SRI International

PRELIMINARY REPORT ON A THEORY OF PLAN SYNTHESIS

Technical Note 358

August 1985

By: Edwin P.D. Pednault
Artificial Intelligence Center

APPROVED FOR PUBLIC RELEASE: DISTRIBUTION UNLIMITED

The research reported herein was supported by the Air Force Office of Scientific Research under Contract No. F49620-82-K-0031, by the Office of Naval Research under Contract Nos. N00014-80-C-0296 and N00014-85-C-0251, and through scholarships from the Natural Sciences and Engineering Research Council Canada and le Fonds F.C.A.C. pour l'aide et le soutien à la recherche, Quebec, Canada.

SRI International®
333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
AUG 1985		**00-08-1985 to 00-08-1985**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Preliminary Report on a Theory of Plan Synthesis	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **SRI International,333 Ravenswood Avenue,Menlo Park,CA,94025**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**64**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

1

Introduction

Classical planning problems have the following form: given a set of goals, a set of actions, and a description of the initial state of the world, find a sequence of actions that will transform the world from any state satisfying the initial-state description to one that satisfies the goal description. In principle, a problem of this type may be solved by a very simple procedure: merely enumerate all possible sequences of actions and test each until one is found that achieves the intended goals. By this procedure, we will eventually find a solution if one exists. However, in practice, not only do we want to find a solution, we want to do so expeditiously. Quick and efficient problem solving is desirable primarily for reasons of economy: the less time it takes to solve a problem, the more productive one can be. Furthermore, in some situations, the time it takes can mean the difference between success and failure, as is the case when the problem is part of a scholastic exam or when the problem is to prevent meltdown in a nuclear reactor.

Previous work aimed at developing efficient planning techniques has been highly experimental in nature, the standard methodology being to explore ideas by constructing computer programs. For the most part,[1] very little theoretical analysis has been done to determine why the programs work, when they are applicable, and whether they can be generalized to solve larger classes of problems.

In my thesis [8], I venture to the opposite extreme and examine the question of efficient planning from a rigorous, mathematical standpoint. My analysis is based on the premise that one of the main impediments to efficient planning is search, and that exhaustive search can be avoided only if

1 The exceptions to this are Warren's analysis of his WARPLAN program [17] and, just recently, Chapman's logical reconstruction of nonlinear planning [2, 3]. Warren's analysis is primarily concerned with proving the correctness of WARPLAN. Chapman, on the other hand, has analyzed previous work in nonlinear planning and, on the basis of this analysis, has constructed a program called TWEAK that is provably correct.

the problem being solved has properties that can be exploited to constrain the search. Accordingly, my methodology has been to construct a mathematical framework in which to study planning problems, to explore this framework for theorems that can be used to constrain the search for a solution, and then to construct planning techniques based on the theorems found. The techniques are described in precise, mathematical terms and are capable of solving any problem that may be expressed in the framework, provided a solution exists. While the techniques may be implemented in a straightforward manner, there are a number of implementational issues identified, but not addressed, in my thesis that need to be resolved before an efficient program can be obtained.

Although we have been working independently and in parallel, my work can be viewed as a significant extension of work resently reported by Chapman [2, 3]. While our approaches are similar, the framework I have developed encompasses a much broader class of problems and addresses some of the representational issues that Chapman identifies. In addition, I have been able to unify many more ideas in automatic planning and show how they arise from first principles. These ideas include not only nonlinear planning [11, 12, 15, 19], means-ends analysis [4], and opportunistic planning [6], which are incorporated into Chapman's technique, but also goal protection [14, 16, 17], goal regression [9, 16], constraint formulation and propagation [12], and hierarchical planning [10, 11, 12, 15, 19].

This report is intended to provide a glimpse of my thesis research. Only about a quarter of the topics presented in my thesis, however, are covered here. It would therefore appear advisable at this point to summarize the topics I bave included and those I have not.

In the next chapter, an intuitive explanation of the mathematical framework is provided and a language introduced for describing the effects of an action. In the framework presented here, actions are assumed to be deterministic—in the sense that performing an action transforms the world from its current state to a uniquely determined succedent state. The synthesis techniques, however, do not require determinism, and in my thesis I present a more general framework that permits actions to be nondeterministic.

The language for describing actions is interesting in that it combines the generality of the situation calculus [7] with the notational convenience of STRIPS [5]. This allows the frame problem of the situation calculus to be circumvented to the same extent that it can be done in STRIPS. As I show in my thesis, but not in this report, any problem that can be described in the situation calculus has an equivalent formulation using this language, and vice versa—with the restriction that the problem specification contain only a description of the initial state, a description of the goal state, and a description of the allowable actions. Also, in my thesis, I extend the syntax of the language to enhance the parsimony of action descriptions. For example, the description of the Put operator presented in Section 2.2 could be rewritten in the extended language as follows:

Put(p, q)

PRECOND: $p \neq q$, $p \neq TABLE$, $\forall z\,(\neg\,\mathrm{On}(z, p))$, $[q = TABLE \vee \forall z\,(\neg\,\mathrm{On}(z, q))]$

ADD: $\mathrm{On}(p, q)$

DELETE: $\mathrm{On}(p, z)$ for all z such that $z \neq q$

Chapter 2 also shows how the correctness conditions for a plan may be expressed in terms of regression operators, and how regression operators may be constructed from action descriptions. The regression equations presented here, though, tend to produce rather long formulas that may often be reduced to much simpler ones. In my thesis, I show how to add simplification rules to the regression equations to overcome this problem. The thesis also presents a number of theorems on regression operators that do not appear in this report, including a theorem that characterizes the kinds of actions that may be described in the language in terms of the regression operators for those actions.

Chapter 3 of this report shows how a simple planning technique may be derived from a particular theorem of the classical planning problems. The technique combines aspects of means-ends analysis, opportunistic planning, goal protection, goal regression, and constraint formulation and propagation (what Stefik called constraint formulation and propagation corresponds to secondary preconditions and regression in my framework). In my thesis, I expand the technique by incorporating partially ordered (i.e., nonlinear) plans, instantiation variables (i.e., formal objects), and

a variant of hierarchical planning in which abstract operators are constructed dynamically. These devices have the effect of introducing the principle of least-commitment, as they are used to defer search as long as possible. In addition, in my thesis, I remove the various assumptions that are incorporated into the technique presented here, such as the assumption that the initial state is completely known.

2

Formalization

In formalizing the classical planning problems, we shall draw a distinction between a state of the world and a description of a state. The state of the world is an abstract concept referring to the totality of all that is true of the world and all that is false. To know the state is to be omniscient. A description, on the other hand, is more concrete: it is a collection of facts about the state expressed in some language. Furthermore, a description need not be complete: certain details might be left out, either because they are not known or because they are thought to be unimportant. Hence, there can be more than one state satisfying a given description.

The distinction between states and state descriptions is not new. For example, the distinction was made by McCarthy and Hayes in developing their situation calculus [7]. The reason for emphasizing it here is that it is crucial to the proper characterization of actions. Since actions are assumed to alter the world in a deterministic fashion, performing an action will transform the world from one state to a uniquely determined succedent state. Actions can therefore be characterized as functions mapping states of the world into other states of the world. This is the traditional view of actions, yet, when implementing practical planning systems, many researchers have chosen to characterize actions as functions that map a ***description*** of one state into a ***description*** of its successor state. In Section 2.3 we will see that there appear to be actions for which the description of the succedent state would have to be infinite to reflect all of the state changes in their entirety. This is unacceptable from a practical standpoint. Hence, systems that treat actions as functions on state descriptions must necessarily limit the range of problems they can solve. None of this is an issue, however, when actions are treated as functions on states.

2.1 FIRST-ORDER LOGIC FORMALIZATION

The formalization of states, state descriptions, and actions that will now be presented is based on first-order logic. First-order logic was chosen because it provides a very general framework for expressing and solving classical planning problems. In this formalization, states are identified with algebraic structures, state descriptions with well-formed formulas, and actions with functions on algebraic structures. An algebraic structure is a complete account of which relations hold among which objects, and thus determines the truth value of every formula in the language. An algebraic structure therefore corresponds to the notion of a state in that both represent the totality of all that is true and all that is false. Well-formed formulas are used to describe facts about algebraic structures; hence, the relationship between algebraic structures and well-formed formulas is identical to the relationship between states and state descriptions. Consequently, it seems natural to equate states with algebraic structures and state descriptions with well-formed formulas. Actions become formally characterized as functions on structures as a consequence of equating states with structures. In keeping with tradition, we will refer to actions in this framework as *operators* so as to distinguish between the formal characterization of an action and the event that actually takes place in the "real world."

Let us consider how a planning problem would be stated, given the above formalization. Initial-state and goal descriptions are both descriptions of states and, hence, are expressed as sets of well-formed formulas. Thus, we will have a set of formulas Γ describing the initial state and a set G describing the goal state. Operators are described in two parts. The first part states the *preconditions* that must be met before the operator can be applied. For example, in many block-stacking problems, a block can be moved only if no other block is on top of it. Preconditions are just state descriptions and, hence, are expressed as a set of well-formed formulas π.

The second part of an operator description is a description of a function on algebraic structures. This function defines how the operator affects the state of the world when it is applied. Unfortunately, there is no standard way of expressing functions on structures, as they are not an integral part of first-order logic. An appropriate language for specifying operators must therefore

be developed. Before considering how to construct such a language, we need to examine the notion of a structure more closely. An algebraic structure consists of the following elements:

(1) A nonempty set (class) of objects D called the *domain* of the structure.

(2) An n-ary relation r on D (i.e., a set-theoretic relation with n arguments whose components are elements of D) for every n-ary relation symbol R.

(3) An n-ary function f on D for every n-ary function symbol F.

(4) A distinguished object c in D for every constant symbol C.

The relation/function/object associated with symbol $R/F/C$ is called the *interpretation* of $R/F/C$. As an example, suppose that we have a blocks world consisting of a $\mathcal{TABLE}$ and three blocks $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, where blocks $\mathcal{A}$ and $\mathcal{B}$ are resting on the $\mathcal{TABLE}$ and block $\mathcal{C}$ is stacked on top of block $\mathcal{A}$. Suppose, further, that our language for talking about this world has four constant symbols, A, B, C, and $TABLE$, corresponding to the objects in the world, and one relation symbol On, where $\mathrm{On}(x, y)$ means that x is on top of y. Then the structure representing this world would have $\{\mathcal{A}, \mathcal{B}, \mathcal{C}, \mathcal{TABLE}\}$ as its domain, $\mathcal{A}$ as the interpretation of A, $\mathcal{B}$ as the interpretation of B, $\mathcal{C}$ as the interpretation of C, $\mathcal{TABLE}$ as the interpretation of $TABLE$, and $\{\langle \mathcal{A}, \mathcal{TABLE}\rangle, \langle \mathcal{B}, \mathcal{TABLE}\rangle, \langle \mathcal{C}, \mathcal{A}\rangle\}$ as the interpretation of On. Viewed semantically, x is on top of y if and only if the ordered pair $\langle x, y\rangle$ appears in the interpretation of On.

To arrive at a practical way of specifying functions on structures, we shall place a number of restrictions on the kinds on functions that may be defined. The first restriction is that a function may not alter the domain of a structure. That is, if M is a structure and f is a function on structures, then the domain of $f(M)$ is identical to the domain of M. This restriction is of concern only when we wish to describe the effects of an action that creates or destroys objects in the world. An example of such an action would be the GENSYM function in LISP, which creates new LISP atoms. The difficulty here is that the restriction prevents us from modeling the creation and destruction of objects by adding and deleting elements of the domain. However, we

can obtain the same effect by introducing a unary relation, say U, where $U(x)$ is true if and only if x "actually" exists. The domain of the structure would include all objects that could possibly exist; objects would be "created" and "destroyed" by modifying the interpretation of U. Note that this is precisely how GENSYM is implemented in a real computer: GENSYM does not create LISP atoms "out of thin air," but rather it locates an area of unused memory and claims it for use as a new atom. Clearly, the restriction that an operator must preserve the domain of a structure does not affect the kinds of behavior that may be considered; it only influences the way in which the behavior is simulated.

The second restriction is that a function on structures may not alter the language used to describe the world. That is, relation, function, and constant symbols may neither be introduced nor eliminated by an operator. This restriction is implicit in all work done in planning to date. It has never been stated explicitly, since it is hard to imagine a situation it which altering the language would make any sense. Yet, if one really wanted to, one could obtain the effect of modifying the language by introducing relations, functions and constants as objects in the domain (axiomatic set theory [13] provides a convenient way of doing this) and then "creating" and "destroying" them in a manner similar to that described in the preceding paragraph.

The motivation for this second restriction is that it allows a function on structures to be decomposed into a collection of functions—one function for each relation symbol, function symbol, and constant symbol. Each function in the collection defines the interpretation of the corresponding symbol, in the succedent state, in terms of the state of the world that existed prior to the application of the operator. In other words, if f_S is the function corresponding to symbol S and if M is the structure defining the current state of the world, then the interpretation of S in the succedent state is given by $f_S(M)$.

To provide a way of specifying these functions, let us introduce our third and final restriction: each function must be representable as a well-formed formula. That is, each function f_S corresponding to symbol S is defined by a well-formed formula φ_S such that

(1) For each n-ary relation symbol R, $R(x_1, \ldots, x_n)$ is true in the succedent state if and only if $\varphi_R(x_1, \ldots, x_n)$ was true previously (where $x_1, \ldots, x_n$ are the free variables of φ_R)

(2) For each n-ary function symbol F, $F(x_1, \ldots, x_n) = w$ is true in the succedent state if and only if $\varphi_F(x_1, \ldots, x_n, w)$ was true previously.

(3) For each constant symbol C, $C = w$ is true in the succedent state if and only if $\varphi_C(w)$ was true previously.

For example, suppose we have an operator that places block B on top of block C. After this operator is applied, B becomes situated on top of C and every block except B remains where it was. Therefore, $\mathrm{On}(x, y)$ is true after the application of the operator if and only if $(x = B \wedge y = C) \vee (x \neq B \wedge \mathrm{On}(x, y))$ was true previously. In other words, the interpretation of On in the succedent state is the set of ordered pairs $\langle x, y \rangle$ such that $(x = B \wedge y = C) \vee (x \neq B \wedge \mathrm{On}(x, y))$ is true in the current state. If this operator were applied to the blocks world described earlier, where the interpretation of On was $\{\langle A, TABLE \rangle, \langle B, TABLE \rangle, \langle C, A \rangle\}$, the resulting interpretation of On would then be $\{\langle A, TABLE \rangle, \langle C, A \rangle, \langle B, C \rangle\}$.

With the planning technique discussed later in this paper, it is important to know exactly what modifications an operator makes in a structure to select the appropriate operators for achieving the intended goals. Therefore, we shall express the φ_R's, φ_F's and φ_C's defined above in terms of other formulas that make the modifications explicit and then deal exclusively with these other formulas. For relation symbols, this means expressing each φ_R associated with an operator a in terms of two other formulas, α_R and δ_R, which, respectively, describe the additions to and the deletions from the interpretation of R: if $\alpha_R(x_1, \ldots, x_n)$ is true when operator a is applied, the tuple $\langle x_1, \ldots, x_n \rangle$ is added to the interpretation of R, and if $\delta_R(x_1, \ldots, x_n)$ is true then $(x_1, \ldots, x_n)$ is deleted from the interpretation of R. For this to make sense, $\alpha_R(x_1, \ldots, x_n)$ and $\delta_R(x_1, \ldots, x_n)$ cannot be true simultaneously, as we are not requiring that the additions and deletions be performed in any particular order. Given α_R and δ_R, $R(x_1, \ldots, x_n)$ is true after operator a is applied if and only if

$$\alpha_R(x_1, \ldots, x_n) \vee (\neg\, \delta_R(x_1, \ldots, x_n) \wedge R(x_1, \ldots, x_n)) \tag{2.1}$$

was true beforehand. In other words, $\langle x_1, \ldots, x_n \rangle$ is in the interpretation of R after applying a if and only if it was added or it was in the interpretation of R beforehand and not deleted. Formula

(2.1) is therefore equivalent to φ_R. Note that appropriate α_R's and δ_R's can be found to make (2.1) equivalent to φ_R for any arbitrary φ_R. For example, we can let $\alpha_R(x_1, \ldots, x_n)$ be the formula $\varphi_R(x_1, \ldots, x_n)$ and $\delta_R(x_1, \ldots, x_n)$ be $\neg\, \varphi_R(x_1, \ldots, x_n)$. For efficient problem solving, though, α_R and δ_R should be chosen to reflect the actual additions to and deletions from the interpretation of R. For example, for the block-stacking operator described previously, a suitable $\alpha_{\mathrm{On}}(x, y)$ would be $(x = B \wedge y = C)$ and a suitable $\delta_{\mathrm{On}}(x, y)$ would be $(x = B \wedge y \neq C)$. Note that $\delta_{\mathrm{On}}(x, y)$ cannot be $(x = B)$, since $\alpha_{\mathrm{On}}(x, y)$ and $\delta_{\mathrm{On}}(x, y)$ are not allowed to be true simultaneously.

The formulas defining the interpretations of the function symbols in the succedent state can be restructured in much the same way as the formulas for relation symbols. In the case of functions, though, we can take advantage of the fact that a function must be defined everywhere, as required by the definition of an algebraic structure. Consequently, $F(x_1, \ldots, x_n) = w$ is true after an operator has been applied if and only if the operator changed the value of $F(x_1, \ldots, x_n)$ to w or the operator preserved the value of $F(x_1, \ldots, x_n)$ and $F(x_1, \ldots, x_n) = w$ was true previously. These changes can be described by a single formula μ_F, where $\mu_F(x_1, \ldots, x_n, w)$ is true if and only if the value of $F(x_1, \ldots, x_n)$ is to be updated to w when the operator is applied. Since functions have unique values, μ_F must have the property that either there is a unique w for which $\mu_F(x_1, \ldots, x_n, w)$ is true or there are no w's for which $\mu_F(x_1, \ldots, x_n, w)$ is true. Given such a μ_F, $F(x_1, \ldots, x_n) = w$ is true after the operator is applied if and only if

$$\mu_F(x_1, \ldots, x_n, w) \vee (\neg\, \exists v[\mu_F(x_1, \ldots, x_n, v)] \wedge F(x_1, \ldots, x_n) = w) \qquad (2.2a)$$

was true previously; that is, $F(x_1, \ldots, x_n) = w$ is true after the operator is applied if and only either if the value of $F(x_1, \ldots, x_n)$ was updated to w, or $F(x_1, \ldots, x_n) = w$ was true beforehand and the operator preserved the value of $F(x_1, \ldots, x_n)$. Formula (2.2a) is therefore equivalent to φ_F. As with α_R and δ_R, an appropriate μ_F can be found to make (2.2a) equivalent to φ_F for any arbitrary φ_F (e.g., let $\mu_F(x_1, \ldots, x_n, w)$ be $\varphi_F(x_1, \ldots, x_n, w)$). However, for efficient problem solving, μ_F should be chosen to reflect the actual updates of the interpretation of F. As an example, suppose we wished to model the assignment statement $U \leftarrow V$, where U and V are

program variables. To do so, we could have a function Val mapping program variables to their values, plus an operator that updates Val(U) to be the value of Val(V). An appropriate update condition $\mu_{\mathrm{Val}}(x, w)$ for this operator would then be $(x = U \wedge w = \mathrm{Val}(V))$.

Constant symbols are handled in exactly the same way as function symbols, since constants are simply functions without arguments. Therefore, $C = w$ is true in the succedent state if and only if

$$\mu_C(w) \vee (\neg \exists v[\mu_C(v)] \wedge C = w) \tag{2.2b}$$

was true previously. Note that Formula (2.2b) is simply a special case of Formula (2.2a).

When dealing with several operators, we will need to distinguish the add, delete, and update conditions of one operator from those of another. This we will do by using superscripts: we will write α_R^a and δ_R^a to mean, respectively, the add and delete conditions defining the interpretation of relation symbol R after operator a is applied, and we will write μ_F^a to mean the update condition defining the interpretation of function symbol F after the application of operator a (likewise for constant symbols). We will also use superscripts to distinguish the preconditions of one operator from those of another. Thus, π^a is the set of preconditions of operator a.

2.2 OPERATOR SCHEMATA

When formulating a planning problem, one quite often encounters groups of operators whose add, delete, and update conditions would be identical given an appropriate substitution of terms. For example, the operator described earlier for stacking block B atop block C has as its add and delete conditions for On(x, y) the formulas $(x = B \wedge y = C)$ and $(x = B \wedge y \neq C)$, respectively. Similarly, an operator for stacking block A on top of block C would have as its add and delete conditions $(x = A \wedge y = C)$ and $(x = A \wedge y \neq C)$. These formulas are identical except that, wherever B appears in one pair of formulas, A appears in the other. Instead of requiring that each and every operator in such a group be defined separately, we will introduce *operator schemata* so that the group may be defined collectively. Schemata allow one to define parametric classes of

operators by introducing parameters as placeholders for terms in the various formulas that make up an operator definition. A schema is then specialized to a particular operator by substituting the appropriate terms for the parameters. For example, we could define a block-stacking schema with parameters p and q, where p is to be stacked on top of q. The add and delete conditions for $\mathrm{On}(x, y)$ in the schema definition would then be $(x = p \wedge y = q)$ and $(x = p \wedge y \neq q)$, respectively. Substituting B for p and C for q yields an operator that stacks block B on top of block C.

It would be useful at this point to introduce a standard notation for defining operators and operator schemata. This notation is illustrated below. A schema definition consists of the name of the schema, a parameter list, and four groups of formulas labeled PRECOND, ADD, DELETE and UPDATE. If the parameter list is empty, the schema defines a single operator.

$Name(p_1, \ldots, p_m)$

PRECOND: $\pi_1(p_1, \ldots, p_m), \ldots, \pi_n(p_1, \ldots, p_m)$

ADD: $R_1(x_1, \ldots, x_{n_1})$ for all $x_1, \ldots, x_{n_1}$ such that $\alpha_{R_1}(x_1, \ldots, x_{n_1}, p_1, \ldots, p_m)$

$R_2(x_1, \ldots, x_{n_2})$ for all $x_1, \ldots, x_{n_2}$ such that $\alpha_{R_2}(x_1, \ldots, x_{n_2}, p_1, \ldots, p_m)$

...

DELETE: $R_1(x_1, \ldots, x_{n_1})$ for all $x_1, \ldots, x_{n_1}$ such that $\delta_{R_1}(x_1, \ldots, x_{n_1}, p_1, \ldots, p_m)$

$R_2(x_1, \ldots, x_{n_2})$ for all $x_1, \ldots, x_{n_2}$ such that $\delta_{R_2}(x_1, \ldots, x_{n_2}, p_1, \ldots, p_m)$

...

UPDATE: $F_1(x_1, \ldots, x_{n_1}) \leftarrow w$

for all $x_1, \ldots, x_{n_1}, w$ such that $\mu_{F_1}(x_1, \ldots, x_{n_1}, w, p_1, \ldots, p_m)$

$F_2(x_1, \ldots, x_{n_2}) \leftarrow w$

for all $x_1, \ldots, x_{n_2}, w$ such that $\mu_{F_2}(x_1, \ldots, x_{n_2}, w, p_1, \ldots, p_m)$

...

The PRECOND group, which specifies the precondition of the schema, consists of a set of well-formed formulas $\pi_1(p_1, \ldots, p_m), \ldots, \pi_n(p_1, \ldots, p_m)$ whose free variables are the schema parameters. The ADD group specifies the add conditions α_R for each relation symbol R. The conditions are

specified by a set of statements of the form

"(add) $R(x_1, \ldots, x_n)$ for all $x_1, \ldots, x_n$ such that $\alpha_R(x_1, \ldots, x_n, p_1, \ldots, p_m)$"

where the x_i's are distinct variables and are different from the parameters $p_1, \ldots, p_m$. The x_i's, together with the parameters, constitute the free variables of α_R. The format of the DELETE group is identical to that of the ADD group. The DELETE group, however, specifies the delete conditions δ_R for each relation symbol R. The UPDATE group specifies the update conditions μ_F and μ_C for each function symbol F and each constant symbol C respectively. These conditions are expressed by a set of statements each of which is of the form

"(update) $F(x_1, \ldots, x_n) \leftarrow w$ for all $x_1, \ldots, x_n, w$ such that $\mu_F(x_1, \ldots, x_n, w, p_1, \ldots, p_m)$"

for function symbols or, alternatively,

"(update) $C \leftarrow w$ for all w such that $\mu_C(w, p_1, \ldots, p_m)$"

for constant symbols. As with the ADD and DELETE groups, w and the x_i's are distinct variables and are different from the parameters.

As an example of what an actual schema might look like, consider the following schema, which defines a class of operators Put(p, q) for stacking block p on top of q, where q may be another block or the table:

Put(p, q)

PRECOND: $p \neq q$, $p \neq TABLE$, $\forall z(\neg \mathrm{On}(z, p))$, $[q = TABLE \vee \forall z(\neg \mathrm{On}(z, q))]$

ADD: $\mathrm{On}(x, y)$ for all x, y such that $(x = p \wedge y = q)$

DELETE: $\mathrm{On}(x, y)$ for all x, y such that $(x = p \wedge y \neq q)$

UPDATE: $A \leftarrow w$ for all w such that *FALSE*

$B \leftarrow w$ for all w such that *FALSE*

$C \leftarrow w$ for all w such that *FALSE*

$TABLE \leftarrow w$ for all w such that *FALSE*

The precondition states that p and q must be distinct, that p cannot be the table, that no object may be on top of p, and that either q must be the table or no object may be atop q. These are the usual constraints one finds in block-stacking problems.

Since it is often not the case that an operator will modify the interpretation of every symbol in the language, we will introduce the following notational convention: if any α_R, δ_R, μ_F or μ_C is not specified, then we shall take it to be the formula *FALSE*. For example, Put(p, q), as defined above, does not modify the interpretations of A, B, C, or *TABLE*. Therefore, we could define Put(p, q) more succinctly as follows:

Put(p, q)

PRECOND: $p \neq q$, $p \neq TABLE$, $\forall z\,(\neg\,\mathrm{On}(z, p))$, $[q = TABLE \vee \forall z\,(\neg\,\mathrm{On}(z, q))]$

ADD: $\mathrm{On}(x, y)$ for all x, y such that $(x = p \wedge y = q)$

DELETE: $\mathrm{On}(x, y)$ for all x, y such that $(x = p \wedge y \neq q)$

In essence, the convention is to presume that the interpretation of a symbol is not modified unless specified otherwise. This convention has all the benefits of the "STRIPS assumption" [5]; however, because it is merely a notational convention and we are dealing with functions on states and not functions on state descriptions, it has none of the drawbacks of the STRIPS assumption [16].

We will also adopt as a notational convention that, if no preconditions are given for an operator, then the precondition is taken to be the formula *TRUE*. In other words, we will assume that the operator may be applied in any state.

2.3 VALID PLANS

The statement of a planning problem consists of a set of well-formed formulas Γ describing the initial state of the world, a set of formulas G describing the goals to be achieved, and a set of operator schemata. The object is to find an appropriate sequence of operators (i.e., instantiated schemata) that will transform any structure satisfying Γ into a structure that satisfies G. We shall call such a sequence of operators *a valid plan for achieving G, given Γ*, or simply *a valid plan for*

achieving G when the intended Γ is understood. This section examines the validity conditions in detail and explores ways of testing a plan for validity.

Two conditions must hold for a plan to be valid: first, the preconditions of an operator must be satisfied when that operator is applied; second, the goals must be satisfied after the entire plan has been executed. To state these conditions more precisely, we shall introduce the following definitions. Let ϵ denote the empty sequence—that is, the sequence containing no operators. Let the sequence σ be called a *prefix* of a sequence θ if and only if there exists a sequence γ such that $\theta = \sigma\gamma$ (i.e., θ is equal to the concatenation of σ followed by γ). For example, the prefixes of the sequence $a_1a_2\cdots a_n$ are ϵ, a_1, a_1a_2, $a_1a_2a_3, \ldots, a_1a_2\cdots a_n$. Finally, let us write $\Gamma\{\theta\}\varphi$ to mean that, if every formula in the set Γ is true before the sequence of operators θ is applied, then the formula φ will be true after θ is applied. More formally, if we let $a(M)$ denote the structure obtained when operator a is applied to structure M, then

(1) $\Gamma\{\epsilon\}\varphi$ holds if and only if every structure satisfying Γ satisfies φ, and

(2) $\Gamma\{a_1a_2\cdots a_n\}\varphi$ holds if and only if $a_n \circ a_{n-1} \circ \cdots \circ a_1(M)$ satisfies φ for every structure M satisfying Γ,

where "$\circ$" denotes function composition. Given the above definitions, the validity conditions may be stated as follows: θ is a valid plan for achieving G given Γ if and only if

(1) $\Gamma\{\theta\}g$ holds for all formulas $g \in G$, and

(2) For every prefix σa of θ, $\Gamma\{\sigma\}\pi_i$ holds for every formula $\pi_i \in \pi^a$, where a is an operator and π^a is the set of preconditions of a.

Unfortunately, it is usually not possible to apply the definition of $\Gamma\{\theta\}\varphi$ directly when testing a plan for validity, as Γ may have an infinite number of models. What we need to do, therefore, is restate the definition of $\Gamma\{\theta\}\varphi$ in terms of theorem proving, so that we may then prove the validity of a plan without having to consider the models of Γ.

Progression Operators

We will consider two possible ways in which the definition of $\Gamma\{\theta\}\varphi$ might be restated in terms of theorem proving. The first approach is to find a *progression operator* [9] for each operator a. Progression operators map the conditions that exist before an action is performed into those that exist after its performance. Thus, if a^{+1} is the progression operator for a, then $\Gamma\{\theta\}\varphi$ holds if and only if φ is a theorem of $a^{+1}(\Gamma)$. If a progression operator can be found for each operator a, then the definition of $\Gamma\{\theta\}\varphi$ could be restated as follows:

(1) $\Gamma\{\epsilon\}\varphi$ if and only if φ is a theorem of Γ, and

(2) $\Gamma\{a_1 a_2 \cdots a_n\}\varphi$ if and only if φ is a theorem of $a_n^{+1} \circ \cdots \circ a_1^{+1}(\Gamma)$.

Unfortunately, progression operators have a major problem: while it is possible to define an appropriate a^{+1} for any operator a, there appear to be operators and finite Γ's for which $a^{+1}(\Gamma)$ is necessarily infinite. By definition, $a^{+1}(\Gamma)$ must be an axiomatization of the set of postconditions of Γ; that is, $a^{+1}(\Gamma)$ must axiomatize $\{\varphi \mid \Gamma\{a\}\varphi\}$. We could simply define $a^{+1}(\Gamma)$ to be this set, but this definition is not practical, as the set of postconditions of Γ is infinite: for computational reasons, we would much prefer a finite axiomatization of the postconditions. Unfortunately, there appear to be cases in which the postconditions cannot be axiomatized finitely, even though Γ may be finite. For example, let Γ be the set of formulas

Q1: $\forall x\,(s(x) \neq 0)$

Q2: $\forall x\,y\,(s(x) = s(y) \rightarrow x = y)$

Q3: $\forall x\,(x = 0 \vee \exists y\,(s(y) = x))$

Q4: $\forall x\,(x + 0 = x)$

Q5: $\forall x\,y\,(x + s(y) = s(x + y))$

Q6: $\forall x\,(x \cdot 0 = 0)$

Q7: $\forall x\,y\,(x \cdot s(y) = (x \cdot y) + x)$

H1: $\forall x\,(H(x) \leftrightarrow \mathcal{A}(x))$

where $A(x)$ is a formula that does not contain the symbol H, and let a be the operator whose schema is

$$\text{UPDATE: } x + y \leftarrow w \text{ for all } x, y, w \text{ such that } w = 0$$
$$x \cdot y \leftarrow w \text{ for all } x, y, w \text{ such that } w = 0$$

Formulas Q1 through Q7 are essentially the axioms of Peano arithmetic without the induction axioms. Formula H1 defines the unary relation symbol H in terms of 0, s, $+$, and $\cdot$ by means of the formula $A(x)$, which will be described below. Operator a leaves the interpretations of 0, s, and H unaltered, but redefines $+$ and $\cdot$ to be zero everywhere after a is applied (i.e., $x + y = x \cdot y = 0$ for all x and y in the succedent state). Since $+$ and $\cdot$ would no longer correspond to addition and multiplication after a is applied, it seems plausible that, if $A(x)$ made heavy use of addition and multiplication, it might not be possible to finitely axiomatize the postconditions involving H. We will now construct an $A(x)$ that appears to have just this property.

Let us write $s^n(0)$ as shorthand for the nth successor of 0 (i.e., $s^0(0) = 0$, $s^1(0) = s(0)$, $s^2(0) = s(s(0))$, $s^3(0) = s(s(s(0)))$, etc). Then it can be shown [1] that, for any partial recursive function $p : N^k \rightarrow N$ on the natural numbers, there exists a formula $A_p(x_1, \ldots, x_k, y)$ involving only 0, s, $+$, and $\cdot$ such that $p(n_1, \ldots, n_k) = m$ if and only if $A_p(s^{n_1}(0), \ldots, s^{n_k}(0), s^m(0))$ is a theorem of formulas Q1–Q7. The formula A_p is said to *represent* the function p. Furthermore, it can be shown that, if $T_1, T_2, \ldots$ is a recursive enumeration of Turing machines, then there exists a partial recursive indicator function $h : N \rightarrow N$ such that $h(n) = 0$ if and only if T_n eventually halts when started on a blank tape. Let $T_1, T_2, \ldots$ be a recursive enumeration of Turing machines and let $A(x)$ be the formula $A_h(x, 0)$, where h is the partial recursive indicator function defined above and $A_h(x, y)$ is a formula representing h. Having defined $A(x)$ to be the formula $A_h(x, 0)$, we have as a result that $H(s^n(0))$ is a theorem of Γ if and only if T_n halts on a blank tape. Furthermore, since a does not affect the interpretations of 0, s, or H, $H(s^n(0))$ is a postcondition of Γ if and only if $H(s^n(0))$ is a theorem of Γ. Let Γ' be an axiomatization of the postconditions of Γ. Then $H(s^n(0))$ is a theorem of Γ' if and only if T_n halts on a blank tape. Since $+$ and $\cdot$ are zero everywhere after operator a is applied, we can decompose Γ' into an equivalent set of formulas

$\Gamma'_1 \cup \Gamma'_2$, where Γ'_1 is the set

$$\{\forall x\, y\, (x + y = 0) \wedge \forall x\, y\, (x \cdot y = 0)\}$$

and Γ'_2 is is obtained from Γ' by substituting 0 for all terms of the form $t_1 + t_2$ or $t_1 \cdot t_2$ in every formula of Γ'. Thus, s and H do not appear in Γ'_1, and $+$ and $\cdot$ do not appear in Γ'_2. Furthermore, the cardinality of Γ'_2 is less than or equal to the cardinality of Γ'. Since $\Gamma'_1 \cup \Gamma'_2$ is equivalent to Γ', it follows that $H(s^n(0))$ is a theorem of Γ' if and only if $H(s^n(0))$ is a theorem of $\Gamma'_1 \cup \Gamma'_2$. But s and H do not appear in Γ'_1. Therefore, the formula $H(s^n(0))$ is true in all structures satisfying $\Gamma'_1 \cup \Gamma'_2$ if and only if it is true in all structures satisfying Γ'_2. Hence, $H(s^n(0))$ is a theorem of Γ' if and only if $H(s^n(0))$ is a theorem of Γ'_2. Hence, T_n halts on a blank tape if and only if $H(s^n(0))$ is a theorem of Γ'_2. But $+$ and $\cdot$ do not appear in any formula of Γ'_2. Therefore, Γ'_2 must axiomatize H by using only 0 and the successor function s. This seems too weak a language, however, for defining the set of Turing machines that halt on blank tapes without effectively enumerating all such Turing machines. Thus, we make the following conjecture:

Conjecture. Γ'_2 is infinite.

If this conjecture is true, Γ' must be infinite since the cardinality of Γ' is greater than or equal to the cardinality of Γ'_2. Therefore, all axiomatizations of the postconditions of Γ must be infinite; in particular $a^{+1}(\Gamma)$ must be infinite. Although it appears unlikely that the conjecture is false, it has not yet been formally proved.

Regression Operators

The second approach to restating the definition of $\Gamma\{\theta\}\varphi$ is essentially the opposite of the first: instead of advancing Γ forward through the plan using progression operators, we will move φ backwards using *regression operators* [9, 16]. This involves finding for each operator a a function a^{-1} mapping formulas into formulas such that φ is true after a is applied if and only if $a^{-1}(\varphi)$ was true beforehand; that is, for every structure M, M satisfies $a^{-1}(\varphi)$ if and only if $a(M)$ satisfies φ. If such functions exist then the definition of $\Gamma\{\theta\}\varphi$ could be restated as follows:

(1) $\Gamma\{\epsilon\}\varphi$ if and only if φ is a theorem of Γ, and

(2) $\Gamma\{a_1 a_2 \cdots a_n\}\varphi$ if and only if $a_1^{-1} \circ \cdots \circ a_n^{-1}(\varphi)$ is a theorem of Γ.

In general, a regression operator maps a postcondition into the weakest sufficient precondition that must exist before an operator is applied in order for the postcondition is true afterward. In the case of the a^{-1}'s, though, we are insisting that the weakest sufficient precondition must also be a necessary precondition.

Unlike progression, there are no difficulties in computing regressions. To see why this is so, consider the following construction. First, let us augment our language with an additional set of relation, function, and constant symbols, i.e., one new symbol for each existing symbol. We are thereby adding a new relation symbol R' for each existing relation symbol R, a new function symbol F' for each existing function symbol F, and a new constant symbol C' for each existing constant symbol C. The new symbols we will call primed, the old ones nonprimed. The primed symbols will be used to describe the state of the world that exists after operator a is applied, while the nonprimed symbols will describe the state of the world before a is applied. To axiomatize the relationship between the primed and nonprimed symbols, we can make use of Formulas (2.1) and (2.2) discussed in Section 2.1. These formulas define the interpretation of each symbol after an action has been applied in terms of the previous state of the world. Thus, we have the following axioms for each primed symbol:

$$\forall x_1 \cdots x_n \left[R'(x_1, \ldots, x_n) \leftrightarrow \alpha_R^a(x_1, \ldots, x_n) \vee (\neg\, \delta_R^a(x_1, \ldots, x_n) \wedge R(x_1, \ldots, x_n))\right] \quad (2.3a)$$

$$\forall x_1 \cdots x_n\, w \left[(F'(x_1, \ldots, x_n) = w) \leftrightarrow \mu_F^a(x_1, \ldots, x_n, w) \vee (\neg\, \exists v(\mu_F^a(x_1, \ldots, x_n, v)) \wedge F(x_1, \ldots, x_n) = w)\right] \quad (2.3b)$$

$$\forall w \left[(C' = w) \leftrightarrow \mu_C^a(w) \vee (\neg\, \exists v[\mu_C^a(v)] \wedge C = w)\right] \quad (2.3c)$$

The reason this construction is valid is that operators preserve the domains of the structures to which they are applied: if $\mathcal{M}$ is a structure, then the domain of $a(\mathcal{M})$ is precisely the domain of $\mathcal{M}$. Therefore, we can construct a composite structure whose domain is the domain shared by $\mathcal{M}$ and $a(\mathcal{M})$, and whose relations, functions, and distinguished elements are the combined relations,

functions, and distinguished elements of M and $a(M)$. To construct a language for this composite structure, we need only add a new set of symbols to the existing language—one new symbol for each existing symbol, just as was done above.

Now suppose φ is a formula that contains only primed symbols and, hence, describes some condition that might hold after operator a has been applied. Using the axioms given above, we can transform φ into an equivalent formula ψ containing only nonprimed symbols. Since ψ is equivalent to φ and contains only nonprimed symbols, it expresses the necessary and sufficient conditions that must exist before a is applied so that φ will be true afterward. Thus, ψ corresponds to $a^{-1}(\varphi)$.

The transformation of φ into an equivalent nonprimed formula can be done in two steps. The first step is to transform φ into an equivalent canonical form in which every atomic subformula of the canonical φ is either of the form $R'(x_1,\ldots,x_n)$, $F'(x_1,\ldots,x_n) = w$ or $C' = w$ for some collection of variables $x_1,\ldots,x_n,w$. Once in canonical form, φ can be transformed into its nonprimed equivalent by replacing the atomic subformulas of φ with their equivalent nonprimed formulas, as defined in the axioms (2.3). In other words, we replace all occurrences of

$$R'(x_1,\ldots,x_n) \text{ with } \alpha_R(x_1,\ldots,x_n) \vee (\neg\, \delta_R(x_1,\ldots,x_n) \wedge R(x_1,\ldots,x_n))$$

$$F'(x_1,\ldots,x_n) = w \text{ with } \mu_F(x_1,\ldots,x_n,w) \vee (\neg\, \exists v(\mu_F(x_1,\ldots,x_n,v)) \wedge F(x_1,\ldots,x_n) = w)$$

$$C' = w \text{ with } \mu_C(w) \vee (C = w \wedge \forall v\, \neg\, \mu_C(v))$$

These substitutions are justified, since we may always substitute a formula for one that is equivalent. To transform φ into its canonical form, we make use of the following theorem of first-order logic: if $\lambda(\tau)$ is a formula containing the term τ, and if x is neither a free variable of $\lambda(\tau)$ nor a bound variable in the scope of τ, then $\lambda(\tau)$ is logically equivalent to $\exists x\,(\lambda(x) \wedge \tau = x)$. Therefore, we can replace any occurrence of

$$R'(\ldots,\tau,\ldots) \text{ with } \exists x\,(R(\ldots,x,\ldots) \wedge \tau = x)$$
$$F'(\ldots,\tau,\ldots) = \varpi \text{ with } \exists x\,(F(\ldots,x,\ldots) = \varpi \wedge \tau = x)$$
$$F'(\ldots) = \tau \text{ with } \exists x\,(F(\ldots) = x \wedge \tau = x)$$
$$C' = \tau \text{ with } \exists x\,(C = x \wedge \tau = x),$$

where ϖ is an arbitrary term, τ is a term that is not a variable, and x is a variable that appears in neither $R'(\ldots,\tau,\ldots)$, $F'(\ldots,\tau,\ldots) = \varpi$, $F'(\ldots)$ nor $C = \tau$. To put φ in canonical form, we merely apply these substitutions repeatedly until no further substitutions are possible.

As an example of how a primed formula is transformed into its nonprimed equivalent, suppose we have the $\mathrm{Put}(B, C)$ operator discussed in Section 2.2 and that φ is $\forall u \neg \mathrm{On}'(u, A')$. To transform φ into its canonical form, we merely need to replace $\mathrm{On}'(u, A')$ with $\exists v\,(\mathrm{On}'(u, v) \wedge A' = v)$. This produces

$$\forall u \neg \exists v[\mathrm{On}'(u, v) \wedge A' = v].$$

With φ in its canonical form, all that remains is to replace the atomic subformulas of φ with their nonprimed equivalents. Recall from the definition of $\mathrm{Put}(B, C)$ that $\alpha_{\mathrm{On}}(x, y)$ is $(x = B \wedge y = C)$ and $\delta_{\mathrm{On}}(x, y)$ is $(x = B \wedge y \neq C)$. Therefore, all occurrences of $\mathrm{On}'(x, y)$ are replaced by

$$(x = B \wedge y = C) \vee [(x \neq B \vee y = C) \wedge \mathrm{On}(x, y)].$$

Also, since $\mathrm{Put}(B, C)$ does not affect the interpretation of A, μ_A is the formula $FALSE$. Hence, all occurrences of $A' = w$ are replaced by $A = w$. These substitutions produce

$$\forall u \neg \exists v\big[\big((u = B \wedge v = C) \vee [(u \neq B \vee v = C) \wedge \mathrm{On}(u, v)]\big) \wedge A = v\big],$$

which simplifies to

$$A \neq C \wedge \forall u(u = B \vee \neg \mathrm{On}(u, A)).$$

Thus, no block is on top of A after $\mathrm{Put}(B, C)$ has been applied if and only if A and C are distinct blocks and there were no blocks on top of A before the application of $\mathrm{Put}(B, C)$, except possibly block B.

The above method for transforming primed formulas into their nonprimed equivalents leads to the following recursive definition for a^{-1}. In the ground case, we obtain

$$a^{-1}[R(x_1, \ldots, x_n)] \equiv \alpha_R^a(x_1, \ldots, x_n) \vee (\neg\, \delta_R^a(x_1, \ldots, x_n) \wedge R(x_1, \ldots, x_n)) \qquad (2.4a)$$

$$a^{-1}[F(x_1, \ldots, x_n) = w] \equiv \mu_F^a(x_1, \ldots, x_n, w) \vee [\neg\, \exists v\,(\mu_F^a(x_1, \ldots, x_n, v)) \wedge F(x_1, \ldots, x_n) = w] \qquad (2.4b)$$

$$a^{-1}(C = w) \equiv \mu_C^a(w) \vee [\neg\, \exists v\,(\mu_C(v)) \wedge C = w], \qquad (2.4c)$$

where $x_1, \ldots, x_n$ and w are variables. These equations correspond to replacing atomic subformulas with their nonprimed equivalents. The following equations transform atomic formulas into their

canonical forms:

$$a^{-1}[R(\ldots,\tau,\ldots)] \equiv \exists x\,(a^{-1}[R(\ldots,x,\ldots)] \wedge a^{-1}(\tau = x)) \tag{2.4d}$$
$$a^{-1}[F(\ldots,\tau,\ldots) = \varpi] \equiv \exists x\,(a^{-1}[F(\ldots,x,\ldots) = \varpi] \wedge a^{-1}(\tau = x)) \tag{2.4e}$$
$$a^{-1}[F(\ldots) = \tau] \equiv \exists x\,(a^{-1}[F(\ldots) = x] \wedge a^{-1}(\tau = x)) \tag{2.4f}$$
$$a^{-1}(C = \tau) \equiv \exists x\,(a^{-1}[C = x] \wedge a^{-1}(\tau = x)), \tag{2.4g}$$

where ϖ is an arbitrary term, τ is a term that is not a variable, and x is a variable that does not appear in $R(\ldots,\tau,\ldots)$, $F(\ldots,\tau,\ldots) = \varpi$, $F(\ldots) = \tau$, or $C = \tau$. Finally, we have the following equations, which allow (2.4a–g) to be applied to all atomic subformulas in a formula:

$$a^{-1}(\neg\,\varphi) \equiv \neg\,a^{-1}(\varphi) \tag{2.4h}$$
$$a^{-1}(\varphi \wedge \psi) \equiv a^{-1}(\varphi) \wedge a^{-1}(\psi) \tag{2.4i}$$
$$a^{-1}(\varphi \vee \psi) \equiv a^{-1}(\varphi) \vee a^{-1}(\psi) \tag{2.4j}$$
$$a^{-1}(\varphi \rightarrow \psi) \equiv a^{-1}(\varphi) \rightarrow a^{-1}(\psi) \tag{2.4k}$$
$$a^{-1}(\varphi \leftrightarrow \psi) \equiv a^{-1}(\varphi) \leftrightarrow a^{-1}(\psi) \tag{2.4l}$$
$$a^{-1}(\forall x\,\varphi) \equiv \forall x\,a^{-1}(\varphi) \tag{2.4m}$$
$$a^{-1}(\exists x\,\varphi) \equiv \exists x\,a^{-1}(\varphi) \tag{2.4n}$$

3

Plan Synthesis

This chapter presents a technique for solving a subclass of the classical planning problems. The first section establishes the fundamental concepts upon which the technique is based. A particular property of the classical planning problems is identified and an example given illustrating how one might exploit this property when synthesizing a plan. Section 3.2 shows how a simple planning technique can be derived from the property, and, in Section 3.3, a detailed example is provided to demonstrate the technique.

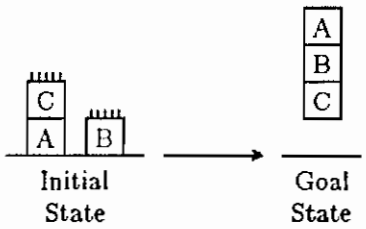
3.1 BASIC CONCEPTS

There are two basic assumptions built into the classical planning problems that can be exploited when a plan is synthesized. The first is that the world can change only as the result of an action. This assumption permits actions to be modeled as state transformations. Furthermore, it forces all plans to have the following property: if some condition is true at one point in a plan but not at an earlier point, then at some point in between there is an operator that causes the condition to become true. This is an important consequence from the point of view of plan synthesis, as it allows one to postulate the existence of operators that cause certain goals to become true. The second assumption is that we are capable of performing only a finite number of actions in a finite amount of time. Consequently, any plan for achieving a particular goal must be finite, as the goal must become true at a definite point in time for it to be achieved. Taken together, these two assumptions imply the following:

Property 3.1. If a condition φ is true at a point p in a sequence of operators but not at an earlier point, then at some point in between there exists an operator that causes φ to become true and φ remains true thereafter until at least point p.

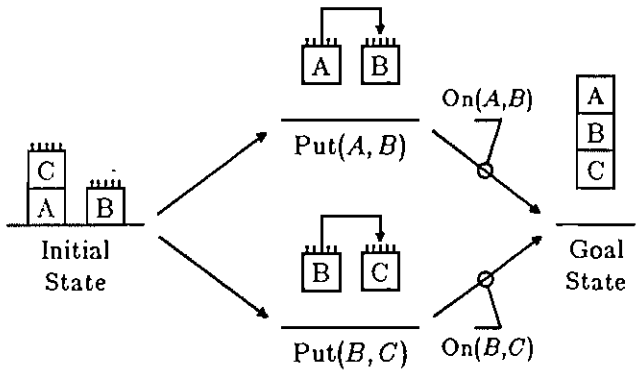
In other words, if some condition is true at one point in a plan but not at an earlier point, then not only must there be an operator somewhere in between that causes the condition to become true, but there must be a final such operator since the number of intervening operators is finite. This combined property turns out to be quite useful during plan synthesis, as we will now demonstrate. A more formal treatment of Property 3.1 appears at the end of this section.

To illustrate how Property 3.1 may be exploited when a plan is being synthesized, let us consider a typical block-stacking problem. Suppose we have the blocks world described in Chapter 2, in which blocks A and B are initially on the *TABLE* and block C is atop block A. Suppose, further, that our goal is to have A on top of B and B on top of C, and that the only operators available are those defined by the Put schema of Section 2.2. The diagram below depicts the initial state and the goal. "bristles" on top of a block signifies that the block is known to be clear (i.e., no other block is on top of it), while a block "floating" above the table signifies that the object supporting the block is not known. The arc from the initial state to the goal signifies that the initial state precedes the goal state in time.

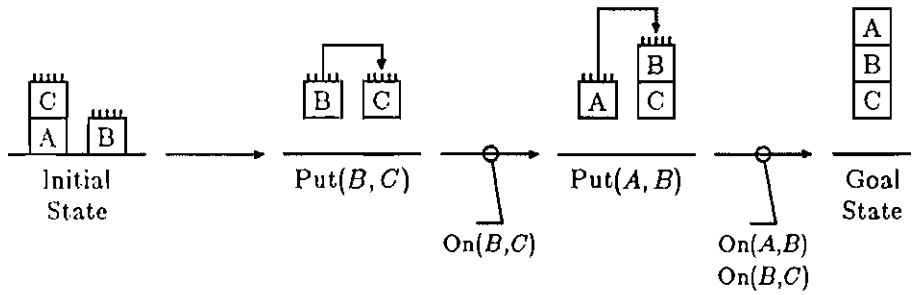


Neither of our goals is satisfied in the initial state; therefore, by Property 3.1 there must be a final point in our plan at which A becomes situated on top of B, and a final point at which B becomes situated on top of C. The only operators available for moving A onto B and B onto C are Put(A, B) and Put(B, C), respectively. Hence, there must exist a point at which we apply Put(A, B), after which A remains on top of B, plus another point at which we apply Put(B, C),

after which B remains on top of C. This is depicted in the diagram below. The conditions that must remain true during particular intervals are identified by labeling the appropriate arcs.

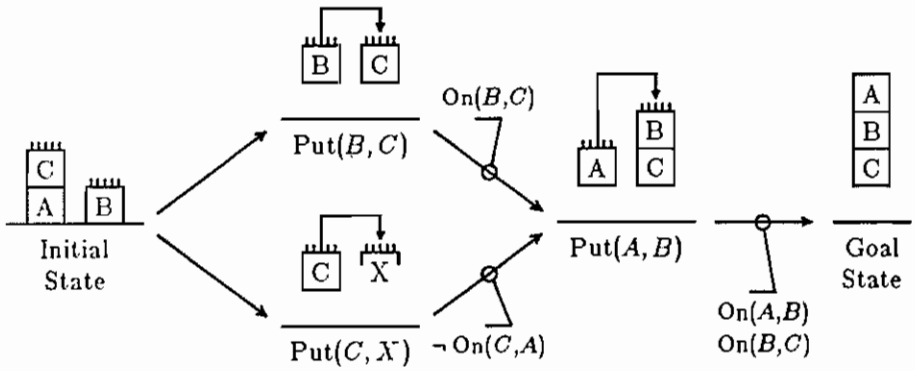


In the final plan, Put(A, B) will come either before Put(B, C) or after Put(B, C). The former case can be ruled out, however, since, with this ordering, the requirement that A remain on top of B after Put(A, B) has been executed contradicts one of the preconditions of Put(B, C), which is that no block be on top of B when Put(B, C) is applied. Therefore, we must perform Put(A, B) after performing Put(B, C).

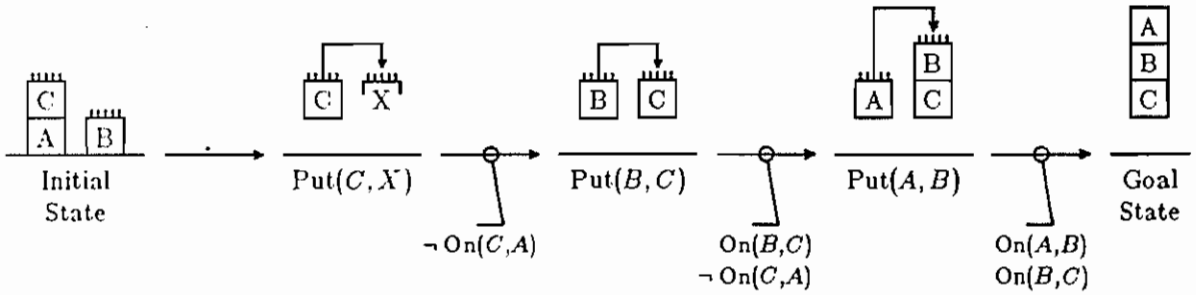


Examining the plan in its current state of development, we find that the goals are now satisfied but that one of the preconditions of Put(A, B) has not been. In particular, C is on top of A in the initial state, which contradicts the requirement that no block be on top of A when Put(A, B) is performed. Therefore, by Property 3.1, there must exist an operator preceding Put(A, B) that causes C to be removed from A, after which C remains off A at least until Put(A, B) is performed. The only operators available for removing C from A are those of the form Put(C, X). Hence, there must exist a point preceding Put(A, B) at which we perform Put(C, X), after which C remains off

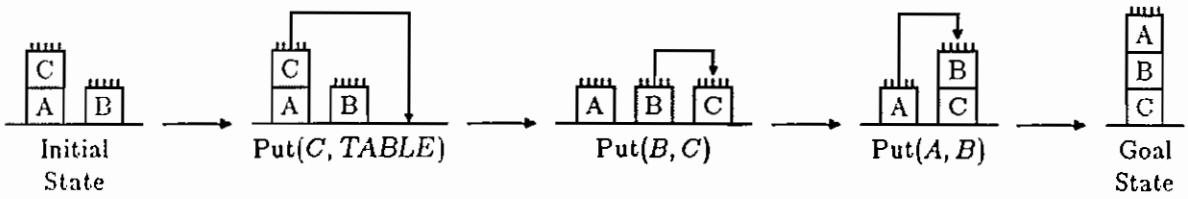
A at least until Put(A, B) is performed. For the moment, let us defer the choice of a particular value for X.



In the final plan, Put(C, X) will come either before Put(B, C) or after Put(B, C). The latter case can be ruled out, however, since, with this ordering, the requirement that B remain on top of C after Put(B, C) has been executed contradicts one of the preconditions of Put(C, X), which is that no block be on top of C when Put(C, X) is applied. Therefore, Put(C, X) must be applied before Put(B, C).



If we now examine the plan, we find that every goal and precondition would be satisfied if we were to let X be the *TABLE*. Therefore, let it be so. This gives us the following plan for stacking A atop B and B atop C: put C on the *TABLE*, then put B on top of C and, finally, put A on top of B.



As the foregoing example illustrates, Property 3.1 contributes to the planning process in two ways. First, it establishes a causal connection between the operators in a plan and the conditions we wish to bring about. This causal linkage permits us to build plans incrementally, introducing operators as needed to satisfy our goals as well as the preconditions of operators previously introduced. The choice of operators is governed by the changes that must be made in the world to bring about the desired conditions; operators that are not essential to constructing a valid plan are not even considered. The result is a tremendous reduction in search compared with that required by an exhaustive search strategy. Furthermore, Property 3.1 does not restrict us to building plans in any particular order, as do forward-chaining and backward-chaining strategies. Instead, operators are inserted *as* needed and *where* needed in an opportunistic fashion (c.f., Hayes-Roth et al. [6]).

The second way in which Property 3.1 contributes to the planning process is by constraining the placement of operators in a plan. When we insert an operator at some point p in a plan so that a particular condition will be true at some later point q, we are considering the last point p preceding q at which that condition becomes true. We can thus *protect* the condition from point p to point q; that is, we can assert that the condition must remain true in the interval between p and q. The advantage of protection is that it enables us to detect impossible orderings of operators: if an operator has the precondition φ, it cannot possibly appear at a point in the plan during which $\neg\varphi$ must remain true. Protection therefore contributes to the minimization of search by allowing us to eliminate impossible orderings from consideration. In fact, in the block-stacking example, protection was so effective that search was avoided altogether.

Protection Through the Ages

Historically, the idea of protecting goals and preconditions was first introduced by Sussman [14] and later refined by Waldinger [16], Warren [17, 18], and others. Sussman developed goal protection as a method for detecting faulty plans. As he explains, using a programming metaphor,

> *... a program is operating correctly, in that it accurately reflects the intent of the programmer, only when each step achieves those goals that the programmer intended it to, and each of those goals remains true at least until the steps which depend upon its being true are run (or the end of the program block if this step is a contributor to the purpose of the program).*

Therefore, if, in the course of plan execution, a goal is violated that was intended to remain true, that plan is then faulty and must be "debugged." It is apparent from the foregoing quote that Sussman had in mind something very much like Property 3.1 when he developed his protection mechanism. However, Sussman viewed protection as being intimately tied to the intent of the programmer, whereas here it is seen as arising from a fundamental principle that is independent of intent (of course, in its use, protection does tend to reflect intent). Furthermore, Sussman employed protection only as a means of detecting faulty plans, not as a guide to ordering operators as done here. Had he recognized this use of protection, he probably would not have had to treat the block-stacking problem presented earlier as an "anomalous situation" requiring special consideration.

Unlike Sussman, Waldinger did employ protection as a guide to ordering operators. However, Waldinger was somewhat overzealous in its application. If a goal or precondition were true in the initial state and not made false by any of the operators currently in the plan, Waldinger's scheme would call for that goal or precondition to be protected without considering the possibility that the goal or precondition might have to be violated and then reestablished in order to solve the overall problem. An example of a problem in which this possibility would have to be considered is the Towers of Hanoi, in which the goal of having the smallest ring on top of the second smallest ring is true in the initial state, but the first ring must be removed from the second so that the other goals can be realized. Waldinger acknowledged that his protection mechanism had drawbacks, but he

did not recognize their source. Instead, he proposed a scheme that circumvented the hidden defect by considering goals in various sequences until a solution was obtained. Although the scheme does work, it is terribly inefficient. Furthermore, as Warren points out [18], reodering is unnecessary if we simply avoid protecting goals that are already satisfied.

Like Waldinger, Warren also used protection as a guide to ordering operators. In Warren's approach, though, a goal is protected only when an operator is inserted that makes the goal true. Warren's scheme therefore operates in accordance with Property 3.1.

Strengthening Property 3.1

It turns out that Property 3.1 is too weak for solving arbitrary planning problems. While it works fine for problems in which the effects of an action are independent of the state in which the action is performed, as in the blocks world, it neglects an important case that must be considered when the effects of an action depend on the state of the world at the time the action is performed. Taking this second case into account, we obtain the theorem stated below. This theorem says that a condition is true after a sequence of operators has been executed *if and only if* (1) there exists an operator at some point in the sequence that causes the condition to become true, and the condition remains true thereafter, or (2) the condition is true initially and never becomes false. Therefore, during plan synthesis, not only must we consider incorporating operators to cause a goal or precondition to become true (Clause 1), but we must also consider the possibility of incorporating operators to prevent a goal or precondition from becoming false if it is true initially (Clause 2). Property 3.1 merely provides a set of sufficient conditions for Clause (1) to hold. The theorem further tells us that a planning technique is fully general if and only if it takes these two possibilities into account, as a goal or precondition cannot be satisfied otherwise.

Theorem 3.2. Let φ be a formula, Γ be a set of formulas, and θ be a sequence of operators. Then $\Gamma\{\theta\}\varphi$ holds if and only if one of the following is true:

(1) There exists a prefix σa of θ, where a is an operator, such that $\Gamma\{\sigma\}\varphi$ is false but $\Gamma\{\sigma a\gamma\}\varphi$ is true for all sequences γ such that $\sigma a\gamma$ is a prefix of θ.

(2) $\Gamma\{\sigma\}\varphi$ for all prefixes σ of θ.

Proof. First we will show that, if either Clause (1) or Clause (2) holds, then $\Gamma\{\theta\}\varphi$ must hold as well. If σa is a prefix of θ, there exists a γ such that $\sigma a\gamma = \theta$. Therefore, Clause (1) implies $\Gamma\{\theta\}\varphi$. If Clause (2) holds, $\Gamma\{\theta\}\varphi$ follows immediately, since θ is a prefix of itself.

To complete the proof we need to show that, if $\Gamma\{\theta\}\varphi$ holds, then either Clause (1) or Clause (2) holds. This we will do by induction on the length of θ. In the base case, θ is the empty sequence ϵ. The only prefix of ϵ is ϵ itself; therefore, if $\Gamma\{\theta\}\varphi$ holds for $\theta = \epsilon$, then Clause (2) must hold. For the induction step, let us assume that, for all θ of length less than or equal to n, $\Gamma\{\theta\}\varphi$ implies (1) or (2). Let θ' be a sequence of operators of length $n+1$, and suppose that $\Gamma\{\theta'\}\varphi$ holds. Let a be an operator and θ'' be a sequence of length n such that $\theta' = \theta''a$. Consider $\Gamma\{\theta''\}\varphi$. Either $\Gamma\{\theta''\}\varphi$ is true or it is false. If it is false, Clause (1) must hold for $\theta = \theta'$ (i.e., consider the case when $\sigma = \theta''$). If $\Gamma\{\theta''\}\varphi$ is true, then, by the induction hypothesis, either Clause (1) or Clause (2) holds for $\theta = \theta''$. If (2) holds for $\theta = \theta''$ then (2) must also hold for $\theta = \theta'$, since we have assumed that $\Gamma\{\theta'\}\varphi$ holds. Likewise, if (1) holds for $\theta = \theta''$, (1) must also hold for $\theta = \theta'$, since, if $\Gamma\{\sigma a\gamma\}\varphi$ is true for all γ such that $\sigma a\gamma$ is a prefix of θ'', then $\Gamma\{\sigma a\gamma\}\varphi$ must be true for all γ such that $\sigma a\gamma$ is a prefix of θ'. Therefore, if $\Gamma\{\theta''\}\varphi$ holds, either Clause (1) or Clause (2) holds for $\theta = \theta'$. But, as shown previously, if $\Gamma\{\theta''\}\varphi$ does not hold, then Clause (1) must hold for $\theta = \theta'$. Therefore, either Clause (1) or Clause (2) holds for $\theta = \theta'$. Since the choice of θ' was arbitrary, it follows that $\Gamma\{\theta\}\varphi$ implies (1) or (2) for all θ of length $n+1$. Hence, by induction, $\Gamma\{\theta\}\varphi$ implies (1) or (2) for all θ. □

Property 3.1 follows as a corollary to Theorem 3.2. Property 3.1 can be stated and proved formally as follows.

Corollary (*Property 3.1*). Let φ be a formula, Γ a set of formulas, θ a sequence of operators, and τ a prefix of θ. Then the following holds: if $\Gamma\{\theta\}\varphi$ is true but $\Gamma\{\tau\}\varphi$ is false, then there exists a prefix σa of θ such that τ is a prefix of σ, a is an operator, $\Gamma\{\sigma\}\varphi$ is false, and $\Gamma\{\sigma a\gamma\}\varphi$ is true for all sequences γ such that $\sigma a\gamma$ is a prefix of θ.

Proof. Suppose that $\Gamma\{\theta\}\varphi$ holds but that $\Gamma\{\tau\}\varphi$ does not. Then either Clause (1) or Clause (2) of Theorem 3.2 holds. But Clause (2) cannot hold, since $\Gamma\{\tau\}\varphi$ is false. Therefore, only Clause (1) holds; that is, there exists a prefix σa of θ, where a is an operator, such that $\Gamma\{\sigma\}\varphi$ is false but $\Gamma\{\sigma a\gamma\}\varphi$ is true for all sequences γ such that $\sigma a\gamma$ is a prefix of θ. It remains only to show that τ must be a prefix of σ. Suppose that τ is not a prefix of σ. Since τ and σa are both prefixes of θ, this implies that σa must be a prefix of τ. Therefore, there exists a sequence γ such that $\tau = \sigma a\gamma$. Therefore, $\Gamma\{\tau\}\varphi$ must be true, since $\Gamma\{\sigma a\gamma\}\varphi$ is true for all sequences γ such that $\sigma a\gamma$ is a prefix of θ. But, by hypothesis, $\Gamma\{\tau\}\varphi$ is false. Contradiction! Therefore, τ must be a prefix of σ. □

3.2 A SIMPLE PLANNING TECHNIQUE

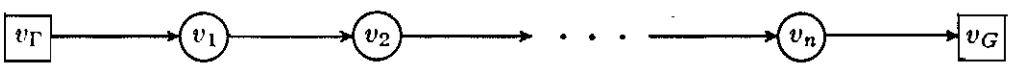
Let us now consider a technique for constructing plans that is based on Theorem 3.2. With this technique, plans are synthesized in much the same way as in the preceding example: we begin with the empty plan (i.e., containing no operators) and add operators until a valid plan is obtained. At each stage in the process, we will have some *current plan*. This plan is analyzed to identify those goals and preconditions not yet satisfied and to determine what additional operators are needed to bring them about. The appropriate operators are then inserted, producing a new current plan and initiating a new cycle of analysis and modification. This process of repeatedly analyzing and modifying the current plan continues until all goals and preconditions have been satisfied. In situations where there are multiple ways of causing a particular goal or precondition to become true, the analysis produces a set of alternative modifications of the current plan. In this case, one of the alternatives must be selected before the plan is modified. However, not all alternatives necessarily lead to solutions, since some ways of effecting one goal or precondition may make it impossible to achieve another. It may thus be necessary to explore a number of alternatives before a valid plan is found.

The technique we shall consider incorporates a number of simplifying assumptions. These assumptions are not essential and, in my thesis [8], I show how they can be lifted to obtain a

completely general synthesis technique. The first assumption is that the initial state is completely known. This makes the validity conditions for a plan decidable (in general, they are undecidable). The second assumption is that function symbols and constant symbols do not change interpretation when an operator is applied (i.e., μ^a_S is false for every operator a and every function symbol or constant symbol S). This makes it easier to decompose a complex goal into simpler subgoals. The last assumption is that, for each object x in the world, there is a constant symbol e_x, called the *standard name* of x, that denotes x in the initial state. Given the preceding assumption, the standard name of x will also denote x at every point in a plan. The reason for this last assumption is that it simplifies the handling of quantifiers.

Representing Plans, Goals, and Protections

To begin, we must establish a representation for plans, goals, and protected conditions. As suggested by the block-stacking example of the previous section, we will represent a plan as a directed acyclic graph, called a *plan graph*, with a single root vertex and a single leaf vertex. The root vertex of a plan graph represents the initial state, while the leaf vertex represents the goal state. The intermediate vertices represent operators. The edges of a plan graph are directed and define a partial ordering of the vertices. From a semantic standpoint, a plan graph asserts that certain operators must appear in the final solution in a certain relative order. Although the representation permits arbitrary partial orders to be specified, we will for the sake of simplicity consider only linear (i.e., totally ordered) plan graphs. An example of a linear plan graph appears below. The diagram uses boxes and circles to distinguish between the root and leaf vertices, on the one hand, and the intermediate vertices on the other.



In our discussion of plan graphs we will adopt the following conventions. We will write v_Γ to denote the root vertex of a plan graph and v_G to denote the leaf vertex, where Γ is the set of

formulas describing the initial state and G is the set of formulas defining our goals. If v_1 and v_2 are vertices, we will write $v_1 \blacktriangleright v_2$ to indicate that there is an edge from v_1 to v_2. The plan graph illustrated above can then be written as $v_\Gamma \blacktriangleright v_1 \blacktriangleright \cdots \blacktriangleright v_n \blacktriangleright v_G$. We will say that a vertex v_1 precedes a vertex v_2, written $v_1 \prec v_2$, to mean that there is a path from v_1 to v_2, and we will write $v_1 \preceq v_2$ as shorthand for $v_1 \prec v_2$ or $v_1 = v_2$. Finally, we will say that a formula φ is true at a vertex v to mean either (1) that φ is true in the initial state, if $v = v_\Gamma$, or (2) that φ is true after execution of the plan, if $v = v_G$, or (3) that φ is true when the operator associated with v is applied, if v is an intermediate vertex. In other words, if our plan graph is $v_\Gamma \blacktriangleright v_1 \blacktriangleright \cdots \blacktriangleright v_n \blacktriangleright v_G$ and a_i is the operator associated with vertex v_i, then (1) φ is true at v_Γ if and only if φ is a theorem of Γ, (2) φ is true at v_i if and only if $\Gamma\{a_1 \cdots a_{i-1}\}\varphi$ holds, and (3) φ is true at v_G if and only if $\Gamma\{a_1 \cdots a_n\}\varphi$ holds.

Protected conditions will be represented as a set P of ordered triples of the form $\langle \varphi, v_1, v_2 \rangle$, where φ is a formula and v_1 and v_2 are vertices such that $v_1 \prec v_2$. P will be referred to as the *protection set*. In semantic terms, each triple in the protection set is an assertion that a particular formula must remain true over some interval in the final solution. More precisely, if $\langle \varphi, v_1, v_2 \rangle \in P$, then φ must be true at every vertex v in the final plan such that $v_1 \prec v \preceq v_2$. In particular, φ must be true at vertex v_2. It will be necessary during plan synthesis to consider all protected formulas that must be true at a particular vertex v. Therefore, let us define ρ_v to be this set: that is,

$$\rho_v = \{\varphi \mid \langle \varphi, v_1, v_2 \rangle \in P \text{ and } v_1 \prec v \preceq v_2\} \tag{3.1}$$

Goals and preconditions will be represented as a set A of ordered pairs of the form $\langle \varphi, v \rangle$, where φ is a formula and v is a vertex. We will refer to this set as the *agenda*. From a semantic standpoint, each ordered pair on the agenda is an assertion that a particular formula must be true at a particular vertex in the final plan. In other words, if $\langle \varphi, v_\Gamma \rangle \in A$, then φ must be true in the initial state, and, if $\langle \varphi, v \rangle \in A$, where $v \neq v_\Gamma$, then one of our goals is to achieve φ at vertex v. The set of all conditions we wish to achieve at a particular vertex v is given by

$$g_v = \{\varphi \mid \langle \varphi, v \rangle \in A\} \tag{3.2}$$

Together, a plan graph, a protection set, and an agenda define a set of constraints that a plan must satisfy to be considered a solution. To synthesize a plan, we take an initial set of constraints defined by an initial plan graph, protection set, and agenda; then, through an appropriate process, we add further constraints until we obtain a complete specification of a plan. At the beginning of the process, our only constraint is for the final plan to achieve every formula g in the goal set G, given that the initial state is described by the set of formulas Γ. Therefore, the initial plan graph is the graph $v_\Gamma \blacktriangleright v_G$, the initial protection set is empty, and the initial agenda is the set $\{\langle g, v_G\rangle \mid g \in G\}$. The problem is then to augment each of the three components—the plan graph, the protection set, and the agenda—until the sequence of operators defined by the plan graph satisfies all of the assertions listed in the protection set and the agenda. For convenience, let us refer to the combination of a plan graph, a protection set, and an agenda as a *partial plan.*

The Technique

The synthesis technique we shall consider is an iterative process by which the initial partial plan is incrementally modified until a solution is obtained. The basic loop involves finding a goal on the agenda that would not be satisfied, given the current partial plan, and then modifying the plan so that the goal will be achieved. This process continues until all goals on the agenda have been satisfied. At each step, the current partial plan is modified in such a way that, if all of the goals on the agenda are satisfied, then all of the assertions in the protection set will likewise be satisfied. This guarantees that, once all of the goals have been attained, we will have constructed a valid plan consistent with the protections.

The modifications made of the current partial plan are governed by a set of rules. For every goal that may be expressed in the logic, there is a corresponding rule. Each rule defines a set of alternative modifications for realizing the corresponding goal at the desired point in the final plan. Each set of alternatives covers all possible solutions, so that, if a rule is applicable and if a solution can be obtained from the current partial plan, at least one of the alternatives defined by that rule is guaranteed to lead to a solution. Consequently, the rules may be applied in any order without

backtracking and without affecting the final solution. Of course, search *is* required to explore the alternatives expressed by a rule. As search is fairly well understood, this report will focus on the modification rules and leave open the issue of an appropriate search strategy.

The rules for modifying plans are based in part on Theorem 3.2. According to this theorem, a goal is satisfied at some point p in the final solution if and only if (1) there is an operator that causes the goal to become true and the goal remains true thereafter until at least point p, or (2) the goal is true initially and never becomes false before point p. This suggests two ways of modifying a partial plan in order to achieve a goal: one is to insert an operator that causes the goal to become true, the other is to prevent the goal from becoming false. There is, however, a third option: since plans are built incrementally, the operator that causes a goal to become true in the final solution may already appear in the current partial plan; therefore, another way of achieving a goal would be to establish the appropriate enabling conditions to allow an existing operator in the plan to cause the goal to become true. These three alternatives are illustrated by the following example.

Suppose we have a world consisting of a briefcase, a dictionary, and a paycheck, each of which may be situated in one of two locations: the home or the office. Operators are available for putting the dictionary and the paycheck into the briefcase and for taking them out, as well as for carrying the briefcase between the two locations. Initially, the briefcase, the dictionary, and the paycheck are at home, and the paycheck is in the briefcase but the dictionary is not. The goal is to have the briefcase and the dictionary at the office, but the paycheck at home. We begin the synthesis process with the empty plan. Let us first consider the goal of having the briefcase at work. Since this goal is not true initially, we must have an operator in our final plan that causes the goal to become true. As the current plan is empty, the only option is to insert the operator that causes the briefcase to be brought to work. Let us next consider the goal of having the dictionary at work. This goal is not satisfied, given the current plan of bringing the briefcase to work. However, if we were to put the dictionary into the briefcase before leaving home, the dictionary would be brought to the office as a side effect. In this case, the operator that causes the dictionary to be at the office (i.e., bringing the briefcase to work) already appears in the plan and an additional

operator is inserted to establish the appropriate enabling condition (i.e., having the dictionary in the briefcase). After making these modifications, we are left with only one more goal to consider, which is to have the paycheck remain at home. Unfortunately, the current plan of putting the dictionary in the briefcase and then bringing the briefcase to the office causes the paycheck to be brought to the office as a side effect. However, if we were to remove the paycheck from the briefcase before leaving home, we would prevent the paycheck from changing locations. Our goal would then be achieved by virtue of the fact that it would never become false. If we choose to remove the paycheck from the briefcase before we put in the dictionary, then our final plan will be to remove the paycheck from the briefcase, put the dictionary in the briefcase, and bring the briefcase to the office.

The three ways of modifying a partial plan illustrated above cover all possible solution paths. This fact is expressed by the theorem that appears below. This theorem may be paraphrased as follows: a condition φ is true at a point p in the final plan if and only if one of the following conditions holds: (1) there exists an operator in the final plan that already appears in the current plan that causes φ to become true, and φ remains true thereafter until at least point p, (2) there exists an operator in the final plan that does not appear in the current plan that causes φ to become true, and φ remains true thereafter until at least point p, or (3) φ is true in the initial state, and remains true until at least point p.

Theorem 3.3. Let θ be a sequence of operators and θ' an expansion of θ. That is, for an appropriate set of operator sequences $\{\beta_1, \beta_2, \ldots\}$, if $\theta = a_1 a_2 \cdots a_n$, then $\theta' = \beta_1 a_1 \beta_2 a_2 \cdots \beta_n a_n \beta_{n+1}$, and, if $\theta = \epsilon$, then $\theta' = \beta_1$. Let $\sigma_1 = \beta_1$ and $\sigma_i = \beta_1 a_1 \cdots \beta_{i-1} a_{i-1} \beta_i$ for $i > 1$. Then $\Gamma\{\theta'\}\varphi$ holds if and only if one of the following is true:

(1) There exists a σ_i such that $\Gamma\{\sigma_i\}\varphi$ is false, but $\Gamma\{\sigma_i a_i \gamma\}\varphi$ is true for all sequences γ such that $\sigma_i a_i \gamma$ is a prefix of θ'.

(2) There exists a prefix σa of θ' such that σ is not a σ_i and $\Gamma\{\sigma\}\varphi$ is false, but $\Gamma\{\sigma a \gamma\}\varphi$ is true for all sequences γ such that $\sigma a \gamma$ is a prefix of θ'.

(3) $\Gamma\{\sigma\}\varphi$ holds for all prefixes σ of θ'.

Proof. The above theorem follows directly from Theorem 3.2, as Clauses (1) and (2) together are equivalent to Clause (1) of Theorem 3.2 and Clause (3) is equivalent to Clause (2) of Theorem 3.2. □

To modify a partial plan to achieve some goal, we merely have to choose one of the three cases described Theorem 3.3 and assert that it holds with respect to the current partial plan and the final solution. In other words, if (φ, v) is a goal on the agenda and if φ is not true at vertex v in the current partial plan, then we can (1) assert that the operator associated with some existing vertex $v' \prec v$ causes φ to become true, and protect φ from v' to v, (2) insert an operator that causes φ to become true, and protect φ up to vertex v, or (3) protect φ from the initial state to vertex v.

To make these assertions, we need to introduce the notion of a *secondary precondition.* A secondary precondition for an operator is a condition that must be true at the time the operator is applied for the operator to have the desired effect. By imposing the appropriate secondary precondition on an operator, we can force that operator to preserve some condition or to cause some condition to become true. For example, in the briefcase example discussed earlier, the act of bringing the briefcase to work causes the dictionary to be brought to work as a side effect only if the dictionary happens to be in the briefcase at the time. Therefore, we can achieve the goal of having the dictionary at the office by requiring that the dictionary be in the briefcase when the briefcase is moved. Similarly, to prevent the paycheck from changing locations, we need only require that the paycheck not be in the briefcase at the time the briefcase is moved. To determine which secondary preconditions are appropriate in any given situation, we need to examine more closely the circumstances under which a condition is preserved or is made true by an operator.

For a condition φ to remain true between two points in a plan, all of the intervening operators must preserve the truth of φ; that is, if φ is true when each such operator is applied, then φ must be true afterward. In Section 2.3 we saw that φ is true after an operator a is applied if and only if

$a^{-1}(\varphi)$ was true just prior to the application. Therefore, a will preserve the truth of φ if and only if $\varphi \rightarrow a^{-1}(\varphi)$ is true when a is applied. Given that, in the final plan, φ will be true when a is applied, any formula $\mathbb{P}^a_\varphi$ such that $\varphi \rightarrow (\mathbb{P}^a_\varphi \leftrightarrow a^{-1}(\varphi))$ is an appropriate secondary precondition to impose on a to ensure that a will preserve φ in the final plan. This is justified by the following lemma:

Lemma 3.4. Let $\mathbb{P}^a_\varphi$ be a formula such that $\varphi \rightarrow (\mathbb{P}^a_\varphi \leftrightarrow a^{-1}(\varphi))$. Then the following holds: if φ is true before a is applied, φ will be true after a is applied if and only if $\mathbb{P}^a_\varphi$ is true beforehand.

Proof. By hypothesis, if φ is true before a is applied, then $\mathbb{P}^a_\varphi$ is true before a is applied if and only if $a^{-1}(\varphi)$ is true before a is applied. But φ will be true after a is applied if and only if $a^{-1}(\varphi)$ is true beforehand. Therefore, if φ is true before a is applied, then φ will be true after a is applied if and only if $\mathbb{P}^a_\varphi$ is true beforehand. □

Let us now consider the conditions that must hold for an operator to cause a formula to become true. Given that the initial state is known completely,[1] an operator a causes a closed formula φ to become true if and only if φ is false before a is applied and true afterward. In Section 2.3 it was shown that φ will be true after a is applied if and only if $a^{-1}(\varphi)$ is true beforehand. Therefore, a causes φ to become true if and only if $\neg\varphi \wedge a^{-1}(\varphi)$ is true when a is applied. Although we would guarantee φ to be true after a is applied and false beforehand if we were to impose $\neg\varphi \wedge a^{-1}(\varphi)$ as a secondary precondition for a, it is sufficient to impose a weaker precondition Σ^a_φ, where Σ^a_φ is any formula such that $\neg\varphi \wedge a^{-1}(\varphi) \rightarrow \Sigma^a_\varphi$ and $\Sigma^a_\varphi \rightarrow a^{-1}(\varphi)$. Σ^a_φ has the property that, if φ is false when a is applied, then a will cause φ to become true if and only if Σ^a_φ is true when a is applied; if both φ and Σ^a_φ are true when a is applied, then a will preserve the truth of φ. Imposing Σ^a_φ as a secondary precondition therefore guarantees that φ will become true if it is false. Σ^a_φ is weaker than $\neg\varphi \wedge a^{-1}(\varphi)$, as it allows the possibility that φ will be true when a is applied. The

1 If the initial state were not completely known, it would be possible for φ to be false before a is applied for some of the worlds satisfying the initial state description and true for others. Therefore, the modifications described here apply only when the initial state is known completely. When the initial state is only partially known, $a^{-1}(\varphi)$ must be asserted as the secondary precondition.

reason we would want to impose Σ^a_φ instead of $\neg\varphi \wedge a^{-1}(\varphi)$ is that we can often find a formula Σ^a_φ that is much simpler than $\neg\varphi \wedge a^{-1}(\varphi)$ and, consequently, is easier to deal with. The justification for using Σ^a_φ instead of $\neg\varphi \wedge a^{-1}(\varphi)$ is provided by the following theorem, which is analogous to Theorem 3.2 except that it is stated in terms of Σ^a_φ and $\mathbb{P}^a_\varphi$.

Theorem 3.5. Let φ be a formula not containing free variables, let Γ be a complete description of the initial state of the world, and let θ be a sequence of operators. Then $\Gamma\{\theta\}\varphi$ holds if and only if one of the following holds:

(1) There exists a prefix σa of θ, where a is an operator, such that $\Gamma\{\sigma\}\Sigma^a_\varphi$ holds and $\Gamma\{\sigma a\gamma\}\mathbb{P}^b_\varphi$ holds for all sequences γ and all operators b such that $\sigma a\gamma b$ is a prefix of θ.

(2) φ is a theorem of Γ and $\Gamma\{\sigma\}\mathbb{P}^a_\varphi$ holds for every prefix σa of θ.

Proof. If Clause (1) holds, then $\Gamma\{\sigma a\}\varphi$ holds for an appropriate prefix σa of θ. Furthermore, by induction and by using Lemma 3.4, $\Gamma\{\sigma a\gamma b\}\varphi$ must hold for all sequences γ and all operators b such that $\sigma a\gamma b$ is a prefix of θ. Therefore, (1) implies $\Gamma\{\theta\}\varphi$. If Clause (2) holds, then, by induction, $\Gamma\{\sigma\}\varphi$ holds for every prefix σ of θ. Therefore, (2) implies $\Gamma\{\theta\}\varphi$. Hence, if either Clause (1) or Clause (2) holds, or if both hold, then $\Gamma\{\theta\}\varphi$ must hold as well.

For the converse, suppose that $\Gamma\{\theta\}\varphi$ holds. Then, by Theorem 3.2, one of the following holds:

(i) There exists a prefix σa of θ, where a is an operator, such that $\Gamma\{\sigma\}\varphi$ is false but $\Gamma\{\sigma a\gamma\}\varphi$ is true for all sequences γ such that $\sigma a\gamma$ is a prefix of θ.

(ii) $\Gamma\{\sigma\}\varphi$ for every prefix σ of θ.

Suppose that (i) holds for a suitable prefix σa of θ. Then $\Gamma\{\sigma\}\varphi$ is false and $\Gamma\{\sigma a\}\varphi$ is true. But $\Gamma\{\sigma a\}\varphi$ is true if and only if $\Gamma\{\sigma\}a^{-1}(\varphi)$ is true. Furthermore, since Γ is a complete description of the initial state of the world and since φ does not contain free variables, $\Gamma\{\sigma\}\varphi$ is false if and only if $\Gamma\{\sigma\}\neg\varphi$ is true. Therefore, $\Gamma\{\sigma\}(\neg\varphi \wedge a^{-1}(\varphi))$ holds. Hence, $\Gamma\{\sigma\}\Sigma^a_\varphi$ holds. Furthermore, (i)

implies that both $\Gamma\{\sigma a\tau\}\varphi$ and $\Gamma\{\sigma a\tau b\}\varphi$ hold for all sequences τ and all operators b such that $\sigma a\tau b$ is a prefix of θ. Therefore, $\Gamma\{\sigma a\tau\}(\varphi \wedge a^{-1}(\varphi))$ holds for all τ and b such that $\sigma a\tau b$ is a prefix of θ. Hence, $\Gamma\{\sigma a\tau\}\mathbb{P}^b_\varphi$ holds for all τ and b such that $\sigma a\tau b$ is a prefix of θ. Hence, Clause (1) of the theorem holds.

Suppose that (ii) holds. Then φ is a theorem of Γ, since $\Gamma\{\}\varphi$ holds. Furthermore, both $\Gamma\{\sigma\}\varphi$ and $\Gamma\{\sigma a\}\varphi$ hold for all σ and a such that σa is a prefix of θ. Therefore, $\Gamma\{\sigma\}\mathbb{P}^a_\varphi$ holds for all σ and a such that σa is a prefix of θ. Hence, Clause (2) of the theorem holds.

These two cases give us the following: if $\Gamma\{\theta\}\varphi$ holds, then either Clause (1) holds or Clause (2) holds or both. But, from before, if (1) or (2) hold, $\Gamma\{\theta\}\varphi$ must hold as well. Therefore, $\Gamma\{\theta\}\varphi$ holds if and only if either (1) holds or (2) holds or both. □

As with Theorem 3.2, Clause (1) of Theorem 3.5 can be divided into two subcases: one in which the operator that makes φ true if it is false must be added to the current partial plan; the other in which the operator that makes φ true if it is false already appears in the plan. Thus, we have the following corollary to Theorem 3.5:

Corollary 3.6. Let θ be a sequence of operators and θ' an expansion of θ. That is, for an appropriate set of operator sequences $\{\beta_1, \beta_2, \ldots\}$, if $\theta = a_1 a_2 \cdots a_n$, then $\theta' = \beta_1 a_1 \beta_2 a_2 \cdots \beta_n a_n \beta_{n+1}$, and, if $\theta = \epsilon$, then $\theta' = \beta_1$. Let $\sigma_1 = \beta_1$ and $\sigma_i = \beta_1 a_1 \cdots \beta_{i-1} a_{i-1} \beta_i$ for $i > 1$. Let Γ be a complete description of the initial state and let φ be a formula containing no free variables. Then $\Gamma\{\theta'\}\varphi$ holds if and only if one of the following is true

(1) There exists a σ_i such that $\Gamma\{\sigma_i\}\Sigma^{a_i}_\varphi$ holds and $\Gamma\{\sigma_i a_i \gamma\}\mathbb{P}^b_\varphi$ holds for all sequences γ and all operators b such that $\sigma_i a_i \gamma b$ is a prefix of θ'.

(2) There exists a prefix σa of θ such that σ is not a σ_i, $\Gamma\{\sigma\}\Sigma^a_\varphi$ holds, and $\Gamma\{\sigma a\gamma\}\mathbb{P}^b_\varphi$ holds for all sequences γ and all operators b such that $\sigma a\gamma b$ is a prefix of θ.

(3) φ is a theorem of Γ and $\Gamma\{\sigma\}\mathbb{P}^a_\varphi$ holds for every prefix σa of θ.

Proof. The above theorem follows directly from Theorem 3.5, as Clauses (1) and (2) together are equivalent to Clause (1) of Theorem 3.5, and Clause (3) is equivalent to Clause (2) of Theorem 3.5. □

Let us now consider the precise modifications that must be made in the plan graph, the agenda, and the protection set to protect φ from the initial state, to force an existing operator to make φ true if it is false, and to insert a new operator that makes φ true if it is false.

According to Clause (3) of Corollary 3.6, if φ is to remain true from the initial state to vertex v in the final plan, then φ must be true in the initial state, and $\mathbb{P}^a_\varphi$ must be true when operator a is applied for every operator a prior to vertex v. Therefore, the following modifications must be made to protect φ from the initial state to vertex v:

(1) $\langle \varphi, v_\Gamma \rangle$ must be added to the agenda to require that φ be true in the initial state.

(2) $\langle \mathbb{P}^a_{\varphi'}, v' \rangle$ must be added to the protection set for each vertex v' such that $v_\Gamma \prec v' \prec v$ to guarantee that every operator preceding vertex v will preserve the truth of φ.

(3) $\langle \varphi, v_\Gamma, v \rangle$ must be added to the protection set to assert that φ is protected from the initial state to vertex v.

If any of the foregoing additions contradict existing goals and preconditions, no amount of further modification will lead to a solution. This is because it is impossible to simultaneously achieve contradictory goals and protections. Therefore, we can rule out the option of protecting φ from the initial state to vertex v if φ is not true in the initial state or if requiring that the intervening operators preserve the truth of φ contradicts existing goals and protections. Stated more precisely, φ cannot be protected from the initial state to vertex v if

(1) $\{\varphi\} \cup \Gamma$ is inconsistent, or

(2) $\{\varphi, \mathbb{P}^a_{\varphi'}\} \cup g_{v'} \cup \rho_{v'}$ is inconsistent for any vertex v' such that $v_\Gamma \prec v' \prec v$,

where $\rho_{v'}$ is the current set of protected conditions that must be true at v', as defined in Equation 3.1, and $g_{v'}$ is the set of goals currently on the agenda that must be achieved at vertex v', as

defined in Equation 3.2. Note that there is no danger in not detecting these inconsistencies if they are present, as it is impossible to obtain a plan that satisfies inconsistent goals and protections. The only reason for the test is to reduce the search space by eliminating impossible solution paths from consideration. This is fortunate, as the test itself is only partially decidable; that is, while it is always possible to detect inconsistencies if they are present, it is not generally possible to confirm their absence if they are not present. Consequently, detecting an inconsistency requires an unbounded amount of computation. Since the only reason for the test is to prune the search space, spending too much time on it can be worse than not performing the test at all. The compromise is to balance amount of computation spent eliminating alternatives against the amount of computation saved in searching a smaller space, in effect limiting the range of inconsistencies that can be detected. The optimum balance, though, is highly dependent on the problem being considered, so it is hard to make any general statements about where the optimum lies.

Let us next consider how to force an existing operator in the plan to cause φ to become true if it is false, and how then to protect φ up to vertex v. If the existing operator $a_{v'}$ is associated with vertex v', then, by Clause (1) of Corollary 3.6, $\Sigma^a_{\varphi^{v'}}$ must be true when $a_{v'}$ is applied, and $\mathbb{P}^a_\varphi$ must be true when operator a is applied for every operator between v' and v. Therefore, the following modifications have to be made to force the operator associated with vertex v' to guarantee that φ will be true and to protect φ up to vertex v:

(1) $\langle \Sigma^a_{\varphi^{v'}}, v' \rangle$ must be added to the agenda guarantee that φ will be true after applying $a_{v'}$.

(2) $\langle \mathbb{P}^a_{\varphi^{v''}}, v'' \rangle$ must be added to the agenda for each vertex v'' such that $v' \prec v'' \prec v$ to guarantee that every operator between v' and v will preserve the truth of φ.

(3) $\langle \varphi, v', v \rangle$ must be added to the protection set to assert that φ is protected between vertices v' and v.

If these new goals and protections contradict their existing counterparts, it will be impossible to obtain a solution if the modifications are made. Therefore, we can rule out the possibility of forcing $a_{v'}$ to make φ true if it is false, and then protecting φ up to vertex v, if

(1) $\{\neg\varphi, \Sigma^{a_{v'}}_{\varphi}\} \cup g_{v'} \cup \rho_{v'}$ is inconsistent, or

(2) $\{\varphi, \mathbb{P}^{a_{v''}}_{\varphi}\} \cup \rho_{v''} \cup g_{v''}$ is inconsistent for any v'' such that $v' \prec v'' \prec v$.

As before, these conditions are only partially decidable, so we must balance the amount of computation spent pruning the search space against the amount saved in searching a smaller space.

The third and final way of modifying a partial plan to achieve a goal is to insert a new operator that causes the goal to become true if it is false and then to protect the goal up to the point we wish it to be true. Since we are considering only linear plan graphs, the insertion must preserve linearity. Therefore, the new operator must be inserted between two consecutive vertices v_1 and v_2 in the current plan graph (i.e., there must be an edge from v_1 to v_2 in the current plan graph). This is done by creating a new vertex v', removing the edge from v_1 to v_2, and adding two new edges, one from v_1 to v' and the other from v' to v_2. The new operator $a_{v'}$ is then associated with v'. The modifications of the agenda and the protection set are then very much like those for forcing an existing operator to make φ true if it is false. As with forcing, we must guarantee that $a_{v'}$ will cause φ to become true if it is false and that all of the operators between vertices v' and v will preserve φ, and we must assert that φ is protected between v' and v. However, we must also guarantee that the preconditions of the new operator $a_{v'}$ will be true when the operator is applied, and we must guarantee that $a_{v'}$ will preserve all of the conditions protected between vertices v_1 and v_2. The former is accomplished by adding the preconditions of $a_{v'}$ to the agenda. For the latter, the set of conditions protected between v_1 and v_2 is given by

$$\begin{aligned}&\{\psi \mid \langle\psi, v_3, v_4\rangle \in P \text{ and } v_3 \preceq v_1 \text{ and } v_2 \preceq v_4\}\\ &\quad = \{\psi \mid \langle\psi, v_3, v_4\rangle \in P \text{ and } v_3 \prec v_2 \preceq v_4\}\\ &\quad = \rho_{v_2}\end{aligned}$$

Therefore, $a_{v'}$ must preserve every formula in ρ_{v_2}. Thus, the complete set of modifications of the agenda and the protection set are as follows: if operator $a_{v'}$ is being inserted at a new vertex v' between vertices v_1 and v_2 so that φ will be true at vertex v, then

(1) $\langle\pi_i, v'\rangle$ must be added to the agenda for every π_i in the set $\pi^{a_{v'}}$ of preconditions of $a_{v'}$ to guarantee that the preconditions will be satisfied when $a_{v'}$ is applied.

(2) $\langle \Sigma^{a_{v'}}_{\varphi}, v' \rangle$ must be added to the agenda guarantee that φ will be true after $a_{v'}$ is applied.

(3) $\langle \mathbb{P}^{a_{v'}}_{\psi}, v' \rangle$ must be added to the agenda for every formula $\psi \in \rho_{v_2}$ to ensure that $a_{v'}$ will preserve all conditions protected between vertices v_1 and v_2.

(4) $\langle \mathbb{P}^{a_{v''}}_{\varphi}, v'' \rangle$ must be added to the agenda for each vertex v'' such that $v_2 \preceq v'' \prec v$ to guarantee that every operator between the new vertex v' and vertex v will preserve the truth of φ.

(5) $\langle \varphi, v', v \rangle$ must be added to the protection set to assert that φ is protected between vertices v' and v.

If these new goals and protections contradict their existing counterparts, it will be impossible to obtain a solution if the modifications are made. Therefore, we can rule out the possibility of inserting $a_{v'}$ between vertices v_1 and v_2 if

(1) $\{\mathbb{P}^{a_{v'}}_{\psi} \mid \psi \in \rho_{v_2}\} \cup \{\neg\, \varphi, \Sigma^{a_{v'}}_{\varphi}\} \cup \pi^{a_{v'}} \cup \rho_{v_2}$ is inconsistent, or

(2) $\{\varphi, \mathbb{P}^{a_{v''}}_{\varphi}\} \cup \rho_{v''} \cup g_{v''}$ is inconsistent for any v'' such that $v_2 \preceq v'' \prec v$.

As before, these conditions are only partially decidable, so we must balance the amount of computation spent pruning the search space against the amount saved in searching a smaller space.

The Rules

While the three ways of modifying a partial plan, as described above, are valid for any formula φ, we will perform these modifications only for formulas of the forms $R(t_1, \ldots, t_n)$ and $\neg\, R(t_1, \ldots, t_n)$, where R is a relation symbol and the t_i's are ground terms (i.e., terms without variable symbols). Goals containing connectives and/or quantifiers will be decomposed into simpler formulas and, ultimately, into one of the two forms above. The reason for doing this is that it leads to a more efficient planning technique, primarily because it is easier to identify operators that cause atomic formulas to become true or false than it is to identify the appropriate operators for arbitrary formulas.

To construct modification rules for atomic formulas and their negations, we can take advantage of our earlier assumption that we will be dealing only with problems in which constant and function symbols cannot change interpretation. This assumption gives rise to the following secondary preconditions for atomic formulas and their negations:

$$\begin{aligned}
\mathbb{P}^a_{R(t_1,\ldots,t_n)} &\equiv \neg\, \delta^a_R(t_1,\ldots,t_n) \\
\mathbb{P}^a_{\neg R(t_1,\ldots,t_n)} &\equiv \neg\, \alpha^a_R(t_1,\ldots,t_n) \\
\Sigma^a_{R(t_1,\ldots,t_n)} &\equiv \alpha^a_R(t_1,\ldots,t_n) \\
\Sigma^a_{\neg R(t_1,\ldots,t_n)} &\equiv \delta^a_R(t_1,\ldots,t_n)
\end{aligned}$$

In other words, a preserves the truth of $R(t_1,\ldots,t_n)$ if and only if a does not delete $\langle t_1,\ldots,t_n\rangle$ from the interpretation of R, a preserves the truth of $\neg R(t_1,\ldots,t_n)$ if and only if a does not add $\langle t_1,\ldots,t_n\rangle$ to the interpretation of R, a causes $R(t_1,\ldots,t_n)$ to become true if it is false if and only if a adds $\langle t_1,\ldots,t_n\rangle$ to the interpretation of R, and a causes $\neg R(t_1,\ldots,t_n)$ to become true if it is false if and only if a deletes $\langle t_1,\ldots,t_n\rangle$ from the interpretation of R.

In stating the modification rules for $R(t_1,\ldots,t_n)$ and $\neg R(t_1,\ldots,t_n)$, we will treat the case in which R is the symbol '=' separately from the general case. Since we have assumed that no operator can change the interpretation of any constant symbol or function symbol, and since by definition no operator can change the interpretation of '=', $\alpha^a_{=}(t_1,t_2)$ and $\delta^a_{=}(t_1,t_2)$ are both *FALSE*. Therefore, it is impossible to make a goal of the form $t_1 = t_2$ or $t_1 \neq t_2$ true if it is not already true. This gives us the following rule:

Rule 1. If $\langle t_1 = t_2, v\rangle$ or $\langle t_1 \neq t_2, v\rangle$ is an unsatisfied goal on the agenda, no further modification of the current partial plan will lead to a solution. Therefore, a different solution path must be considered.

This rule tells us to abandon the current branch in the search space if a goal of the form $t_1 = t_2$ or $t_1 \neq t_2$ is found to be unsatisfied. If all branches are found to lead to dead ends, a solution does not exist.

When R is not the symbol '=', the following two rules apply. Each of the modifications described in these rules is carried out as discussed previously.

Rule 2. If $\langle R(t_1, \ldots, t_n), v\rangle$ is an unsatisfied goal on the agenda and R is not the symbol '=', remove $\langle R(t_1, \ldots, t_n), v\rangle$ from the agenda and effect one of the following modifications:

(1) Protect $R(t_1, \ldots, t_n)$ from the initial state to vertex v, if $R(t_1, \ldots, t_n)$ is true in the initial state and protecting $R(t_1, \ldots, t_n)$ does not contradict existing goals and protections.

(2) For some vertex v' such that $v_\Gamma \prec v' \prec v$, force the operator associated with vertex v' to cause $R(t_1, \ldots, t_n)$ to become true if it is false, and then protect $R(t_1, \ldots, t_n)$ up to vertex v, provided neither modification introduces an inconsistency.

(3) Insert a new operator that causes $R(t_1, \ldots, t_n)$ to become true if it is false, at a point preceding vertex v that does not contradict existing goals and protections, and then protect $R(t_1, \ldots, t_n)$ up to vertex v.

Rule 3. If $\langle \neg R(t_1, \ldots, t_n), v\rangle$ is an unsatisfied goal on the agenda and R is not the symbol '=', remove $\langle \neg R(t_1, \ldots, t_n), v\rangle$ from the agenda and effect one of the following modifications:

(1) Protect $\neg R(t_1, \ldots, t_n)$ from the initial state to vertex v, if $\neg R(t_1, \ldots, t_n)$ is true in the initial state and protecting $\neg R(t_1, \ldots, t_n)$ does not contradict existing goals and protections.

(2) For some vertex v' such that $v_\Gamma \prec v' \prec v$, force the operator associated with vertex v' to cause $\neg R(t_1, \ldots, t_n)$ to become true if it is false, and then protect $\neg R(t_1, \ldots, t_n)$ up to vertex v, provided neither modification introduces an inconsistency.

(3) Insert a new operator that causes $\neg R(t_1, \ldots, t_n)$ to become true if it is false, at a point preceding vertex v that does not contradict existing goals and protections, and then protect $\neg R(t_1, \ldots, t_n)$ up to vertex v.

Note that it is possible for a situation to arise in which none of the modifications described in Rule 2 is consistent with the existing goals and protections. When this happens, no further modification of the partial plan will lead to a solution, since, in virtue of Corollary 3.6, $R(t_1, \ldots, t_n)$

can be achieved *if and only if* one of the modifications described in Rule 2 is consistent with existing goals and preconditions. Therefore, when such an inconsistency is detected, we must abandon the current partial plan and try an alternative solution path. This also applies to Rule 3.

Note also that Rules 2 and 3 call for the unsatisfied goal to be removed from the agenda. This is permitted, as the goal will be satisfied in the final plan if the new assertions are satisfied. The rules that follow also call for the removal of unsatisfied goals, for precisely the same reason, once the appropriate modifications have been made.

The remaining rules are used to decompose complex formulas into simpler ones. To decompose a goal of the form $\varphi \wedge \psi$, we make use of the fact that $\varphi \wedge \psi$ is true at some point in a plan if and only if φ and ψ are both true at that point. This leads to the following rule for conjunctive goals:

Rule 4. If $\langle \varphi \wedge \psi, v \rangle$ is a goal on the agenda, remove $\langle \varphi \wedge \psi, v \rangle$ from the agenda and insert $\langle \varphi, v \rangle$ and $\langle \psi, v \rangle$.

It is recommended that this rule be applied regardless of whether $\varphi \wedge \psi$ is satisfied or not, as φ and ψ may then be considered separately at later stages in the synthesis process.

For disjunctive goals, we can make use of the assumption that the initial state is completely known. As a result, $\varphi \vee \psi$ is true at some point in a plan if and only if either φ or ψ is true at that point, or both are (note that this does not necessarily hold when the initial state is not completely known as $\dot{\Gamma}\{\theta\}(\varphi \vee \psi)$ can be true without either $\Gamma\{\theta\}\varphi$ or $\Gamma\{\theta\}\psi$ being true). This gives us our fifth rule:

Rule 5. If $\langle \varphi \vee \psi, v \rangle$ is an unsatisfied goal on the agenda, remove $\langle \varphi \vee \psi, v \rangle$ from the agenda and insert either $\langle \varphi, v \rangle$ or $\langle \psi, v \rangle$.

The rule for goals involving implication is merely a special case of the preceding rule, since $\varphi \rightarrow \psi$ is equivalent to $\neg \varphi \vee \psi$.

Rule 6. If $\langle \varphi \rightarrow \psi, v \rangle$ is an unsatisfied goal on the agenda, remove $\langle \varphi \rightarrow \psi, v \rangle$ from the agenda and insert either $\langle \neg \varphi, v \rangle$ or $\langle \psi, v \rangle$.

The rule for goals involving the equivalence connective is obtained from Rules 4 and 5 by making use of the fact that $\varphi \leftrightarrow \psi$ is equivalent to $(\varphi \wedge \psi) \vee (\neg \varphi \wedge \neg \psi)$.

Rule 7. If $\langle \varphi \leftrightarrow \psi, v \rangle$ is an unsatisfied goal on the agenda then remove $\langle \varphi \leftrightarrow \psi, v \rangle$ from the agenda and insert either $\langle \varphi, v \rangle$ and $\langle \psi, v \rangle$ or $\langle \neg \varphi, v \rangle$ and $\langle \neg \psi, v \rangle$.

For quantified goals, we can make use of the assumption that every object in the world has a standard name (i.e., there is a constant symbol denoting that object at every point in a plan). Because of this assumption, if $\{e_1, \ldots, e_n\}$ is the set of standard names of all objects, then $\forall x\, \varphi(x)$ is true if and only if $\varphi(e_i)$ is true for all $e_i \in \{e_1, \ldots, e_n\}$, and $\exists x\, \varphi(x)$ is true if and only if $\varphi(e_i)$ is true for some $e_i \in \{e_1, \ldots, e_n\}$. Unsatisfied goals of the form $\forall x\, \varphi(x)$ are handled by separating the cases for which $\varphi(e_i)$ is false from those for which $\varphi(e_i)$ is true in a manner that permits each false case to be considered individually:

Rule 8. If $\langle \forall x\, \varphi(x), v \rangle$ is an unsatisfied goal on the agenda and $\varphi(e_i)$ is false at vertex v for each standard name $e_i \in \{e_{i_1}, \ldots, e_{i_m}\}$, then remove $\langle \forall x\, \varphi(x), v \rangle$ from the agenda and insert $\langle \varphi(e_{i_1}), v \rangle, \ldots, \langle \varphi(e_{i_m}), v \rangle$, and $\langle \forall x\, (x = e_{i_1} \vee \cdots \vee x = e_{i_m} \vee \varphi(x)), v \rangle$.

An example of the use of Rule 8 may be found in the block-stacking example appearing in Section 3.1. In that example, no block may be on top of block A when $\mathrm{Put}(A, B)$ is performed. However, this requirement is not satisfied, given the plan of placing B on top of C and then A on top of B. This is because C is on top of A both in the initial state and after $\mathrm{Put}(B, C)$. Therefore, we would use Rule 8 to decompose $\forall z\, \neg\, \mathrm{On}(z, A)$ into $\neg\, \mathrm{On}(C, A)$ and $\forall z\, (z = C \vee \neg\, \mathrm{On}(z, A))$. The subgoal $\neg\, \mathrm{On}(C, A)$ would then be achieved by inserting a new operator $\mathrm{Put}(C, X)$ and protecting $\neg\, \mathrm{On}(C, A)$ in the interval between the $\mathrm{Put}(C, X)$ and the $\mathrm{Put}(A, B)$ operators. The subgoal $\forall z\, (z = C \vee \neg\, \mathrm{On}(z, A))$ is serendipitously true and no further action need be taken to achieve it.

For an unsatisfied goal of the form $\exists x\, \varphi(x)$, we must make $\varphi(e_i)$ true for some standard name e_i. From the standpoint of minimizing the search space, it would be preferable to defer the choice of e_i by introducing a variable as a placeholder for the appropriate e_i and then instantiating

this variable at some later point in the synthesis process. The mechanisms needed to handle instantiation variables, however, are beyond the scope of this report and are covered in my thesis [8]. To keep our planning technique simple, we will explicitly consider each and every choice for e_i.

Rule 9. If $\langle \exists x\, \varphi(x), v \rangle$ is an unsatisfied goal on the agenda and $\{e_1, \ldots, e_n\}$ is the set of standard names of all objects in the world, then remove $\langle \exists x\, \varphi(x), v \rangle$ from the agenda and insert $\langle \varphi(e_i), v \rangle$ for some $e_i \in \{e_1, \ldots, e_n\}$.

The remaining rules deal with negated goals. They are obtained from the previous rules in an obvious fashion by making use of De Morgan's laws and similar theorems of first-order logic.

Rule 10. If $\langle \neg(\varphi \wedge \psi), v \rangle$ is an unsatisfied goal on the agenda, remove $\langle \neg(\varphi \wedge \psi), v \rangle$ from the agenda and insert either $\langle \neg\, \varphi, v \rangle$ or $\langle \neg\, \psi, v \rangle$.

Rule 11. If $\langle \neg(\varphi \vee \psi), v \rangle$ is a goal on the agenda (be it satisfied or not), remove $\langle \neg(\varphi \vee \psi), v \rangle$ from the agenda and insert $\langle \neg\, \varphi, v \rangle$ and $\langle \neg\, \psi, v \rangle$.

Rule 12. If $\langle \neg(\varphi \rightarrow \psi), v \rangle$ is a goal on the agenda (be it satisfied or not), remove $\langle \neg(\varphi \rightarrow \psi), v \rangle$ from the agenda and insert $\langle \varphi, v \rangle$ and $\langle \neg\, \psi, v \rangle$.

Rule 13. If $\langle \neg(\varphi \leftrightarrow \psi), v \rangle$ is an unsatisfied goal on the agenda, remove $\langle \neg(\varphi \leftrightarrow \psi), v \rangle$ from the agenda and insert either $\langle \neg\, \varphi, v \rangle$ and $\langle \psi, v \rangle$ or $\langle \varphi, v \rangle$ and $\langle \neg\, \psi, v \rangle$.

Rule 14. If $\langle \neg(\forall x\, \varphi(x)), v \rangle$ is an unsatisfied goal on the agenda and if $\{e_1, \ldots, e_n\}$ is the set of standard names of all objects in the world, remove $\langle \neg(\forall x\, \varphi(x)), v \rangle$ from the agenda and insert $\langle \neg\, \varphi(e_i), v \rangle$ for some i, $1 \leq i \leq n$.

Rule 15. If $\langle \neg(\exists x\, \varphi(x)), v \rangle$ is an unsatisfied goal on the agenda and $\neg\, \varphi(e_i)$ is false at vertex v for each standard name $e_i \in \{e_{i_1}, \ldots, e_{i_m}\}$, remove $\langle \neg(\exists x\, \varphi(x)), v \rangle$ from the agenda and insert $\langle \neg\, \varphi(e_{i_1}), v \rangle, \ldots, \langle \neg\, \varphi(e_{i_m}), v \rangle$, and $\langle \forall x (x = e_{i_1} \vee \cdots \vee x = e_{i_m} \vee \neg\, \varphi(x)), v \rangle$.

3.3 AN EXAMPLE

To illustrate how a plan would be synthesized by applying the rules just introduced, let us formulate and solve the briefcase problem discussed earlier. The reader will recall that there are three objects, a briefcase, a dictionary and a paycheck, and two locations, the home and the office. Each object is at one of the two locations; furthermore, the dictionary and the paycheck may be in or out of the briefcase. In our formulation, we will have five constant symbols, B, D, P, H and O, corresponding, respectively, to the briefcase, dictionary, paycheck, home, and office. We will also have two relation symbols, 'At' and 'In'. 'At' is a binary relation such that $\mathrm{At}(x, y)$ is true if and only if object x is at location y, and 'In' is a unary relation such that $\mathrm{In}(x)$ is true if and only if object x is in the briefcase. Initially, the three objects are at home; the paycheck is in the briefcase but the dictionary is not. Therefore, the initial state description Γ contains the following formulas:

(1) $e_1 \neq e_2$ for all $e_1, e_2 \in \{B, D, P, H, O\}$ such that e_1 and e_2 are distinct

(2) $\forall x (x = B \vee x = D \vee x = P \vee x = H \vee x = O)$

(3) $\forall x\, y\, (\mathrm{At}(x, y) \leftrightarrow [(x = B \vee x = D \vee x = P) \wedge y = H])$

(4) $\forall x (\mathrm{In}(x) \leftrightarrow x = P)$.

The formulas defined in Item (1) assert that B, D, P, H and O represent distinct entities, and the formula in Item (2) asserts that these are the only entities in existence. Formula (3) asserts that the only entities that have locations are B, D and P, and they are all at H. Finally, (4) asserts that the only entity in the briefcase is P.

Our objective is to have the briefcase and the dictionary at the office, and the paycheck at home. Therefore, the goal description consists of the formulas $\mathrm{At}(B, O)$, $\mathrm{At}(D, O)$ and $\mathrm{At}(P, H)$. To achieve these goals, we may put objects into the briefcase, remove objects from the briefcase, and move the briefcase between the two locations. We will therefore have three operator schemata, $\mathrm{PutIn}(z)$, $\mathrm{TakeOut}(z)$ and $\mathrm{MovB}(l)$, corresponding to the three allowable actions. These schemata

are defined as follows:

PutIn(z)

PRECOND: $\exists x(\text{At}(z,x) \wedge \text{At}(B,x))$

ADD: In(p) for all p such that $p = z$

TakeOut(z)

PRECOND: $\exists x(\text{At}(z,x) \wedge \text{At}(B,x))$

DELETE: In(p) for all p such that $p = z$

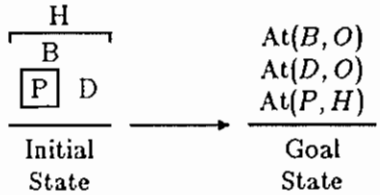
MovB(l)

ADD: At(p,q) for all p,q such that $q = l \wedge (p = B \vee \text{In}(p))$

DELETE: At(p,q) for all p,q such that $q \neq l \wedge (p = B \vee \text{In}(q))$

PutIn(z) causes In(z) to become true and requires as a precondition that z and B be at the same location. TakeOut(z) causes In(z) to become false and also requires as a precondition that z and B be at the same location. MovB(l) causes the briefcase and everything in it to be moved to location l. Unlike PutIn(z) and TakeOut(z), MovB(l) has no precondition and may be applied in any state. If the briefcase and its contents are already at location l, MovB(l) has no effect.

The initial partial plan is illustrated below. The initial state is depicted graphically and the goals are simply listed above the goal vertex. In general, goals on the agenda will be listed above the appropriate vertices and entries in the protection set will be indicated by labeling the appropriate edges.



In this partial plan, At(P,H) is satisfied but At(B,O) and At(D,O) are not; in other words, $\Gamma\{\}$At(P,H) holds but $\Gamma\{\}$At(B,O) and $\Gamma\{\}$At(D,O) do not. Rule 2 can therefore be applied to

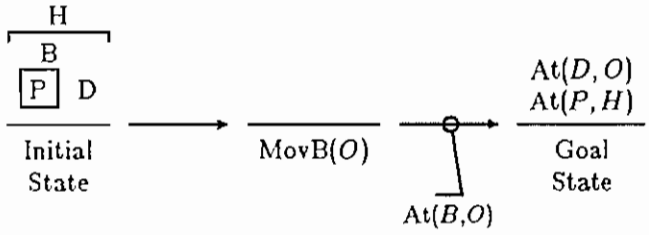
either At(B,O) or At(D,O). It does not matter which we choose to work on first; therefore, let us arbitrarily choose At(B,O). Since At(B,O) is false in the initial state and we are starting with the empty plan, we can rule out protecting At(B,O) from the initial state or forcing an existing operator to make At(B,O) true. Therefore, we have no choice but to insert an operator to make At(B,O) true. $\Sigma^{\text{PutIn}(z)}_{\text{At}(p,q)}$ and $\Sigma^{\text{TakeOut}(z)}_{\text{At}(p,q)}$ are both *FALSE* since both $\alpha^{\text{PutIn}(z)}_{\text{At}(p,q)}$ and $\alpha^{\text{TakeOut}(z)}_{\text{At}(p,q)}$ are *FALSE*. Hence, neither PutIn(z) nor TakeOut(z) can make At(B,O) true. However,

$$\Sigma^{\text{MovB}(l)}_{\text{At}(p,q)} \equiv \alpha^{\text{MovB}(l)}_{\text{At}}(p,q) \equiv [q = l \wedge (p = B \vee \text{In}(p))].$$

Therefore,

$$\begin{aligned}\Sigma^{\text{MovB}(l)}_{\text{At}(B,O)} &\equiv [O = l \wedge (B = B \vee \text{In}(B))] \\ &\equiv (O = l).\end{aligned}$$

Hence, the only operator that can make At(B,O) true is MovB(O). Inserting MovB(O) into the plan produces the following plan:



Note that no additions were made to the agenda, since the precondition for MovB(O) is *TRUE* and $\Sigma^{\text{MovB}(O)}_{\text{At}(B,O)} \equiv (O = O) \equiv$ *TRUE* (*TRUE* is always true and thus need not he placed on the agenda). At(B,O), however, was removed from the agenda. This is reflected in the diagram by removing At(B,O) from the goal vertex.

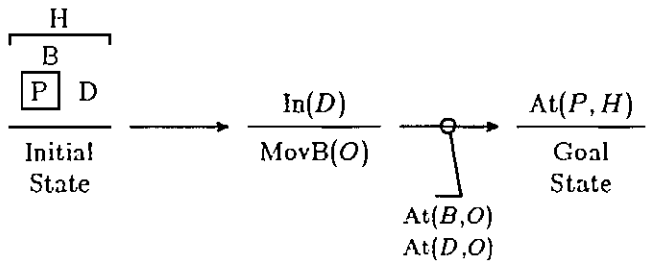
In the above plan, both At(D,O) and At(P,H) are unsatisfied. Choosing to work on At(D,O) first and applying Rule 2, we find that At(D,O) is not true in the initial state; consequently, we must either insert a new operator to make At(D,O) true or force an existing operator to make At(D,O) true. $\Sigma^{\text{PutIn}(z)}_{\text{At}(D,O)}$ and $\Sigma^{\text{TakeOut}(z)}_{\text{At}(D,O)}$ are both *FALSE*, but

$$\Sigma^{\text{MovB}(l)}_{\text{At}(D,O)} \equiv [O = l \wedge (D = B \vee \text{In}(D))].$$

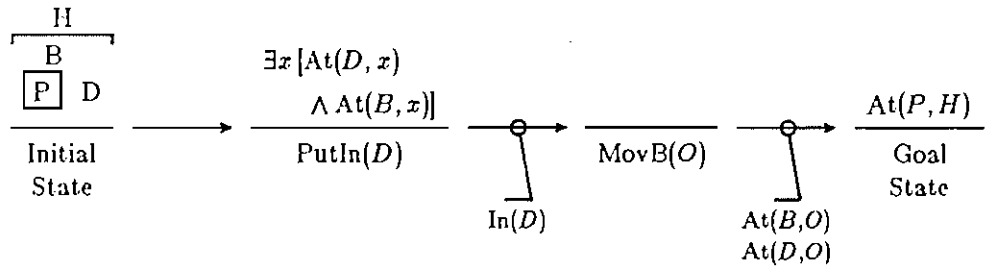
Since $D \neq B$ in the initial state and none of the operators changes the interpretations of either D or B, $\Sigma^{\mathrm{MovB}(l)}_{\mathrm{At}(D,O)}$ simplifies to

$$\Sigma^{\mathrm{MovB}(l)}_{\mathrm{At}(D,O)} \equiv (O = l \wedge \mathrm{In}(D)).$$

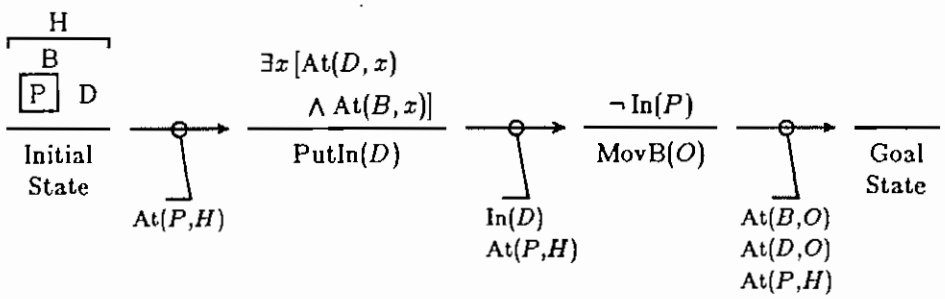
Hence, the only operator that can make $\mathrm{At}(D,O)$ true is $\mathrm{MovB}(O)$. Therefore, we must either insert a new $\mathrm{MovB}(O)$ operator or force the existing $\mathrm{MovB}(O)$ operator to cause $\mathrm{At}(D,O)$ to become true. Choosing to do the latter, and being prepared to backtrack if this does not work out, we obtain the following partial plan. Note that $\mathrm{In}(D)$ is added as a secondary precondition to $\mathrm{MovB}(O)$, since $\Sigma^{\mathrm{MovB}(O)}_{\mathrm{At}(D,O)}$ simplifies to $\mathrm{In}(D)$.



In this partial plan, both $\mathrm{In}(D)$ and $\mathrm{At}(P,H)$ are unsatisfied. Choosing to work on $\mathrm{In}(D)$ first and applying Rule 2, we find that $\mathrm{In}(D)$ is not true in the initial state and there are no operators preceding $\mathrm{MovB}(O)$ in the partial plan. Therefore, our only option is to insert a new operator to make $\mathrm{In}(D)$ true. $\Sigma^{\mathrm{TakeOut}(z)}_{\mathrm{In}(D)}$ and $\Sigma^{\mathrm{MovB}(l)}_{\mathrm{In}(D)}$ are both *FALSE*, but $\Sigma^{\mathrm{PutIn}(z)}_{\mathrm{In}(D)} \equiv (D = z)$. Therefore, $\mathrm{PutIn}(D)$ is the only operator that can make $\mathrm{In}(D)$ true. Inserting $\mathrm{PutIn}(D)$ gives us the following partial plan. Note that $\exists x\,(\mathrm{At}(D,x) \wedge \mathrm{At}(B,x))$ is placed on the agenda, as it is a precondition for $\mathrm{PutIn}(D)$.



In this plan, $\exists x\,(\mathrm{At}(D,x) \wedge \mathrm{At}(B,x))$ is satisfied but $\mathrm{At}(P,H)$ is not. Applying Rule 2 to $\mathrm{At}(P,H)$, we find that either we can protect $\mathrm{At}(P,H)$ from the initial state, since it is true initially, or we can insert a new operator $\mathrm{MovB}(H)$ with secondary preconditions $\mathrm{In}(P)$, since $\Sigma^{\mathrm{MovB}(l)}_{\mathrm{At}(P,H)} \equiv (H = l \wedge \mathrm{In}(P))$ and $\Sigma^{\mathrm{PutIn}(x)}_{\mathrm{At}(P,H)} \equiv \Sigma^{\mathrm{TakeOut}(x)}_{\mathrm{At}(P,H)} \equiv \mathit{FALSE}$. Choosing to do the former, and preparing to backtrack if necessary, we obtain the following partial plan:

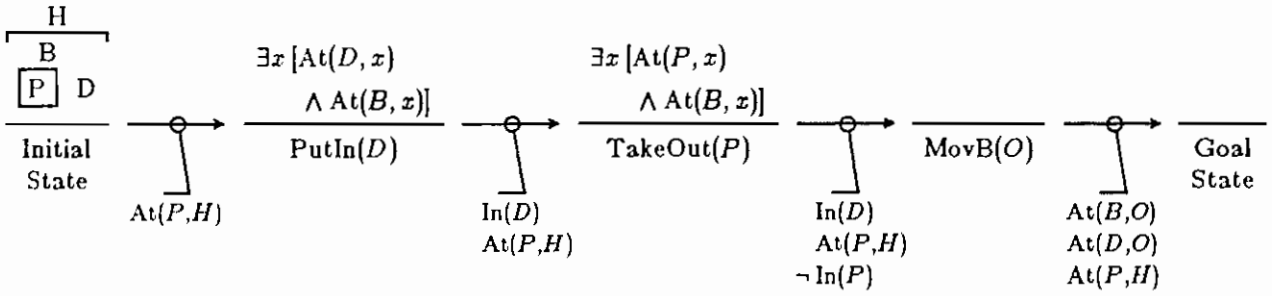


Note that, in protecting $\mathrm{At}(P,H)$ from the initial state, $\neg\,\mathrm{In}(P)$ is added as a secondary precondition for $\mathrm{MovB}(O)$, since

$$\begin{aligned}\mathbb{P}^{\mathrm{MovB}(O)}_{\mathrm{At}(P,H)} &\equiv \neg\,\delta^{\mathrm{MovB}(O)}_{\mathrm{At}}(P,H)\\ &\equiv \neg[H \neq O \wedge (P = B \vee \mathrm{In}(P))]\\ &\equiv \neg\,\mathrm{In}(P)\end{aligned}$$

No other preconditions are imposed on $\mathrm{PutIn}(D)$, since $\mathbb{P}^{\mathrm{PutIn}(D)}_{\mathrm{At}(P,H)} \equiv \mathit{TRUE}$.

At this point, only $\neg\,\mathrm{In}(P)$ is unsatisfied. Applying Rule 3, we find that our only option is to insert a new operator $\mathrm{TakeOut}(P)$ either before $\mathrm{PutIn}(D)$ or after $\mathrm{PutIn}(D)$. Choosing to do the latter, and preparing to backtrack if necessary, we obtain the following partial plan:



All outstanding goals on the agenda are now satisfied. Therefore, the plan just explicated, which consists of putting the dictionary into the briefcase, removing the paycheck from the

briefcase, and then bringing the briefcase to the office, satisfies all of our goals, preconditions, and protections.

Acknowledgments

I would like to thank my advisors, Bob Moore and Gio Wiederhold, for the many long discussions that helped me crystallize my ideas, and for creating an environment that made this research possible. I am especially indebted to Bob Moore for keeping me on track and for providing criticism when needed. Certainly, without Bob's influence, the nature of my work would have been quite different. I have also benefited from discussions with Alfred Aho, David Chapman, Peter Cheeseman, Tom Dean, Mike Georgeff, Amy Lansky, Vladimir Lifshitz, Mike Lowry, John Mohammed, Nils Nilsson, Stan Rosenschein, Marcel Schoppers, Yoav Shoham, Reid Simmons, Albert Visser, Richard Waldinger, and Dave Wilkins.

The research reported herein was supported by the Air Force Office of Scientific Research under Contract No. F49620-82-K-0031, by the Office of Naval Research under Contract Nos. N00014-80-C-0296 and N00014-85-C-0251, and through scholarships from the Natural Sciences and Engineering Research Council Canada and le Fonds F.C.A.C. pour l'aide et le soutien à la recherche, Quebec, Canada. The views and conclusions expressed in this document are those of the author and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the Office of Naval Research, the Natural Sciences and Engineering Research Council Canada, le Fonds F.C.A.C., the U.S. Government, the Quebec Government, or the Canadian Government.

References

[1] Boolos, G.S. and R.C. Jeffrey, *Computability and Logic*, 2nd edition (Cambridge University Press, Cambridge, England, 1980).

[2] Chapman, D., "Nonlinear Planning: A Logical Reconstruction," *Proc. IJCAI 9*, University of California at Los Angeles, Los Angeles, California, pp 1022–1024 (August 1985).

[3] Chapman, D., "Planning for Conjunctive Goals," Tech. Report AI TR–802, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts (forthcoming).

[4] Ernst, G.W. and A. Newell, *GPS: A Case Study in Generality and Problem Solving* (Academic Press, New York, New York, 1969).

[5] Fikes,. R.E. and N.J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," *Artificial Intelligence*, Vol 2, pp 189–208 (1971).

[6] Hayes-Roth, B., F. Hayes-Roth, S. Rosenschein and S. Cammarata, "Modeling Planning as an Incremental Opportunistic Process," *Proc. IJCAI 6*, Tokyo, Japan, pp 375–383 (August 1979).

[7] McCarthy, J. and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in *Machine Intelligence 4*, Meltzer, B. and D Michie eds., pp 463–502 (Edinburgh University Press, Edinburgb, Scotland, 1969).

[8] Pednault, E.P.D., *Toward a Mathematical Theory of Plan Synthesis*, Ph.D. thesis, Department of Electrical Engineering, Stanford University, Stanford, California (forthcoming).

[9] Rosenschein, S.J., "Plan Synthesis: A Logical Perspective," *Proc. IJCAI 7*, University of British Columbia, Vancouver, Canada, pp 331–337 (August 1981).

[10] Sacerdoti, E.D., "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, Vol. 5, No. 2, pp 115–135 (Summer 1974).

[11] Sacerdoti, E.D., *A Structure for Plans and Behavior* (Elsevier, New York, New York, 1977).

[12] Stefik, M., "Planning With Constraints (MOLGEN: part 1)," *Artificial Intelligence*, Vol. 16, No. 2, pp 111–140 (May 1981).

[13] Suppes, P., *Axiomatic Set Theory* (Dover, New York, New York, 1972).

[14] Sussman, G.J., "A Computational Model of Skill Acquisition," Tech. Report AI TR-197, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, Massachusetts (August 1973).

[15] Tate, A., "Project Planning Using a Hierarchic Non-Linear Planner," Research Report No. 25, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, (1976).

[16] Waldinger, R., "Achieving Several Goals Simultaneously," in, *Machine Intelligence 8*, Elcock, E. and D. Michie eds., pp 94–136 (Ellis Horwood, Edinburgh, Scotland, 1977).

[17] Warren, D.H.D., "WARPLAN: A System for Generating Plans," Memo No. 76, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland (June 1974).

[18] Warren, D.H.D., "Generating Conditional Plans and Programs," *Proc. AISB Summer Conference*, University of Edinburgh, Edinburgh, Scotland, pp 344–354 (July 1976).

[19] Wilkins, D.E., "Domain-Independent Planning: Representation and Plan Generation," *Artificial Intelligence*, Vol. 22, No. 3, pp 269–301 (1984).