reprinted from the Proc. of the Second International Conference on Principles of Knowledge Representation and Reasoning, Cambridge, MA April, 1991

Planning as Temporal Reasoning

James F Allen
Department of Computer Science
University of Rochester
Rochester, NY 14627

Abstract

This paper describes a reasoning system based on a temporal logic that can solve planning problems along the lines of traditional planning systems. Because it is cast as inference in a general representation, however, the ranges of problems that can be described is considerably greater than in traditional planning systems. In addition, other modes of plan reasoning, such as plan recognition or plan monitoring, can be formalized within the same framework.

1 INTRODUCTION

There is strong interest currently in designing planning systems that can reason about realistic worlds. In moving from the toy-world domains that characterized early work, researchers are looking at a wide range of issues, including reasoning in uncertain worlds, interacting with processes and events beyond the agent's direct control, and controlling mechanisms in real-time (i.e. robotics). One of the problems faced in extending existing frameworks is the weak expressiveness of the representation of the actions, events and the external world. This paper describes a reasoning system based on a temporal logic that can solve planning problems along the lines of traditional planning systems. Because it is cast as inference in a general representation, however, the ranges of problems that can be described is considerably greater than in traditional planning systems.

The key observations motivating this development are the following: 1) Actions take time - very few actions are instantaneous, 2) More than one action may occur at the same time, 3) Complex plans of activity may involve complex ordering constraints, and 4) Actions may interact with external events beyond the agent's direct control.

Of these problems, the most central one is that of dealing with simultaneous action. Without simultaneous action, the range of problems that can be addressed is very limited, mainly restricted to specialized situations such as computer programming and game playing. It makes little sense to study planning in a dynamic changing world if the planner cannot act while some other natural event is occurring. The problems that arise when an action and a external event occur simultaneously exactly parallel the problems of simultaneous action. To understand why this problem is so difficult, it is important to look at the assumptions underlying the world representation in most current planning systems. This is examined in detail in the next section.

2 BACKGROUND: ACTIONS AS STATE CHANGE

The predominant approach to modeling action in artificial intelligence and computer science has been to view action as state change. This view underlies all the state-based planning systems (e.g. STRIPS and its successors), formal models of planning (e.g. the situation calculus [McCarthy & Hayes 1969], and work in dynamic logic for the semantics of programs (e.g. [Harel 1974] and [Rosenschein 1981]. In this view, the world is modelled by a succession of **states**, each **state** representing an instantaneous "snapshot" of the world. Actions are defined as functions from one state to another. Propositions in such models are relative to states. The notion of a proposition independent of a state is modelled as a function from states to truth values. In the situation calculus [McCarthy & Hayes 1969], such functions are called **fluents**. For example, *On(A,B)* is a fluent that when applied to a state S is a proposition that is true if A is on B in state S.

In state-change models such as STRIPS, actions are instantaneous and there is no provision for asserting what is true while an action is in execution. Also, since the state descriptions do not include information about action occurrences, such systems cannot represent the situation where one action occurs while some other event or action is occurring. Finally, there is no reasoning about the future in these models except by searching through different possible action sequences.

Many n**on-linear planners** suffer from the same deficiencies, although some (e.g. [Tate 1977, Vere 1983, & Wilkins 1988]) allow simultaneous actions if the two actions are independent of each other. In such cases, the effect of the two acts performed together is the simple union of the individual effects of the acts done in isolation. The problem with this solution is that it excludes common situations of interest in realistic domains. In particular, interesting cases of interaction occur when the effect of two actions done together is *different* from the sum of their individual effects. In particular, two actions may have additional effects when performed together, or they may partially interfere with each other's effects.

Here's one example concerning the door to the Computer Science Building at Rochester. The door is designed so that it requires both hands to open it, very annoying since you have to put down whatever you are carrying! There is a spring lock that must be held open with one hand, while the door is pulled open with the other hand. If we try to formalize this in a STRIPS-like system, we find there is no way to assert that unless the lock is held open it will snap shut.

An approach to this problem has been used in several systems (e.g. [Vere 1983]). The interaction of two actions is encoded in a special state. We'll call this technique **state encoding**. In particular, in the above example, we might introduce a fluent that is true only if the agent is holding the lock open. The action of holding the lock would be transformed into two actions, one to start holding the lock, and another to release it. Pulling the door simply has a precondition that the agent is holding the lock open. The fluent "holding lock open", once asserted by the TURN-LOCK action, remains true until a RELEASE-LOCK action deletes it. While this might solve this particular problem, there are many potential disadvantages with this approach. The first objection is that it is ad-hoc. While it may be the case that one can invent predicates to cover every specific example someone proposes, each must be done after the fact on an individual case by case basis. It is also not clear how the technique could be generalized to additional complexities involving simultaneous actions. More importantly, *holding the lock open* is intuitively an action - it may take effort on the part of the agent to maintain and must explicitly be part of the plan. This is not reflected in a representation where holding the lock open is simply a fluent that will remain true until the agent does something to stop it.

In certain applications, where a detailed causal theory is known, state encoding approaches can be very powerful. If we cannot specify such a complete causal theory, however, or if we simply don't know enough about a situation to be able to use a causal theory, then other reasoning techniques must be explored. We would claim that both these problems arise in everyday planning situations: first, we have no detailed causal theory of the world, and second, we would not know the values for the parameters to the theory if such a theory was known.

2 TIMES, EVENTS AND ACTIONS

We do not have the space for an extensive discussion of time here. I will use interval temporal logic as developed in Allen [1983] and Allen and Hayes [1985]. In this logic, time is constructed out of one primitive object, the **time period,** and one primitive relation called **Meets**. Two periods meets if one precedes the other but there is no time between them. From the *Meets* primitive, many different temporal relations can be defined. In this paper uses the following:

In(i,j) - period *i* is contained in period j;
Disjoint(i,j) - *i* and *j* do not overlap in any way;
Starts(i,j0 - period *i* is an initial subsegment of *j*;
Finishes(i,j) *i* is a final subsegment of *j.*
SameEnd(i,j) - periods *i* and *j* end at the same time.
Overlaps(i,j) - *i* starts before but overlaps *j*

Time periods can be divided into the non-decomposable periods called **moments**, and decomposable periods, called **intervals.**

We attach times to predicates by adding an extra argument to each predicate as in Bacchus, Tenenberg, and Koomen [1989]. For example, the proposition *Green(FROG13,T1)* is true only if the object named by *FROG13* was green over the time named by *T1*.

By allowing time intervals as arguments we open the possibility that a proposition involving some predicate P might neither be true nor false over some interval t. In particular, consider a predicate P such that p is true during some subinterval of t, and also false in some other subinterval of t. In this case, there are two ways we might interpret the proposition ~P(t). In the **weak** interpretation, ~P(t) is true iff it is not the case that P is true throughout interval t, and thus ~P(t) is true if P changes truth values during t. In the **strong** interpretation of negation, ~P(t) is true iff ~P is true throughout t, and thus neither P(t) nor ~P(t) would be true in the above situation. Thus, a logic with only strong negation has truth gaps.

We use the weak interpretation of negation, as do Shoham [1987] and Bacchus, Tenenberg and Koomen

[1989], to preserve a simple two-valued logic. Of course, we can still make assertions equivalent to the strong negation. The fact that P is false throughout t can be expressed as follows, where *In* is true if its first temporal argument is contained in its second:

$$\forall t' . In(t',t) \supset \sim P(t').$$

This logic is still insufficient to conveniently capture many of the situations that we need to reason about, however. In particular, we need to introduce events as objects into the logic. Davidson [1967] argued that there are potentially unbounded qualifications that could be included in an event description. For example, the event of Jack lifting a particular ball might be asserted to occur at some time by a predicate *Lift*, as follows:

$$Lift(JACK34,BALL26,T1).$$

The problem arises in now representing the event "Jack lifted the ball onto the table". Either we need to add another argument to the *Lift* predicate, or we need to introduce a new predicate that represents a variant of lifting that includes an extra argument. Either is unsatisfactory. Davidson suggested the solution of reifying events, whereby additional modifiers would simply become additional predications on the event. Thus, the event of Jack lifting the ball to the table with the tongs might be represented as

$$\exists e . Lift(JACK34,BALL26,e,T1) \wedge Dest(e) = TABLE555 \wedge Instrument(e) = TONGS1.$$

We will represent knowledge about action in several ways. The first is by defining the necessary conditions for the event consisting of the action occurring (as in [Allen(1983]) For example, consider the action *stack(a,b)*, which involves stacking block *a* on block *b*. For this action, we define a predication *Stack(a,b,e,t)*, that is true if e is an event consisting *stack(a,b)* occurring over time period t.

The event variable plays a central role - all the other parameters can be defined in terms of the event variable. For example, every instance of a stacking event uniquely defines the blocks that it involves, and the times relevant to the properties that define it. As a convention, we will denote the times for properties corresponding to preconditions by functions *pre1, pre2* and so on, those corresponding to effects by *eff1, eff2*, and so on, and those corresponding to conditions that hold while the event is occurring by *con1, con2,* and so on.

The stacking action in a typical STRIPS-style system is defined by its preconditions: (both blocks must be clear), and its transition function: (delete the formula *Clear(y)* and add the formula *On(x,y))*. We can use the STRIPS definition to motivate the conditions of the world that necessarily must be true whenever such a stacking event occurs.

In particular, each event type defines a set of temporal functions that define the structure of the temporal intervals involved in the event. For example, the class of stacking events uses functions to produce times corresponding to the properties involved in the action's preconditions and effects. We can define the structure of the stacking event as follows (see Figure 1):

Stacking Axiom 0: Temporal Structure

$$\forall e , \exists a, b, i . Stack(a, b, e, i) \supset Overlaps(pre1(e), i) \wedge Finishes(con1(e), i) \wedge Meets(pre1(e), con1(e)) \wedge Meets(i, eff1(e)) \wedge SameEnd(i, pre2(e)) \wedge Meets(i, eff2(e)).$$

With this temporal structure defined for every stacking event, the axiom defining the necessary conditions for the event's occurrence now can be expressed as:

Stacking Axiom 1: Necessary Conditions

$$\forall i, a, b, e . Stack(a, b, e, i) \supset Clear(a,pre1(e)) \wedge Holding(a, con1(e)) \wedge Clear(a,eff1(e)) \wedge Clear(b, pre2(e)) \wedge On(a, b, eff2(e)).$$

The above axiom asserts what is true whenever a stacking event occurs, independent of the situation. Other knowledge about action is relevant only in certain situations. For instance, if the block being moved in a stacking action was initially on another block, then this other block becomes clear (at least momentarily). This is easily expressed in the logic by the following axiom, which states that if block *a* was initially on another block *c*, then *c* becomes clear when *a* is moved:

Stacking Axiom 2: Conditional Effects

$$\forall i,a,b,c,t,e . Stack(a,b,e,i) \wedge On(a,c,t) \wedge Overlaps(t,i) \supset Clear(c,eff3(e)) \wedge Meets(t,eff3(e)) \wedge Meets(t,con1(e)).$$

This axiom applies in a situation with three blocks, say *A, B* and *C* where *A* is originally on block *C*. The conditions for the action *Stack(A, B, E1, T1)* are shown graphically in Figure 1. Note that this definition does not assert that the block C will be clear at the end of the stacking event. In particular, if two stacking events overlap in time (say *Stack(A,B,E1,T1)* and *Stack(D,C,E2,T2)*) then this may not be the case, for *D* may be placed onto *C* before *A* is placed on *B*. Such subtleties cannot be represented easily in a STRIPS-style representation.

The development so far has not captured any sense of causality. In particular, the axioms above do *not* state what properties are caused by the stacking action, or what properties simply must be true whenever the action succeeds. This is the distinction that STRIPS makes between preconditions and effects. Intuitively, it is clear that the stacking action causes block *a* to be on block *b* in situations where both blocks are clear at the start of the action. Furthermore, the stacking action

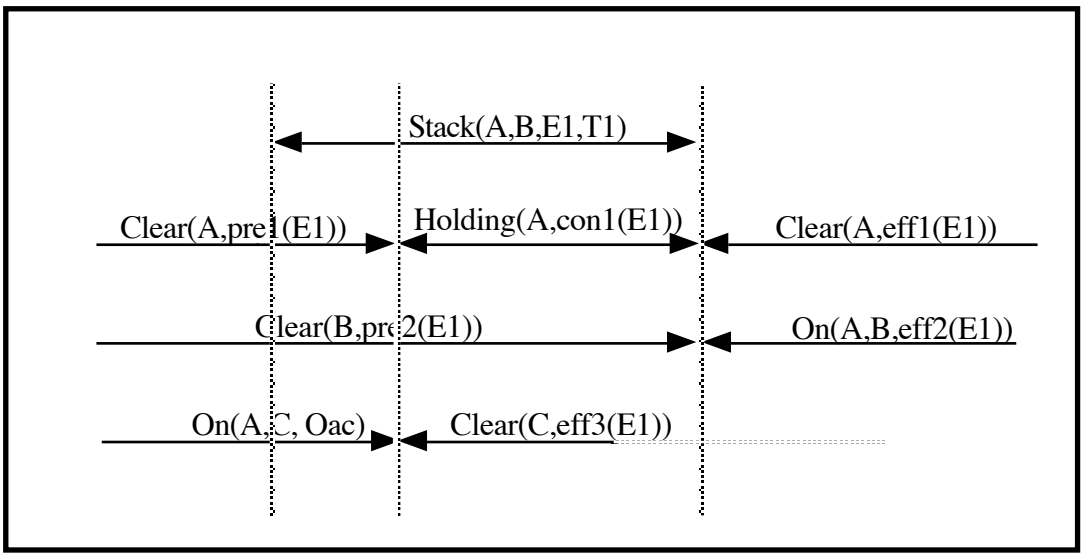

Figure 1: The necessary conditions for Stack(A,B,E1,T1) in a situation where A is originally on C (using stacking axioms 1 and 2)

causes block *b* to become not clear while it doesn't affect the condition that block *a i*s clear.

To encode such knowledge, we need to be able to reason about action attempts (cf. [McDermott 1986]). The logic developed so far can express the fact that a certain event occurred, but not that an agent attempted to do some action. The predicate *Try* is defined such that *Try(a,e,t)* is true whenever the action *a* is attempted by the agent at time *t* in order that event *e* occurs. Of course, if the conditions are not right then the action will not succeed and the event does not occur. For example, *Try(stack(a,b),e,t)* does not necessarily imply *Stack(a,b,e,t)*. The relationship between the two is defined by axioms corresponding to precondition assertions: In particular, we can assert that wherever the agent tries to stack *a* on *b* starting from an initial situation where *a* and *b* are clear, then a stacking event occurs:

Stacking Axiom 3: Prerequisites
$\forall$ *i,j,k, a,b,e . Try(stack(a,b),e,i)* $\wedge$ *Clear(a,j)* $\wedge$
Overlaps(j,i) $\wedge$ *Clear(b,k)* $\wedge$ *SameEnd(i,k)*
$\supset$ *Stack(a,b,e,i)* $\wedge$ *pre1(e)=j* $\wedge$ *pre2(e)=k.*

3 THE PLANNING FORMALISM

A planning system can now be specified using the temporal logic developed above. This system can reason about certain classes of interacting simultaneous events and it has a limited capability for reasoning about the future. In particular, while it cannot plan to change any external events predicted to occur in the future, it can construct plans that take future events into account and reason about interactions with such events. Pelavin [1988] and Allen et al [1991] present an extended logic that can represent future possibilities as well.

In order to construct plans, an agent needs to predict future states of the world. STRIPS-like problem solvers do this by using the add and delete lists to transform the current state into the next state. With a representation based on an explicit temporal logic, however, it is more complicated. In particular, if a proposition P is asserted to hold at a time T1 and then some action A occurs after T1 that makes P false, it is still true that P held at time T1. So the representation of the world should still contain this assertion. What has changed once the new action is introduced is some prediction about whether P holds in the future. For example, before A is known about, the agent might have predicted that P still holds in the future. Once the action A is expected, however, this prediction might change.

Thus it is the predictions (or expectations) about the future that change as an agent plans. Since an agent may change its mind about what future actions it might do, most conclusions about the future must be retractable. This suggests that some form of non-monotonic reasoning is necessary in order to maintain the world model and some models such as deKleer's ATMS [deKleer 1986] might be useful. But there is a simpler route: a model can be outlined that views all predictions about the future as conditional statements based on what the agent assumes about the future including its own actions. Given an initial world description *W* and a goal statement *G*, the plan is a set of assumptions $A_1,...,A_n$ such that

$$W^{-}(A_1 \wedge A_2 \wedge ... \wedge A_n \supset G.)$$

Of course, if the Ai's are inconsistent then this statement is vacuously true, so we must also add the condition that

$$A_1 \wedge A_2 \wedge ... \wedge A_n \text{ is consistent.}$$

Finally, we want to avoid assuming the problem away. For instance, if A_1 is simply equivalent to the goal statement G, the above conditions are true but we can't say we've solved the problem! This is handled by restricting the form of assumptions that the planner can make, as described below.

With this analysis, we can view the planning problem as consisting of two types of reasoning:

- prediction - what is the world like (based on a given set of assumptions)
- planning - what assumptions should the agent make about its future behavior, and the future world.

These two types of reasoning are explored in detail in the remainder of this section.

3.1 Predicting the Future

If an agent had full knowledge about a world, then predicting the future would be a relatively well-defined task. The agent would simply simulate the future course of events starting from the present state. In practice, however, the agent never has such detailed knowledge about the world - the agent's knowledge of the present state is partial, and the world is not well-understood enough to make precise predictions. Even qualitative models, such as those discussed in Bobrow [1985], assume a level of knowledge about the state of the world and the processes that change it that are not realizable in most situations.

Here we develop a very conservative model of prediction based on maintaining limited consistency of the agent's beliefs about the world. Essentially, given some set of beliefs about the future, the predictions are simply what is inferable from those beliefs using the agent's knowledge of the structure of the world and the definitions of actions. The particular system we will specify uses a forward chaining strategy on Horn clauses coupled with constraint propagation techniques for time (Allen, 1983, Koomen, 1989) to make the predictions.

To drive the predictor, we need knowledge about the actions, such as defined in the last section, as well as general knowledge of the domain. For instance, to reason about the door latch problem given at the start of this paper, we would have to know that a door cannot be open and shut at the same time. This motivates a forward chaining rule that guarantees that this cannot occur:

Domain Constraint 1
$\forall t1,t2 \; . \; DoorOpen(,t1) \wedge DoorClosed(,t2) \supset Disjoint(t1,t2)$.

Similarly, the latch is either open or shut, but not both:

Domain Constraint 2
$\forall t1,t2 \; . \; LatchShut(t1) \wedge LatchOpen(t2) \supset Disjoint(t1,t2)$.

Allen & Koomen (1983) show that a prediction algorithm using action definitions and domain constraints similar to those presented here can capture many important aspects of non-linear planning systems.

3.2 Making Assumptions

There are two main classes of assumptions that the planner must make. It must decide what actions it will attempt to perform, and it must make assumptions about how the external world will behave. While the planner may assume it can attempt any action at any time, the action will only succeed if the appropriate conditions hold. As we'll see below, this is implemented by allowing the planner to add an assertion of the form *Try(a,e,t)* without proof.

The assumptions about the external world are limited at present to *persistence* assumptions [Dean & McDermott 1987], that once a property is established, it tends to remain true until explicitly changed. More precisely, a literal $P(i_1,...,i_n,t)$ can be proven **by persistence** if there is a literal in the database of form $P(i_1,...,i_n,t')$ where it is possible that t=t'. This definition not only allows persistence into the future, it also allows persistence into the past. It will be examined further after the basic planning algorithm is presented.. Note that we have a constant-time method of heuristically checking whether t=t' is possible given the network representation of temporal information: the system simply checks if "=" is one of the disjuncts still present on the arc connecting the node for t and the node for t' (see [Allen 1983]).

4 THE PLANNING SYSTEM

To a first approximation, the planning algorithm is simply a backwards-chaining proof strategy driven by the goal statement, where assumptions about action attempts and persistence can be made without further proof. While very simple, this planner is similar in power to the regression planners (e.g. Waldinger, 1977).

To distinguish the logic developed so far from the system, which involves heuristic reasoning on a restricted formalism, we will use a different notation. A literal consists of a predicate name and a list of arguments enclosed in square brackets. Thus the literal corresponding to the formula *On(A,B,G)* is:

[On A B G].

Knowledge about actions is captured by a set of **planning rules**, which are a modified notation of Horn clauses using "?" as a prefix to indicate variables. In general, a planning rule is of the form

C <<< D1 D2 ... Dn such that A1, ..., Ak

and can be interpreted formally as a Horn clause: the consequent literal C is true (or provable) if the antecedent literals D1,...,Dn are true and the constraints A1,...,Ak are true. The system, however, will treat the antecedents and constraints differently to produce an efficient inference strategy.

Two planning rules for the action of pulling the door open are as follows: First, pulling on door when the latch is open results in the door being open:

(PullOpen.1)
[DoorOpen ?t'] <<<
[PullOpen ?e ?t] such that [EQ eff1(?e) ?t'].

Second, you may open the door any time you try to, if it's closed and the latch is unlocked:

(PullOpen.2)
[PullOpen ?e ?t] <<<
[DoorClosed pre2(?e)]
[LatchOpen pre1(?e)]
[Try [pull] ?e ?t]

The temporal structure for the *PullOpen* action must also be defined in the system. Rather than present details we will summarize such information by an axiom in the original logic that defines two precondition intervals (for the latch being open, and the door being shut), and one effect interval (for the door being open):

PullOpen Axiom 0: Temporal Structure
$\forall e,t$. *PullOpen(e,t)* $\supset$ *OverlapsDuring(pre1(e), t)* $\wedge$
Meets(t,pre2(e)) $\wedge$ *Starts(eff1(e),t)*

Planning rules will be used by the system in both a forwards (i.e. from antecedent to consequent) and a backwards (from consequent to antecedent) chaining manner. To apply a rule in a backwards chaining manner, the rule's consequent C is unified with the goal. Then the constraints A1,..., An are added to the database and the antecedent literals D1,...,Dn are introduced as subgoals. To apply the rule in a forward manner, if a literal is added to the database that unifies with some antecedent literal Di, and all the other Dj ($j \neq i$) and the constraints A1,...,An are in the database, then we also assert the consequent C. For instance, rule (PullOpen.1) could be used to suggest that a goal [DoorOpen Do1] could be accomplished by an event E1 if we can prove [PullOpen E1 T1] under the constraint eff1(E1)=Do1. The same rule is also used to predict the consequence of the same event E1 occurring at time T1: ie., if [PullOpen E1 T1] is added then add [DoorOpen eff1(E1)].

This simple example illustrates the basic technique for generating plans - planning rules are used to backward chain to suggested actions, and then in a forward manner to compute the consequences of those actions. In addition, all the domain prediction rules are also used in a forward chaining manner to compute additional consequences of the action. For example, Domain constraint 1 above would be asserted as the following forward chaining rule:

[Disjoint ?t1 ?t2] <forward<
[DoorOpen ?t1] [DoorClosed ?t2].

All the other domain constraints can be expressed similarly.

There are several additional issues to consider before the final algorithm is specified. First, the planner must be able to create event structures as needed, since the appropriate events will not generally be known to occur in advance of the planning. This is accomplished during the backwards chaining phase: whenever a literal containing an unbound event variable is to be introduced as a goal, a new event constant is generated and the temporal structure associated with that event is added to the database together with any other constraints specified in the planning rule. As an example, given the goal [DoorOpen Do1], rule (PullOpen.1) suggests a subgoal of proving [PullOpen ?e ?t]. Before this is considered further, a new event, say E1, and a new time, say T1, are created and the following constraints are added to the database from the definition of the temporal structure of Stack events (StackingAxiom 0):

[OverlapsDuring pre1(E1) T1]
[Meets T1,pre2(E1)]
[Start eff1(E1) T1].

What we have done is create the temporal structure for an event that could accomplish the goal clause. We have not asserted that this event yet occurs. This will require further chaining to prove [PullOpen E1 T1]. This process of creating event and temporal constants to replace the unbound variables will be called *instantiating* the planning rule.

4.1 The Algorithm

The following algorithm defines a planner that does not commit to the persistence assumptions until the entire plan is otherwise complete. It uses the simple backwards chaining technique from the goals as described informally above, and forward chaining to compute the consequences of its assumptions about persistence and action attempts. Because the temporal aspects of the plan are independent of the planning algorithm, the simple back-chaining strategy does not restrict the plans that can be produced.

It consists of two main parts: the **plan generator**, which creates a particular plan and the **assumption verifier**, which takes a suggested plan and evaluates whether the persistence assumptions that support it still appear to be consistent. Let **GS** be the goal stack, which is initialized to the initial set of goals when the

algorithm is invoked. The output of this algorithm is a set of actions to be attempted by the agent (the **action list**), a set of assumptions about the world (the **assumption list**), and the world state generated by the prediction reasoner. Each persistence assumption consists of a literal P true over some time period T and an equality relation involving T that captures the persistence assumption.

Plan Generation

This is a non-deterministic version of the planning algorithm. A PROLOG-style search strategy to iterate through all possible proof paths can be added in the obvious way.

(0) Do until **GS** is empty; then go to verification stage: remove the top element of **GS** and call it **G**;

(1) Choose:

(1.1) If a formula unifying with **G** is found in the database, then bind any variables in **G** as necessary;

(1.2) If **G** can be proven by a persistence assumption, then pass **G** to the prediction reasoner, and add **G** together with the equality assertion that justifies the assumption to the assumption list;

(1.3) If **G** is of the form *Try(A,e,t)*, for some action A, then add **G** to the action list, pass **G** to the prediction reasoner;

(1.4) Find a planning rule **R** whose consequent unifies with **G**, instantiate the rule as defined above (i.e. binding the event and temporal variables and adding the constraints) and push the antecedents onto **GS**.

Verifying Assumptions

This algorithm uses the temporal reasoner to check that all the persistence assumptions appear to be globally consistent. It does this by first re-checking the temporal constraints for each assumption individually to see if it is still possible. It then adds all the assumptions together to see if they appear to be globally consistent (according to the temporal reasoning algorithm). If some assumptions are no longer consistent, the planning stage is re-activated.

(2.1) Check each persistence assumption individually to see if it is still possible given the current temporal network generated by the prediction reasoner. If not, add the literal associated with each assumption that is now impossible to **GS** and restart at step (0).

(2.2) (Given that step (5) succeeded) Add the persistence assumptions to the prediction reasoner.

Unless the prediction reasoner returns an inconsistency, we are done. If an inconsistency is found, then we must select an assumption to retract. Designing a good strategy for this is left as future work. For now we simply select an assumption at random. Remember that assumptions consist of a literal P, and an equality assertion t=t'. Given the selected assumption, add t≠t' to the prediction reasoner, add P to **GS** and restart at step (0).

5 THE DOOR-LATCH PROBLEM

One of the major goals of this work was allowing plans that necessarily required simultaneous actions. The Door-Latch Problem was posed as the simplest example of this type of situation. In this section, we show how the domain can be formalized and a plan constructed from first principles that will open the door. First we present the planning axioms that define the actions, and then give an overview of the solution.

Remember that the complication in this example is that the agent must realize that it must continue to hold the latch open while it is pulling on the door. The actions needed are turning the latch, holding the latch open, and pulling the door. Unless the latch is held open, it snaps shut. Given that the planner uses persistence assumptions about the world, some technique must be introduced to prevent the planner from using this technique to infer that the latch stays open. This would best be handled by adding some causal reasoning to the predictor, but a simpler technique can be used in this class of situations. We will define the turn-latch action such that its effect holds exactly for a moment, i.e. a non-decomposable period. Thus any action that requires the latch to be open for an extended period of time cannot accomplish this by persistence, since a moment cannot be equal to an interval by definition. The hold-latch action is then defined so that it requires the latch to be open at the time the action starts (which may be a moment) and has the effect that the latch stays open for the duration of the action. Specifically, we have the planning rules below which are used by the predictor to maintain the world representation.

The temporal structures for each event are axiomatized below and shown graphically in Figure 2. The *PullOpen* action was defined earlier. *TurnLatch* events have a precondition interval (for the latch being closed) and an effect moment (for the latch being open):

TurnLatch Axiom 0: Temporal Structure
$\forall e,t \; . \; TurnLatch(e,t) \supset Finishes(t,pre1(e)) \wedge Moment(eff1(e)) \wedge Meets(t,eff1(e))$.

HoldingLatch events define a single precondition period (for the latch being open) and an effect interval simultaneous with the event interval:

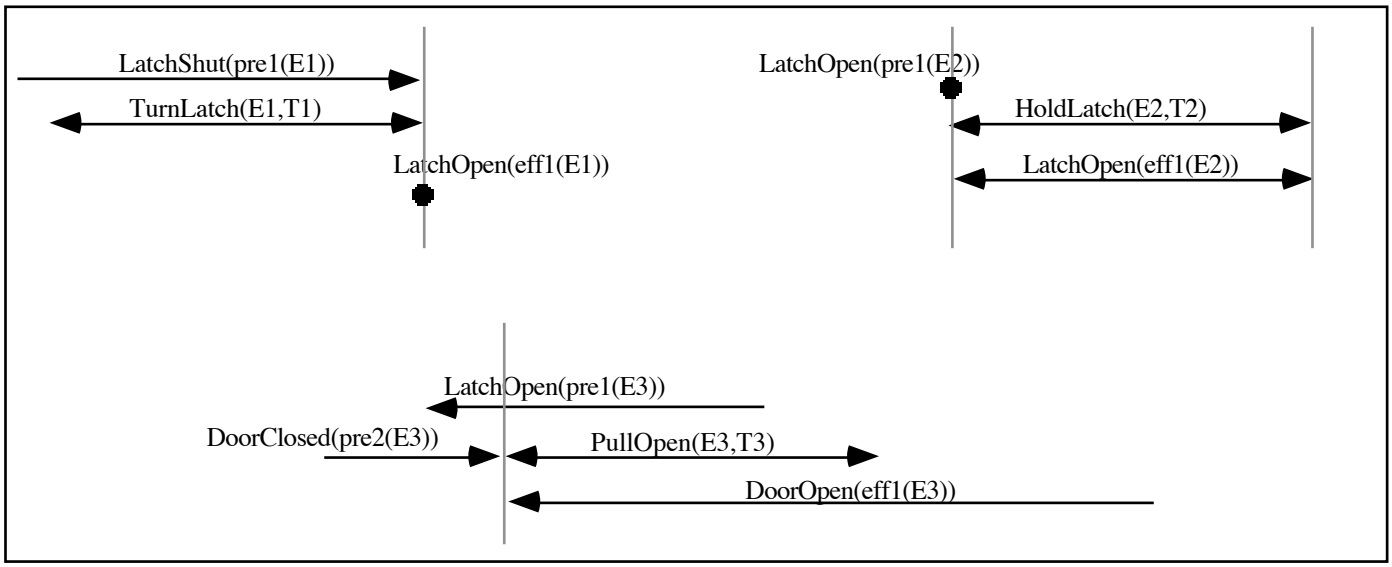

Figure 2: The temporal structure for three events in the door problem

HoldingLatch Axiom 0: Temporal Structure
∀ e,t . HoldOpen(e,t) ⊃ Meets(pre1(e),t) ∧ EQ(eff1(e),t)

The planning rules for these actions are as follows: Turning the Latch has the effect that the latch is momentarily open:

(TurnLatch.1)
[LatchOpen ?t'] <<<
[Moment ?t']
[TurnLatch ?e ?t]
such that [EQ ?t' eff1(?e)].

Turning the latch can be accomplished by trying to do it when the latch is shut:

(TurnLatch.2)
[TurnLatch ?e ?t] <<<
[LatchShut pre1(?e)]
[Try [turnlatch] ?e ?t].

The latch remains open if and only if it is held open. In particular, note that the effect and the action in this rule must be simultaneous:

(HoldOpen.1)
[LatchOpen ?t'] <<<
[Interval ?t']
[HoldOpen ?e ?t]
such that [EQ eff1(?e) ?t'].

Holding the latch open succeeds whenever the latch is open at the start of the holding act:

(HoldOpen.2)
[HoldOpen ?e ?t] <<< [LatchOpen pre1(?e)]
[Try [holdopen] ?e ?t]

Assuming a situation, in which the agent is near the door , the initial world description would be as follows, where I is the current time, and G is the time when the goal must hold:

[In I ls1] [LatchShut ls1]
[In I dc1] [DoorClosed dc1]
[Before I G].

The goal is simply to have the door open over time G, i.e. [DoorOpen do1] such that [In G do1]. The initial planning situation is shown in Figure 3.

Here's a brief sketch of the planner in operation. Given the goal [DoorOpen do1], rule (PullOpen.1) applies and introduces the subgoals after instantiation:

GS: [PullOpen E1 T1]

where [EQ eff1(E1) do1] and the temporal

Figure 3: The Door-latch problem

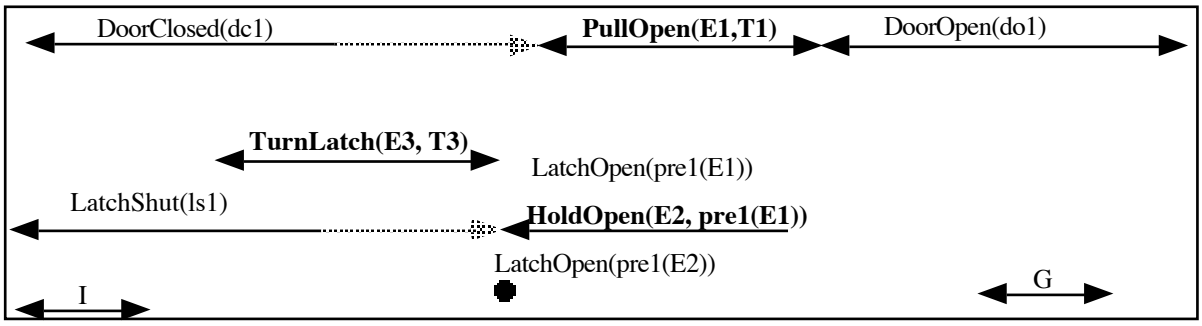

Figure 4: The solution to the Door-Latch Problem

constraints for the new PullOpen event E1, i.e. [OverlapsDuring pre1(E1) T1}, [Meets T1 pre2(E1)] and [Starts eff1(E1) T1] are added to the database

The subgoal [PullOpen E1 T1] can be proven by rule (PullOpen.2) producing the new set of subgoals:

GS: [DoorClosed pre2(E1)]
[LatchOpen pre1(E1)]
[Try [pull] E1 T1].

The first subgoal is proven by persistence from the initial state, using the assumption that pre2(E1)=dc1. The second subgoal, [LatchOpen pre1(E1)], requires further planning. Rule (TurnLatch.1) cannot apply here as it requires the interval pre1(E1) to be a moment. Rule (HoldOpen.1) does apply, however, and introduces the subgoal [HoldOpen E2 pre1(E1)] (note that by the planning rule pre1(E1)=eff1(E2), which in turn equals the time of the HoldOpen event by its definition). Rule (HoldOpen.2) then applies to this subgoal and introduces the following subgoals after instantiation:

GS:[LatchOpen pre1(E2)]
[Try [holdopen] E2 pre1(E1)]
[Try [pull] E1 T1].

This time, rule (TurnLatch.1) can apply (since pre1(E2) can be a moment) and the action [TurnLatch E3 T3] is introduced. After using rule (TurnLatch.2) to reduce this goal, the following subgoals remain:

GS: [LatchShut pre1(E3)] [Try [turnlatch] E3 T3]
[Try [holdopen] E2 pre1(E1)]
[Try [pull] E1 T1].

The first of these subgoals can be proven by persistence, since it is possible that pre1(E3)=ls1, and the remaining three subgoals are trivially proven as they are under the control of the planner. As each of these is assumed, it is added to the database triggering the forward-chaining prediction rules. As a result, the door is predicted to be open at time do1.

Finally, the persistence assumptions must be verified, and then added to the predictor producing the final plan as shown in Figure 4, with the persistence assumptions shown in grey. Note that the Pull action must start within the time when the HoldOpen action occurs, as desired. If this were not the case, the final effect, namely that the door is open, would not be predicted by the prediction mechanism. Thus we've shown that the planner can find the correct plan, and that it would not accept the faulty plan that would arise from a STRIPS-style planner, or from a naive persistence mechanism.

6. PLANNING WITH EXTERNAL EVENTS

Another simple example of some interest shows that the planner can co-ordinate with external events that it knows will occur sometime in the future. For example, consider a different initial situation, which is the same as before except that the planner knows that the door is unlocked automatically between 8AM and 9PM every day, and the goal is to get the door open sometime between 7AM and 9AM. This situation is shown in Figure 5, where the times of day are represented by moments.

The initial database consists of the following assertions:

[In I ls1]
[LatchShut ls1] [Meets ls1 lo1]
[LatchOpen lo1] [Meets ls1 8AM]
[In I dc1] [DoorClosed dc1]
[Before I 7AM] [Before 7AM 8AM
[Before 8AM 9AM] [Before 7AM G]
[Before G 9AM] [Moment 7AM]
[Moment 8AM] [Moment 9AM].

The initial goal is as before, to accomplish [DoorOpen do1] such that [In G do1]. Using rule (PullOpen.1) we get the subgoal of [PullOpen E1 T1], where eff1(E1)=do1. Rule (PullOpen.2) gives two preconditions for this action, namely

[DoorClosed pre2(E1)]
[LatchOpen pre1(E1)].

In this case, both can be proven by persistence. [DoorClosed pre2(E1)] would be true if pre2(E1)= dc1, and [LatchOpen pre1(E1)] would be true if pre2(E1)=lo1. Adding these assumptions creates a plan that involves pulling the door after 8AM (since the

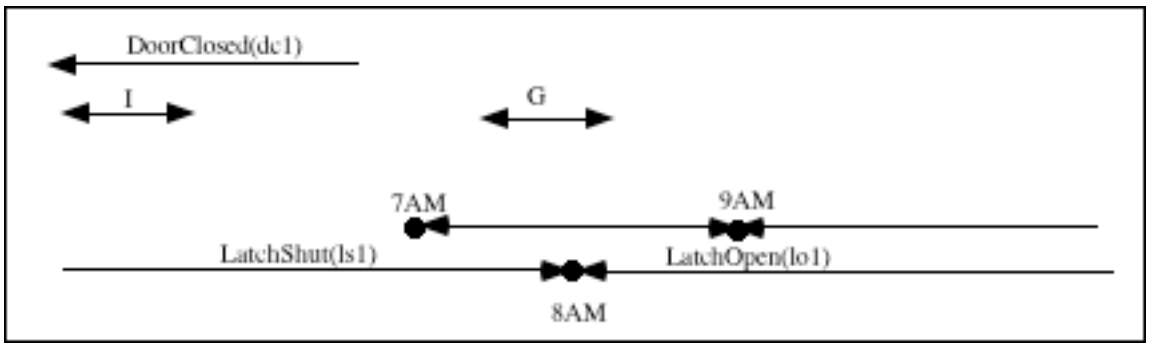

Figure 5: The door problem with an automatic latch system

Latch must be open) and before 9AM (to satisfy the goal conditions on G). Thus we have constructed a plan that takes advantage of the automatic latch, a known event in the future, by scheduling its own actions to take advantage of the latch being open. If, on the other hand, the goal had been to open the door before 8AM, i.e. G is constrained to be before 8AM, then this plan will not be suggested since the persistence assumption pre1(E1)=lo1 is not consistent with the database.

7 DISCUSSION AND EXTENSIONS

7.1 Persistence Assumptions

One of the critical techniques used to construct plans was the use of assumptions about the equality of times. In essence, this technique allows a time period to extend as far as possible given its constraints to other periods. Thus, if a proposition P is known to be true over an interval that contains a time t, and nothing is known to prevent P remaining true after t, then an assumption may be made that P is true after t if needed in a plan. Similarly, if nothing prevents P from being true before t, an assumption might be made that P was true some time before t. This capability to extend forwards or backwards in time might seem strange at first, but is quite useful in tasks that require plan recognition, or planning in worlds where information has to be acquired.

In Allen et al (1991), we show that this technique corresponds to the technique of Dean & McDermott [1987] if the persistence is into the future. The differences arise in two cases: first when there is uncertainty as to the starting time of the property, and second when a property is proven by extending backwards. Thus, the interval persistence rule is considerably more liberal than the rule used by Dean and McDermott. To handle these latter cases, Dean introduces another mechanism based on abduction. The different cases, however, seem to all reflect the same interactions, so a single uniform method for handling them seems preferable. In addition, the interval persistence rule is considerably simpler to describe and analyze. Situations requiring the more general rule appear frequently in everyday situations.

For example, if we are told that our airline tickets will be at the Bursars Office at 3 on Tuesday, then that suggests that they might be there earlier - it depends on the unknown information about when they were delivered. Similarly, being there at 3 suggests that the tickets will be there after 3 as well, and how long depends on unknown information about when they were picked up. In a single-agent domain, we have a high degree of confidence that the tickets remain at the office until we pick them up, since no other agent exists to pick them up. Note, of course, in a single agent domain, there wouldn't be an agent to deliver the tickets to the office in the first place, so that the tickets would need to be at the office in the initial situation. Thus, with a single agent, and a completely defined initial world, there is a strong bias to persistence only into the future. With multi-agent worlds, and partially defined situations, extension into the past becomes an equally important technique.

Of course, a simple persistence technique like this has problems. Using logical consistency is too weak a measure for the plausibility of a persistence assumptions. Rather, it would be better to evaluate the likelihood of a persistence using a causal theory, or a probabilistic method such as in Dean & Kanazawa[1988]. Note that since the assumptions are explicitly part of the final plan, such techniques could be introduced into this framework to produce a likelihood that a given plan will succeed if attempted.

7.2 A Hierarchical Planning Algorithm

The representation and algorithm above can easily be extended to include reasoning based on action decomposition as found in hierarchical planners such as NONLIN (Tate, 1977), SIPE (Wilkins, 1988) or FORBIN (Dean, Firby & Miller, 1989).

We could do this by introducing axioms that allow us to prove formulae of the form *[Try a e t]* rather than assuming them. But it turns out that the algorithm is easier to specify if we introduce a new predicate

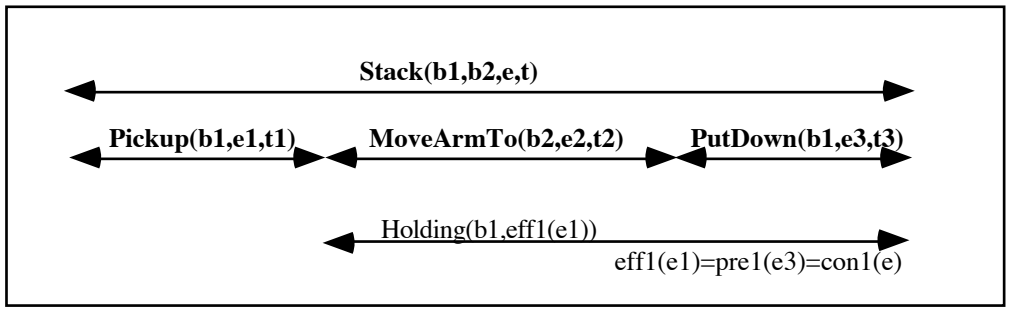

Figure 6: The decomposition of the Stack Action

Decomp on actions. The relation between *Decomp* and *Try* is that you try an action by performing one of its decompositions, ie.

$$\forall e,t,a \;.\; Decomp(a,e,t) \supset Try(a,e,t).$$

For example, a stacking action is accomplished by moving the arm to the block desired, opening the hand, lowering the hand over the block, grasping the block, raising the arm and moving it to the desired destination, and then lowering the arm and opening the hand again. Figure 6 shows the decomposition of the Stack action together with the necessary persistence assumptions required to make the decomposition effective. In particular, the effect of picking up b1, namely *Holding(b1,eff1(e1))*, must extend to satisfy the precondition on the PutDown action. In addition, this effect is identical to the constraint originally defined for the Stacking action.

We capture this information by adding a planning rule that specifies this as one way to decompose the action:

Stack Decomposition Axiom

[Decomp [stack ?x ?y] ?e ?t] <<< [Pickup ?x ?e1 ?t1] [MoveArmTo ?y ?e2 ?t2] [PutDown ?x ?e3 ?t3] such that [Meets ?t1 ?t2][Meets ?t2 ?t3][Starts ?t1 ?t] [Finishes ?t3 ?t] [EQ eff1(?e1) pre1(?e3)] [EQ eff1(?e1) con1(?e)].

The only other complication is that the initial algorithm used a second stage to verify that all persistence assumptions made in the plan were still consistent. We could leave this second stage until the entire plan is decomposed, but it is more in line with traditional hierarchical planners to verify these assumptions at each decomposition level before the next level is constructed. This can be accomplished simply in the new algorithm by adding a "dummy" goal on the bottom of the goal stack that invokes the consistency checking algorithm. When this goal rises to the top of the stack, one complete level of decomposition has been completed. The constraints are checked and the dummy goal is added again at the bottom of the goal stack to signal the end of the next level of decomposition. We will call this dummy goal [VerifyAssumptions]. A precise specification of the algorithm is as follows:This algorithm is a slight variation of the earlier algorithm. It differs in how action attempts are treated, and in the time that assumptions are verified. As before, this is a non-deterministic version of the algorithm, and the goal stack **GS** is initialized to the goal statement.

Do until **GS** is empty:

(0) Remove the top element of **GS** and call it **G**;

(1) Choose

(1.1) If a formula unifying with **G** is found in the database, bind any variables in **G** as necessary.

(1.2) Otherwise, if **G** can be proven by a persistence assumption, then pass **G** to the prediction reasoner, and add **G** together with the equality assertion that justifies the assumption to the assumption list.

(1.3) Otherwise, if **G** is of the form *Try(A,e,t)*, for some action A, add **G** to the action list and pass **G** to the prediction reasoner. Also, if there are axioms with a consequence of form *Decomp(A,e,t)* in the database, add *Decomp(A,e,t)* to the *end* of **GS**.

(1.4) If **G** = [VerifyAssumptions], then invoke the assumption verifier. (Note, if verifying the assumptions fails, then **G** is not achieved and the algorithm backtracks). Unless **GS** is now empty, add a new goal [VerifyAssumptions] to the *end* to **GS**.

(1.5) Otherwise, find a planning rule **R** whose antecedent unifies with **G**, instantiates the rule as defined above and push the antecedents of **R** onto **GS**.

This algorithm expands a plan level-by-level through a decomposition hierarchy, validating the consistency of the plan at each level before the next level was addressed. Constraints imposed by the higher levels

makes the accomplishment of the actions at the lower levels considerably easier.

8 CONCLUSIONS

We showed how traditional planning systems could be recast fairly directly as a specialized inference process on a temporal logic. By doing this, we have produced a framework that is much easier to understand and extend. By separating the temporal aspects of a plan from the procedural aspects of plan construction, for example, we found that even the simplest backwards chaining planning algorithm can generate non-linear plans. Similarly, a hierarchical planner can be generated by changing the set of assumptions about action attempts that the system is willing to make at any given time. As such, this work provides a uniform framework for examining many of the different planning frameworks developed to date.

While the actual system described duplicated the abilities of traditional planning algorithms, the situations that can be represented and reasoned about are more general than can be represented in a state-based model. In particular, we can reason about plans involving complex interactions between overlapping actions. It can reason about the effects of simultaneous actions that are not the effect of any one of the actions individually. The representation is limited, however, in representing partial interference between actions. This is because the current representation cannot explicitly capture the notion of possibility (as found in branching time models) and from the simple technique used for generating persistence assumptions.

By separating the domain reasoning from the plan construction algorithm, we have developed a general representation for reasoning about action that is independent of the particular application that is driving it. A plan recognition system could use the same action definitions in the same representation. Plan recognition can be viewed as just another specialized inference process on this same world representation.

More details on all these topics can be found in Allen et al (1991).

Acknowledgements

Thanks to George Ferguson and Nat Martin for comments on a draft of this paper. This work was supported in part by the Air Force Systems Command, RADC (now Rome Laboratory) and the Air Force Office of Scientific Research, under contract no. F30602-85-C-0008, and NSF grant N00014-90-J-1811.

References

Allen, J.F. "Maintaining knowledge about temporal intervals," *CACM 26*, 11, 832-843, 1983.

Allen, J.F. "Towards a general theory of action and time," *Artificial Intelligence 23*, 2, 123-154, 1984.

Allen, J.F. and P.J. Hayes. "A common-sense theory of time," *Proc. IJCAI 85*, Los Angeles, CA 1985.

Allen, J.F. and J.A. Koomen, "Planning using a temporal world model", *Proc. IJCAI 83*, Karlsruhe, Germany, 1983.

Allen, J.F.,H. Kautz, R. Pelavin, and J. Tenenberg *Reasoning.About Plans* Morgan Kaufmann, 1991.

Bacchus, F., J. Tenenberg and H. Koomen, "A non-reified temporal logic", *Proc. of the First Int'l Conf. on Principles of Knowledge Representation and Reasoning,* Morgan Kaufmann, 1989.

Davidson, D. "The logical form of action sentences," in N. Rescher (ed.). *The Logic of Decision and Action.* U. Pittsburgh Press, 1967.

DeKleer, J. "An assumption-based TMS," *Artificial Intelligence 28*, 127-162, 1986.

Dean, T., J. Firby and D. Miller. "Hierarchical planning involving deadlines, travel time and resources," *Computational Intelligence,* 1990.

Fikes, R.E., and N.J. Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving," Artificial Intelligence 2 231-272, 1971.

Harel, D. "Dynamic logic," in *Handbook of Philosophical Logic, Vol. II*. Reidel, 1984.

Koomen, J.A. "Localizing temporal constrain propagation", *Proc. of the First Int'l Conf. on Principles of Knowledge Representation and Reasoning,* Morgan Kaufmann, 1989.

McCarthy, J. and P. Hayes "Some philosophical problems from the standpoint of artificial intelligence" in *Machine Intelligence 4*. Edinburgh University Press, 1969.

McDermott, D. "A temporal logic for reasoning about processes and plans," *Cognitive Science 6,* 2, 101-155, 1982.

McDermott, D. "Reasoning about plans," in J.R. Hobbs and R.C. Moore (eds.). *Formal Theories of the Commonsense World*, Ablex, 1986.

Pelavin, R. "A formal approach to planning with concurrent actions and external events," TR 254, Computer Science Dept., U. Rochester, 1988.

Rosenchein, S.J. "Plan synthesis: A logical perspective," *Proc. IJCAI,* 331-337. Vancouver, British Columbia, 1981.

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

Shoham, Y. "Temporal logics in AI: Semantical and ontological considerations," *Artificial Intelligence 33,* 1, 89-104, 1987.

Tate, A. "Generating project networks," *Proc. IJCAI,* 888-93, Cambridge, MA, 1977.

Vere, S. "Planning in time: Windows and durations for activities and goals," *IEEE Trans. Pattern Analysis Mach. Intell. 5,* 3, 246-67, 1983.

Waldinger, R. "Achieving several goals simultaneously", in Elcock,E & Michie, D (eds), *Machine intelligence 8*, Ellis Horwood, pp 94-136.

Wilkins, D. *Practical Planning,* Morgan Kaufmann, 1988