A Formal Model for Classical Planning*

Jens Christensen
Computer Science Department
Stanford University
Stanford, CA 94305
jens@cs.stanford.edu

Adam Grove
Computer Science Department
Stanford University
Stanford, CA 94305
grove@cs.stanford.edu

Abstract

In this paper, we describe a formal semantic model that applies to many "classical" planning systems. This gives a unifying framework in which to study diverse planners, and motivates formal logics that can be used to study their properties. As an example of the model's utility, we present a general truth criterion which tests for the necessary truth of a proposition at arbitrary points in the planning process.

1 Introduction

In this paper we investigate a formal model for a broad class of A.I. planning systems. We define the range of applicability of our model later; here, we simply note that it is relevant to such well-known "classical" planners as STRIPS [Fikes and Nilsson, 1971], NOAH [Sacerdoti 1977], NONLIN [Tate, 1977], TWEAK [Chapman, 1987 and SIPE [Wilkins, 1988]. Each of these planners has its own very distinctive features. But there are also many similarities, and we can define a useful *formal* model of the concept of "plan" used by all of them.

The model can be used to provide *semantics* for much of the planning process. That is, we can develop a logic which is interpreted as making assertions about plans. Relative to the model and the logic, claims about particular plans or planning systems can be proven true or false. Section 3 gives a concrete application of this: we develop a *truth criterion* for a broad class of planning systems. The concept of a truth criterion (TC) is due to Chapman [Chapman, 1987]. In fact, many of the ideas in this paper were inspired by Chapman's work and so we spend the rest of this introduction contrasting the goals of his work and ours.

In general, a planner's truth criterion is the test it uses to check whether a particular proposition holds at some point in a proposed plan. It is easy to see how important this test is. For example, we are always interested in whether the *goal* propositions will be true after execution of a proposed plan. A truth criterion tells the planner when to stop planning for the current goal. Aside from computational efficiency, there are three desirable properties a truth criterion can have. First, it should be correct: when it tells us that a goal is achieved, we can safely stop planning (for this goal). Second, a TC can be sufficient: if the goal is achieved, it will tell us. Sufficiency makes planning faster, because we can avoid unnecessary work. Finally, and somewhat more vaguely, a truth criterion may be informative. By this we mean that, when a proposition fails to hold, the TC gives us information about why it does not and we can use this to guide further planning steps.

Chapman presented a model for one particular planning system (his "TWEAK" formalism). Using this model, assertions *about plans such as "proposition p* necessarily holds after execution of action a" are given a precise meaning. Chapman gave an interesting test for the necessary truth of a proposition which is, relative to his model, provably correct and sufficient. Furthermore, this TC is informative; it is possible to "read off" all the useful modifications we might make to a plan to ensure the proposition's truth. Given any reasonable search strategy that explores the suggested possibilities, the resulting planner will be correct and complete.[1]

We also give a precise semantic model for planning, in Section 2. However, whereas Chapman discusses just one, quite restrictive, planning system, we discuss a general theory for all "classical" planners. This reflects our goal to develop a theory that is useful for comparing many different planners. In contrast, Chapman is able to develop a new, straightforward, and provably correct planner by taking advantage of features particular to the system he studies.

We discuss a logic for describing and reasoning about our models. The idea of using a logic was implicit in Chapman's work, but never developed; for although he attempts to give a version of his TC in "logical notation", the formula presented contains minor errors and so does not quite correspond to his criterion. This has no effect on his results, because the real criterion, its proof

*The first author was partially supported by NASA Grant NCC2-494 and by Texas Instruments Contract No. 7554900. The second author is supported by an IBM Graduate Fellowship.

Present address: Teknekron, 530 Lytton Ave, Palo Alto, CA 94301.

[1] That is, it always finds a working plan if this is possible, and never terminates with a plan that doesn't satisfy all goals.

and all discussion of its application, are in English. But in general, a well-defined formal language is useful because it can prevent ambiguity. Furthermore, we can use a formal logic to give syntactic proofs of assertions about the planner (such as the correctness of a proposed truth criterion). Theorem proving is much easier to automate than subtle semantic arguments like those used by Chapman.

2 The Model and Logic

Our model applies to planning systems that depend on a *single-agent* and *situation-based* world model and work by *refining nonlinear plans.* Before presenting the model itself, we look at these issues.

Plans are constructed for agents that operate in some well-defined environment. We only consider planners whose world model is like the situation calculus ([McCarthy, 1968]). More precisely, we suppose that a particular state of the world is described by a set of objects, often called *propositions,* which throughout we consider to come from some fixed set *V?* We say that a proposition holds at, or is true in, a state if it is a member of that state. We assume that basic operations in the world, the actions, are functions mapping one state to another resulting state (i.e., they are functions from subsets of *V* to subsets of *V).* Of the class of all possible actions, *A,* the agent we are planning for can presumably execute some known subset, say R. If the agent executes a sequence of actions starting in state I, the final state must be what results from applying the composition of the actions to /.

The purpose of a planner is to find a sequence of actions in R which ensures that, when the world contains a given collection of *initial* propositions, then after execution it is certain that all of some given collection of *goal* propositions holds. We will call any sequence of actions, together with the set /, a *complete plan.* We are interested in planners that proceed by considering *sets* of complete plans, and slowly narrow the size of the sets until all the complete plans that remain guarantee the goals (i.e., this is the least-commitment approach to planning). Because it is never practical to manipulate arbitrary sets of plans, a planner always uses some compact representation system, which we will call the *plan representation language,* to describe such sets. Here, we make the assumption that the planners use a language which describes *nonlinear* plans. A nonlinear plan is given by:

ICV, which is the initial state.

- A fixed, finite collection *P* of *plan steps* $A_1,\ldots, A_n$. Each plan step is associated with a collection of actions in R (an *action-set).* Where there can be no confusion, we will blur the distinction between a plan step and its associated action-set.
- A partial order on plan steps, -<.
- A set of constraints on the n plan steps. The constraints specify which combinations of actions in A_1 x A_2 x ,., x A_n are actually permitted.

It is easy to see how a nonlinear plan describes a set of complete plans: it stands for all such plans obtainable by extending -< into a total order on the A_i, and then choosing (subject to the constraints) one action from each A_i. The term "nonlinear" is used because we need only give a partial order on the plan steps.

Virtually all the planners mentioned in Section 1 manipulate some form of nonlinear plan.[3] Within the framework described so far (which has just concerned itself with representations) planners differ in three main ways. First, many planners only allow certain special types of actions to occur in R. A very common restriction on R is what we call the STRIPS assumption ([Fikes and Nilsson, 1971]), Under this assumption, an action a € R must have a very simple structure: in any situation, a has the effect of first removing some fixed set of propositions (traditionally called the *delete list)* then adding all of some other fixed set (the *add list).* Second, plan representation languages differ in how they describe the action-sets associated with a plan step in a partial plan. One common technique, used in Chapman's system among others, allows action "schemas" such as Puton(x.y) which stand for all actions obtained instantiating the *variables* x,y; we discuss such systems in somewhat more detail later. Finally, planners can differ significantly in their constraint schemes (for example, Chapman uses a simple language that constrains the values his variables can be instantiated to). Note that the constraint language and the action-set representation language are interrelated.

In our model, these differences are represented within a uniform framework. We first define a partial plan, which is a tuple *{I,P,* -<) (where each component is as described above). The structure of partial plans can be viewed as a plan representation language, but it is very weak because we make no provision for constraints. Our model is based on the observation that a nonlinear plan in a general language, possibly with constraints, can be

[2] Propositions are often thought of as being formulas in some formal logic. However, it is not always possible or useful to take this view.

[3] Some of these planners, such as NOAH, NONLIN, and SIPE, also make use of a hierarchical structure on actions That is, there is a concept of "high level" actions and/or propositions, which can be expanded into a more detailed and more complex objects at a lower abstraction level. Our model does not capture this aspect of a plan representation system. This does not invalidate its usefulness however, because, as Wilkins [Wilkins, 1988] points out, all hierarchical planners operate on a series of *planning levels,* where on any one level the degree of abstraction never changes. Within one level our model is usually applicable. Nevertheless, giving a deeper theory for these concepts would probably be the most interesting extension to the model we are proposing here.

[4] In addition, we sometimes specify a set of *preconditions,* which are propositions that must be true before an action *is* regarded as being truly executable. Considerations of executability are ignored in the model we present (and, in effect, this is also true for Chapman's model). During planning, preconditions become (sub-)goals that must be achieved, and so typically only influence the goal selection component of the planner.

described using a collection of partial plans.

Definition: A *plan-structure* is a finite tree, where the nodes are partial plans. If $\mathbf{n}_i = (I_i, P_i, \prec_i)$ is an ancestor of, or equal to, $\mathbf{n}_j = (I_j, P_j, \prec_j)$, then we write $\mathbf{n}_i \rightarrow \mathbf{n}_j$ and say that $\mathbf{n}_j$ is a specialization of $\mathbf{n}_i$. Whenver this is the case, we insist that $\prec_i \subseteq \prec_j$, $I_i \subseteq I_j$, and $P_i = P_j$. However, although $P_i = P_j$, we do not assume that the action-sets associated with a particular plan step A_i are identical in the two partial plans, but we do require that the action-set for A_i in $\mathbf{n}_i$ is a superset of the corresponding action-set in $\mathbf{n}_j$, for all i. From this condition it follows that complete plans (i.e., partial plans that represent a single complete plan) must be leaf nodes in the tree. Conversely, we also require that every leaf must be complete.

From any n_i, there are a large number of potential specializations (formed by constraining the partial order in some way, and restricting the action-sets associated with some of the plan steps). However, not all these possibilities can be considered by the planner. This is because the planner has a limited language for describing action-sets, so that not all possible subsets of *R* can be captured, and also because of constraints. This is perhaps the key point: the main differences between different planners' representations system can be captured by differences in how the specialization relation works.

We can define a formal language, which makes assertions about plan-structures. Because the model is so general, the language covers many planning systems. To shorten the presentation, we introduce our language informally. The key features are:

- The language is a based on a sorted first-order logic, with the usual connectives, and quantification for the two sorts *plan-steps* and *propositions.* At any node in the model, the domain of *plan-steps* is, of course, just the plan-steps at the node, and the domain of the sort *propositions* is P.[5]
- The language has two primitive modal operators, $\Box_S$ and $\Box$. At a node in a plan, $\Box_S\phi$ is true just if ϕ is true in all nodes which are specializations of the first, and $\Box\phi$ is true if ϕ holds in all specializations which are complete plans. We read $\Box\phi$ as "ϕ is necessarily true". The "necessary" modality is the key to our logic; a necessary property of a nonlinear plan is true of all the complete plans it describes. The truth criterion we give later is of interest because it shows that, sometimes, we can discover whether a formula is necessarily true without considering all these complete plans individually.

  We sometimes use the duals of $\Box$ and $\Box_S$, written $\Diamond$ and $\Diamond_S$, which are defined in the usual way as $\Diamond = \neg\Box\neg$ and $\Diamond_S = \neg\Box_S\neg$ respectively.
- There is a binary predicate $\prec$ between plan steps. At $\mathbf{n}$, $a \prec b$ if in fact the plan step (denoted by) a precedes b.
- There is one unary predicate Initially on propositions. If p is a term denoting a proposition, Initially(p) is true at a node if the denotation of p is in I (for that node).
- There are three binary predicates, Asserts, Denies and Holds which relate propositions to plan steps. First suppose that, n is a complete plan, say (I,P,-<), and that p denotes a proposition in *V.* Then Holds(p, a) is true at n if the proposition denoted by p is true immediately prior to the plan step denoted by a. This is well defined because of the nature of complete plans, and because we know the initial state I. In a complete plan, Asserts(a,p) is true if Holds(p, a) is false but p holds immediately after *a* is executed. (Denies is defined similarly; p must be true before o but false afterwards.)

  When n is not complete, the denotations of Asserts, Denies and Holds can be defined arbitrarily.

There are several interesting formulas that are valid in all plan-structures. One of the most important is the following axiom, that captures the fundamental connection between Asserts, Denies, Holds, and Initially as we have defined them.

$$\Box(\mathbf{Holds}(p,a) \equiv (\neg\exists b(b \prec a) \wedge \mathbf{Initially}(p)) \vee \exists c\,((c \prec a) \wedge \neg\exists d((c \prec d) \wedge (d \prec a)) \wedge (\mathbf{Asserts}(c,p) \vee (\mathbf{Holds}(p,c) \wedge \neg\mathbf{Denies}(c,p)))))$$

Later, we will refer to this formula as the *Holds axiom.*

All nonlinear planning languages share some common weaknesses and our language can help make these precise. For example, it is a consequence of the use of partial orders on plansteps that formulas like the following are valid:

$$\forall x,y,z\ \Box_S((x \prec z) \wedge (\neg((x \prec y) \wedge (y \prec z)))) \Rightarrow \Box_S(y \prec x) \vee \Box_S(z \prec y)$$

That is, using partial orders does not allow us to impose a constraint that simply says "x must come immediately before z"; we must also commit to whether x and z are before or after every other action y.

There are also many formulas that are not valid in the general case, but are true of particular systems. If the STRIPS assumption about actions in $\mathcal{R}$ is assumed, then additional axioms like the following hold.

$$\forall x,y,z\ \Diamond_S(x \prec y) \wedge \Diamond_S(\mathbf{Asserts}(z,p)) \Rightarrow \Diamond_S((x \prec y) \wedge (\mathbf{Asserts}(z,p) \vee \mathbf{Holds}(p,z))$$

Such formulas say that the propositions asserted (or denied) by an action are essentially independent of the ordering imposed on plan-steps. This is a very strong property which simplifies planning enormously when it is reasonable. Most planners use STRIPS-like actions.

Another special axiom that is true in systems like Chapman's TWEAK is:

$$\forall x_1, x_2, \ldots, x_n\ \Box(\mathbf{Asserts}(x_1,p) \vee \ldots \vee \mathbf{Asserts}(x_n,p)) \Rightarrow (\Box\mathbf{Ensures}(x_1,p) \vee \ldots \vee \Box\mathbf{Ensures}(x_n,p))$$

[5] Note that these domains are independent of the node in a plan-structure. In a presentation of the formal properties of the logic (for which we have insufficient space here) this observation has some important consequences (such as the truth of the "Barcan" axiom; see [Hughes and Cresswell, 1968]).

(In this axiom, Ensures(z,p) is simply an abbreviation for Asserts(x,p) V Holds(p,x).) This axiom can be understood as showing that the constraint language in TWEAK is rather weak; it cannot be used to constrain a group of actions to assert p, except in the rather trivial case where one particular action is guaranteed to assert *p* anyway. We have insufficient space to explain in detail how such axioms arise in Chapman's system. In brief, Chapman's action-set language makes use of "propositions" with *variables,* which are essentially placeholders that are instantiated during planning. For example, in the "blocks world" a plan step might be associated with action schema Pickup(x), with add list Holding(x) (and perhaps some preconditions, such as Clear(x)). This stands for all actions in R obtainable by instantiating x to a real object,[6] It turns out to be very important for Chapman's TC that variables can be instantiated in infinitely many ways; and the reason for this is essentially to guarantee the truth of axioms like the above, although this can appear quite mysterious until Chapman's proof is read. In our language we express many of the important properties Chapman required without using planner specific features such as variables. It is not possible to reprove Chapman's TC exactly in our system, because his TC makes use of features specific to TWEAK. However, a TC which is very similar in spirit can be proven just by making assumptions, such as the formula just given, whose truth can be tested for any nonlinear planning system.

3 Truth Criterion

Chapman proved that his TWEAK truth criterion is correct and sufficient for his model. However, as he points out, this criterion fails for any planning system just somewhat more expressive than his. One basic shortcoming of Chapman's truth criterion is its requirement that, whenever a proposition necessarily holds at some point in the plan, it is necessarily asserted by one particular plan step. But in general, it is possible for multiple plan steps to act together to guarantee the truth of a proposition, even where no one step alone is enough.

In this section, we discuss a new truth criterion that is correct and sufficient for any planning system within the scope of our model.

3.1 Presentation

Intuitively our truth criterion can be read as follows. A proposition *p* is necessarily true immediately before plan step *a* iff, *in* all specializations *of* the plan, *two* conditions hold; (1) If a is necessarily the first plan step, then it is necessarily the case that p is initially true, and (2) whenever there is a plan step *c* that is necessarily immediately before *a*, then if *c* does not possibly assert *p* then *p* must hold at *c* and not possibly be denied by *c*.

In our logic this TC is given as:

$$\Box_S \Big((\neg\exists b\ \Diamond(b \prec a) \Rightarrow \Box\textsf{Initially}(p)) \wedge \forall c\ (\Box(c \prec a) \wedge \neg\exists d\ \Diamond((c \prec d) \wedge (d \prec a))) \Rightarrow (\neg\Diamond\ \textsf{Asserts}(c,p) \Rightarrow \Box\textsf{Holds}(p,c) \wedge \neg\Diamond\textsf{Denies}(c,p)) \Big)$$

Using our logic, it is possible to give a (lengthy) syntactic proof showing that the TC holds if and only if $\Box\textsf{Holds}(p,a)$ (see [Christensen, 1990]). In principle, this proof could be checked by an automatic theorem prover, and perhaps even generated by one. Here, we simply give a sketch of a semantic proof.

We first show that whenever $\Box\textsf{Holds}(p,a)$ fails to hold, our TC is false also. So suppose $\neg\Box\textsf{Holds}(p,a)$. Consider one of the completions where $\neg\textsf{Holds}(p,a)$. There are two cases: either *a* is the first step of the plan or it is not. If the former, the Holds axiom shows that $\neg\textsf{Initially}(p)$. But then our TC fails since this completion is a specialization where *a* is necessarily the first plan step and $\neg\Box\textsf{Initially}(p)$.

If *a* is not the first plan step then there must be some other plan step, *c*, which is immediately before *a*; then the Holds axiom shows that $\neg\textsf{Asserts}(c,p)$ and either $\neg\textsf{Holds}(p,c)$ or $\textsf{Denies}(c,p)$. It follows that $\neg\Diamond\textsf{Asserts}(c,p)$ holds in this completion, but one of $\Box\textsf{Holds}(p,c)$ and $\neg\Diamond\textsf{Denies}(c,p)$ is false. Therefore our TC does not hold.

Next, we show if our TC does not hold, then neither does $\Box\textsf{Holds}(p,a)$. Suppose the TC is false in a partial plan. Then there must be some specialization of the plan where either (1) *a* is necessarily the first plan step and $\neg\Box\textsf{Initially}(p)$ or (2) there is some plan step *c* immediately before *a* with $\neg\Diamond\textsf{Asserts}(c,p)$ and either $\Diamond\textsf{Denies}(c,p)$ or $\neg\Box\textsf{Holds}(p,c)$. If (1), there must be a completion where *a* is the first plan step and $\neg\textsf{Initially}(p)$. Then, by the Holds axiom, $\neg\textsf{Holds}(p,a)$ is true in this completion and therefore $\neg\Box\textsf{Holds}(p,a)$ *holds in our plan. If* (2) *there must be some completion* where either (i) there is some plan step *c* immediately before *a* and $\neg\textsf{Asserts}(c,p)$ and $\textsf{Denies}(c,p)$ or (ii) there is some plan step *c* immediately before *a* and $\neg\textsf{Asserts}(c,p)$ and $\neg\textsf{Holds}(p,c)$. In either case, by the Holds axiom, $\neg\textsf{Holds}(p,a)$, which implies that $\neg\Box\textsf{Holds}(p,a)$.

3.2 Algorithm for Checking our Truth Criterion

The straightforward implementation of our TC involves examining every specialization of a plan. This is clearly not desirable. In fact, the most naive truth criterion, namely applying the Holds axiom to every completion of a partial plan, is no less efficient. However, the potential for efficiency improvements results from noticing two important properties. First, we only need to examine those specializations where the following holds:

$$\neg\exists b\ \Diamond(b \prec a) \vee \exists c\ (\ \Box(c \prec a) \wedge \neg\exists d\ \Diamond((c \prec d) \wedge (d \prec a)) \wedge \neg\Diamond\textsf{Asserts}(c,p)\ )$$

In the following, we will refer to the expression inside the initial $\Box_S$ of our TC as $\textsf{CTC}(p,a)$. It is easy to see

[6]Note that expressions like Holdtng(z) are part of the plan representation language, and are meaningless unless considered in the context of the planning process. This contrast with true propositions in *V,* for example Holding(BLOCKA), which can be used to describe the state of the world. In languages like Chapman's, it is important to keep in mind that "variables" like *x* are just formal objects used by the planner, and do not connote any formal first-order quantification,

To test if Holds(p,a):

For every weakest specialization such that there is a plan step c necessarily immediately before a which does not possibly assert *p:*

Check that *c* does not possibly deny *p* and necessarily Holds(p, c).

For any weakest specialization such that *a* is the first plan step:

Check that Initially(p) necessarily holds.

Figure 1: Algorithm for checking truth criterion.

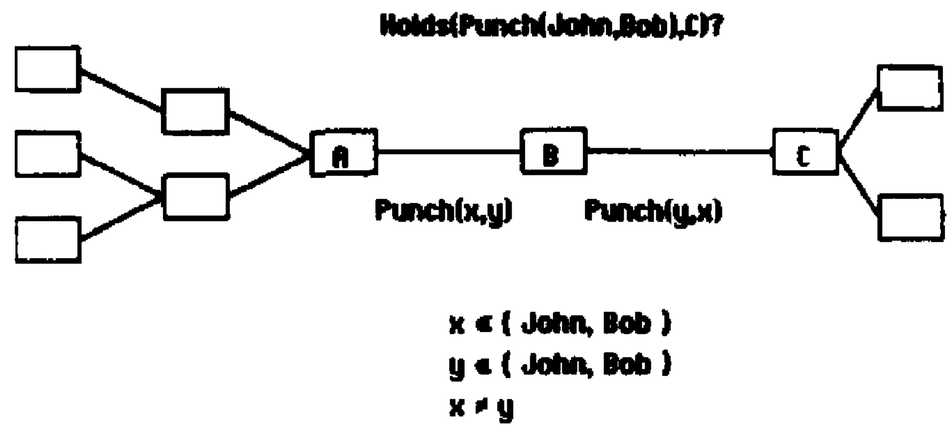

Figure 2: Example where Chapman's TC fails.

that if the above expression does not hold in a specialization then CTC(p, a) must hold there.

Second, we only need to examine the *weakest* such specializations. A specialization is weaker than another if the latter is a specialization of the former. If *CJC(p,a)* does not hold in such a specialization we can stop, since in that case $\neg\Box_S$CTC(p, a). If CTC(p, a) does hold in such a specialization then it holds for all specializations of that specialization. For a proof of this, see [Christensen, 1990]. Using these two properties we arrive at the algorithm in figure 1.

The algorithm for checking the truth criterion requires time exponential in the length of the plan. Chapman proved that determining $\Box$Holds(p, *a)* in any system where the schema representation is sufficiently strong to include conditionals is NP-Hard.[7] It is therefore extremely unlikely that any polynomial time algorithm will ever be found for verifying the truth criteria of such systems. However, our algorithm will often be considerably more efficient than the application of the Holds axiom to all the completions of a plan, as we shall see shortly. Furthermore, as we discuss later, it represents a good starting point in the development of incomplete or unsound algorithms which might overcome the complexity barrier.

3.3 Restricted Ranges of Variables

Chapman gives an example of an extension to TWEAK that causes his TC to fail. Recall that TWEAK uses "variables", which are placeholders the planner uses to reason about groups of actions. If TWEAK is extended with a mechanism for restricting the range of variables, then Chapman's TC is no longer sufficient. For example, in TWEAK, if CLEAR(x) is denied by a TWEAK plan step, this step can be specialized into infinitely many actions which deny propositions that satisfy the CLEAR(i) schema, e.g. CLEARS), CLEAR(B), CLEAR(C), *CLEAR(Harry),* etc., where *A,* B, C, and *Harry* are constants. Restricting the range of variables might be desirable to ensure that variables adhere to a certain type, e.g. we might wish to restrict variable *x* to the values {A, B, C}, thereby eliminating the possibility CLEAR(Harry).

Suppose we have the plan in figure 2. In this example, Punch(Bob, John) necessarily holds in the final situation. However, it is not necessarily asserted by any one step, and so fails Chapman's truth criterion.

On the other hand, our truth criterion succeeds. Furthermore, our truth criterion algorithm requires that only two specializations of the plan be checked. To see this let us step through the algorithm. We would begin by evaluating Holds(Punch(Bob, John),C). Clearly, there are no specializations where C is the first plan step. However, there are two distinct weakest specialization such that there is a plan step immediately before C and that plan step does not possibly assert Punch(Bob. John). These are simply the specializations where the constraints $y \neq$ Bob and $x \neq$ John have been respectively added. We will examine the former specialization. In this specialization plan step B must not possibly deny Punch(Bob, John). This is in fact so, since plan step B does not deny anything.

Furthermore, we want $\Box$Holds(Punch(Bob, John),B). This is checked with another application of the TC. There are no specializations where B is the first plan step. However, in this case there are no specializations where there is a plan step immediately before B that does not possibly assert Punch(Bob, John). To guarantee this would mean adding either $x \neq$ Bob or $y \neq$ John. If we add the former then both *y* and *x* must denote John, which is not allowed. If we add the latter then *y* can't possibly denote any object, which is also disallowed.

The evaluation of the other specialization proceeds in an analogous manner. As can be seen from stepping through the algorithm, it is somewhat reminiscent of resolution theorem proving, in that we continually attempt to disprove the proposition until a contradiction is found. It is different from resolution theorem proving in that it is guaranteed to terminate if a contradiction is not found in this manner.

3.4 Conditional Actions

Figure 3 shows a plan which, while simple, cannot be represented by TWEAK, or by most other classical planning systems. One exception is [Pednault, 1988], who presents a plan language that supports conditional actions and a method for synthesizing linear plans using these actions.

The actions in figure 3 might be described as being *conditional* because their effect depends on the state of the world in which they are executed. In contrast

Conditional actions, which can be modeled in our system, will be described later.

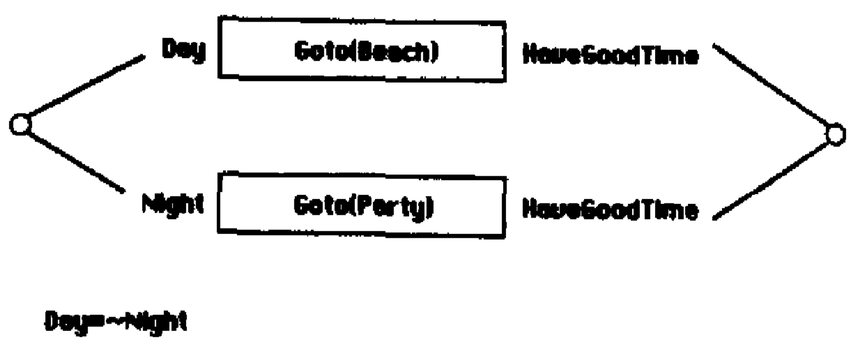

Figure 3: Example with conditional actions.

STRIPS-like actions have a constant effect (in any world where their preconditions are true).[8] In the figure, one can always go to the beach; there are no executability preconditions as such. However, nothing will be achieved at night. This is a very simple conditional action representation and nothing in our model precludes *UE* from having arbitrarily complex conditional actions; our model allows any action that is a function mapping states to states.

The algorithm for determining our truth criterion would begin by considering one of the specializations where there is an action immediately before the final situation which does not possibly assert HaveGoodTime. Assume it considers Goto(Beach) first. We must constrain -day to be true, or else HaveGoodTime would possibly be asserted. Then, we verify that Goto(Beach) cannot deny HaveGoodTime and recurse on Holds(HaveGoodTime,Goto(Party)). However, there are no specializations with an action immediately before Goto(Party) where HaveGoodTime is not possibly asserted. If there were then both —"night and -day must hold, which is impossible.

3,5 Conclusion and Further Work

In practical terms, one might argue, our truth criterion is not useful since the algorithm for verifying it appears to be inherently exponential. This contrasts with Chapman's truth criterion, which can be verified in polynomial time. However, the value of a polynomial time truth criterion is somewhat questionable, given that the planning process is still undecidable. Furthermore, TWEAK achieves efficiency in the truth criterion at the cost of limitations in expressive power.

As domains become more complex it becomes less and less likely that the TWEAK representation can be used to adequately model them. For example," TWEAK is unable to efficiently model HACKERS cubical blocks world, and completely unable to solve problems whose solution requires conditional actions.

There is an interesting tradeoff between expressivity of a planning language and the search process using that language. As expressivity is added to the language the number of choice points at each plan node in the search tree increases. Furthermore, efficient truth criteria such as Chapman's are less likely to exist. On the other hand, because of the finer resolution greater expressivity allows, over-commitment is more easily avoided, and so the amount of backtracking can be reduced. Also, the increased expressivity allows more knowledge to be encoded at each node, thereby allowing for possibly more informed choices. So it is far from clear what the optimal point along this spectrum of expressivity is. We note that much of planning research has concerned itself with developing planners which support powerful plan representations.

Of course, such planners are generally not based on provably correct truth criteria. Our truth criterion could in principle be used instead. However, for efficiency reasons, it is likely that the actual algorithm used would be tailored to take advantage of the planner's specific plan representation. It might even be the case that completeness, or even in some circumstances, correctness, is sacrificed in order to achieve efficiency gains. However, the designer of the truth criterion would have a sound and complete reference point from which to start, and would therefore be well aware of the corners that were cut to achieve efficiency. It remains to be seen what the appropriate tradeoff between efficiency, correctness, completeness, and expressive power of the plan representation is.

[5]The idea of modifying the STRIPS paradigm, to allow for "preconditions" which determine action's effects, can be found occasionally in the literature; for instance, these are Pednault's *secondary preconditions.*

References

[Chapman, 1987] Chapman, D., "Planning for Conjunctive Goals," *Artificial Intelligence,* 32 (July 1987) 333-378.

[Christensen, 1990) Christensen, J., "Automatic Abstraction in Planning," PhD Dissertation, Stanford University, 1990.

[Fikes and Nilsson, 1971] Fikes, R. E. and Nilsson, N, J., "STRIPS: A new Approach to the Application of Theorem Proving to Problem Solving," *Artifictal Intelligence,* 2(3-4) (1971) 189-208.

[Hughes and Cresswell, 1968] Hughes, G.E, and Cresswell, M.J., "An Introduction to Modal Logic,[11] Methuen and Co, London, England, 1968.

[McCarthy, 1968] McCarthy, J., "Programs with Common Sense," in: Minsky, M. (Ed.), *Semantic Information Processing,* MIT Press, Cambridge, MA, 1968.

[Pednault, 1988] Pednault, E., "Synthesizing Plans that Contain Actions with Context-Dependent Effects," *Computational Intelligence,* 4(4) (1988) 356-372.

[Sacerdoti, 1977] Sacerdoti, E., *A Structure for Plans and Behavior,* Elsevier, North-Holland, New York, 1977.

[Tate, 1977] Tate, A., "Generating Project Networks; *Proceedings IJCAI-77,* Cambridge, Massachusetts, (1977) 888-893.

[Wilkins, 1988] Wilkins, D. E., *Practical Planning,* Morgan Kaufman, San Mateo, California, 1988.