PLAN SYNTHESIS: A Logical Perspective

Stanley J. Rosenschein

Artificial Intelligence Center
SRI International

ABSTRACT

This paper explores some theoretical issues of robot system planning from the perspective of propositional dynamic logic. A generalized notion of "progression" and "regression" of conditions through actions Is developed. This leads to a bidirectional single-level planning algorithm that Is easily extended to hierarchical planning. Multiple pre-/postcondition pairs, complex (e.g., conjunctive, disjunctive) goals, goals of maintenance and prevention, and plans with tests are all handled in a natural way. The logical framwork Is used to clarify gaps in existing "nonlinear" and "hierachical" planning strategies.

I INTRODUCTION

Although the connection between the artificial-Intelligence (AI) plannin-oblem and automated program synthesis is widely edged, relatively little planning researc made explicit use of concepts from *t* c of programs. Such logic, however, offers .etIcal insight into various Issues in AI plann iuch as compound goals and levels of ractIon [11, 12, 14, 4]. Many of these issue .rise in their purest form in domains descrIba e in the propositional calculus (e.g. simple blocks worlds), as evidenced by the literature on the subject [13, 11, 15]. Thus, for clarity and continuity, we choose the propositional setting to develop a unified, abstract treatment of these issues using propositional dynamic logic (PDL) as our primary logical tool [7, 5, 8, 9, 3J.

PDL is a decIdable modal propositional logic for reasoning about binary state relations induced by programs. In theory, the existence of such a logic provides an immediate "solution" to the propositional planning problem: One could systematically substitute all possible plans into a schema (the specification) that asserts the desired property of the plan. The resulting expressions could be tested for validity to filter out non-solutions. Unfortunately, this fact is of little practical consequence, as such a procedure is certain to be grossly inefficient. The approach developed here imposes additional structure by (1) considering a class of problems that require, in effect, only nonmodal reasoning, and by (2) using suitable "progression" and "regression" operators to structure the search for a solution and allow early pruning of hopeless paths.

Surprisingly, even our restricted formulation covers a more general class of problems than are handled by most comparable AI planning methods. For Instance, we allow goals to be arbitrary wffs, so that disjunctive goals (cited as an unsolved problem by Sacerdoti [12]) require no special treatment at all. The approach provides a theoretical basis for hierarchical plan generation that ties in directly with current Ideas on hierarchical program development (see Section III C.) In addition, the use of program logic constitutes a formal basis for specifying and verifying the plan-generating system Itself.

Although our work can be generalized along several dimensions (propositional axiom schemata, plans with loops, quantified pre-/po8tcondItIons, etc.), these are beyond the scope of the current paper, which focuses instead on the essential structure of the approach. At the same time, it should be noted that the use of axiom schemata seems to be a minimal requirement for practical application. This paper should be regarded as a foundational study aimed at deepening our understanding of planning; a separate paper will discuss problems of implementation [10].

II PRELIMINARIES

This section contains a brief presentation of the basic concepts of a loop-free fragment of propositional dynamic logic (PDL). The Interested reader Is referred to [7, 5, 3] for a more comprehensive treatment of dynamic logic. Although dynamic logic is ordinarily used to reason about programs, it is equally appropriate for reasoning about plans (in the AI sense); thus, in this paper the terms program and plan are used i nte rchangeably.

A. Syntax

Let (? and Q. denote two symbol sets: atomic propositions and atomic actions, respectively. Define wffs Pf * and programs A^^ simultaneously (deleting subscripts for convenience):

1. <?SP
2. a * A
3. If p,q€ P, then -*p, pVq€ P
4. If pfc P and *£ A, then <*>p£ P

The research described In this paper was supported by the Office of Naval Research under Contract Number N00014-80-C-0296.

5. $\Lambda \in A$
6. If $p \in P$ and p is nonmodal (see below), then $p? \in A$
7. If $\alpha, \beta \in A$, then $\alpha;\beta$, $\alpha \cup \beta \in A$

A formula is nonmodal if it contains no subformula of the form $\langle\alpha\rangle p$. We abbreviate $\neg(\neg p \vee \neg q)$ as $p \wedge q$, $\neg p \vee q$ as $p \supset q$, $(p \supset q) \wedge (q \supset p)$ as $p \equiv q$, $\neg\langle\alpha\rangle\neg p$ as $[\alpha]p$, $p \vee \neg p$ as true, and $p \wedge \neg p$ as false. Parentheses are used conventionally as required.

B. Semantics

A structure S is a triple (W,π,m) where W is a nonempty set of "worlds," $\pi : \Phi \to 2^W$, and $m : \Delta \to 2^{W \times W}$. That is, π assigns to each atomic proposition p the subset of W where p holds, and m assigns to each atomic action a the binary relation over W representing the next-state relation for a. Given a structure S, meanings can be assigned to arbitrary programs and formulas by extending m and π:

Meanings of Programs

1. $m(\Lambda) = \{(s,s) \mid s \in W\}$
   (identity relation over W)
2. $m(p?) = \{(s,s) \mid s \in \pi(p)\}$
   (identity relation restricted to worlds where p holds)
3. $m(\alpha;\beta) = m(\alpha) \circ m(\beta)$
   (composition of relations)
4. $m(\alpha \cup \beta) = m(\alpha) \cup m(\beta)$
   (union of relations considered as sets)

Meanings of Formulas

1. $\pi(\neg p) = W - \pi(p)$
2. $\pi(p \vee q) = \pi(p) \cup \pi(q)$
3. $\pi(\langle\alpha\rangle p) = \{ s \in W \mid \exists t \in W.\ (s,t) \in m(\alpha) \text{ and } t \in \pi(p)\}$

The last equation asserts that $\langle\alpha\rangle p$ is true in those worlds s from which another world t is reachable via α's next-state relation such that p holds in t. In general, our formulas will involve the dual of $\langle\alpha\rangle$, namely $[\alpha]$. $[\alpha]p$ can be read "after α, p." The formal meaning of $[\alpha]p$ is $\{ s \in W \mid \forall t \in W.\ (s,t) \in m(\alpha) \text{ implies } t \in \pi(p)\}$. In other words, $[\alpha]p$ holds in a world if p holds in all worlds accessible from that world via α. As expected, $\pi(\text{true}) = W$ and $\pi(\text{false}) = \emptyset$.

A formula p is valid in a structure $S = (W,\pi,m)$ (written $S \models p$) iff $\pi(p) = W$; p is valid (written $\models p$) iff it is valid in every structure.

C. Axiomatics

The following system captures the semantics given in the previous section:

Axioms

1. Axioms of the propositional calculus
2. $[\alpha](p \supset q) \supset ([\alpha]p \supset [\alpha]q)$
3. $[\Lambda]p \equiv p$
4. $[p?]q \equiv p \supset q$
5. $[\alpha;\beta]p \equiv [\alpha][\beta]p$
6. $[\alpha \cup \beta]p \equiv [\alpha]p \wedge [\beta]p$

Rules of Inference

1. From p, $p \supset q$ derive q (modus ponens)
2. From p, derive $[\alpha]p$ (necessitation)

If a formula p follows from these axioms under the stated rules of inference, we say it is provable and write $\vdash p$; if p can be proved from a set of assumptions, Q, we write $Q \vdash p$.

D. A Restricted Class of Programs

PDL breaks the ordinary conditional statement into more primitive notions of "test" (?) and "nondeterministic choice" (U). Though we allow the primitive actions to be nondeterministic, we shall be interested only in deterministic combining forms. Thus, we limit the use of ? and U to contexts of the form $(p?;\alpha) \cup (\neg p?;\beta)$ and require (for convenience only) that p be atomic. This corresponds to the ordinary conditional, so we abbreviate this program form to $p \to \alpha, \beta$ and call the class of programs obeying these syntactic restrictions C-programs (symbolically $\hat{A}_{\Phi,\Delta}$). The requirement that p be atomic is not restrictive, since arbitrary Boolean combinations of tests can be expressed by appropriate use of (possibly nested) conditionals. For example, $\neg p \to \alpha, \beta$ is equivalent to $p \to \beta, \alpha$. $(p \wedge q) \to \alpha, \beta$ is equivalent to $(p \to (q \to \alpha, \beta), \beta)$, and so forth.

An important property of our combining forms is that they preserve termination; if the primitives always terminate, every C-program over those primitives will always terminate. (Loops are conspicuously absent.) In PDL, the fact that a program always terminates is expressed $\langle\alpha\rangle$true, since with true holding in every state, the only way for this formula not to hold is for there to be no states accessible via α--i.e., for α not to terminate.

III A PLANNING METHOD

Having described a suitable language and logic, we are now in a position to discuss planning methods. Section A contains the formal definition of a (single-level) "planning problem" and the corresponding notion of a "solution." This leads directly (Section B) to a bidirectional (single-level) planning algorithm based on "progressing" and "regressing" conditions through actions. Section C describes how the hierarchical planning problem can be regarded as a succession of single-level problems in a way that makes the connection between the levels logically precise.

A. Definitions

A planning problem is a triple $(V,Q,R(u))$, where

$V = (\Phi,\Delta)$ is the vocabulary of the problem,

consisting of the atomic propositions and the atomic actions.*

Q — is a finite set of axioms, which we will refer to as domain constraints. Q is partitioned into two subsets: static constraints, which are nonmodal formulas, and dynamic constraints, which are always of the form $p \supset [a]q$, p and q being arbitrary nonmodal wffs and a an atomic action. We implicitly assume an axiom of the form $\langle a \rangle true$ for every atomic action a; this expresses the fact that the action a always terminates, though Q may only partially specify in what state a terminates. We also assume that Q is consistent.

R(u) — is a finite set of formulas called the plan constraints. Like the dynamic domain constraints, each of these is of the form $p \supset [u]q$ for nonmodal p and q. The symbol u is a distinguished atomic action not contained in $\mathcal{A}$.

A solution to a planning problem (V,Q,R(u)) is an expression α in the programming language $\hat{A}_V$ such that for every $r(\alpha)$ (obtained by substituting α for u in R), $Q \vdash r(\alpha)$. That is, it is provable from the domain constraints Q that α satisfies all the plan constraints.** (Because of the termination constraints on the atomic actions, α is guaranteed to terminate. Therefore, in the language of program logic, we are talking about "total correctness.")

B. Finding Solutions

Having defined "solutions," we turn our attention to methods for discovering them. A natural way of organizing the search for solutions is to follow the syntactic structure of the programming language.

Let us recall that a program is either Λ, an atomic action a, or a composite of the form $\alpha;\beta$ or $t \rightarrow \alpha,\beta$ where α and β are programs and t is an atomic proposition. It will simplify the algorithm to consider only programs in a normal form, which we now define.

1. Normal Form for Conditional Plans

A program is in normal form if it consists of a sequence* of zero or more atomic actions followed optionally by a conditional program, both branches of which are in normal form, followed in turn by 0 or more further atomic actions. More formally stated, a program is in normal form if it can be written as $A_1;\ldots;A_n$, $n \geq 0$, with at most one A_i not atomic, in which case A_i is of the form $t \rightarrow B_1, B_2$ where both B_1 are themselves in normal form.** The null sequence is identified with Λ, and we take $\Lambda;\alpha = \alpha;\Lambda = \alpha$.

We have not precluded any essential solutions by insisting on this form, since every C-program can be put into normal form by transforming the longest (length > 1) sequence of steps whose first and last steps are conditionals into a single conditional as follows:

$(s \rightarrow A,B); \ldots ; (t \rightarrow C,D)$
$\quad ==> \quad (s \rightarrow A;\ldots;(t \rightarrow C,D),\ B;\ldots;(t \rightarrow C,D))$

and applying this transformation recursively to A, B, and the residual $\ldots;(t \rightarrow C,D)$.

2. An Algorithm

Let us suppose we are looking for a normal-form program α that satisfies one of the dynamic constraints $p \supset [\alpha]q$ in R. We consider the following cases corresponding to the possible forms of α:

1. $\alpha = \Lambda$. This is a solution if $Q \vdash p \supset q$.
2. $\alpha = a;\beta$ or $\alpha = \beta;a$ for some atomic action a. In the former case, α is a solution if $Q \vdash p/a \supset [\beta]q$, where p/a represents the strongest provable post-condition of p and a. Analogously, in the second case, α is a solution if $Q \vdash p \supset [\beta]a\backslash q$, where $a\backslash q$ is the weakest provable precondition of a and q. We call the former case "progression" and the latter "regression."
3. $\alpha = t \rightarrow \beta_1, \beta_2$. In this case, α is a solution if $Q \vdash p \wedge t \supset [\beta_1]q$ and $Q \vdash p \wedge \neg t \supset [\beta_2]q$.

We see that (1) defines success, (2) suggests forward and backward strategies for sequential steps, and (3) suggests a forward strategy for conditionals.

---

*For some applications it is desirable to constrain the programming language to use only a designated subset of the propositions as tests in conditionals. This requires a straightforward modification of our definition and will not be pursued here.

**Equivalently in semantic terms: Structures that satisfy the domain constraints also satisfy the α-instantiated plan constraints.

*Since ";" is associative, we write sequences a;b;...;c without indicating the order of association.

**Warren's method [17] for introducing conditionals produces plans of an even more restricted form: the conditional must be the last action in the sequence. That is, a plan, once split, may never rejoin. This is not an essential limitation, but it introduces a somewhat greater degree of redundancy than does our form. We note in passing that Warren's view of the conditional test as an action has much in common with PDL's p? action.

In addition, we observe that there are several obvious ways to limit the search. First, if $p \supset p/a$, the forward search need not consider action a. (A special case of this arises when p/a = true.) Dually, if $a\backslash q \supset q$, the backward search need not consider a. (Here we have a special case when a\q = false.) These checks eliminate self-loops. We can eliminate cycling in the search space altogether if we are willing to pay the price of checking whether $p_i \supset p/a$ for any p_i in the leading chain of preconditions. Likewise we can check whether $a\backslash q \supset q_i$ for any q_i in the trailing chain of postconditions. It should also be noted that if $p \supset t$ or $p \supset \neg t$, the forward conditional search involving t need not be pursued. p/a can never be false, since this would imply failure of a to terminate, contradicting our assumptions about the domain constraints Q.

These observations lead directly to the following nondeterministic algorithm for computing solutions for the single constraint $p \supset [\alpha]q$: (Multiple constraints will be discussed later.)

BIGRESSION* ALGORITHM

Assume p, q are not false.

Solve(p,q) = Bigress(p,q,Λ,Λ).

Bigress(pre,post,leader,trailer):

IF $Q \vdash pre \supset post$ THEN RETURN(leader;trailer).
CHOOSE:
 CHOOSE <a, pre/a> from LiveForward(pre):
  RETURN(Bigress(pre/a, post, leader;a, trailer))
 CHOOSE <a, a\post> from LiveBackward(post):
  RETURN(Bigress(pre, a\post, leader, a;trailer))
 CHOOSE t from NonTriv(pre):
  RETURN(leader;C;trailer)
   where C = (t $\rightarrow$ Bigress(pre$\wedge$ t, post,Λ,Λ),
              Bigress(pre$\wedge\neg$t, post,Λ,Λ))

LiveForward(p):
 IF S $\neq \emptyset$
  where S = { <a, p/a> | $a \in \mathcal{A}$, $Q \vdash p \supset p/a$}
 THEN RETURN(S)
 ELSE FAIL().

LiveBackward(q):
 IF S $\neq \emptyset$
  where S = { <a, a\q> | $a \in \mathcal{A}$, $Q \vdash a\backslash q \supset q$}
 THEN RETURN(S)
 ELSE FAIL().

NonTriv(p):
 IF S $\neq \emptyset$
  where S = { t | $t \in \mathcal{P}$, $Q \vdash p \supset t$, $Q \vdash p \supset \neg t$}
 THEN RETURN(S)
 ELSE FAIL().

The algorithm as presented finds solutions for a single plan constraint. However, the extension to the general case is straightforward: To ensure that all the plan constraints are satisfied, a "Cartesian product" version of this algorithm must be run. A failure in any of the constraint components counts as failure and serves to prune that branch.

The bigression algorithm makes use of three additional auxiliary functions: "$Q\vdash$", "/", and "\". "$Q\vdash$" is a procedure that takes as input a nonmodal formula p and decides whether p is provable from Q. If the static axioms are rich enough,* this check can be done using only nonmodal reasoning, i.e., ordinary propositional decision methods. The functions "/" (progression) and "\" (regression) are the subject of the next section.

3. Progression and Regression

Ideally, we would like p/a to compute the strongest postcondition of condition p and action a. Similarly, we would like a\q to compute the weakest precondition. [1, 15] In PDL the weakest precondition of p and a can be expressed simply as [a]p, which is obviously the weakest formula implying "after a, p." The strongest postcondition can be expressed using a "converse" operator that we have not described. (See [5].)

However, given the restricted form of our dynamic axioms, there will be no propositional formula provably equivalent to either of these modal formulas. On the other hand, we can effectively compute the weakest precondition pre and strongest postcondition post for which it is provable from Q that pre implies "after a, q" and p implies "after a, post." It is these propositional formulae that we label p/a and a\q.

The formula p/a is found by taking the conjunction of the set of formulas each of which is a disjunction of a set of q_i drawn from the "right-hand side" of the dynamic axioms of Q ($p_i \supset [a]q_i$) such that the disjunction of the corresponding p_i's is implied by p. Dually, a\q is found by taking the disjunction of the set of formulas, each of which is a conjunction of a set of p_i drawn from the "left-hand side" of the dynamic axioms of Q ($p_i \supset [a]q_i$) such that that conjunction of the corresponding q_i's implies q.

Let us consider the following sample axioms:

A $\supset$ [a] (B $\vee$ C)
G $\supset$ [a] $\neg$B
(F $\wedge$ E) $\supset$ [a] D

In this case, $a\backslash(C \vee D) = (A \wedge G) \vee (F \wedge E)$. The reason for this is that (B $\vee$ C) conjoined with $\neg$B implies (C $\vee$ D), so the conjunction of the corresponding left-hand sides (A $\wedge$ G) is one disjunct of a\(C $\vee$ D). Likewise, the formula D alone implies (C $\vee$ D), making the corresponding

*"Bigression" stands for "bidirectional progression and regression."

*Specifically, the static axioms must generate all the nonmodal formulas generated by all of Q. In certain pathological cases, such as when Q contains a dynamic axiom of the form $p \supset$ [a]false, Q would have to be extended to include extra static axioms, since $p \supset$ [a]false and <a>true together imply $\neg p$--a nonmodal formula derivable only through modal reasoning.

left-hand side (F A E) the second disjunct. These two cases exhaust the possibilities for getting (CVD).

The reason the formulas p/a and a\q defined in this way are not exactly equivalent to the strongest postcondition and the weakest precondition lies In the nature of our atomic actions. Briefly stated, In the context of programming languages one typically begins with primitives whose semantics are fully characterized and focuses on characterizing the derived operations (sequencing, etc.) [1]. For example, the weakest precondition for the assignment primitive is given by the equation: wp("x:= E", P(x)) - P(E), which asserts that the weakest precondition for condition P and action "x gets E" is precisely P with E substituted for x.

In our case, however, the primitive actions are specified only by axioms stating one-way implications. Thus, unless we make assumptions of a "non-monotonlc" nature, we would generally be able to consistently add axioms that "weaken[1]* the precondition or "strengthen" the postcondition of an action. Since "provably weakest" is unattainable, we make do with "weakest provable." This does not affect the completeness of the search algorithm, since we are looking only for programs that provably satisfy the specifications.

C. Hierarchical Planning

The key observation to be made in extending the single-level algorithm to multilevel, hierarchical planning is that an atomic action at level k Is a plan to be solved for at level k+1. This point of view is possible because of the way the planning problem was formalized. Specifically, an atomic action is described by a set of dynamic axioms in Q. Likewise, the desired program is described by a set of dynamic axioms in R. Since the same formal objects—namely, sets of dynamic axioms—are Involved in both cases, it is natural to assume as primitive some action with given properties at level k and then to solve for a program having those properties at level k+1.

In formal terms, a hierarchical planning problem is a tree of single-level problems. If $\langle V_k=\langle P_k,Q_k\rangle,Q_k,R_k(u_k)\rangle$ is the problem at nonleaf node k, then node k has one successor for each $a_{k,i}$ in Q_k, and that successor's problem has the form $\langle V_{k+1},Q_{k+1},Q'(a_{k,i})\rangle$, where Q′ denotes the subset of dynamic axioms of Q_k having the form $p \supset [a_{k,i}]q$. In other words, the domain constraints on the primitive "a" at level k become plan requirements at level k+1. A solution is a plan using the vocabulary of the leaf nodes that satisfies the requirements of the root node. That is, is a solution if it solves $\langle V_0,Q_0,R_1(u_1)\rangle$. The propositional vocabulary and the action vocabulary can change from level to level, provided that the domain axioms have enough inferential structure to make the transfer from level to level meaningful.

Obviously, for any node k, only the successor nodes corresponding to actions actually used in k's solution need be solved. Furthermore, finding a solution for each of these nodes guarantees that we have an overall solution.

As with other hierarchic planners, the main benefit of levels In our approach is heuristic: The selection of Intermediate vocabularies and domain axioms constitutes a choice of "planning islands." Any algorithm that tries to solve a problem by solving the nodes in the hierarchy is, in essence, searching for a plan constrained to go through the states defined by the domain constraints of the intermediate actions. The main benefit of logic here is to define a reasonable relation between the levels, namely the relation: "correctly Implements•'*

For a fixed determination of levels and a small number of actions it would be possible to precompute solutions to the subproblems, in which case, after solving the problem at the top level, the system would act more like a compiler than a problem solver. In dynamic situations in which the lower-level actions (in effect, the "tools" for solving the problem) are changing or when few actions are ever actually used, it seems more natural to solve subproblems as they arise.

IV DISCUSSION

[A#] Modeling Actions: The Legacy of STRIPS

Much of the research into the control of planning has been carried out in the STRIPS paradigm (2, 6J. In this approach, actions are regarded not as mappings from states to states, but rather as syntactic transformations of state descriptions to other state descriptions, where state descriptions are logical formulas. One consequence Is the oft-cited advantage of not having to mention the various "frame conditions," i.e., the properties that are invariant under an action. Unfortunately, the need for operators to be sensitive to the syntax of state descriptions led researchers to consider only very simple state descriptions (e.g. sets of atomic propositions) and very simple transformations (e.g., addlists and deletelists).

As an example of an action that is difficult to specify with a single addlist/deletelist pair, consider the action toggle, described by a pair of dynamic axioms:

$$\text{On(light)} \supset [\ \text{toggle(switch)}\ ]\neg\text{On(light)}$$
$$\neg\text{On(light)} \supset [\ \text{toggle(switch)}\ ]\ \text{On(light)}$$

Since the consequents depend conditionally on the antecedents, it cannot be determined in isolation whether toggle adds or deletes the wff On(light). Nonetheless, a planning system given these axioms might reason that after two toggles, the light would surely have gone on at least once. Similar remarks hold for actions with disjunctive postconditions.

These possibilities notwithstanding, many

However, these invariants need not be as large an obstacle to practical implementation as is commonly supposed (see [10]).

planning systems do make the assumption that the truth of a given atomic proposition In the state resulting from applying a sequence of operators is a determinate, calculable thing. Techniques that rely critically on these assumptions are sometimes difficult to adapt to less constraining assumptions. We offer two illustrations from NOAH [11].

B. Nonlinear Planning: Problems with Partial Orders and Shuffles

The basic idea behind nonlinear planning is the following. To solve a conjunctive goal Gl & G2, find a sequence Sl = a;b;...;c that achieves Gl and another sequence S2 • d;e;...;f that achieves G2. Represent the overall plan as a network of partially ordered actions with Sl and S2 as parallel branches* Now use the "resolve-conflicts critic" to detect interference between the plans and impose additional ordering constraints upon the actions to eliminate the interference. The network encodes the subset of the possible shuffles of S1 with S2 that are believed to achieve the overall goal Gl & G2.

For the resolve conflicts critic to filter interference correctly, it must know what is true at each node of the network. Unfortunately, for nodes that occur after joins, what is true depends critically on the ultimate linearization of the parallel branches. In the general case, the best that can be done is to represent the disjunction of the strongest oostconditions of the alternative linearizations. This requires considering the alternatives, of which there are ("Jj[11])* where m and n represent the lengths of the action sequences In the two parallel branches. Since It is easy to imagine cases in which resolve-conflicts criticism would be an expensive operation, the belief that using a nonlinear strategy is computationally efficient seems to be grounded in the empirical hypothesis that operators encountered in practice will permit easy detection of conflicts.

C. Hierarchical Planning: Problems with Heuristic Decompositions

The justification for partial orderings in NOAH is linked with a desire not to commit the system prematurely to a particular linear order of actions which, though seemingly correct at one level, may expand into incorrect plans at lower levels. This possibility can arise, of course, only if the relation between levels ("Plan A achieves the same effect as Action a") is not exact. However, such Inexactness undermines the original rationale for hierarchical planning—namely, reducing complexity by means of factorization—since it destroys composltlonallty and requires that we check complex lower-level plans for "unexpected" global interactions. Here too, an empirical hypothesis is presumably invoked, namely, that by some suitable metric the plan comes "close" to implementing the abstract action. (It is not Immediately obvious, though, what metric could be meaningful for the space in question.)

D. Some Benefits of Bigresslon

Some of the benefits of regression were first discussed by Waldinger [15] and appreciated by Warren [16]. These benefits accrue dually by Including progression, which completes the logical symmetry and allows bidirectional search. As we have described them, the progression and regression operations handle arbitrary Boolean formulas, thus solving conjunctive and disjunctive goals as particular applications of a more general strategy. Goals of maintenance and prevention can be incorporated into the algorithm as well by expressing as (nonmodal) wffs the condition to be maintained (m) and the condition to be prevented (v). Since the planning algorithm actually develops a descriptive wff (d) for each state reachable during plan execution, it is straightforward to add a check to the procedures LiveForward, LiveBackward, and NonTrlv eliminating paths through states where Qt»d:>-imVv • This simple approach will work in situations in which no dynamic replanning is anticipated; goals of maintenance and prevention involving execution monitoring, feedback, and replanning require more complex strategies. Further research should be directed at these problems as well as at efficiency issues, especially clarification of the role of heuristics and the compilation of deductive processes.

ACKNOWEDCEMENTS

I have profited considerably from discussions with Richard Waldinger, Vaughan Pratt, Kurt Konollge, Dave Wilklns, Jerry Hobbs, and Bob Moore.

Actually, NOAH does not represent disjunctive postconditions—which may explain why disjunctive goals are considered problematical.

For a more thoroughgoing treatment of reasoning about processes with Intermediate states, see [9].

REFERENCES

1. Dijkstra, Edsger W. Guarded Commands, Nondetermlnacy and Formal Derivation of Programs. Communications of the ACM 18, 8 (August 1975), 453-457.

2. Fikes, R.E. and N.J. NIlsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 2, 3-4 (Winter 1971), 189-208.

3. Harel, David. Lecture Notes in Computer Science. Volume 68: First Order Dynamic Logic. Springer-Verlag, 1979.

4. Hayes-Roth, B. and F. Hayes-Roth,

5. Rosen8cheln, and S. Cammarata. Modeling Planning as an Incremental, Opportunistic Process. Proceedings of 6th International Joint Conference on Artificial Intelligence, Tokyo, August, 1979, pp. 375-383.

5. Utvintchouk, S.D. and V.R. Pratt. A Proof-Checker for Dynamic Logic. Proceedings of 5th International Joint Conference on Artificial Intelligence, Massachusetts Institute of Technology, August, 1977, pp. 552-558.

6. NIlsson, Nils J. Principles of Artificial Intelligence. Tioga Publishing Co., 1980.

7. Pratt, Vaughan R. Semantical Considerations on Floyd-Hoare Logic. Proceedings of the 17th IEEE Symposium on Foundations of Computer Science, October, 1976, pp. 109-121.

8. Pratt, Vaughan R. A Near-Optimal Method for Reasoning about Action. MIT/LCS/TM-113, Massachusetts Institute of Technology, September, 1978.

9. Pratt, Vaughan R. Six Lectures on Dynamic Logic. MIT/LCS/TM-117, Massachusetts Institute of Technology, December, 1978.

10. Rosenscheln, Stanley J. Hierarchical Planning: Implementation Considerations. SRI Technical Report. Forthcoming.

11. Sacerdotl, Earl D. The Nonlinear Nature of Plans. Advance Papers of 4th International Joint Conference on Artificial Intelligence, Tbilisi, Georgia, USSR, September, 1975, pp. 206-214.

12. Sacerdotl, Earl D. A Structure for Plans and Behavior. Elsevier, 1977.

13. Sussman, G.J. A Computer Model of Skill Acquisition. American Elsevier, New York, 1975.

14. Tate, Austin. Generating Project Networks. Proceedings of 5th International Joint Conference on Artificial Intelligence, Massachusetts Institute of Technology, August, 1977, pp. 888-893.

15. Waldinger, Richard. Achieving Several Goals Simultaneously. Technical Note 107, SRI, International, Artificial Intelligence Center, July, 1975.

16. Warren, David H.D. WARPLAN: A System for Generating Plans. Department of Computational Logic Memo 76, University of Edinburgh, July, 1974.

17. Warren, David H.D. Generating Conditional Plans and Programs. Proceedings of Summer Conference on Artificial Intelligence and Simulation of Behavior, University of Edinburgh, July, 1976, pp. 344-354.