Lansky's (1987) work points out the need for a rich vocabulary for reasoning about intentions: what the planner is committed to doing, how important individual commitments are, and how various commitments are related to one another.

Until recently, the work in situated activity and strategic planning has proceeded along separate tracks, driven largely by orthogonal issues and run by separate groups of devotees. It appears from the workshop discussions that each of the groups is finally conceding that the issues which are of central concern to the other are important and deserve attention. This broadening view should set the stage for a synthesis of ideas resulting in a new generation of situated planning systems.

An Emerging Paradigm

In this subsection, we consider some ideas that might form the basis for a paradigm for planning research. A useful paradigm for planning would provide an analytic framework for designing, comparing, and communicating ideas about planning systems. A small group of workshop participants engaged in a lengthy and productive discussion aimed at developing just such a framework. In the following paragraphs, I will summarize some of the basic ideas put forth in these discussions.

In this framework, planning is a two-phase process, consisting of projection (a term owed to Wilensky 1983) and refinement. The planner has at all times an abstract plan that it is working on. *Projection* involves quantifying over instances of such abstract plans in order to determine how best to further specify (or refine) the plan. *Refinement* consists of transforming an abstract plan into another abstract plan. An abstract plan is likely represented as a set of sentences in some language expressive enough to capture facts about time. An instance is likely represented as a superset of the set of sentences corresponding to the abstract plan. It is useful, however, to speak directly in terms of the model theory.

A model of a set of sentences in the chosen language corresponds to one or more time lines (McDermott 1982) or possible worlds (Lewis 1973). An abstract plan, then, is a set of possible worlds, and an instance is a member of such a set. A transformation is just a function from sets of possible worlds to sets of possible worlds. The transformations used by existing planners include instantiating operators (adding sentences that introduce new action events) and ordering operator instances (adding sentences which restrict the order of action events). Still, this (emerging) paradigm is incomplete. We have said nothing about how the examination of instances of abstract plans affords any insight into how the abstract plan should be transformed. We haven't mentioned comparing plans, choosing a good plan, or choosing an optimal plan. Nevertheless, the approach, even as it stands, appears promising. The approach seems to provide an appropriate framework for categorizing planners (for example, most of the participants believed that NOAH and ABSTRIPS could profitably be analyzed using such a framework). The approach could be used as a tool for synthesizing new planners; some discussion took place of planners (Finger 1987) that manipulate sets of sentences which are inconsistent, that is, the abstract plan corresponds to the empty set of possible worlds. A number of techniques already exist that could be cast directly into the above paradigm (Genesereth 1984) as well as a growing literature describing methods for directly analyzing formal systems in terms of their model theoretic properties (Shoham 1986).

Conclusions

Initially, there was some reluctance on the part of the workshop participants to serve on the paradigms panel. The impression was that the older methods have little to offer us in the way of foundations, especially with regard to STRIPS and hierarchical planning. In retrospect, it seems that just the opposite is true. Those approaches which have been around the longest have the most to offer in terms of a set of ways for thinking about planning. New problems have been discovered and the older methods found wanting, but many of the basic intuitions that gave rise to the older methods have stood the test of time. Today, these basic and intuitively appealing methods are known largely by association with archaic and poorly understood programs; these methods cry out for careful reexamination and precise formulation. To be fair, some of the foundations have already been laid (Nilsson 1980). A great deal remains to be done, however.

The newer approaches (for example, CBP and situated activity) have yet to sort out what problems they are designed to solve. The researchers promoting these approaches have only recently succeeded in convincing the rest of the research community that the issues they consider important are not adequately addressed in existing approaches. The paradigm discussed in the previous subsection is by no means complete. It does not attempt to address the basic problems in controlling processes; it says nothing about gathering information concerning processes not under the direct control of the planner. In fact, our fledgling paradigm says nothing at all about computation, search, or sensing, aspects that cannot be neglected in any complete paradigm. A single paradigm that addresses all the planning issues is still needed.

A great deal of work must be done, and unfortunately, only a small number of competent researchers are working on the problem. In the early 1970s, a flurry of activity took place in planning research. This initial surge of interest was followed by nearly 10 years of inactivity, largely the result of the major funding agencies losing interest and withholding their support. The promise of planning technology for military and industrial applications is enormous. For the small investment made, a great deal of progress in planning has already been made in the 1980s, but without continued support for basic research, this progress will not continue.

Plan and Goal Representations

Theodore Linden

Chair: Theodore Linden, Advanced Decision Systems. *Speakers*: Reid Simmons, Massachusetts Institute of Technology; Barbara Hayes-Roth, Stanford University; Drew McDermott, Yale University; Nils Nilsson, Stanford University; and Phil Agre, Massachusetts Institute of Technology

This session focused on plan and goal representations as one of the key problems when building planning systems that integrate multiple planning methods. Not one of the planning paradigms discussed in the previous session offers a complete solution for most practical planning applications, and the effective integration of multiple planning methods depends on a common plan representation that is shared by the different methods.

In most planning systems, a plan representation is used for communication between a planner and a plan executor. In addition, many planning systems (especially those which use blackboard-based or transformational approaches to planning) represent intermediate states of the plan and then allow multiple planning methods to evolve the plan into a form suitable for execution. Thus, the plan representation not only communicates the plan to the plan executor but also serves as the primary means by which different planning methods cooperate.

Material covered in this summary is based on written material prepared by the panelists and by James Allen in preparation for the session, presentations during the session, and other relevant discussions and reflections from the workshop.

What Is a Plan?

Planning researchers are currently facing many new challenges, and the basic planning paradigms are changing. Thus, it is no surprise that a controversy exists about what a plan is.

As chairman, I tried introducing an analogy between plan representations and programming languages, with the idea that plans could be described as similar to programs except.... I was expecting responses along the lines of the following, which came from Simmons: "The type of language one would use to tell the executor how to carry out actions is analogous to a programming language, such as LISP. However, just as research in automatic programming (e.g., Programmer's Apprentice) has found that programming languages are impoverished with respect to the types of information needed to do program synthesis and explanation (e.g., they need data flow as well as control flow, design history, etc.), so too we find that, although planning and plan execution share a common representational framework, the language needed to do planning must be much more expressive than that needed for execution."

The idea that plans are like programs was radically challenged by Agre. Agre argued against any model where a smart planner gives detailed instructions to a dumb plan executor. A plan executor needs to be smart enough to deal with most situations that might arise, and once the user of the plan is this smart, then the language in which a planner communicates with the plan user depends strongly on the competence and the needs of the plan user.

Hayes-Roth also disagreed with the analogy between plans and programs. She characterized a plan as a "temporally organized pattern of intended action descriptions." The action descriptions and the temporal relations between them are much less rigid than the operators and control flows that can be expressed in most programming languages. The work by Hayes-Roth and her colleagues in applying the BB1 architecture to planning applications represents some of the earliest and most long-standing work on integrating diverse planning methods. Her recent work on declarative representations of actions and plans is implemented in several planning applications, and it shows the integration of skeletal planning, hierarchical planning, linear (assembly) planning, CBP, and planning by analogy.

The idea of having a denotational semantics for plan representations was raised during the session. This direction seems a good one for research; however, this idea was not developed by any of the panelists. I return to the question about what a plan is at the end of this summary; first, I cover some additional points made by the panelists.

Information Content for Plans

Greater agreement existed about the kinds of information that should be represented as part of a plan. Simmons's elaboration on the minimal information content needed in a plan was consistent with comments from many others.

Plan representations are needed for two separate but related tasks: planning and plan execution. For plan execution, certain basic knowledge is needed.

- A list of actions: The actions should have some duration and must describe how the action can be executed in the real world, for example, by being associated with a program that sends a sequence of commands to the robot.
- Temporal information constraining the order of actions; This information should include the full range of temporal orderings between sequences of commands to the robot.
- Parameter bindings for the actions, describing which objects participate in an action.
- Preconditions of action: It is desirable to represent the action's preconditions separately from the actual body of the action so that the plan executor can use this information to decide if and when to perform sensory operations to decide whether an action is executable.

In addition to these four items, three more types of information are needed for planning:

- The effects of actions: These effects must be represented in a form that the planner can reason about.
- The goals one is trying to achieve.
- Dependency information relating the effects of actions to the achievement of goals and subgoals: This information is necessary for efficiently determining how changes to one part of the plan affect the plan's global character.

This framework should be relatively noncontroversial. However, it is relatively weak in that many plans might be difficult to represent in the framework. Some additional information is needed to ease the burden of planners (and executors):

- Abstraction: This is widely recognized and does not need much explanation. One observation is that plan executors, as well as planners, can benefit from the abstraction hierarchy, for example, by checking whether the preconditions of an abstract action are met before attempting to execute its detailed subactions.
- The planning decisions and assumptions underlying the model of causality and the model of planning should be made explicit. For example, the planner (and, in particular, the replanner) should have access to information describing why, in achieving a particular goal, one action was chosen over another or what assumptions helped decide an action ending at time T1 to achieve a goal at time T2.

Goal Representation

Goal representations appear in the interface between the planner and the human. Although these representations could be represented differently from the plans appearing in the interface between the planner and the plan executor, good reasons exist to look for a single wide-spectrum representation scheme that can represent intentions at all levels, from abstract goals to the lowest level of executible plans. A wide-spread representation encourages generality in the planning methods by allowing each method to be applied at any level of abstraction.

McDermott, Simmons, and James Allen discussed the need to represent goals or intentions that cannot be expressed as a simple conjunction of statements about a single achievable state. As they pointed out, it is possible to write statements about the future that deal with sequences of goals, define goals which are to remain true or false while other goals are achieved, and so on. The problem is not in writing complex statements about future states of the world. The problem, quoting McDermott, is that "every planning method we know of relies on dividing a problem into pieces, producing plans for the pieces, and reconciling them. It is hard to divide an arbitrary statement (replete with quantifiers, connectives, and modalities) into pieces that make sense." Thus, just as there seems to be no practical, ultimately general planning method, it appears also there is no useful, ultimately general plan representation that allows goals to be decomposed in domain-independent ways.

Control Structures for Plans

It was generally agreed that for many applications, the plans produced by a planner need to be able to express conditional, iterative, and concurrent action executions. Partially developed plans need to be able to represent general temporal constraints between actions.

Nilsson is "exploring the alternative of creating plans expressed in a production-system programming language rather than in ordinary programming languages. Such programs have a short sense-act cycle and have more built-in conditionality. Thus, they can respond to environmental changes in a more timely fashion. They can also be easily modified by adding and/or subtracting production rules, and thus planning and plan execution processes can be more tightly integrated. In this view, we regard the planning process as one of maintaining the production rules (governing the system's actions) in rational balance with the systems beliefs and goals."

The Role of Plans in Controlling Actions

Let us return to the basic question about what a plan is. I focus specifically on the role of planning in robotics and other dynamic control applications. (Many other planning applications exist, such as the Defense Advanced Research Projects Agency battle management applications, where execution of the plan is not automated; these questions then do not arise in the same form).

A simplistic view was that a planner provides the intelligence and is the source of all decisions about what a robot should do. The problem with this view is that traditional planners build models of possible futures and then generate and evaluate plans with respect to these models. Most robotic control decisions must be made quickly; furthermore, it is often impossible (and unnecessary) to foresee enough to model it in any reasonable way.

Everyone probably agrees that most of the decisions which need to be made by a robot (especially lower-level decisions) do not require the construction of elaborate models about possible futures. So the question comes down to whether we want to view these simpler decisions as based on some new, simplified form of planning (called reactive planning, situated planning, or whatever) or whether we want to admit that most activity does not require planning. The latter seems a reasonable choice, but this question really needs to be discussed in the context of a more sophisticated model of hierarchically structured control systems.

Hierarchical Systems for Controlling Activity

During the workshop, abstraction levels and hierarchies were mentioned frequently; however, when talking about hierarchical control systems, it is important to distinguish different hierarchies within a planning system. Hierarchical levels can be based on subtasking, abstraction levels, and reflection.

These principles—largely orthogonal—interact in complex ways and they are principles by which a system can be decomposed into levels. Planning systems, of course, make extensive use of goal decomposition (subtasking) and abstraction levels. Note that *abstraction* is the technique which offers a way out of the exponential growth in complexity that characterizes most planning algorithms (see, for example, Korf 1987).

Reflection is the principle that distinguishes between acting and thinking about acting. If we take action as basic (a point advocated by Agre), then planning is reflection about acting, and metaplanning is reflection about planning. Thus, we can view planning as metaactivity; it is also reasonable to assume that there should be an order of magnitude or more of activity than there is of metaactivity.

A reflection hierarchy is orthogonal to an abstraction hierarchy. This distinction was made clear in Molgen, where there was reasoning about domain objects at multiple levels of abstraction as well as metaplanning. In Molgen, one could view the reflection hierarchy as primary, with the domain objects organized in an abstraction hierarchy. For robot control, one might rather view abstraction levels as primary, leading to an opportunity for reflection (planning) in the control decisions at each abstraction level. A robot's abstraction levels might include levels for deciding where to go, avoiding obstacles while traveling, extracting linear features from an image, and so on. Discussions about planning versus execution should be relative to a given abstraction level. What is execution on one level of abstraction might well involve planning on a lower level of abstraction. Thus, no single distinction exists between a planner and an execution agent; rather, at each level of abstraction, control decisions must be made (possibly by a planner).

This proposal that the distinction between planning and execution should be relative to abstraction levels is one approach toward implementing the kinds of smart plan users that Agre argued for. However, Agre explicitly argued against this approach toward implementing smart plan users because it is difficult (the alternatives are not yet clear).

We should not assume that all decisions about actions are made by planning. At each abstraction level, the builder of a robot can choose whether the system should just act or whether it should first think about how to act. This decision involves the usual trade-off between performance and flexibility. At any level of abstraction, if we build the decision criteria into the control system, then we will have better performance and, typically, less flexibility. If the run-time system first thinks about alternative actions it could take, it has greater opportunities to take additional factors into account when deciding how to act but not without a clear run-time performance cost. Furthermore, reflection is useful only when there is enough information available to build and reason about a world model. When this information is available, reflection enables decisions that are more flexible and more complex than the kind of decision-making which can be handled in practice by building decision criteria directly into the run-time system. Reflection is more likely to be useful at higher levels of abstraction, where flexibility is more important and performance is less important; however, the choice of whether to act or reflect applies at all abstraction levels.

The distinction between acting and thinking about acting is somewhat elusive; a hierarchy based on reflection does depend on one's viewpoint. All hierarchical structures are simply design issues; in principle, a system with the same functionality can be built without using any of the hierarchical design principles. Indeed, one of the standard effects of compilation is to map a system designed with hierarchical structures into a flat, high-performance system. Stan Rosenschein, Nilsson, and others are working on ways to compile planning systems into high-performance action systems, achieving much of the flexibility of the planning system and preserving the performance of systems that do not have to reflect on their own activity at run time.

These "reflections" lead to the conclusion that plans are representations for proposed courses of action which can be reasoned about and evaluated before the action is undertaken. Planning is the most important technique available for building systems that can act in flexible and intelligent ways; however, planning is not the only way to achieve flexible systems, and planning is not a prerequisite for acting.

The Use of Truth Maintenance in Planning Systems

Richard Fikes

Chair: Richard Fikes, Price Waterhouse. *Speakers*: Phil Agre, Massachusetts Institute of Technology; Paul Morris, IntelliCorp; Yoav Shoham, Stanford University; and Dave Smith, Stanford University.

In this session, we examined ways in which truth maintenance technology has been and might be used in the formation, monitoring, and repair of plans. *Truth maintenance technology* enables efficient maintenance of explicit records of the assumptions made in a model and of the justifications for beliefs derived from these assumptions. It promises to be a valuable tool in planning systems because of the centrality of making assumptions about an external environment while a plan is being constructed and of responding to variations from these expectations as the plan is executed. In addition to exploring this potential, the session highlighted a collection of pitfalls that occur in the use of truth maintenance for planning. This discussion is an overview of the potential of the technology; some of the pitfalls of the technology are also discussed.

Truth Maintenance Systems

The development of truth maintenance systems (TMSs) seems to have been motivated primarily by two perceived defects in previous inference techniques. First, the deficiencies of chronological backtracking had become apparent from experience with languages in the PLANNER family (Bobrow and Raphael 1974). Second, it was recognized that reasoning in many practical situations requires the making of assumptions which could be retracted later if they turned out to be incorrect. Because this reasoning based on assumptions meant the set of believed facts could diminish as well as increase, it was referred to as *nonmonotonic reasoning*.

Nonmonotonic reasoning allows conclusions to be drawn in the absence of contrary information. For example, if we leave our car in the parking lot, we expect that it will still be there when we return. If a meeting takes place every week in a certain room, we expect this schedule to continue unless we are notified otherwise. These conclusions are really assumptions, which might turn out to be wrong. Further reasoning can proceed from these assumptions. To allow for recovery in case of error, the system needs to maintain a record of dependencies so that it can undo all the consequences of invalidated assumptions.

The need to retain records of justifications for beliefs led to the development of a TMS by Doyle (1979) that maintained explicit records of the assumptions made in a model and the justifications for beliefs derived from these assumptions. By keeping track of these justifications, a TMS is able to determine the assumptions on which a conclusion is based, withdraw a conclusion if these assumptions cease to be believed, and efficiently reinstate the conclusion if the assumptions are believed once again. In the event of a contradiction, the assumptions underlying it can be quickly determined so that they can be revised as appropriate.

The ability of a TMS to identify the assumptions that cause a contradiction led to the development of a search technique called *dependency-directed backtracking* (Stallman and Sussman 1977), which is much more efficient than chronological backtracking in many situations. The reduction in search is obtained by representing each choice made by the searcher as an assumption and supplying justifications to the TMS for all derivations based on these choices. When a contradiction is derived, the TMS identifies which choices are responsible for the failure so that the searcher can proceed directly to changing them, even if they are not the most recent choices made. Note that in general, a TMS implicates a set of choices in the failure. The problem of which of the implicated choices to consider changing remains.

Truth Maintenance and Planning

The creation of a plan is based on a set of assumptions about task goals, the external environment, and the actions available as plan steps. The planner makes choices during plan creation that can be considered further assumptions.

During plan creation, when a situation that is expected to be produced by executing plan steps is inconsistent or is a dead end with respect to achieving the plan's goals, a TMS can be used to implicate the assumptions responsible for the failure. A point made in the session is that for such plan creation failures, assumptions representing planner choices are typically the ones to be reconsidered, not the assumptions about the environment or action repertoire. Changing assumptions about the environment or action repertoire to fix an expected failure corresponds to a form of wishful thinking. For example, a planner might predict that opening the door of a bird cage would produce the desirable result of allowing the bird to be fed and the undesirable result of letting the bird escape. We would not want the planner to avoid the undesirable result by simply changing its environmental assumption that the bird wants to escape. Instead, we want it to change its choice about opening the door.

Note that in situations where alternative plan choices are highly undesirable (for example, costly or dangerous), questioning critical assumptions about the environment or action repertoire can be a reasonable strategy (for example, "Is that bridge really out?" "Can the truck carry additional cargo?"). In such cases, the planner can recommend information-gathering actions to confirm or refute the assumptions. In extreme situations, it might be necessary to change or ignore environmental assumptions ("Damn the torpedoes!") and proceed at risk.

In contrast, failures during plan execution are typically recognized as undesirable properties of the current situation. Because it is no longer meaningful to change the choices that produced the current situation, analysis of such failures involves determining which of the environmental or action assumptions implicated by TMS are invalid (for example, "The book wasn't in the library." "The car would not start."). Diagnosis might need to be done to determine which assumptions must be revised to account for the unexpected state (for example, which component of the car failed). The justification structures and built-in search capabilities of an assumption-based truth maintenance system can be used as the basis for a diagnostic procedure to do this task (as in the work on multiple-fault diagnosis by de Kleer and Williams 1986).

Danger, Proceed with Caution

Although the integration of truth maintenance technology into planning systems seems attractive, planning system designers need to be aware that there are subtle problems involved in its use which can cause significant difficulties in planning systems. A good example is the much publicized Hanks and McDermott (1986) anomaly in which an intuitive formulation in the default logic of an action sequence produces a nonintuitive interpretation of the events.

In this session, we explored two such problems. The first, presented by Morris, was the wishful thinking issue described earlier. The second, presented by Smith, was what he considers to be a fundamental problem with using truth maintenance when reasoning about temporal quantities.

Suppose that a fact A is true at time t0 (denoted as [A,t0]), and we have the implication (ForAll t) ([A,t] implies [B,t]), allowing us to conclude [B,t0]. Now suppose that an action takes place causing A to become false at t1. What happens to our belief in B? As it turns out, this question does not have a simple answer; cases exist where B should be retracted at t1, but cases also exist where B should be preserved at t1 (even when no other supporting justification is present for B). Indiscriminantly applying truth maintenance can result in the loss of important information. For example, we can apply the transitivity property of LeftOf to conclude that because X is LeftOf Y and Y is LeftOf Z that X is LeftOf Z. Now, if we move Y so that it is no longer LeftOf Z, then TMS would remove the conclusion that X is LeftOf Z, even though the movement of Y does not affect the relationship between X and Z.

In order to solve this problem, we require additional information about the connection between the propositions. The question becomes, "What information is required and how do we supply it? We can fix the problem by supplying frame axioms or explicitly listing all the ramifications of each action. However, this solution is hardly satisfactory because of the epistemological and computational problems that result.

Dependencies and Planning

TMSs maintain dependencies between derived information and the explicit assumptions on which the derivations are based. Planning systems can exploit a general notion of dependency links in several additional ways. For example, a planner might have identified for each plan step a set of propositions whose truth before this step assures that nominal execution of the remainder of the plan achieves the task goals (that is, the notion of a kernel as developed in Fikes, Hart, and Nilsson 1972). An important augmentation of the basic kernel idea is to associate with each kernel entry (that is, proposition) the goal(s) in the plan the entry is expected to satisfy (that is, which task goal or plan step precondition) (as in Tate's goal structures).

Given such dependency information associated with kernel entries, the monitor can, for example, respond to a failed kernel by invoking a dependency-directed replanning effort in which new plan fragments are constructed only for the goals affected by the failed kernel entry. For example, if two towers are to be constructed, and a failure occurs during the construction of the first, then a new plan is constructed only for the remainder of the first tower; the existing plan for the second tower is left intact. Such dependency information is important basic information about the structure and intent of the plan that can be useful in most replanning efforts.

Another important use for dependency information is in linking planner decisions to environmental assumptions. If a plan monitor knew which environmental assumptions were used during plan construction, then it could monitor these assumptions and reinvoke the planner when any of them were invalidated. For example, if an obstacle is unexpectedly removed from a path, then the plan steps to circumvent the obstacle could be replaced by more direct steps. In order to identify and monitor all the environmental features that were used to choose a given plan fragment over all others, the planner would need to provide the monitor with TMS-style justifications for choosing one plan step over another, one ordering of steps over another, and so on.

It is doubtful whether a system can effectively monitor all such environmental features in nontrivial domains because the justifications need to include in some form all the subgoals considered in all the alternative plans, because if any of the subgoals in an alternative plan completion become true during plan execution, then this alternative needs to be reevaluated.[6] In addition, the monitor needs to know what facts cause new alternatives to become available to the planner (for example, a new block being put on the table). For example, a new operator instance could become available that satisfies a given goal. That occurrence might happen if the operator repertoire is added to or changed. More interestingly, the occurrence might happen if a new binding becomes available for an operator parameter (for example, a new block is discovered).

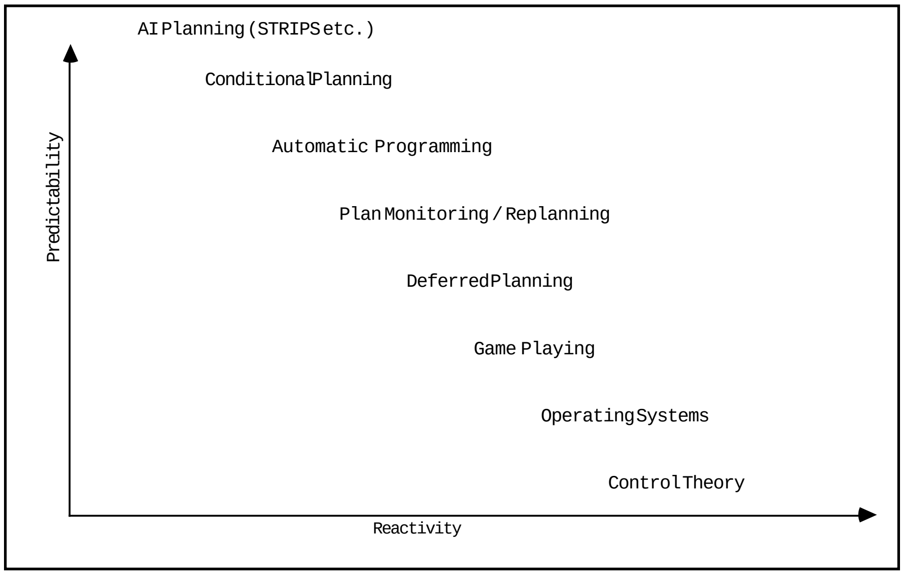

Figure 2: The Predictability-Reactivity Trade-Off

A practical approach to monitoring for the opportunity to change to an alternative plan will apparently require identifying, during plan creation, only the subset of alternatives and factors that would make a significant difference in some way (for example, in resource expenditures or risk) and monitoring only these. In any case, the TMS-style dependency structures will play a central role in the process.

Uncertainty and Planning: A Summary

Peter Cheeseman

Chair: Peter Cheeseman, NASA/Ames Research Center. *Speakers*: Larry Fagan, Stanford University; Ben Wise, Dartmouth University; and Mike Genesereth, Stanford University.

The panel on probability in planning focussed primarily on planning under uncertainty. A summary of the themes of the presentations and subsequent discussions is shown in figure 2. This summary is entirely the view of the author and should not be blamed on the panelists.

The basic idea behind this diagram is that there is a trade-off between the predictability of the environment in which the plan is to be executed and the degree of reactivity which is necessary to successfully achieve the goals of the plan. Traditional AI planning lies at one extreme of this trade-off, where complete knowledge of the world and the effects of possible actions on the world is assumed. Traditional control theory lies at the other end of this trade-off because here there is almost no look ahead (advance planning); instead, the system reacts immediately to differences between its predicted state and its observed state. In order of increasing reactivity, the kinds of planning represented in this trade-off are traditional AI planning, conditional planning, automatic programming, plan monitoring and replanning, deferred planning, and game playing.

Traditional AI Planning. Traditional AI planning includes such well-known systems as STRIPS, NOAH, NONLIN, SIPE, and DEVISOR as well as many scheduling systems. The important characteristic of these systems is how much predictability they assume about their domain. Typically, all the relevant aspects of the initial world are assumed known, and the effects of actions (as represented by operators) are assumed correct. Few real-world domains meet these strong predictability requirements. The only ones I know of are manmade cases, such as remote spacecraft (complete isolation) or machine setups (where the environment can be forced to agree with assumptions) and scheduling. Even with these assumptions, many problems of theoretical interest exist, in particular, the frame problem, temporal persistence, search control of a potentially huge search space, truth maintenance, interaction detection and resolution, temporal representation and reasoning (that is both expressive and efficient), goal representation (including trade-offs between goals), constraint handling, and so on. Although these problems are far from solved, the consensus at the meeting was that the main problems in building realistic planning systems lie in the relatively neglected area of planning under uncertainty.

Conditional Planning. If a traditional AI planner is missing information about the world at plan time, or nondeterministic operators exist with multiple outcomes, then standard planning methods will not work. A method of coping with a limited amount of such uncertainty is conditional planning. Conditional planning involves generating alternative plans for each possible outcome or world state, including an appropriate test in the plan to choose from the alternatives. Successful conditional planning requires solving a number of problems. The main problem is the explosion of possible worlds if many conditions exist. This explosion occurs because the world is in a different state, depending on which branch is taken; thus, subsequent choices can depend on earlier choices, leading to a potential exponential growth in the number of possible worlds. The explosion can largely be avoided if actions are taken to ensure that the world is effectively in the same state regardless of which branch is taken (that is, the branches rejoin, avoiding an exploding tree of possible worlds). In general, conditional branches should be generated for all situations where failure is considered likely, and the number of possibilities is small. Note that the introduction of conditions removes the uncertainty as far as the planner is concerned because every branch in the plan is completely determined. The loss of predictability is the result of a lack of knowledge about which branches will actually execute at run time.

Automatic Programming. Automatic programming is usually regarded as a separate area of research in AI, rather than as a form of planning. This situation is unfortunate because both areas are closely related. In automatic programming, the execution system is always a particular computer or a particular target language (for example, LISP). Because a computer is simple minded and predictable in its behavior, it is possible to produce plans (programs) with a large number of steps and prove properties of program execution (for example, termination) that would be impractical for an error-prone real-world execution system. Conditional planning, described earlier, corresponds to IF statements in programming, and simple commands (for example, assignment, +, and CONS) correspond to actions in AI planning. The main difference from AI planning is that automatic programming is more concerned with planning for a whole class of possible goals (for example, inverting a matrix and reversing a list) than planning for a set of specific goals. Also, in program loops, the world only changes as dictated by the program; in planning, however, iterative loops are more likely to fail because of unpredictable world events. These differences are only of emphasis and not a fundamental distinction; so, many of the methods developed for automatic programming could be applied to planning, and vice versa. As for conditional planning, the only uncertainty concerns what will actually execute in particular cases—the program must be correct for all the allowed input.

Plan Monitoring and Replanning. An AI planner not only produces a plan (that is, a [partially] ordered set of actions) but also a model of the world over the period covered by the plan. During execution, it is important that the observed situation be checked against the expected world state(s) and that any mismatches be discovered. This check can be achieved by the planner inserting sensor steps in the plan to verify that the desired world state is achieved and remains achieved. To do such sensor planning, the planner must have a model of what the sensors can detect and what conditions are necessary for their application. This requirement implies that the available sensors have well-defined capabilities. The sensor capabilities can be captured as operators similar to the standard action operators in AI planning; the main difference is that the effect of a sensor operator is to generate information, not change the physical state of the world. The modeling of sensors as operators allows the planner to reason about information goals using the same procedures it uses for state goals. This integration of action and sensor planning allows the use of active sensors that require actions in order to perceive. For example, a camera attached to a robot arm requires movement of the arm to see difficult areas; this robot motion can potentially interfere with other actions. This example makes it clear that sensor planning must be integrated with action planning and generally cannot be done as a separate postplanning step.

An alternative to the planner doing all the sensor planning is to allow the execution system to take some of the responsibility for correct plan execution. This possibility is the only one if prior information is insufficient to decide appropriate sensor choices at plan time. For example, possible camera occlusion is often difficult to decide in advance; so, the choice of camera location should be deferred to execution time when occlusions can easily be detected and alternative locations selected. Sensor steps can be included in predefined operators in a form transparent to the planner. For example, if there is a sensor that can track a seam, a weld operator can be specified which only requires the robot welder to be positioned at the beginning of the seam; the weld operator does the rest through sensor feedback. Without such a sensor, the planner would have to do detailed geometric reasoning to compute a welding path. The optimal balance between advance sensor planning and execution time sensor selection depends strongly on the context.

Replanning is necessary when new information is found that invalidates the previous plan (before or during plan execution), or situations arise during execution which were not planned for. Replanning is not different from planning; the main difference is in the trade-off between speed and optimality. During planning, considerable effort can be expended to search for an optimal (or near optimal) plan; in online replanning, however, it is usually more important to produce a correct plan quickly, even if it is suboptimal. If time is critical, it might be necessary to sacrifice correctness because to act on a plan that has not been fully checked is usually better than to not act at all. If during planning, particular execution failures are considered likely, the planner should preplan responses—a form of reflex response. These reflexes can be elaborate, precompiled conditional plans and should be considered conditional planning rather than a replanning response.

Deferred Planning. If important information is missing in the initial world description, it is possible to insert into the plan a call to the planner to achieve the unplanned goal(s) at execution time. This approach is possible if the missing information is expected to be available when (if) the planner is called at execution time. For example, it might not be known at plan time where free space will be to put something. Instead of failing at this point, the planner should insert a call to itself in the plan, so that when (if) it gets called, it is able to decide, in the particular context, where the placement should be. This procedure is possible because at execution time the information about the location of objects will be available through sensors. This deferred planning greatly extends the range of situations that can be planned for. However, for it to work, the planner should check (at plan time) that the information and resources needed to plan (at run time) will be available, that is, deferred goals are likely to be achievable at execution time. If most of the planning is deferred, the result is a high-level plan that defines an overall structure, with all the detailed actions to be decided during execution. Humans seem to rely heavily on such high-level planning, and automatic planners can also benefit in many situations by deferring decision making to execution time, when additional information is available.

Game Playing. The essential difference between game playing and planning is the lack of predictability as a result of the uncertainty of other players' responses. Multiagent planning is a type of cooperative game, where the planner could be one of the agents or a dictatorial agent that to some degree can control the other agents through a centralized enforcement. Planning under these circumstances is difficult primarily because of the possible interactions and uncertainties in the other agents' behavior. This uncertainty is true even for perfect information games, such as chess. When planning is extended to include actively hostile other agents (opponents), the goals of these opponents give some degree of predictability to their behavior (moves), but the range of possible responses (and your response to their response, and so on) makes detailed long-range planning impossible. Under these circumstances, planning is reduced to taking immediate actions (moves) that are likely to advance the planners' goals regardless of what the opponent(s) do. based on lookahead. When there is uncertainty about the world and the effects of operators, traditional AI planning is perhaps better viewed as a game against nature. This view shows, for example, the implausibility of plans consisting of long chains of actions, each dependent on the previous so that game playing is necessarily highly reactive.

Discussion

The trade-off between predictability and reactivity shown in figure 2 leads to a number of consequences, as defined in the following paragraphs.

Reactivity and response time. The more reactive a system is, the shorter the available response time tends to be, leading to faster feedback cycles. Because of the shorter cycle time, highly reactive systems have little time to think before responding; so, precompiled stimulus-response-type reflexes are necessary, as with standard control theory. Note that where a possibility exists of the system going into oscillation because of overreaction, the methods of standard control theory are needed. AI approaches tend to assume that if there is a divergence between the model and the observed world, then it is only necessary to apply the appropriate operator to correct the problem. This kind of "bang-bang" control is not always appropriate.

Plan length and predictability. The more predictable the execution environment, the longer the plan can be with a reasonable expectation for successful completion. Automatic programming, in particular, can produce plans (programs) millions of steps long that usually complete successfully. Producing such big plans is only possible because a computer is such a predictable environment; the main source of unpredictability here concerns the input to the program.

Sensor information and predictability. Highly reactive situations tend to require large amounts of sensor data to compensate for the lack of predictability. The quality and quantity of sensor data that is needed varies inversely with the ability to model (predict) the world; if there is sufficiently good sensor information, it might not be necessary to model the world at all.

Reactivity and hierarchical planning. The higher the level in the planning and control hierarchy, the more predictable the environment tends to be. The hierarchy tends to be chosen specifically to make this statement true.

Correctness versus robustness. Traditional AI planning systems tend to concentrate on producing correct plans. Although this method is appropriate for highly predictable environments, it is much more important to produce robust plans in realistic situations. *Robustness* means that the plan is likely to succeed no matter what unanticipated conditions arise; that is, robust plans avoid including commitment to courses of action which allow few options if they fail in execution. For example, we try to avoid travel plans that include tight airline connections even though such a plan is "correct."

Many sources of unpredictability in planning require greater reactivity. The main source could be called model uncertainty, which includes lack of information about the initial state, uncertainty about the effects of operators (for example, nondeterministic operators), and uncertainty about the actions of other agents (if present). Sometimes, this uncertainty can be reduced by performing actions that force the world into a known state. For example, shaking a table ensures that no object is on top of another object or offering sufficient rewards persuades other agents to act in specified ways, and so on. However, a planning system must cope with the fact that it is not always possible to achieve a particular set of goals; so, finding a suitable trade-off between the utility of a partially achieved goal set and the cost of achieving them becomes the main problem.

The assumption behind the discussion so far is that it is always better to do advanced planning if the available information permits, instead of just reacting as needed. Advanced planning can avoid situations where it is necessary to undo actions because of a lack of forethought. For example, it would not be desirable for the members of a NASA mission to Mars to discover that the exit hatch can only be opened from the outside. However, several workshop participants noted that humans manage well without a great deal of advanced planning, largely because we rely on generalized precompiled plans (such as a "go-to-work" plan) that include steps whose effects are not needed until later. For example, the general "go-shopping" plan includes the "pick-up-keys" step, even though they are not needed at this stage of the plan. Presumably, such steps have been learned and compiled as a result of experience. It is because we live in a reasonably benign world that we can usually achieve our goals no matter what happens (or do we change our goals to fit what does happen?). For those cases where we would be seriously inconvenienced by a lack of forethought, we seem to rely on our generalized experience to avoid such situations. Whether machines can be trusted to learn from their experience, given their greater potential for damage and general lack of commonsense, is another question.

Recommendations

In the following paragraphs, I outline

what I consider to be the main recommendations from the panel and discussions.

The AI planning community should make much greater use of probabilities to represent and reason about uncertainties in planning, including uncertainty in temporal intervals, spatial uncertainty, and uncertainty of possible outcomes. A major advantage in explicitly representing uncertainty in planning is that it provides useful search control information: It can show when the uncertainty has accumulated to the point where further advance planning is useless.

Increased use should be made of decision analytic methods. In particular, the AI use of goals needs to be extended to utilities, so that trade-offs between competing goals can be arrived at. This approach allows planning to produce useful plans, even when the given list of goals cannot be satisfied.

AI planners should increase their use of compiled procedural knowledge to avoid rediscovering the wheel with every planning problem. Many technical difficulties exist in suitably generalizing previous plans, but precompilation represents an important escape from the usually exponential search problem inherent in most difficult planning problems.

Planning and Execution

Drew McDermott

Chair: Drew McDermott, Yale University. *Speakers*: Phil Agre, Massachusetts Institute of Technology; David Payton, Hughes Research Laboratories; and David Miller, Virginia Polytechnic and State University.

It was once thought that there was a problem with interleaving planning and execution. To a large extent, our perceptions of this problem have changed so much that it is hard to recognize it any more. Of course, we have new problems to worry about in its place, but they might be easier to solve or, at least, more interesting.

The problem of interleaving planning and execution arose when planning was thought of as the process of generating a sequence of action descriptions at the level of (PUTON A B). Much of this research abandoned execution altogether and focused on methods for arriving at provably correct action sequences. When people did think about execution, they tended to ask questions such as, What if the world weren't the way the planner thought it would be? What if actions didn't always have their intended effects? Asking the questions this way encouraged a focus on inserting extra steps to ensure step preconditions were met and to make unexpectedly false preconditions true.

This discussion revealed that these questions have been superseded by a whole new viewpoint.[7] Experience with real agents acting in the world, that is, robots, has shown that the level of abstraction at which planners have traditionally worked is of no interest, at least not in isolation. In order to control real behavior, it does no good to issue a command such as (PUTON A B). Instead, one must be prepared to engage in behavior tightly coupled to the world through sensory feedback. Finding A and B, making sure A is held properly, following a collision-free path, and verifying that A made it to B must all be part of the behavior the agent engages in.

Thus, at the very least, we must rid ourselves of the idea that sensory feedback is put in as a separate phase having to do with execution monitoring. The behavior must involve sensory monitoring from the beginning.

I use the word behavior here with caution. A chance exists that the whole concept of plans and planning might be relatively unimportant. Instead, perhaps we should focus on an entirely different set of issues. For instance, why do we want our robot to put A on B? Can we really expect it to know precisely which objects are denoted by these symbols? Suppose what we really care about is getting shiny metallic objects on top of tables. Getting the particular shiny object A on top of a particular table B then ceases to be important. We can build a machine to home in on shiny objects and then gravitate toward objects that look like tables. In Rod Brooks's immortal phrase, we can just try to produce a program that "does the right thing." After all, if we can't write this program, we certainly can't write a planner that does what the program does.

This tilt toward the realities of robot programming is clearly healthy. It makes us realize that planning algorithms, no matter how clever or deep, will not compensate for incompetence in getting around in the world. A robot must be equipped with good sensors and behavior controllers (in hardware or software) that can respond sensibly to most conditions which occur as a prerequisite to planning. Planning, when it is necessary, must go on against a backdrop of behavior that is already in progress. This image raises the possibility that we can dispense with the whole idea of planning and just focus on competent behavior. There are some (Rod Brooks, Phil Agre, and David Chapman) who would do just this.

However, most researchers believe that planning exists and can be studied, and it is hard to disagree (although there is room for disagreement on effective research directions). Once we decide to study planning, then the following issues become important: (1) What is planning for? (2) When must a planner step in and alter what the agent is doing? What do such alterations consist of? (3) What are the properties of a good domain for studying planning? In particular, can we hope to simulate realistic domains in sufficient detail?

Answering the first question requires that we clarify some concepts. Given that a computer is always executing some program, we must be clear about the distinction between planning and behaving and between plan time and execution time. We define *planning* as reasoning about future events in order to select or, at least, verify the existence of a reasonable series of commitments to undertake in order to accomplish top-level goals (for a robot, the instructions issued by its human masters). The terms plan time and execution time do not refer to separate periods not even interleaved. Instead, we define execution time for an action or a set of actions as the time when they are executed. Plan time for this set of actions is then any prior time when they are reasoned about.[8]

Why should a basically competent

agent engage in planning? The answer seems intuitively obvious. Planning is an attempt to map out future activity so that the agent does not become trapped in local minima as it acts. As David Miller put it, "Planning should be done in order to properly allocate the needed resources for doing execution." For instance, the planner can note that it has several jobs to do at several different locations and arrange for them to be done in a particular order rather than have the execution system just drift from job to job or even thrash among several jobs at once. A planner can consult a map to realize that a tempting path toward a shiny object ends in a cul-de-sac. Its job is to think about the future enough to have appropriate activities ready to go as current activities finish. The agent can intend, not just to engage in a certain sensor-controlled interaction with the world now but to detect when this interaction is finished and go on to another way of interacting.

In other words, the activity-specification language must be rich enough to include semicolon, that is, to specify a sequence of actions. Because traditionally that's all which could be specified, there has been a tendency to reject the whole concept of intended sequences of plan steps or to see sequences of action as being at a different level of abstraction from reactive specifications, such as production systems. No reason really exists to compartmentalize things this way a priori. A behavior-specification language ought to be able to specify loops, sets of productions, and sequences, and it ought to allow these things to be composed in arbitrary ways so that sequences can occur on the right-hand sides of productions, and loops can include sets of productions as steps. We might as well use the term plan to refer to the intended activity, present and future, specified in this notation.[9] (In addition to the intention structure, the plan must also represent estimates of resource consumption, annotations about competing plans that were rejected, and so on.)

It would be outside our topic to dwell on what the planning process consists of. However, we must go into this question to some extent in order to clarify the relation between planning and execution. Planning is almost tautologically a process of plan revision. A planner starts with an empty plan (in some sense) and revises it until it works. Different planners revise in different ways: linearly, hierarchically, transformationally, and so on. Replanning after an execution failure is also a process of plan revision; when an agent fails to reach yonder shiny object, it must at least abandon the goal of reaching it. What is the relationship between these two processes?

Before we can answer this question, we must ask, What is an execution failure? In the no-planning view, there is no such thing. The agent is always acting and always reacting as appropriately as it can to current circumstances. There is no stage in this process to label a failure. (One might choose to point to where a high-level controller steps in to disable or reset a low-level controller, but past this event, execution just continues.) Given a planner, there is a natural model of execution failure. The current plan can specify the proper reaction to a wide range of sensory input, but it can also classify certain input as outside the tolerable envelope. When an input in this range is detected, the current plan just doesn't specify what to do. An event of this type is an execution failure.[10]

When a failure occurs, behavior cannot cease. Presumably, behavior is under the control of a plan consisting of several active processes. A failure in one does not cause the robot to cease to exist. Other modules remain active and take over. Indeed, their activity can be triggered by the failure itself. For example, if a robot has lost track of the road it is supposed to be following, then it might come under the control of a backup plan to sit quietly and periodically send out a distress signal (or wait to detect nearby enemy robots and explode when they get close). Meanwhile, it has a certain amount of time to replan. It might be able to estimate how long it will take to be rescued and how many of its top-level goals will have to be given up once it is restarted.

There is a consensus of workers in this area that (to quote David Payton) "any notion of optimality for run-time replanning must take into account the cost of the replanning time itself. ... Unfortunately, many opportunities may be lost if too much time is spent reconstructing an entirely new plan. These lost opportunities may mean that regenerating a plan from scratch will not yield a truly optimal plan." During the time spent planning, the world is changing. Hence, it is pointless to base planning decisions on the details of the world model at the point where replanning is required.

It is important to be careful about who is taking into account the cost of replanning. The planner itself cannot deeply deliberate about whether to act or contemplate. Any planning algorithm must be run under circumstances where the time it takes is short compared to the rate of change of world states it depends on. Whoever devises such an algorithm (for example, a heuristic traveling salesman procedure or a procedure for choosing a sonar-sampling schedule) must also spell out the circumstances where it makes sense to use it, and it must be trivial to test whether these circumstances apply.

Now let's return to the issue of plan-time versus run-time revision. In some cases, there is a natural "time zero" before a robot is sent on a mission. In this context, it hasn't really started to behave until it first plans. It starts with a complete set of instructions and attempts to generate a plan to carry them out. The generation process, as described earlier, is a search through some space of partial plan descriptions, which terminates in a plan that is specified well enough for execution to begin. In this scenario, once execution is in progress, the system has the option, when a failure occurs, of resuming the plan-generation process, which will now find a different plan.

For example, the planning search might be through a space of plan refinements guided by a heuristic evaluation function. An execution failure changes the robot's world model, altering the value of this function. Hence, the search for a good plan can be restarted with the values assigned to partial plans altered. For example, the planner might be considering two different routes to a destination. Route 1 might look better, but after detecting the destruction of a bridge along this route, the alternative, route 2, might suddenly look better. We restart the planning process, and this time route 2 looks better, gets attention, and ends up being the basis for a revised plan. The planner might also possess operators that transform a plan to eliminate bugs anticipated at plan time (as suggested by Ted Linden). In many cases, an execution failure can be modeled as just another bug, and transformations can be brought to bear to get rid of it. One way to deal with an anticipated protection violation is to insert plan steps to restore the violated condition, and this strategy will work at execution time as well as at plan time (Linden, Reid Simmons, and David Wilkins).[11]

There are limits to how well execution failures can be identified with plan-time bugs. Route 2 might have been the best alternative to route 1, but by the time we have executed half of route 1 and discovered a missing bridge, it is too late to start route 2. The closest we could come is to go back where we started and take route 2 but this option might be one of the worst around. In general, there is nothing special about the alternative plans the agent had before an execution failure or even about the plan it was executing. The failure represents new information, which, in principle, could have an impact on all aspects of the current plan. If time were not an issue, the planner might as well discard its entire plan and rethink it from the state it now finds itself in. However, because time is an issue, we must in practice retain the pieces of the plan that are probably unaffected by the failure.

For this idea to make sense, plans must be decomposable into pieces. At least two notions of decomposability are relevant here. First, a system can have several plans active simultaneously. It can be (1) monitoring the edge of the road and correcting deviations (as a step in a plan to get to a destination), (2) keeping velocity at about 20 km/h, (3) scanning for other vehicles (with the intent of avoiding colliding with them), (4) scanning the horizon for landmarks (to keep track of global position), or (5) monitoring radiation and chemical levels (with the intent of activating emergency escape behavior if high levels are detected).

We can use the term process to describe each of these pieces. Second, the current plan can be described at different levels of detail. The agent can intend to go to Hill 224, with this intention carried out by following a certain route, which involves traveling at 20 km/h and monitoring road edges. Hence, each process can be described at different levels.

To some level of abstraction, the candidate pieces to abandon when a failure occurs are the processes that failed. If the road-following process fails, the agent should kill the route-following process below some level of detail. Choosing the level is probably not too difficult. A planner must associate estimates of the time and the resources that it will consume with every piece of its plan.[12] Whenever it replans a piece, it can check whether the new plan consumes significantly more resources than originally allowed. If so, it can discard the revised plan, ascend to a level in the abstraction hierarchy at which the magnitude of the resource increase is not catastrophic, and replan this level. For instance, suppose the robot detects that a bridge it intended to use has been destroyed, and its first attempt at plan revision is to figure out some other way to get across the river at this point, leaving the rest of the route intact. If another bridge is close by, then the revised plan might not exceed the time allotted for getting across the river, and the rest of the plan can survive. However, if the best alternative for getting to the other side of the bridge involves finding a boat or going miles out of its way, then it will require much more time than originally allotted. If we call this time *t*, then the correct level at which to rethink is the lowest level whose time scale is commensurate with *t* (Payton).

This idea reflects pessimism about our ability to exploit opportunities. The new information we receive might make it possible to do much better at some higher level, but the procedure I outlined won't notice. Suppose that one of the robot's high-level purposes is to cut a road in order to deny it to the enemy. It might have decided to travel down the road a certain distance and lay mines. When it detects a missing bridge, it would be nice if it could realize that its mission is accomplished; instead of finding an alternative route that doesn't cost much more than it originally intended, it needn't make the trip at all. A trade-off exists between how much of the plan we revise and how many opportunities we miss.

Another sense of "opportunism" is even harder to exploit. Suppose our robot is given the mission of attacking enemy vehicles that it believes to be in a certain area. It comes up with a plan of following a certain route to this area and then looking for enemy vehicles to attack. If it passes an enemy vehicle on the way, then it will have overlooked an obvious opportunity. Unfortunately, that's exactly what it will do unless it is already expending some sensory resources to look for enemy vehicles before it reaches its destination. Presumably, the correct approach to this problem is to have high-quality sensors deliver an enormous quantity of information that is continuously monitored against all goals at all levels of abstraction. For the foreseeable future, sensors will probably not be good enough for this idea to be worth pursuing.

Mention of sensors raises a sticky issue. The research issues we want to look at in planning are mostly beyond what current sensor and effector technology can handle. It is easy to talk about road-following plans, but in fact, actually making such plans feasible is a major undertaking, which involves more person-hours (in the ALV program) than all of planning research. Does it make sense to study planning and execution based on ideal models of sensors that might turn out to be completely wrong? My own opinion would be that there are lots of architectural issues which can be investigated independent of the details of individual sensors. However, it is important to keep in mind that planners can make plans which are more grandiose than real executors can execute. Most of these plans end up being executed only by virtual robots in simulated worlds. Of course, the use of simulators is essential in almost all engineering disciplines. What is worrisome is our ignorance about what the sensors will actually look like when they exist.

Conclusions

Everything said in this section should be taken with a grain of salt. First, the research it is based on is in its initial stages. Second, the position presented here is a composite of the positions of several researchers; so, no one will agree with all of it, not even me. I hope the key conclusion is correct: Broadening plan representations to include sensor-controlled behavior gives us a clearer picture of how execution failures are detected and eliminates any special problem of replanning after a failure. Planning and replanning are essentially the same process. An agent is always planning as it acts. The information brought in by its sensors just pushes the planning process in new directions.

References

Bobrow, D., and Raphael, B. 1974. New Programming Languages for Artificial Intelligence Research. *Computer Surveys* 6(3): 153-174.

Brooks, R. A. 1985. A Robust Layered Control System for a Mobile Robot, Technical Report, A.I. Memo No. 864, Artificial Intelligence Laboratory, Mass. Institute of Technology.

Chapman, D. 1987. Planning for Conjunctive Goals. *Artificial Intelligence* 32: 333-377.

Chapman, D., and Agre, P. 1987. Abstract Reasoning as Emergent from Concrete Activity. In Proceedings of the 1986 Workshop on Reasoning about Actions and Plans. Los Altos, Calif.: Morgan Kaufmann.

Charniak, E., and McDermott, D. V. 1985. *Introduction to Artificial Intelligence*. Reading, Mass.: Addison-Wesley.

Chien, R. T., and Weissman, S. 1975. Planning and Execution in Incompletely Specified Environments. In Proceedings of the Fourth International Joint Conference on Artificial Intelligence, 169-174. Los Altos, Calif.: Morgan Kaufmann.

Dean, T. 1987. Intractability and Time-Dependent Planning. In Proceedings of the 1986 Workshop on Reasoning about Actions and Plans, 245-266. Los Altos, Calif.: Morgan Kaufmann.

Dean, T., and Boddy, M. 1987. Incremental Causal Reasoning. In Proceedings of the Sixth National Conference on Artificial Intelligence, 196-201. Menlo Park Calif.: American Association for Artificial Intelligence.

de Kleer, J., and Williams, B. C. 1986. Reasoning about Multiple Faults. In Proceedings of the Fifth National Conference on Artificial Intelligence, 132-139. Menlo Park Calif.: American Association for Artificial Intelligence.

Doyle, J. 1979. A Truth Maintenance System. *Artificial Intelligence* 12(3): 231-272.

Durfee, E., and Lesser, V. 1986. Incremental Planning to Control a Blackboard-Based Problem Solver. In Proceedings of the Fifth National Conference on Artificial Intelligence, 58-64. Menlo Park Calif.: American Association for Artificial Intelligence.

Ernst, G., and Newell, A. 1969. *GPS: A Case Study in Generality and Problem Solving*. New York: Academic.

Fikes, R. E.; Hart, P. E.; and Nilsson, N. J. 1972. Learning and Executing Generalized Robot Plans. *Artificial Intelligence* 3(4): 251-288.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence* 2:189-208.

Finger, J. J. 1987. Exploiting Constraints in Design Synthesis. Ph.D. diss., Dept. of Computer Science, Stanford Univ.

Genesereth, M. R. 1984. Partial Programs, Technical Report, HPP-84-1, Heuristic Programming Project, Dept. of Computer Science, Stanford Univ.

Georgeff, M. P., and Lansky, A. L. 1987. Reactive Reasoning and Planning. In Proceedings of the Sixth National Conference on Artificial Intelligence, 677-682. Los Altos, Calif.: Morgan Kaufmann.

Ginsberg, M. L., and Smith, D. E. 1986. Reasoning about Action I: A Possible Worlds Approach, Technical Report, KSL-86-37, Knowledge Systems Laboratory, Dept. of Computer Science, Stanford Univ.

Green, C. C., and Westfold, S. J. 1982. Knowledge-Based Programming and Deduction in Algorithm Design. *Machine Intelligence* 10: 339-359.

Hammond, K. 1986. CHEF: A Model of Case-Based Planning. In Proceedings of the Fifth National Conference on Artificial Intelligence, 267-271. Menlo Park Calif.: American Association for Artificial Intelligence.

Hanks, S., and McDermott, D. 1986. Default Reasoning, Nonmonotonic Logics, and the Frame Problem. In Proceedings of the Fifth National Conference on Artificial Intelligence, 328-333. Menlo Park Calif.: American Association for Artificial Intelligence.

Korf, R. E. 1987. Planning as Search: A Quantitative Approach. *Artificial Intelligence* 33(1): 65-88.

Laird, J. E.; Newell, A.; and Rosenbloom, P. S. 1987. SOAR: An Architecture for General Intelligence. *Artificial Intelligence* 33:1-64.

Lewis, D. 1973. *Counterfactuals*. Cambridge, Mass.: Harvard University Press.

Lifschitz, V. 1987. Formal Theories of Action. In Proceedings of the 1987 Workshop on the Frame Problem in Artificial Intelligence. Lawrence, Kans.

McDermott, D. V. 1982. A Temporal Logic for Reasoning about Processes and Plans. *Cognitive Science* 6: 101-155.

McDermott, D. V. 1977. Flexibility and Efficiency in a Computer Program for Designing Circuits, Technical Report, AI Memo 402, AI Laboratory, Mass. Institute of Technology.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence*. Los Altos, Calif.:Morgan Kaufmann.

Rosenschein, S.; Kaelbling, J.; and Pack, L. 1987. The Synthesis of Digital Machines with Provable Epistemic Properties. In *Theoretical Aspects of Reasoning about Knowledge, Proceedings of the 1986 Conference*, ed. J. Halpern. Los Altos, Calif.: Morgan Kaufmann.

Rosenschein, S. 1987. Formal Theories of Knowledge in AI and Robotics, Technical Report, CSLI-87-84, Center for the Study of Language and Information, Stanford Univ.

Scerdoti, E. 1977. *A Structure for Plans and Behavior*. New York: American Elsevier.

Shoham, Y. 1986. Chronological Ignorance: Time, Knowledge, Nonmonotonicity, and Causation. In Proceedings of the Fifth National Conference on Artificial Intelligence, 389-393. Menlo Park Calif.: American Association for Artificial Intelligence.

Simmons, R. and Davis, R. 1987. Generate, Test, and Debug: Combining Associational Rules and Causal Models. In Proceedings of the Tenth International Conference on Artificial Intelligence, 1071-1078. Los Altos, Calif.: Morgan Kaufmann.

Stallman, R. M., and Sussman, G. J. 1977. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence* 9: 135-196.

Stefik, M. J. 1981. Planning with Constraints. *Artificial Intelligence* 16:111-140.

Sussman, G. J. 1975. *A Computer Model of Skill Acquisition*. New York: Elsevier.

Tate, A. 1977. Generating Project Net-

Halbrecht Associates, Inc.

Executive Search Consultants

SERVING THE ARTIFICIAL INTELLIGENCE COMMUNITY

Halbrecht Associates, founded in 1957, was the first search firm in the United States to specialize in Information Systems Management, Operations Research, Management Sciences, Telecommunications, Computer Sciences and Decision Support Systems. We are now also in the forefront of the Artificial Intelligence field. Because we have been so extensively involved in this emerging technology, our portfolio of people includes extensive numbers of AI professionals at all levels, from the United States and abroad.

Our client list includes companies coast to coast, large and small, including start-ups with equity opportunities. Accordingly, depending on your needs, we are in a unique position to assist you rapidly with any of your requirements.

Our current assignments range downward from several Chief Scientists, or those who can provide the technical leadership to head up major AI programs, to Senior LISP Programmer. If you would like to be considered to be part of our AI data base, please send us your credentials. This will enable us to give you first notice of appropriate opportunities. All inquiries are in strictest confidence.

Contact: Daryl Furno, Senior Associate

1200 Summer Street • Stamford, CT 06905 • 203-327-5630

For free information, circle no. 193

works. In Proceedings of the Fifth International Conference on Artificial Intelligence, 888-893. Menlo Park Calif.: American Association for Artificial Intelligence.

Wilensky, R. 1983. *Planning and Understanding.* Reading, Mass.: Addison Wesley.

Wilkins, D. 1984. Domain-Independent Planning: Representation and Plan Generation. *Artificial Intelligence* 22: 269-302.

Wilkins, D. 1986. Hierarchical Planning: Definition and Implementation. In Proceedings of the 1986 European Conference on Artificial Intelligence.

Notes

1. This section borrows from the ideas put forth in a position paper written by Matt Ginsberg for the workshop.

2. This section borrows from the ideas put forth in a position paper written by Drew McDermott for the workshop.

3. The interested reader is urged to consult Wilkins (1986) or Charniak and McDermott (1985) for detailed discussions of the issues involved in hierarchical planning.

4. This section borrows from the ideas put forth in a position paper written by Kris Hammond for the workshop.

5. This section borrows from the ideas put forth in a position paper written by Mike Georgeff for the workshop.

6. One need not consider all the subgoals in the alternative plan completions, only those which were not achievable and those which required steps to be added to the plan to achieve.

7. This discussion, most of which took place electronically beforehand, involved a large group of people who took the time to put their views in writing: Phil Agre, Rod Brooks, David Chapman, Tom Dean, Rich Fikes, Paul Lehner, Ted Linden, David Miller, David Payton, Reid Simmons, and David Wilkins. Contrary to one's usual expectation, they all said provocative and intelligent things, which I try to do justice to.

8. There might be some tricky philosophical issues here about what it means to reason about possible events, especially events that never actually take place, but these issues don't have to detain us here.

9. Some people object to this extension of traditional terminology on the grounds that it will confuse the uninitiated and make some carefully drawn distinctions murky. Here, as everywhere in this research area, a pressing need exists for the standardization of terms.

10. We probably want to include items