Universal Plans for Reactive Robots in Unpredictable Environments

M.J. Schoppers

Advanced Decision Systems
201 San Antonio Circle #286
Mountain View, CA 94040

Abstract

To date, reactive robot behavior has been achieved only through manual programming. This paper describes a new kind of plan, called a "universal plan", which can be synthesized automatically, yet generates appropriate behavior in unpredictable environments. In classical planning work, problems were posed with unique initial and final world states; in my approach a problem specifies only a goal condition. The planner is thus unable to commit to any specific future course of events but must specify appropriate reactions for anticipated situations. An alternative conception is that one universal plan compactly represents every classical plan. Which part of the universal plan is executed depends entirely on how the environment behaves at execution time.

Universal plans are constructed from state-space operator schemas by a nonlinear planner. They explicitly identify predicates requiring monitoring at each moment of execution, and provide for sabotage, serendipity and failure without requiring replanning.

1. Introduction

1.1. Scope of this Paper

The work described herein continues my efforts toward synthetic reactivity, i.e. the automatic synthesis of robot programs capable of realtime performance in an unpredictable and/or dynamic world. This paper presents a solution to the problem of achieving a satisfying integration of goal-directed advance planning and sensor-driven reaction, without resorting to human programming.

By now, the problems with the so-called "classical approach" to planning are old news. Without going into tedious detail: state space planning requires a great deal of information up front, is time-consuming, and delays the arrival of suitable actions; its plans must eventually be very detailed or else may risk failure; even so, they are brittle, being unsuited to temporal continuity and uncertainty. The core problem is one of over-commitment.

The use of abstract plans, of partial plans, and of "reactive procedures", side-steps the early commitment problem by interleaving plan refinement and execution. The penalty is that acting on an incomplete plan may make the goal permanently unachievable: the planner may paint itself into a corner.

The work reported in this paper is the first fruits of a larger project concerned with the integration of planning, executing, sensing, and reacting. As a first step, that project will drive a robot arm in an environment that is rather less cooperative than a blocks world. The focus of the work is not on the robotics but on the planning, however, and this paper will be even narrower, concentrating on the plan representation and interpreter I have developed.

1.2. Mischief in the Blocks World

We are to solve the following benchmark problem. I have cast this problem in terms of a blocks world, to clarify (by way of contrast with the familiar) the issues involved.

> As in most blocks worlds, all the blocks are cubical and of unknown size; every block can be stacked on every other; and the usual 1-on-1 stacking rules apply. In addition we have a robot arm with a hand as our only means of moving blocks. On the other side of the table from the robot there is a mischievous baby who will flatten block towers, snatch blocks out of the robot's hand, and even throw blocks at the robot. The robot may not snatch blocks back, may not touch the baby, and cannot keep anything out of the baby's reach. As partial compensation we are given an unlimited set of sensing devices. The robot is to achieve the usual tower of blocks: **a** on **b** on **c**. Devise plan execution software enabling the robot to cope efficiently with this domain.

This problem highlights some issues that are challenges to the state of the art in AI and robotics:

- The robot's knowledge is incomplete, lacking the blocks' sizes. The missing information must be mitigated if the hand is to pick up any blocks.

- The problem does not specify the current state of the world -- neither the current locations of blocks, nor the position of the hand.

- The environment is capricious, allowing failure, sabotage and serendipity. Actions may not achieve their intended effects, and effects already achieved may not persist.

These items all have to do with the robot's behavior in the face of uncertainty, whether due to ignorance or to a dynamic environment beyond the robot's control. The work reported here is the first example of problem-independent synthetic reactivity.

1.3. Solution Approach

Accurate prediction being notoriously difficult, I have moved toward the other extreme, relying heavily on reaction. That in itself is not novel. The novelty of my approach arises from the fact that the representation I use to encode robot behaviors specifies appropriate reactions to every possible situation within a given domain, yet can be synthesized automatically and at moderate computational cost.

Universal plans convey highly conditional advice of the form:

> *If a situation satisfying condition P should ever arise while you are trying to achieve goal G, then the appropriate response is action A.*

There is no commitment to any particular sequence of events; in fact, universal plans contain little sequence of any kind. Instead, the task of the planner is to partition the set of possible situations on the basis of the reaction each situation requires. At execution time the *actual* situation is classified, and the response planned for that class of situation is then performed. Thus the behavior of an agent executing universal plans depends critically on which situations arise at execution time. Such reactivity allows universal plans to generate appropriate behavior even in unpredictable environments.

The behavior of agents executing universal plans is also goal-directed. If the world is cooperative long enough, each action will have its expected effects and the net behavior will be indistinguishable from that generated by a linear plan. No matter how the environment behaves, however, a universal plan always selects the action that would, in a cooperative world, move the current situation toward the goal.

The difference between the classical planning approach and that of universal plans is summarized in Figure 1. In

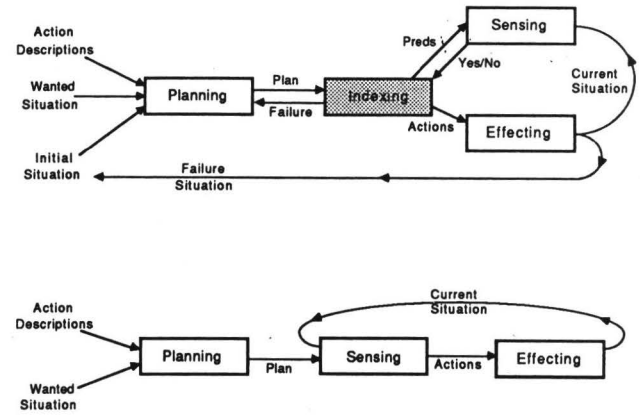


Figure 1: Plan monitoring vs planned reaction.

classical planning work, problems are posed with a specific initial world state, and plans are restricted not only to a particular set of situations but also to a specific ordering on those situations.[1] Any predicates associated with individual actions are used only to determine whether an action has been completed, and whether it was successful. If so, the next action to be executed is determined from the sequence imposed by the planner, not from any knowledge of the environment. When an action fails to produce a "go" outcome, the only recourse is to replanning, beginning with the newly produced situation.

In my approach a problem specifies only the goal condition -- no initial state and no unique final state -- so that the planner cannot over-commit to a specific future course of events. Instead the planner must anticipate possible situations and predetermine its reactions in those situations. The sensing module takes over the function previously served by sequencing, namely that of indexing into the set of available actions. And because the planner does not commit itself to a particular future, replanning is no longer necessary, no matter what serendipity, sabotage or failure takes place.

[1] Triangle tables are restricted to a subset of possible situations but not to a specific ordering. See Section 5.

The fact that universal plans can be generated automatically allows for a rather novel perspective:

> *Planning is the goal-directed selection of reactions to possible situations.*

This view bridges the chasm between goal-directed planning and situation-driven reaction. It makes all behavior reactive, but allows for rationality in the prior selection of reactions. The notion of an action's success becomes irrelevant to the process of plan execution; this reflects the intuition that no plan is knowably foolproof until *after* it has succeeded. Just as success cannot be known until execution time, so the actual course of events cannot be known in advance: the idea of a procedure is an abstraction, practically applicable only in non-real-world environments. To realize plans as procedures is to equate plan structure with the seriality of behavior-through-time. In other words, some particular sequence of events is inescapable at execution time, but is a major over-commitment at planning time.

2. Inputs to the Planner

2.1. Primitive Actions

That the planner must drive a real robot arm has some important consequences. For example: we can no longer pretend that it is irrelevant how big the blocks are. In fact we must consider how to cope with positions and distances in general. The approach I describe here is well below the level of abstraction adopted by previous blocks world planners, and is borrowed in part from the REX project at SRI (cf. Section 5).

The readings returned by all sensors are integers. The speed and acceleration values sent to the robot arm are also integers. In between, the most primitive actions are numeric functions. For example, if the arm is at x_{now} and we want it to stop at x_f in the shortest possible time, we define **speed-up** as in Figure 2a.

The functions for a **slow-down** stage are similar. Whenever we get new values for x_{now} or v_{now}, or even for x_f, the arm's speed and acceleration can be adjusted to suit. Of course, x_{now} and v_{now} are constantly changing. Thus, computing new arm motion parameters from current sensor readings establishes a feedback loop.

We must also specify when each stage should be used (Figure 2b). There are three relevant predicates: is the arm at its destination? moving in the right direction? moving as fast as its destination will allow? Depending on the truth or falsity of these predicates, one of **slow-down** and **speed-up** will be imposed between sensors and effectors.

Notice that these primitive actions are very different from traditional plan primitives in that they represent I/O conditions to be maintained for some unspecified length of time, not conditions to be achieved in the world. How long a given constraint is in force depends entirely on the environment. Moreover, there is no particular order in which **speed-up** and **slow-down** must be executed: that too is determined by the environment.

At this point we must diverge from the REX project mentioned earlier. By design, Figure 2b can be expressed as the "universal plan" of Figure 2c.

A synthesizer of universal plans must determine under what conditions the feedback functions should be imposed so as to achieve some condition in the world. How the necessary information is communicated to the planner is the topic of the next Section, which proceeds with "actions" at the level of "move to X". The main point of the present Section was to show that what lies below that level of abstraction is in fact identical to what lies above: even the most primitive sensori-motor feedback constraints can be controlled by universal plans.

speed-up:

$$d_{now} = x_f - x_{now}$$

$$v_{limit} = \left\lceil \sqrt{a_{MAX} \cdot |d_{now}| + \frac{v_{now}^2}{2}} \right\rceil$$

$$v_{OUT} = \min(v_{limit}, v_{MAX})$$

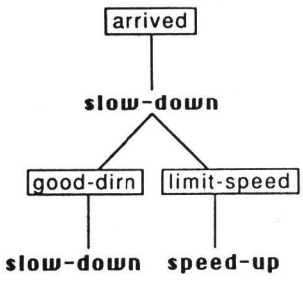
$$a_{OUT} = a_{MAX}$$

a

arrived $= (d_{now}=0 \wedge v_{now}=0)$
good-dirn $= (d_{now} \cdot v_{now} > 0)$
limit-speed $= (v_{now} \geq v_{limit})$

arrived	. . . . .		**no-op**
¬arrived	good-dirn	limit-speed	**slow-down**
¬arrived	good-dirn	¬limit-speed	**speed-up**
¬arrived	¬good-dirn	. . . . .	**slow-down**

b



c

Figure 2: Implementing arm motions with feedback.

2.2. Action Descriptions

The task of decomposing block transfers into more primitive actions reveals an interesting ambiguity in the interpretation of "action". On the one hand, from the robotics perspective it is immediately obvious that five physical motions will suffice, namely: close the hand as far as possible; open the hand completely; lower the hand vertically as far as possible; raise the hand vertically to some maximum height; and move the hand horizontally until above some target location. Each of these motions can be implemented straightforwardly by feedback functions as described in the previous Section. Hence the blocks world domain requires only five physical actions. On the other hand, capturing the blocks world at the level of such actions requires about 30 STRIPS-like operator schemas. Which are we to regard as an action: the actual motion, as identified by the function driving the arm, or what actually happens, as described by the operator schema?

Contrary to AI tradition, I chose to conceive of actions in terms of the robotic functions used to realize them, and to regard operator schemas as *effect* descriptions. Now the same action can produce multiple effects depending on the environment in which it is executed (cp. [side-effects]). Raising the empty hand is precisely the same "action" as raising the hand while holding a block, yet their effects are different, and moreover, each effect is necessary at different times. The problems and benefits of planning with alternative effects cannot be detailed here, however.

```
lowering =
                                  →  ¬top
                            wide  →  around(X)
          holding(X) over(table)  →  on(X,table)
     holding(X) block(Y) clear(Y) →  on(X,Y)

raising =
                                  →  ¬bottom
                       around(X)  →  mid
          holding(X) on(X,table)  →  mid
      holding(X) on(X,Y) block(Y) →  mid clear(Y)

opening =
                     hand-empty   →  ¬closed
          holding(X) ¬on(X,none)  →  around(X)

closing =
                                  →  ¬wide
                       around(X)  →  holding(X)

lateraling =
                            top   →  over(Z)
```

Figure 3: Blocks world effect descriptions.

When an action is described to the planner (Figure OPS), the pre- and post-conditions of its effects need not be complete: my planner utilizes an extension of the logic of Ginsburg [counterfactuals] to infer missing pre- and post-conditions (e.g. that, to lower the hand around block **X**, the hand must be over **X** to begin with).

2.3. Domain Constraints

I will not discuss the logic that determines which world states are possible and impossible; it is intuitively obvious (e.g. if the hand is open it can't be holding anything). What is crucial, however, is that the planner can deal with partially-determined world states. That occasionally involves facts of the form "Something is on block **a**". In predicate logic (and all previous blocks-world planners) such facts would be encoded as existentially quantified formulae within a world state model, and would give rise to considerable difficulties in ensuring a consistent world description. I solve this problem by using a possible-worlds TMS [tms] to encode and compute with world descriptions. This particular TMS will accept clauses of the form

$$\text{on}(b,a) \vee \text{on}(c,a) \vee \text{clear}(a).$$

The fact "Something is on block **a**" is encoded as **¬clear(a)**, which means that one of **on(b,a)** or **on(c,a)** must be true for the above clause to remain valid. There is no need for an explicit existential in the world description: there is an implicit existential in the restriction logic.

The planner manipulates multiple TMS instances, all sharing the same logic but having different truth value assignments. Thus, the planner can represent partial knowledge of the world at any point in the plan. More will be said in Section 4 about the importance of this for universal plan synthesis.

2.4. The Goal

The problem to be solved is posed to the planner as a condition to be achieved, not as a particular world state. For the problem discussed in the next section the goal is **on(a,b)∧top** -- how to stack **a** onto **b** and leave it there -- and there are still 24 world states that satisfy this goal condition (the final location of **c** and the final state of the hand are variable). The initial state of the world is irrelevant at planning time.

3. Universal Plans

3.1. Interpretation

The plan interpreter finds the action that is relevant to the current state of the world by using the completed universal plan as a decision tree. The pre- and post-conditions of operators are evaluated in the current real world, and so determine, at each node, which branch should be traversed next. Exactly how these decisions are made will be detailed below. Eventually the interpreter arrives at the currently appropriate action. Just as that action's location in the tree specifies the path necessary to reach it, so its being chosen as appropriate specifies what conditions are true in the world. As long as those conditions remain true, the chosen action remains appropriate. When they change, the interpreter must search the tree again for the next appropriate action.

The universal plan shown in Figure 4 is a plan to stack one block onto another: it realizes the very block-stacking action that other planners have taken for granted, and it is not trivial. We can extract from the plan a series of partial world descriptions with specifications of an action appropriate to the worlds described. This procedure involves a "method of assumed preconditions". Beginning with the root node conditions, assume them to be true, and note that no action is necessary. Now proceed backwards along the root node conditions, assuming them to be false, and in each case see what action is necessary to reverse the falsified condition. Assume the preconditions of this action (if any) to be satisfied. This generates a partial world description in which the action is appropriate. Now assume that the preconditions of the selected action are false... and so on, recursively. The result is shown in Figure 5.

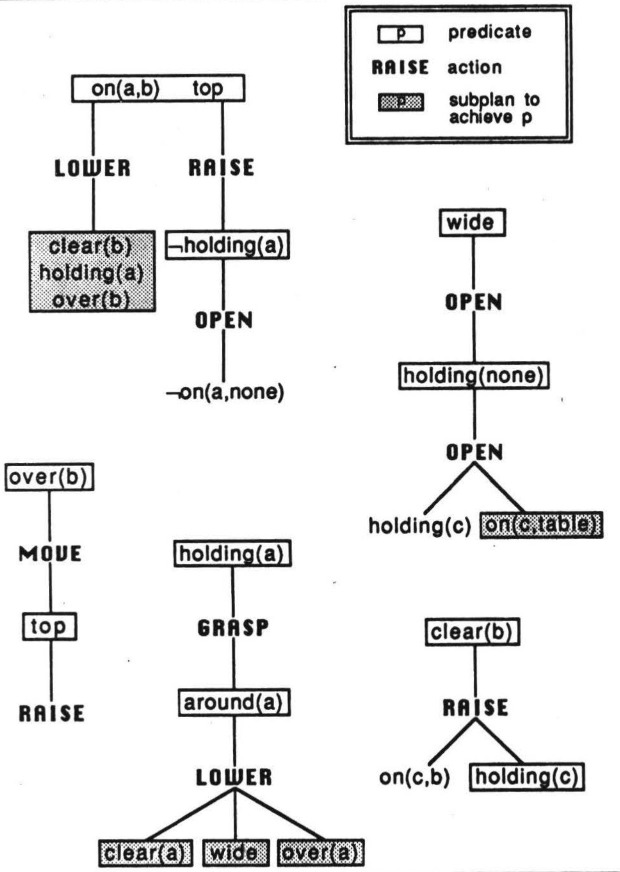


Figure 4: Part of the universal plan for **stack(a,b)**.

```
 on(a,b)  top . . . . . . . . . . . . . . . .  no-op
on(a,b) ¬top  ¬holding(a)   . . . . . . . . .  raise
on(a,b) ¬top ¬¬holding(a)   . . . . . . . . .  open
¬on(a,b)  clear(b)  holding(a)  over(b)  . . .  lower
¬on(a,b)  clear(b)  holding(a) ¬over(b)  top    lateral
¬on(a,b)  clear(b)  holding(a) ¬over(b) ¬top    raise
¬on(a,b)  clear(b) ¬holding(a) . . . . . . . .  ..2..
¬on(a,b) ¬clear(b) . . . . . . . . . . . . . .  ..3..
```

Figure 5: World state partitions in **stack(a,b)**.

```
on(a,b) ?
   T) top ?
   .      T) no-op
   .      F) ¬holding(a) ?
   .             T) raise
   .             F) open
   F) clear(b) ?
          T) holding(a) ?
          .      T) over(b) ?
          .      .      T) lower
          .      .      F) top ?
          .      .             T) lateral
          .      .             F) raise
          .      F) ..2..
          F) ..3..
```

Figure 6: Decision tree form of **stack(a,b)**.

Of course, for efficiency's sake the interpreter tests not these world models but the individual conditions comprising them. The partial world descriptions can in fact be reorganized into a decision tree which allows the interpreter to test only the conditions actually required to select the currently appropriate action, and those conditions are evaluated only once. The decision tree for the **stack(a,b)** plan is shown in Figure 6.

Once the currently appropriate action has been determined, it can be executed continuously until the truth value of some predicate changes, in which case the decision tree must be traversed again from that predicate down to determine the new reaction. In the

simplest case, only the last false condition in the active path will change to true, indicating that the current action has achieved its postconditions, and leading to what would be the next step for a classical sequential plan. It is also possible that some condition changes higher up in the tree. This corresponds either to serendipity or to sabotage.

The decision tree form makes obvious the fact that every possible world state is provided for somewhere, simply because both outcomes of every predicate are classified eventually. Hence the name, "universal plan". An immediate consequence is that universal plans have no preconditions: they always apply. The kind of composition that produces universal plans is very different from the concatenation of sequential plan fragments.

The **stack(a,b)** decision tree is never embodied as a data structure, but is merely a convenient description of how the interpreter executes a universal plan. The universal plan structure is used as a template from which a sequence of predicates is simultaneously extracted and evaluated, following the pattern of the above decision tree.

3.2. Hierarchy

Recall that in Section 2.1 the lateraling action was itself expressed as a universal plan. Hence, universal plans can be used to construct behaviors from the most primitive levels up. Equally importantly, it follows that something viewed as an "action" by the **stack(a,b)** plan can in fact be another universal plan. The ability to reuse a universal plan as a primitive reaction at a higher level gives us a form of abstraction: the planner can remain ignorant of the conditions used within **lateral**. Now it is easy to see that abstraction can also continue upwards beyond the stacking of individual blocks. The plan to build a tower of three blocks can be translated into the decision tree:

```
on(c,table) ?
   T) on(b,c) ?
   .     T) on(a,b) ?
   .     .     T) no-op
   .     .     F) stack(a,b)
   .     F) stack(b,c)
   F) unstack(c)
```

By determining the currently appropriate block transfer action, and hence the currently appropriate arm motion, and hence the currently appropriate feedback constraint, this plan will achieve the **a,b,c** tower from any initial state, and no matter what serendipity or sabotage occurs in the meantime. Notice also that this is the first plan representation capable of capturing the intuition that block towers should *always* be built bottom-up. Sequential plan representations provide no way to express this general heuristic, forcing it to become part of the planner's domain specific knowledge instead.

3.3. Competence

In principle, once we have a universal plan for a non-conjunctive goal **G** (such as **holding(a)**) we actually have two ways to achieve **G**: the plan, and the "primitive" action **A** (in this case, **grasp**) that finally brings about **G**. How do we know which one to use if **G** becomes a goal in subsequent planning?

This problem is an artifact of poor plan representations such as the traditional sequential one. Remember however that a universal plan is applicable in any initial situation. In particular, the universal plan **P** to achieve **G** applies also in situations to which **A** applies, whereas **A**'s applicability remains restricted by its preconditions. Consequently **A**, as a means of achieving **G**, is completely superseded by **P**, and might as well be forgotten. The construction of a universal plan represents a major increase in competence.

4. Plan Synthesis

The planner builds a universal plan by back-chaining from the goal condition, using the effect descriptions as goal reduction operators. There is a subtle difference from ordinary back-chaining, however: when a precondition becomes the goal of subsequent back-chaining, the negation of that precondition must be true of all the situations occurring in that subplan. That is, the subplan to achieve **on(a,b)** need only consider worlds in which **on(a,b)** is *not* true. This is the planning-time version of the execution-time "method of assumed preconditions". Back-chaining terminates when the accumulation of goals above the current locus of control either forces the satisfaction of the preconditions being examined, or leads to contradiction. It is not necessary to consider each possible world state individually: the planner may assign the same reaction to groups of states, considered *en masse* in the form of a world state schema. Hence, universal plan synthesis is at least as efficient as synthesizing linear plans (see Section 6.1).

Whenever an effect description has multiple preconditions, goal conflicts are possible. The reasoning required to resolve goal conflicts is complicated by both the new plan representation and the multiplicity of effects per action. Details must be left for another paper.

Identifying preconditions with goals at planning time can be continued into execution time. When the decision-tree

version of universal plan execution applies predicates to the environment, every false predicate is a failed precondition and hence a planning-time goal. Thus, the locus of control at execution time explicitly determines the goals being pursued, so that even the agent's goals are subject to the environment.

5. Related Work

My experience with using Georgeff's Procedural Reasoning System (PRS) [2, 3] to control an autonomous mobile robot [4] was seminal for this work. The goal of that project was to reduce the amount of advance planning, and hence of advance over-commitment, by decomposing behaviors (by hand) into sequences of goals, and by selecting at run-time the behavior to achieve each goal. In some respects the project succeeded: in navigating the robot around an office building, the planner's floor map became a connection graph, containing no advance knowledge at all about widths of hallways and distances between doorways. That information was acquired *en route* from sensory input. In other respects, however, progress was less than satisfactory. If the robot sensors saw a doorway where none existed it would get stuck just as helplessly as if it had been measuring distance. It succeeded in eliminating dead reckoning by distance maps only to have it reappear as dead reckoning by connection graphs.[3] Armed with our knowledge of universal plans, we can now see that the PRS work did not go quite far enough in its attempt to achieve reactivity: it experimented with situation-dependent selection of *means* to achieve goals, but not with situation-dependent adoption and abandonment of *goals*. The fundamental difficulty was the rigidity of PRS's procedural control structure. Nevertheless, PRS's ability to eliminate dead reckoning from lower abstraction levels was instructive, and on subsequent analysis pointed to the importance of having actions whose *duration* depends on the sensed environment, and to a robotics-oriented view of motion.

The REX project [9, 6] was equally influential: the behavior of a situated automaton is always contingent on the state of the environment. Indeed, the contingency assumes precisely the forms I have adopted for universal plans, with continuously evaluated predicates determining which numeric feedback function should be executed. My approach differs from that of the REX team in that universal plans are produced automatically, and are therefore symbolic and highly constrained structurally, while REX automata are hand-coded. Indeed, symbolic representation of a REX automaton's knowledge is considered not only superfluous but undesirable; instead, the automaton's state is a function of the entire contents of its memory. This lack of symbols is somewhat disconcerting from a planning point of view. The REX project is emphasising analysis of the information content of situated automata synthesized by hand; I am emphasising the automatic synthesis and control of reactive behavior.

Triangle tables [1, 8] are synthesized by extracting, from a conventional linear plan, the set of expected world states and the set of needed operators, then reorganizing the predicates involved to form an index into that set of operators. This reorganization increases the competence of the original plan, from coping with a specific set of situations in a specific order to coping with that set of situations in any order. Thus, triangle tables were first to give the environment a hand in selecting the operator to be performed next. That in itself is not sufficient (see my comments on PRS above), but it is a quirk of the rigidity of linear plans that selecting an operator instance also selects a control state. Serendipitously, triangle tables allow the environment to dictate the interpreter's (apparent) goals. That feature is crucial, and has been made explicit by universal plans.

6. Critique

6.1. Computational Complexity

Universal plans not only anticipate every possible situation in a domain but actually prescribe an action for every initial state; moreover, the prescribed action is usually optimal. These remarks suggest that universal plans must be limited to very small problem spaces.

Universal plan synthesis is not a graph search problem, however. The blocks world as I solved it has >400 possible world states, but the planner makes use of predicates and effect descriptions to decompose the problem space. In fact, universal plan synthesis is closer to a classification problem: given a set of possible situations, with an appropriate reaction already assigned to each situation, what is the complexity of producing a decision tree? In the best case the effort required to classify n distinct situations may be $\mathbf{O}(\log(n))$. Only in the worst case, when each possible situation must be classified at a different leaf node, does the classification effort become $\mathbf{O}(n)$. For comparison, the total effort expended by a planner that produced a new plan for each initial situation would be $\mathbf{O}(n)$ at best.

6.2. Blocks World Problems

The current planner relies on a STRIPS-like effects representation, restricting the plans to domains

[3] I am indebted to Ken Dove of Advanced Decision Systems for the wording of this observation.

representable as state spaces, but this is not necessarily as serious as contemporary AI milieu might think. Clearly, in my case the state space representation has *not* condemned me to a static and predictable environment, nor to dead reckoning, nor to complete knowledge of every world state considered. Such limitations are artifacts of the reasoning engine, not of the problem representation.

Neither have I been condemned to instantaneous actions and discrete worlds. State spaces may be inadequate to *reasoning* about continuous processes, but when sensory information is exploited to control both the current action and the manner of its performance, the increased behavioral competence removes much of that burden from the reasoning engine. The inability to respond to continuous stimuli is an artifact of sensory deprivation.

6.3. The Sensing Bottleneck

Sensing is the only reliable means to obtain feedback from the environment, but for many kinds of sensor, the rate at which readings can be converted to usable information is relatively slow. Universal plan execution, however, relies on sensory feedback continuously, and needs predicates from a variety of sources. The problems are aggravated in domains requiring knowledge of objects other than the robot itself, such as the locations of blocks when those locations cannot be controlled.

If some predicate can only be evaluated infrequently, only domain-dependent considerations can determine how the robot should proceed in the meantime. It may turn out that predicates near the root of the universal plan are less crucial, for example, or that it is more effective to temporarily ignore the values of some predicates than to monitor them closely. At least the decision tree form of universal plans makes very clear what predicates are relevant to each reaction, and so may facilitate the determination of how to proceed at each point. This in itself is a useful contribution.

7. Summary

This paper has shown how to integrate goal-directed planning with situation-driven reaction in the case of robotic motion, namely by redefining plans so as to eliminate the over-commitment, inherent in procedural representations, to a particular course of events. Given suitable sensory input, the resulting plan representation generates appropriate behaviors even in unpredictable environments, allowing the environment to determine the robot's current goals. This achievement encourages a new perspective on planning as choosing reactions for situations that might arise, and on plans as never guaranteeing success. The universal plan structure replaces procedural indexing with sensory indexing; makes explicit the conditions under which actions are applicable; renders notions of success and failure irrelevant at execution time; and encourages hierarchy.

Acknowledgements

This work has been supported in part by an internal R&D grant from Advanced Decision Systems. I especially appreciate the encouragement they provided when my ideas were still emerging. Thanks also to Stan Rosenschein, Mike Georgeff, Amy Lansky and Leslie Kaelbling (SRI AI) for allowing me to participate in their progress, and to John Myers (SRI Robotics) for being a constructive critic of a first draft.

References

1. FIKES, R.E., HART, P.E. AND NILSSON, N.J. "Learning and executing generalized robot plans". *Artif Intel 3* (1972), 251ff.

2. GEORGEFF, M. AND BONOLLO, U. Procedural expert systems. Proc 8th IJCAI, 1983, pp. 151ff.

3. GEORGEFF, M., LANSKY, A. AND BESSIERE, P. A procedural logic. Proc 9th IJCAI, 1985.

4. GEORGEFF, M., LANSKY, A. AND SCHOPPERS, M. Reasoning and planning in dynamic domains: an experiment with a mobile robot. Tech Note 380, AI Center, SRI International, 1986.

5. GINSBERG, M. Counterfactuals. Proc 9th IJCAI, 1985, pp. 80-86.

6. KAELBLING, L. An architecture for intelligent reactive systems. Proc Workshop on Planning and Reasoning about Action, AAAI, 1986.

7. McALLESTER, D.A. A three-valued truth maintenance system. Memo 473, MIT AI Lab, 1978.

8. NILSSON, N.J. Triangle tables: a proposal for a robot programming language. Tech Note 347, AI Center, SRI International, 1985.

9. ROSENSCHEIN, S.J. "Formal theories of knowledge in AI and robotics". *New Generation Computing 3* (1985), 345-357.

10. SRIDHARAN, N.S. AND HAWRUSIK, F. Representation of actions that have side effects. Proc 5th IJCAI, 1977, pp. 265ff.