This is the published version:


Available from Deakin Research Online:

http://hdl.handle.net/10536/DRO/DU:30044793

Reproduced with the kind permissions of the copyright owner.

Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Copyright : 1991, IEEE
Planning with Events and States

Dorota H. Kieronska, Svetha Venkatesh and C.P. Tsang

School of Computing Science
Curtin University of Technology
PO Box U1987
Perth, Western Australia

* Dept. of Computer Science
University of W.A.
Nedlands 6009
Western Australia

We present an overall planning system in which specifications can be described in terms of events and states. The underlying feature of this system is temporal logic, and its expressive power allows us to deal with simultaneous actions and interacting actions. Moreover, we can represent both goal-oriented positive constraints and prevention-oriented negative constraints. The planning system can generate hierarchical plans and the overall model is capable of handling interacting agents.

1. Introduction

The goal of a planning process is to describe a set of actions (or plan) that can take an agent in a system from an initial state to a goal state. To enable such a system to be meaningful, both events and temporal constraints of the domain need to be represented. Several researchers have worked towards the goal of formalising the theory of action and time [1, 2, 3, 12].

In this paper we propose a system within the framework of temporal logic that automatically generates all possible plans of action subject to a given set of causal constraints.

The main contribution of this research is the development of an overall planning system that adequately deals with major problems traditionally associated with domain-independent planning [11, 13]. The intrinsic properties of temporal logic allow for the representation of both goal-oriented positive constraints, as well as prevention-oriented [4] negative constraints. The merits of this formalism lie in the increased flexibility provided to the user, by permitting the specification of constraints in terms of both events and states. Further, the expressive power of this specification method provides the user with the means of expressing constraints to be satisfied after a sequence of actions has occurred. This is a marked improvement from traditional methods, where such constraints can be invoked only due to an occurrence of a single event or state. The plans satisfying the given constraints can be represented as directed graphs. This representation provides natural means of dealing with hierarchical plans.

This paper is structured as follows. In section 2 we describe propositional temporal logic and the planning system. The framework for mapping the user's conceptual model onto mixed event-state formal specifications is defined in section 3. Our conclusions follow in section 4.

2. The Plan Generator

Planning involves describing and reasoning with actions occurring over time. The information about the relative ordering of actions is retained, but the issues of real time are not considered. A natural framework of dealing with the temporal ordering of events is provided by temporal logic.

The underlying model for our planning system is based on Propositional Temporal Logic (PTL) [8]. In this section we briefly describe the syntax and semantics of PTL, followed by the description of a program synthesis method [5, 14] that allows for the generation of all sequences of events satisfying the original specifications. In case of planning, the specifications define all constraints on the desired plan and the result from the synthesis method is a graph representing all plans satisfying these constraints.

2.1. PTL syntax.

Linear time propositional temporal logic is used in this method. The operators used describe only the future. A PTL formula can be recursively defined, for a set of atomic propositions P:

- A proposition p ∈ P is a PTL formula
- for F₁ and F₂ being PTL formulae, ¬F₁, F₁ V F₂, F₁ A F₂, F₁ ⇒ F₂ are also PTL formulae;
- if F₁ and F₂ are PTL formulae then the following also are PTL formulae:
deadlock detection onto propositions in PTL. An individual formula can occur at this point of time. The current state. This is equivalent to deciding which events can occur at this point of time. The next part defines the condition to be satisfied by the remaining sequence of events. The process of graph construction is initiated by an eventuality is a formula containing an operator. A set of PTL formulae, which we will refer to as specifications, define the behaviour of a system of agents. Each agent can perform some actions, which are mapped onto propositions in PTL. An individual formula represents some constraint on the behaviour of one or more agents.

We distinguish constraints as being local with respect to an agent, or global with respect to the complete system in which the agents are operating. While global constraints are not required for the production of plans for any particular agent, they are necessary for the overall system to work.

2.3. Hierarchical plans.

In many cases it is desirable to generate the final plan in stages: first construct an overall plan, where events (PTL propositions) represent some complex actions, then replace some (or all) of these actions with plans of individual (atomic) actions. This can be done in any number of stages, not necessarily two. For example a general plan of action might consists of events like: cross-the-street, enter-the-building, get-to-floor-10, deliver-the-package. Then cross-the-street might involve: decide-if-safe-to-cross, get-to-the-other-side, climb-the-curb. And then again decide-if-safe may consist of: look-to-the-right, look-to-the-left, if-safe-move-forward.

The interpretation of the PTL propositions, which can be thought of as the factor determining the correct level within the hierarchy, has no effect on the graph generation process. Creating a hierarchy of plans can be described as a process involving:

(i) generating a plan using events and constraints defining the actions at the desired level of hierarchy (originally at the coarsest level)

(ii) generating plans consisting of next level actions, to replace some actions at the current level; no action at lower level can have the same name as an action at any of the previous levels;

(iii) replacing each edge in the current global graph, with a graph representing the sub-plan according to the rules specified below.

If the subplan must be executed in an uninterrupted manner then a straightforward replacement of an edge in the graph, with another graph representing a sub-plan is sufficient. However, in general such a replacement would violate the original constraints on the system. Consider the situation where in the graph there is a node with two edges leaving from it: clean-the-table and empty-the-box, i.e. the agent in the state represented by the node has two equally valid options nondeterministic choice. The plan involves performing both actions eventually, but the order does not matter. We will refer to such actions as independent. Now, if we try to replace clean-the-table with a plan involving putting away books, storing pens and throwing away scrap paper, then in such a case each of these new actions is independent of empty-the-box. There

OF₁ ("next" F₁) - F₁ becomes true in the next instant of time.
IF₁ ("always" F₁) - F₁ becomes true in the current instant of time and remains true in all subsequent instants of time.
<>F₁ ("eventually" F₁) - there is an instant of time, now or in the future, in which F₁ becomes true.
F₁ U F₂ (F₁ "until" F₂) - in some instant of time, possibly never, F₂ becomes true, and F₁ is true in all instants of time between now, inclusive, and that instant.

2.2. Global state graph generation.

If propositions represent events, a PTL formula defines a condition to be satisfied by a sequence of events (or actions) occurring in time. PTL is decidable and the decision procedure relies on the construction of all models that satisfy the given formula. These models are represented by a labelled directed graph, where the edges correspond to events. The specification represents a set of constraints and each constraint can be represented as a PTL formula. Since all the formulae must be satisfied, the specification can be treated as a conjunction of these formulae.

Any PTL formula can be decomposed into current and next states, where the current part is a classical Boolean expression and the next part is a PTL formula. The decomposition of PTL formulae is based on the following identities:

\[ \Box X = X \land O(\Box X) \]
\[ <>X = X \lor O(<>X) \]
\[ X_1 U X_2 = X_2 V (X_1 \land O(X_1 U X_2)) \]
\[ OX = true \land OX \]

The current component of the decomposed formula can be used to decide which propositions can be true in the current state. This is equivalent to deciding which events can occur at this point of time. The next part defines the condition to be satisfied by the remaining sequence of events. The process of graph construction is initiated by associating the original specifications with the first node in the graph. The method of graph construction is described in detail in [6]. The algorithms include deadlock detection (a situation where a node in the graph has no edges leaving from it) and unfulfilled eventualities, where an eventuality is a formula containing an <> operator.

The decomposition of PTL formulae is based on the following identities:

\[ \Box X = X \land O(\Box X) \]
\[ <>X = X \lor O(<>X) \]
\[ X_1 U X_2 = X_2 V (X_1 \land O(X_1 U X_2)) \]
\[ OX = true \land OX \]

The current component of the decomposed formula can be used to decide which propositions can be true in the current state. This is equivalent to deciding which events can occur at this point of time. The next part defines the condition to be satisfied by the remaining sequence of events. The process of graph construction is initiated by associating the original specifications with the first node in the graph. The method of graph construction is described in detail in [6]. The algorithms include deadlock detection (a situation where a node in the graph has no edges leaving from it) and unfulfilled eventualities, where an eventuality is a formula containing an <> operator.

A set of PTL formulae, which we will refer to as specifications, define the behaviour of a system of agents. Each agent can perform some actions, which are mapped onto propositions in PTL. An individual formula represents some constraint on the behaviour of one or more agents.

We distinguish constraints as being local with respect to an agent, or global with respect to the complete system in which the agents are operating. While global constraints are not required for the production of plans for any particular agent, they are necessary for the overall system to work.

In many cases it is desirable to generate the final plan in stages: first construct an overall plan, where events (PTL propositions) represent some complex actions, then replace some (or all) of these actions with plans of individual (atomic) actions. This can be done in any number of stages, not necessarily two. For example a general plan of action might consists of events like: cross-the-street, enter-the-building, get-to-floor-10, deliver-the-package. Then cross-the-street might involve: decide-if-safe-to-cross, get-to-the-other-side, climb-the-curb. And then again decide-if-safe may consist of: look-to-the-right, look-to-the-left, if-safe-move-forward.

The interpretation of the PTL propositions, which can be thought of as the factor determining the correct level within the hierarchy, has no effect on the graph generation process. Creating a hierarchy of plans can be described as a process involving:

(i) generating a plan using events and constraints defining the actions at the desired level of hierarchy (originally at the coarsest level)

(ii) generating plans consisting of next level actions, to replace some actions at the current level; no action at lower level can have the same name as an action at any of the previous levels;

(iii) replacing each edge in the current global graph, with a graph representing the sub-plan according to the rules specified below.

If the subplan must be executed in an uninterrupted manner then a straightforward replacement of an edge in the graph, with another graph representing a sub-plan is sufficient. However, in general such a replacement would violate the original constraints on the system. Consider the situation where in the graph there is a node with two edges leaving from it: clean-the-table and empty-the-box, i.e. the agent in the state represented by the node has two equally valid options nondeterministic choice. The plan involves performing both actions eventually, but the order does not matter. We will refer to such actions as independent. Now, if we try to replace clean-the-table with a plan involving putting away books, storing pens and throwing away scrap paper, then in such a case each of these new actions is independent of empty-the-box. There
is no reason why the agent could not proceed to empty the box after putting away books and then resuming cleaning the table.

In order to ensure that the current plan remains correct with respect to the original specifications, the independence of the events must be preserved. The details of the required algorithm can be found in [7].

3. Mapping Planning Constraints

To model planning, the relevant domain is described in terms of events and states. Events represent atomic actions that cause the state of the world to change. States represent the intervals of time that the system resides in, and a collection of all the states is a description of the system at any point in time. Every state change is associated with some event causing this change. And similarly, every event represents some state change (possibly more than one, e.g. firing a gun causes a gun to become empty and a bullet to be in a state of motion). This implicit mapping between states and events allows the user to freely mix state and event constraints within the specification, as they can all be easily translated into the event-based form required by our plan generator.

3.1. The primitive operators.

We define the operators B and E, to derive the starting and terminating events associated with a state as follows:

\[ B_{[\text{state}]} \] - returns the event that starts a state.

\[ E_{[\text{state}]} \] - returns the event that terminates a state.

To represent ordering on events, we define two primitive operations, coincides_with and starts_before, as follows:

\[ B_{[\text{state}1]} \text{ coincides_with } B_{[\text{state}2]} \]

\[ B_{[\text{state}1]} \text{ and } B_{[\text{state}2]} \] represent the simultaneous occurrence of two events. This is mapped onto a single event that causes changes in two states in the following manner. A new event \( B_{[\text{state}1,\text{state}2]} \) is created and each occurrence of \( B_{[\text{state}1]} \) and \( B_{[\text{state}2]} \) is replaced with \( B_{[\text{state}1, \text{state}2]} \). This new event must satisfy the constraints imposed on \( B_{[\text{state}1]} \) and \( B_{[\text{state}2]} \). The concept of two events occurring simultaneously cannot be included in the PTL specifications due to the assumptions underlying the plan generation algorithm, however it is a generic idea, often needed to express naturally occurring situations.

\[ B_{[\text{state}1]} \text{ starts_before } B_{[\text{state}2]} \]

This means that the event \( B_{[\text{state}1]} \) occurs before \( B_{[\text{state}2]} \). It is to be noted that an explicit ordering of the terminating events is redundant since those events can be interpreted as beginning events of the following states, as indicated in figure 1. It can be directly translated into a PTL formula: \( (\neg B_{[\text{state}2]} \cup B_{[\text{state}1]}) \)

![Figure 1. States and events.](image)

In TABLE I, we demonstrate how these simple primitives are sufficient to describe all the temporal relations on intervals as described by Allen [1] (X and Y are intervals).

3.2. Temporal implication.

To formulate all plans that can satisfy the conditions in a given domain, it is necessary to specify the set of possible events, the set of state transitions and a set of temporal constraints relating both actions and states. While some constraints must be satisfied by all events under consideration at all times, others must be satisfied after a particular sequence of events (possibly non-consecutive) occur. For example in any system in order to handle emergencies, it is essential to have the capability to deviate from the normal sequence of operations to take care of the emergency, and, if possible, return to the normal sequence once the emergency has been dealt with. To deal with such constraints, which cannot be expressed in terms of Allen's primitives, we introduce the concept of temporal implication, defined by the following operators:

\[ X \text{ followed_by } Y \]

\( X \) represents a temporal ordering of explicit state transitions \( a_1 ... a_n \) which appear within a sequence of the form: \( a_1 A_1 a_2 A_2 ... A_{n-1} a_n \). Each \( A_i \) is a sequence (possibly empty) of arbitrary transitions/events; \( Y \) is a temporal condition to be satisfied by the sequence of actions that will follow \( X \). Such a condition can be defined in PTL as:

\[ (O a_1 \Rightarrow (O a_2 \Rightarrow ...((O a_n \Rightarrow O Y) U a_n)...) U a_2) U a_1) \]

\[ X \text{ immediately-followed-by } Y \]

\( X \) represents an explicit sequence of transitions \( X \) being followed by a sequence satisfied by \( Y \). Such a condition can be defined in PTL as:

\[ ([a_1 \Rightarrow O(a_2 \Rightarrow ...O(a_n \Rightarrow Y)...))] \]
These operators can be intuitively thought of as temporal implication because of their similarity to classical implication. Whereas $X \Rightarrow Y$ describes a condition to be satisfied if $X$ is true at the current point in time (this is meaningless if $X$ is a temporal formula), $X$ followed by $Y$ is satisfied after a sequence of actions represented by $X$ has occurred.

### 3.3. Event and state based specification.

Earlier formal models for representing temporal causality rules are either strictly event based or state based. For example, in event based systems the specifications express temporal relationships of instantaneous actions, and such models have been used to model atomic database transactions [10]. In the alternative state based approach, where facts are unchanged over intervals of time, the specifications describe the temporal ordering of (possibly overlapping) intervals of time. Both these approaches require users highly skilled in formal specification methods to describe the considered domain. The specification process becomes more natural and simpler if the use of both events and states be permitted.

To start with, each of the events in the specifications directly maps onto a proposition. The states, however, require constraints defining the initial occurrence of the state, and possible subsequent occurrences of the state. For a state $S$ we have:

- a state must begin before it can end:
  
  $$(-E_{S}) \cup B_{S}$$

- once the state begins, in the future it can:
  
  a) never begin again:
  
  $$\langle B_{S} \Rightarrow O(\langle \neg B_{S} \rangle) \rangle$$

  b) or it can begin again subject to some constraint $C$:
  
  $$\langle B_{S} \Rightarrow O(\langle \neg B_{S} \rangle \cup C) \rangle$$

It should be noted that $C$ can be a simple constraint of the form of: "a state can begin as soon as it has finished". That is, $$\langle B_{S} \Rightarrow O(\langle \neg B_{S} \rangle \cup B_{S}) \rangle$$.

In Table 2, we show the mapping of mixed event and state specifications to equivalent event-based descriptions.

### 3.4 Advantages.

Using the mapping in TABLE 1 one can express typical goal-oriented constraints as used by Allen. Using our operators directly, we can also express prevention constraints of the following form:

- (i) $x$ cannot happen during the state $S$  
  
  $B_{S} \Rightarrow O(\neg x \cup B_{S})$  
  
  where $B_{S}$ represents the end of $S$ and beginning of $S'$;

- (ii) $x$ cannot happened until state $S$  
  
  $$\neg x \cup B_{S}$$

Simultaneous actions can be described using Allen's operators in terms of overlapping intervals (states). The approach we have presented here is more powerful and expressive than original due to Allen. Hierarchical planning which is a necessity in realistic and complex cases can be done with ease using our formalism. The user has the freedom to consider any level of hierarchy as appropriate without the need to regenerate a complete set of plans.

Another important issue resolved by our system involves the generation of component plans in the case of interacting agents in a cooperating system. The details are provided in [7].

### 3.3 Example.

To illustrate the use of our approach we present an example involving the Yale Shooting Problem [9].

<table>
<thead>
<tr>
<th>X before Y</th>
<th>$\neg B_{Y} \cup E_{X}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>X equal Y</td>
<td>$(B_{X} \text{ coincides with } B_{Y})$ and $(E_{X} \text{ coincides with } E_{Y})$</td>
</tr>
<tr>
<td>X meets Y</td>
<td>$E_{X} \text{ coincides with } B_{Y}$</td>
</tr>
<tr>
<td>X overlaps Y</td>
<td>$(\neg B_{Y} \cup B_{X})$ and $(\neg E_{Y} \cup E_{X})$ and $(\neg E_{Y} \cup E_{X})$</td>
</tr>
<tr>
<td>X during Y</td>
<td>$(\neg B_{Y} \cup B_{X})$ and $(\neg E_{Y} \cup E_{X})$</td>
</tr>
<tr>
<td>X starts Y</td>
<td>$(B_{X} \text{ coincides with } B_{Y})$ and $(\neg E_{Y} \cup E_{X})$</td>
</tr>
<tr>
<td>X finishes Y</td>
<td>$(E_{X} \text{ coincides with } E_{Y})$ and $(\neg B_{X} \cup B_{Y})$</td>
</tr>
</tbody>
</table>

Table 1: The mapping of temporal relations on intervals as defined by Allen into our system primitives.
problem is traditionally associated with the issue of extended prediction. However, if all possible plans of actions can be generated, an extended prediction can be made on the basis of analysing possible behaviours of the described system, which appear as paths in the generated graph.

The problem at hand involves a simple system consisting of two agents, namely a person called Fred and a gun. Initially at some point in time Fred is alive and the gun is unloaded. At some future point in time the gun is unloaded. The prediction question posed is: is Fred dead or alive? One way to attack this problem is by considering all possible plans of actions that can be taken from the initial state.

To define this problem we use the following events: load, unload, fire and kill. Each of the events is associated with some change(s) of state(s), as given below:

- `alive -> kill dead`
- `empty -> load loaded`
- `loaded -> unload empty`
- `loaded -> fire empty`

where `alive`, `dead`, `empty` and `loaded` are states.

In this simple model, we restrict ourselves to a simple case, where a gun can only be unloaded or fired; once it is fired or unloaded it cannot be fired again (i.e. pulling the trigger in an empty gun has no effect); Fred can only be killed once (i.e. after killing Fred, firing at him will have no effect). The implicit constraints on the state transitions comprise the bulk of the definition of our system. Full specifications in PTL are included in Appendix I. The graph representing all plans satisfying the constraints is shown in figure 2.
3.4 Implementation

The system described in this paper had been implemented in Common Lisp on a SPARC workstation. The input accepted is in the form of a list of possible events and PTL specifications as demonstrated in the example. In case of hierarchical plans, each even (PTL proposition) is also associated with participating agents in order to determine that events are independent [7].

4. Conclusions

In this paper we have presented an overall planning system in which specifications can be described in terms of both events and states. The underlying feature of our system is temporal logic, and its expressive power allows us to deal with simultaneous actions and interacting actions. Moreover we can represent both goal-oriented positive constraints and prevention oriented negative constraints. The overall planning system provides the user with flexible specification process. The model we have presented is capable of handling multiple agents, which is necessary to adequately describe complex systems with cooperating agents. Our current research is directed towards extending this model to incorporate real time.

5. References


Appendix I

PTL specifications of the Yale Shooting Problem.

(¬ kill) U load
(¬ (fire v load)) U load
[load => O (¬ load) U (unload v fire))]
[(fire v unload) => O (¬ (fire v unload)) U load)]
[unload => O (¬ kill) U load)]
[kill => O (¬ kill)]
[(load => O( (¬ kill) U fire) U (fire v unload))]