PDDL+: Modelling continuous time-dependent effects

Derek Long

Proc. 3rd International NASA Workshop on Planning and Scheduling for Space

PDDL+ : Modelling Continuous Time-dependent Effects

Maria Fox and Derek Long
University of Durham, UK
emails: maria.fox@dur.ac.uk and d.p.long@dur.ac.uk

Abstract

The adoption of a common formalism for describing planning domains fosters far greater reuse of research and allows more direct comparison of systems and approaches, and therefore supports faster progress in the field. A common formalism is a compromise between expressive power (in which development is strongly driven by potential applications) and the progress of basic research (which encourages development from well-understood foundations). The role of a common formalism as a communication medium for exchange demands that it is provided with a clear semantics. This paper describes extensions of PDDL2.1 (used in the 3rd International Planning Competition) that support the modelling of continuous time-dependent effects and illustrates why they can play a critical role in modelling real domains.

Introduction

This paper describes PDDL+, a significant extension of PDDL2.1 intended to support the representation of deterministic real time problem domains involving numeric-valued resources. PDDL2.1 supports the modelling of durative actions and other metric quantities and was used in the AIPS 2002 planning competition. It comprises a core drawn from McDermott's PDDL (McDermott & the AIPS'98 Planning Competition Committee 1998; McDermott 2000) supplemented with numeric and durative action extensions (Fox & Long 2002). PDDL2.1 is limited to the discrete modelling of time. The only time points that can be identified in a plan are those associated with the start and end points of actions selected by the planner. The language cannot represent exogenous events and is therefore of rather limited utility for modelling realistic problems in which activity is constrained by factors outside the planner's control.

An important contribution made by PDDL+ is the ability to model predictable exogenous events. This enables a planner to reason about the consequences of certain execution failures and to plan to avoid, or to exploit, consequences of its actions that will be brought about by the world. For example, the event of an instrument malfunctioning will occur when the temperature of a satellite drops below a threshold. A planner can plan to avoid this happening by ensuring that it maintains its temperature above this threshold using a charged battery. This may involve advance planning to ensure the battery is adequately charged to heat the satellite during its entire period out of sunlight.

PDDL+ is powerful enough to model domains containing both discrete and continuous behaviours. The key extension that PDDL+ provides is the ability to express temporal behaviour in terms of the initiation and termination of *processes* that act on the numeric components of states. Concurrent processes can interact resulting in continuous change. Logical state changes are effected by the instantaneous initiation and termination activities marking the end points of active processes. PDDL+ supports the modelling of situations in which PDDL2.1 is insufficiently expressive and which capture important aspects of time-critical concurrent behaviours.

In this paper we discuss the features of PDDL+ and demonstrate its use in modelling situations in which there is continuous change. We compare the operator-centric view of PDDL+ with some related work in the modelling of action and change and explain the advantages of the operator-centric view for planning.

Modelling Continuous Time

In PDDL+ one of the numerically varying quantities is time. The objective is to be able to model continuous processes as these arise in planning problems. An agent must be able to interact with continuously changing quantities and to plan with up-to-date information about their values. The model of time in PDDL+ is therefore real-valued and continuous.

Several researchers have addressed the problem of modelling time and numeric quantities in a discretized way (Bacchus & Kabanza 2000; Smith & Weld 1999). The basic idea is to add a value to an action specification indicating the duration of that action. For example, the action of a rover driving between two waypoints would have associated with it a duration of, say, ten minutes and an asso-

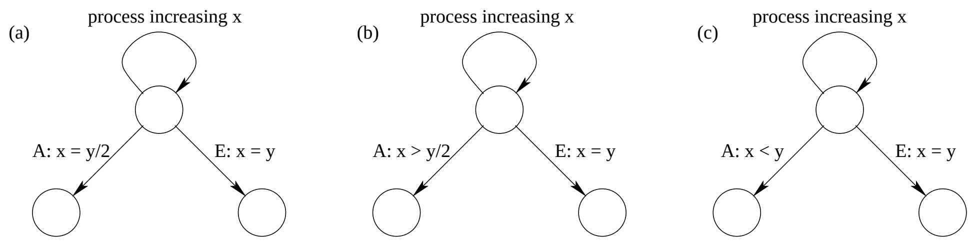

Figure 1: Interactions between actions and events. In each figure, A is an action and E is an event, each with indicated numeric precondition, where y is a constant and x is increasing under the influence of an active process.

ciated energy consumption. A difficulty immediately arises in describing the effect of energy consumption on the energy level of the rover.

Using discretized effects it is necessary to use a step function to model the energy consumption. If the step function is applied at the start of the action execution then the energy level will appear lower than it actually is for the duration of the action, while if it is applied at the end then the energy level will appear much higher. In either case, concurrent actions that depend on the energy level will be forced to consult an inaccurate value leading to possibly flawed behaviour. The safe way to avoid this problem is to adopt the conservative requirement that the energy level is undefined during the execution of the action, forcing actions that must access its value to be sequenced before or after the move.

If an action is associated with a duration then the action will have its effect at the end of that duration without further intervention by the planner. This prevents the modelling of effects that occur as the consequence of an action initiating some process that then needs to be explicitly terminated. In many realistic situations the environment itself brings about change as a consequence (perhaps unintended) of the activities of an executive. For example, sinks will overflow if the water source into them is left open (Shanahan 1990) and soup bowls will spill if they are tilted far enough in one direction without sufficient compensating tilt on the other side (Gelfond, Lifschitz, & Rabinov 1991).

To enable the correct modelling of these situations we have supplemented PDDL2.1 with two additional modelling components: *processes* and *events*. In PDDL+, as in PDDL and its precedents, non-durative actions have instantaneous effects that result in a change in the logical state of the domain. If change is modelled as deterministic the planner has perfect knowledge about action outcomes. Of course, this is a simplification of real world behaviour but it is the classical assumption upon which PDDL+ is based. When grounded (to remove quantifiers and conditional effects), the action semantics can be supplied by straightforward mappings to finite transition systems. Actions can also have numeric pre- and post-conditions which both consult and modify numeric variables.

Like actions, events are modelled as instantaneous state transition functions which can have numeric pre- and post-conditions. They are distinguished from actions only by the fact that the planner cannot select them in the development of a plan. This enables change in the domain to be modelled by means of a transition system in which some of the transitions perform mappings between states (actions and events do this) whilst others model continuous numeric change. PDDL+ domains can be modelled as hybrid automata, providing a simple and clean semantic basis for the language.

Actions and events can initiate the execution of processes that then run over time. A process maintains the logical aspects of a state whilst modifying numeric aspects of the state as time passes. Processes have to be terminated, either by the deliberate action of the executive or by the environment itself. *Events* are not under the control of the executive — their role in a plan is to signal the occurrence of predictable exogenous events. For example, a foreseeable event can be triggered to happen at a given absolute time. Events that are the consequences of initiated actions, such as the event of a sink overflowing, a soup bowl spilling, an instrument becoming over-exposed to sunlight or of a rover becoming sun-lit, occur because their preconditions become satisfied after some period of time in which one or more processes have been active.

In hybrid automata theory no distinction is made between actions (under the control of the executive) and events (under the control of the world). In planning, the key difference between them is that no matter what plan the executive is executing events will always take priority over actions if they become enabled in a state which also satisfies the preconditions of an action. Figure 1 shows how conflicts between actions and events are resolved in the interpretation of PDDL+ plans.

A crucial detail of the semantics is that it ensures that when the preconditions of an event are satisfied then that event must be the next transition executed. This gives rise to several questions concerning the relationship between actions and events. Figure 1 depicts three situations in which there might be considered to be an action/event conflict. In

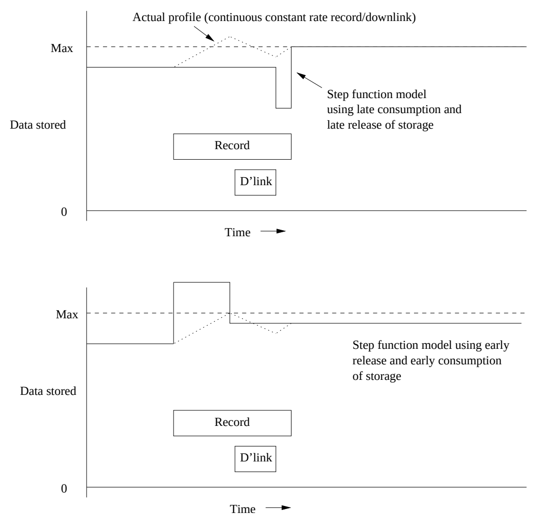

Figure 2: Discrete models of continuous behaviours fail to adequately account for concurrent interactions.

part (a) we consider the action A to be ill-defined because it relies on the executive being able to apply the action at a precise point in time: the precondition $x = y$ is true instantaneously and to exploit it would require the capacity to measure time to an arbitrary degree of accuracy. We argue that the executive does not have access to this level of precision, and that only the world itself is able to synchronize its activity precisely. In part (b) A and E are well-defined and can each occur. If the plan includes A timed to execute at the moment when $x = y$ then this is valid provided that the order in which A and E in fact occur does not affect the outcome (ie: both orderings result in the same state being reached). In part (c) there is no conflict between A and E since they are never applicable at the same instant. Note that events can have preconditions that are precisely synchronized with the world because they are not under the control of an executive — the world has the capacity to respond to situations with arbitrary precision.

Processes and events are distinguished by the fact that the numeric post-conditions of events cannot be time-dependent whilst those of a process always are. A process can be terminated by an action, or by some event that occurs in the world (such as a pan boiling dry, or a vehicle running out of fuel). The introduction of processes and events means that activities resulting in change occuring over time are modelled, in PDDL+, by a three-part structure consisting of a point of initiation, a process and a point of termination. The points of initiation and termination can be the points of application of actions or events or they can be the points at which the effects of active processes cause numeric values to reach critical thresholds. We refer to this as the *start-process-stop* model.

PDDL+ supplies additional expressive power over earlier PDDL variants including PDDL2.1. Its primary contribution is the ability to model the occurrence of events that are produced by the environment instead of as the direct consequences of the executive's action. This makes it possible to model many situations that lie outside the capabilities of less expressive languages. In particular, it is possible to model domains in which events that the world controls can be exploited by the planner to achieve a desired effect.

Processes, Events and Durative Actions

The three examples described in this section demonstrate the need for the three main modelling components of PDDL+: durative actions with flexible durations, events and processes. In the following we will refer to actions the durations of which can be determined by the planner as *flexible* durative actions. All of these components support the modelling of continuous change but flexible durative actions can encapsulate this change when the period of time over which a quantity is changing, and the point at which it will terminate, can be predicted by the planner and the change itself is under the control of the planner. Events are required when changes are triggered that lie outside of the control of the planner, and which the planner might need to plan to avoid. Finally, processes are required when an event initiates a period of change.

We present the three examples by first motivating them and then describing fragments of PDDL+ which successfully capture the intended behaviours. We also present parts of the hybrid automata that model these descriptions, in order to provide some semantic intuitions. A detailed BNF description of PDDL+ is available from (Fox & Long 2002).

Modelling Data Downlink in the Satellite Domain

In the satellite observation scheduling domain one of the important features is that downlink opportunities coincide with windows of time in which the satellite is in communication with a ground station. The times at which these windows will open and close can be computed from other information, such as the orbit of the satellite and the positions of the ground stations. Therefore the opening times can be predicted and a planner must make use of the opportunity to downlink data if it wants to maximise the amount of data collected and transmitted (given that it has a finite amount of storage space on board).

The purpose of this example is to demonstrate why it is necessary to have the ability to model continuous change. Suppose that the behaviour of the on-board storage medium has been abstracted to a level at which reading from and writing to store can be treated as happening concurrently. In this situation it is possible for the satellite to make observations and downlink stored data concurrently if an observation window overlaps any part of a downlink window.

At first sight it might appear that the storage and downlink of data in the situation just described can be modelled using durative actions with step function effects. Figure 2 shows how records and downlinks can be performed concurrently, both accessing a data store with a maximum capacity. As can be seen from the figure, the step function effects make it impossible for the true status of the storage device to be correctly modelled. The storage capacity of the device might be exceeded at the point where the step function is applied. Equally, the data might be downlinked before it has actually been recorded. The reason for the difficulty is that the recording and downlinking of data are really continuously affecting the amount of stored data in a way that tightly interacts.

The planner can determine the point at which the storage capacity would be exceeded and plan to halt the recording process before this consequence occurs. In order to do this it is necessary to model the recording action as a durative action the duration of which is determined by the time it takes to make the desired observation. The action terminates when the invariant condition (that capacity has not been exceeded) is violated. Similarly, the action that models downlinking of data has flexible duration determined by there being data to transmit. These actions both affect the amount of data stored, in a way determined by the rate of storage and transmission, and they can be executed concurrently. At any point during their execution the amount of data stored and the amount so far transmitted can be computed from known values. Figure 3 shows how the two durative actions could be modelled. The special literal $\#t$ is used to denote the time period over which the action is executing. Using $\#t$ it is possible to access arbitrary time points within the execution interval.

```
(:durative-action observe
 :parameters (?r - recorder ?i - instrument
                  ?o - observation)
 :duration (= ?duration (observationTime ?i ?o))
 :condition (and (at start (targetted ?i ?o))
                (over all (targetted ?i ?o))
                (over all (<= (data ?r) (capacity ?r))))
 :effect (increase (data ?r) (* #t (dataRate ?i))))

(:durative-action downlink
 :parameters (?r - recorder ?g - groundStation)
 :duration (> ?duration 0)
 :condition (and (at start (inView ?g))
                (over all (inView ?g))
                (over all (> (data ?r) 0)))
 :effect (and (increase (downlinked)
                           (* #t (transmissionRate ?g)))
             (decrease (data ?r)
                           (* #t (transmissionRate ?g)))
```

Figure 3: Durative actions encoding the continuous behaviours of recording and downlinking data.

Maintaining Satellite Operating Temperatures

In the recording and downlinking example it is possible to model the continuous behaviour using durative actions the durations of which are expressed using *duration inequalities*. The actual duration of any action instance can be computed from variables whose values can be predicted in advance (such as the point at which the storage tape will be filled, given the capacity of the tape, the amount of data it holds at the start of the action and the rate at which data can be stored). However, there are other examples where periods of continuous change cannot be bounded in advance.

When the orbit of a satellite takes it out of the sun it begins to cool at a continuous rate. During the cooling period it must use battery power to keep itself warm enough to prevent instruments from malfunctioning. The temperature of the satellite changes continuously as a result of the interaction between the cooling and warming processes. If the battery runs out of charge at some point the satellite will cool until a point is reached at which the instruments will malfunction. This critical point is brought about by an *event* that triggers when the temperature reaches a critical threshold.

The cooling of the satellite is triggered by an event that occurs when the satellite is removed from all heat sources. This event in turn triggers a process, the cooling process, which continues for as long as the satellite remains in the cold. While the cooling process is in operation the temperature of the satellite is falling and further events may be triggered as thresholds are passed.

It is not possible to model processes that are autonomously triggered by events within the durative action framework. The planner may not know, or care to know, the extent over which these processes will be active. It may be able to plan to counter the cooling effects by, for example, using battery power. An important reason why the cooling process cannot be modelled using a durative action is that it is not under the planner's control to choose the

```
(:action heat
 :parameters (?s - satellite)
 :precondition (and (> (energy ?s) 0) (heaterOff ?s))
 :effect (and (heaterOn ?s) (not (heaterOff ?s))))

(:process cooling
 :parameters (?s - satellite)
 :precondition (inShade ?s)
 :effect (decrease (temperature ?s)
            (* #t (F(temperature ?s)))))

(:event freeze
 :parameters (?s - satellite ?i - instrument)
 :precondition (and (functional ?i) (onBoard ?s ?i)
                    (< (temperature ?s) (safeValue ?i)))
 :effect (not (functional ?i)))

(:process warming
 :parameters (?s - satellite)
 :precondition (and (heaterOn ?s) (> (energy ?s) 0))
 :effect (and (increase (temperature ?s) (* #t (heatRate ?s)))
              (decrease (energy ?s)
                           (* #t (heaterConsumption ?s))))
```

Figure 4: A fragment of a satellite domain encoding the heating and cooling effects. The expression `F(temperature ?s)` indicates the cooling effect due to radiation (and would be written explicitly in a complete encoding). There is a corresponding heating process that occurs due to radiation from the sun when the satellite is not in shade.

cooling effect. The satellite cools because of the physical nature of the world. The planner can plan to counter the effect by ensuring that its battery is charged sufficiently to heat the satellite during its period in the cold, demonstrating the difference between processes that are triggered by *actions* and processes that are triggered by *events*. When the planner has control over how long they will operate, those triggered by actions can often be encapsulated in flexible durative actions. However, the planner may not know or care how long an autonomously triggered process will last so those triggered by events must be modelled differently.

Figure 4 shows the PDDL+ description of the cooling and heating example. The $\#t$ literal is used to denote the time period over which a process instance runs. That is, each separate continuous period over which the preconditions of the process are satisfied will have a duration whose value is denoted by $\#t$ for that process instance. It can be seen that the heating action offsets the effects of the cooling process, affecting the time taken to reach the critical threshold. By planning to store enough charge prior to entering the cold region the planner can avoid the critical threshold being reached at all.

Modelling a Continuous Recharging Process

The recharging situation is similar in detail to the cooling and heating example described above. When the robot enters the sunlight the recharging process begins, triggered by the event of arriving in the sun. The planner need not care how long the recharging process continues, provided that there is enough charge available for all of the robot's activities to be completed. Driving and digging activities

```
(:action activate-charger
 :parameters (?r - rover)
 :precondition (and (in-sun ?r)
                    (< (charge ?r) (capacity ?r)))
 :effect (is-charging ?r))

(:process charging
 :parameters (?r - rover)
 :precondition (and (<= (charge ?r) (capacity ?r))
                    (in-sun ?r)
                    (charging ?r))
 :effect (increase (charge ?r) (* #t (charge-rate ?r))))

(:event stop-charging
 :parameters (?r - rover)
 :precondition (or (= (charge ?r) (capacity ?r))
                   (not (in-sun ?r)))
 :effect (not (charging ?r)))
```

Figure 5: A fragment of the rover domain showing the recharging process with its initiating action and concluding event.

consume charge, resulting in interacting increasing and decreasing effects on charge while these activities are taking place in the sun. The planner can plan to exploit the charging effect of the sunlight, but it cannot dictate how long the charging process will last.

The difference between this example and the cooling example is that it might seem reasonable for the charging effect to be modelled using a durative action (since the charging effect is a desirable one that the planner might choose). However, in fact, charging when in the sun is not controlled by the planner but happens to be a positive effect that the planner can exploit. Furthermore, using a durative action would necessitate identifying an upper bound on the time over which charging would take place. A reasonable upper bound might be suggested by the charge capacity, but, in fact, the potential to continue charging is present for as long as the robot is in the sun (as long as the robot remains in sunlight charging resumes as soon as stored charge is depleted). Modelling charging as a process is more accurate than using durative actions because it makes the world responsible for the length of the charging period and for the interaction between charging and charge consumption. The details can be seen in Figure 5.

Plan Metrics

An important extension that we introduced into PDDL2.1, and which is also available in PDDL+, is an (optional) field within the description of problems to enable the specification of *plan metrics*. Plan metrics specify, for the benefit of the planner, the basis on which a plan will be evaluated for a particular problem. The same initial and goal states might yield entirely different optimal plans given different plan metrics. The following metric evaluates a satellite observation plan according to the amount of data successfully downlinked.

: metric maximize (datadownlinked)

In this expression the value *datadownlinked* is incremented, by the amount of data transmitted, every time a downlinking operation is completed in the plan.

The ability to encode such metrics makes it possible for plans to be evaluated according to more realistic notions of quality than sequential length. Length is a very coarse measure of quality that does not take into account the cost or utility of specific actions. An enriched descriptive power for the evaluation of plans is a crucial extension for the practical use of planners, since it is almost never the case that real plans are evaluated solely by the number of actions they contain.

Semantics of Domain Models

The traditional semantics for planning languages rests on state transition models, so it is natural to consider temporal and metric extensions of finite automata as a foundation for planning language extensions. The theory of hybrid automata (Henzinger 1996; Gupta, Henziner, & Jagadeesan 1997; Henzinger & Raskin 2000), which has been a focus of interest in the model-checking community for some years, provides an ideal formal basis for the development of a semantics for PDDL+. Our examination of this theory revealed that the issues we have considered in the development of PDDL+ have been considered and, in many important cases, resolved, by research done in this field. Our main contribution has therefore been to develop a language for succinct encodings of hybrid automata, in a form that is directed for use in planning. Our hope is that both the formal (semantics and other properties) and practical (model-checking techniques) results in hybrid automata theory will be able to be exploited by the planning community in addressing the problem of planning for discrete-continuous planning domains.

We have developed the semantics of PDDL+ by means of a mapping to hybrid automata, effectively demonstrating how the succinct encoding of a domain in PDDL+ is used to construct an explicit hybrid automaton model. Figure 6 depicts a fragment of the hybrid automaton model of the rover recharging domain. In this example the variable *charge* represents the charge level for the rover and *recharge-rate* represents the rate of recharging. The variable $\frac{d}{dt}charge$ represents the rate of change of *charge* and $charge'$ represents the value of $charge$ after a discrete change. In control mode:

$$[at(rover1, A), in{-}sun(rover1), \frac{d}{dt}charge(rover1) = 0]$$

the rover charge is unchanging (although the rover might not be fully charged). The *activate-charger* control switch causes the rover to enter the new control mode in which the rover is charging and the level of charge is changing according to the rate of recharging over time. This mode can only be maintained while the charge level is less than the capacity of the rover (this constraint is an invariant condition) and while the rover is in the sun. If the sun goes down or the rover is fully charged then charging will stop. Figure 7 depicts the state transition semantics of the cooling and heating example shown in figure 3.

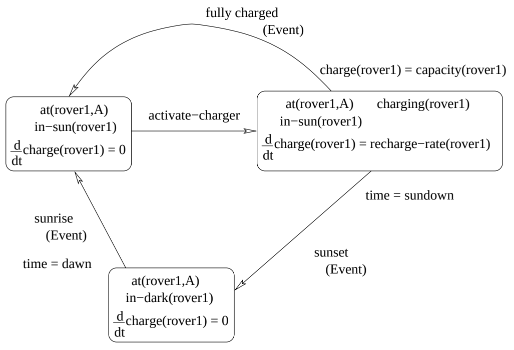

Figure 6: A fragment of the Rover Domain modelled as a hybrid automaton

Related Work

Representation of, and reasoning with, statements about time and the temporal extent of propositions has long been an subject of research in AI including planning research (Allen 1984; McDermott 1982; Sandewall 1994; Kowalski & Sergot 1986; Laborie & Ghallab 1995; Muscettola 1994; Bacchus & Kabanza 2000). Important issues raised during the extension of PDDL to handle temporal features have, of course, already been examined by other researchers, for example in Shanahan's work on continuous change within the event calculus (Shanahan 1990), in Shoham's (Shoham 1985) and Reichgelt's (Reichgelt 1989) work on temporal reasoning and work on non-reified temporal systems (Bacchus, Tenenberg, & Koomen 1991). Vila (Vila 1994) provides an excellent survey of work in temporal reasoning in AI. In this section we briefly review some of the central issues that have been addressed, and their treatment in the literature, and set PDDL2.1 and PDDL+ in the context of research in temporal logics.

Several researchers in temporal logics have considered the problems of reasoning about concurrency, continuous change and temporal extent. These works have focussed on the problem of reasoning about change when the world is described using arbitrary logical formulae. The need to handle complex logical formulae makes the frame problem difficult to resolve, and an approach based on circumscription (McCarthy 1980) and default reasoning (Reiter 1980) is typical. The STRIPS assumption provides a simple solution to the frame problem when states are described using atomic formulae. The classical planning assumption is that states can be described atomically but this is not a general

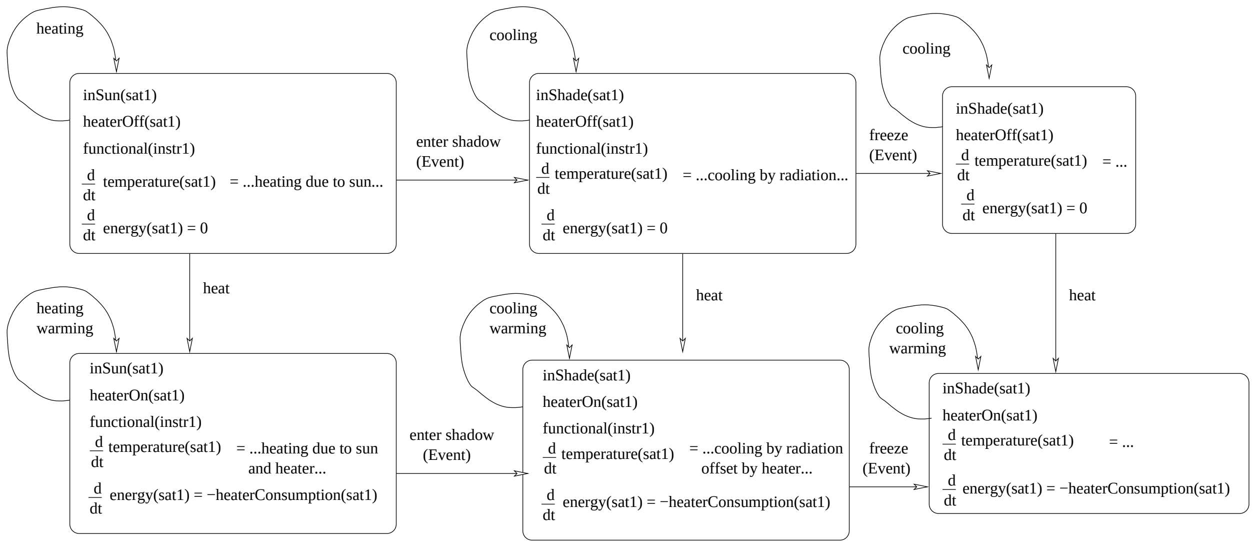

Figure 7: A fragment of the Satellite Domain modelled as a hybrid automaton. Active processes are denoted by loops on the states, corresponding to the processes that contribute non-zero components to the values of the derivatives. Other actions might cause further energy draining processes to affect the derivative of the energy level.

view of the modelling of change. Although simplifying, this assumption is surprisingly expressive. The bench mark domains introduced in the third international planning competition suggest that atomic modelling is powerful enough to capture some complex domains which closely approximate real problems. The temporal reasoning issues we confront are not simplified as a consequence of having made a simplifying assumption about how states are updated. We remain concerned with the major issues of temporal reasoning: concurrency, continuous change and temporal extent.

In the development of PDDL+ we made a basic decision to consider actions and events as instantaneous state transitions. This allows us to concentrate on the truth of propositions at points instead of over intervals. The decision to consider actions and events as instantaneous state transitions is similar to that made by many temporal reasoning researchers (Shanahan 1990; McCarthy & Hayes 1969; McDermott 1982). In the context of PDDL+ the approach has the advantage of smoothly integrating with the classical planning view of actions as state transitions.

In the remainder of this section we compare the PDDL extensions that we propose with previous work in temporal reasoning by considering the three central issues identified above. Our objective is not to claim that our extensions improve on previous work, but instead to demonstrate that the implementation of solutions to these three problems within the PDDL framework makes their exploitation directly accessible to planning in a way that they are not when embedded within a logic and accompanying proof theory.

Continuous change

Several temporal reasoning frameworks began with consideration of discrete change and, later, were extended to handle continuous change. For example, in (Shanahan 1990) Shanahan extended the event calculus of Kowalski and Sergot (Kowalski & Sergot 1986) to enable the modelling of continuous change. This process of extension mirrors the situation faced in extending PDDL, where a system modelling discrete change already existed. It is, therefore, interesting to compare the use of PDDL+ with the use of systems such as the extended event calculus.

In the sink-filling example that Shanahan describes (Shanahan 1990) he discusses the issues of termination of events (self-termination and termination by other events), identification of the level of water in the sink during the filling process and the effect on the rate of change in the level of water in a sink when it is being filled from two sources simultaneously. The behaviour of the filling process and its effects on the state of the sink over time are modelled as axioms which would allow an inference engine to predict the state of the sink at points during the execution of the process.

PDDL+ allows the representation of the complex interactions that arise when a sink is filled from multiple independently controlled water sources by means of decomposition into the initiation of filling, the process of filling and its termination (either by water sources all being turned off, or by flooding). This model is robust, since it easily accommodates multiple water sources, simply modifying the rate of flow appropriately, which then correctly affects the process of filling. In contrast to Shanahan's extension to the event calculus, this approach does not require that the filling process be (at least from the point of view of the logical axiomatisation) terminated and restarted at a new rate when a water source is opened or closed, since the process simply remains active throughout — the change in rate of filling is reflected in a piecewise-linear profile for the depth of water in the sink, just as it is in Shanahan's model.

One of the important consequences of continuous behaviour is the triggering of events. In Shanahan's extensions this is achieved through the axiomatisation of causal relationships — events are not distinguished syntactically from actions, but only by the fact that their happening is axiomatically the consequence of certain conditions. We believe that it is important for a planner to have direct access to the distinction because it determines what the planner can do and what consequences it can expect from interactions within the world. Although Shanahan might add additional axioms to capture the difference, the action-oriented representation of the PDDL tradition makes the distinction natural and convenient.

Concurrency

The opportunity for concurrent activities complicates several aspects of temporal reasoning. Firstly, it is necessary to account for which actions can be concurrent and secondly it is necessary to describe how concurrent activities interact in their effects on the world.

In most formalisms the first of these points is achieved by relying on the underlying logic to deliver an inconsistency when an attempt is made to apply two incompatible actions simultaneously. For example, the axioms of the event calculus will yield the simultaneous truth and falsity of a fluent if incompatible actions are applied simultaneously and consequently yield an inconsistency. Unfortunately, recognising inconsistency is, in general, undecidable, for a sufficiently expressive language. In PDDL+ we adopt a solution that exploits the restricted form of the action-centred formalism, defining the circumstances in which two actions could lead to inconsistency (are mutex) and rejecting the simultaneous application of such actions.

Shanahan (Shanahan 1999) discusses Gelfond's (Gelfond, Lifschitz, & Rabinov 1991) example of the soup bowl in which the problem concerns raising a soup bowl without spilling the soup. Two actions, lift left and lift right, can be applied to the bowl. If either is applied on its own the soup will spill, but, it is argued, if they are applied simultaneously then the bowl is raised from the table and no soup spills. To model this situation Shanahan uses an explicit assertion of the interaction between the lift left and lift right actions to ensure that the spillage effect is cancelled when the pair is executed simultaneously. The assumption is that the reasoner can rely on the successful simultaneity in order to exploit the effect.

In PDDL+ we reject this solution on the grounds that precise simultaneity is outside the control of any physical executive. PDDL+ supports the modelling of the soup bowl situation in the following way. Lift left and lift right both independently initiate tilting processes which, after a measurable amount of time, will result in spillage of the soup. Provided that the two lift actions occur within an appropriate tolerance of one another the tilting will be corrected and the spillage avoided without the need to model cancellation of effects. We argue that an executive can execute the two actions to within a fine but non-zero tolerance of one another, and can therefore successfully lift the bowl.

Temporal extent

A common concern in temporal reasoning frameworks, discussed in detail by Vila and others (Vila 1994; van Bentham 1983), is the *divided instant problem*. This is the problem that is apparent when considering what happens at the moment of transition from, say, truth to falsity of a propositional variable. The question that must be addressed is whether the proposition is true, false, undefined or inconsistently both true and false at the instant of transition. Clearly the last of these possibilities is undesirable. The solution we adopt is a combination of the pragmatic and the philosophically principled. The pragmatic element is that we choose to model actions as instantaneous transitions with effects beginning at the instant of application. Thus, the actions mark the end-points of intervals of persistence of state which are closed on the left and open on the right. This ensures that the intervals nest together without inconsistency and the truth values of propositions are always defined. The same half-open-half-closed solution is adopted elsewhere (Shanahan 1999).

Plan Generation and Validation

The semantics we have provided for PDDL+ suggests that model-checking techniques appropriate for hybrid automata might provide a basis for the development of plan generation algorithms for PDDL+. We have not yet explored the plan generation problem further, but instead have focussed on the question of validation, since it is essential that plans be verifiable efficiently.

It is necessary to automate the validation of plans produced for complex domains since it cannot be considered reliable or even feasible for non-trivial plans to be checked by human experts. The validation problem is decidable for PDDL because plans are finite and can be validated simply by simulation of their execution. The issue is complicated for PDDL2.1 and PDDL+ because the validity of a plan depends on confirming that the actions are applicable in the states that result from any events and/or processes triggered by the initial state or by actions in the plan. Neither events nor processes will be visible in the finished plan, so it will no longer be safe to assume that the actions in the plan will chain together as they do in PDDL plans (actions might rely on preconditions that are established by active processes or events). Plan validation must confirm that actions interact with any active processes and events to perform a successful state transition between the initial and goal states. Since the ability to validate plans is crucial for the practicality of the language, we consider some of the restrictions that might be imposed on PDDL2.1 and PDDL+ to make validation decidable.

Plan validation is decidable for domains including discretized and continuous durative actions, but without events or processes, because all activity is encapsulated with the durative actions explicitly identified by a plan. This makes the trace induced by a plan finite and hence checkable. We therefore observe that the validation problem for PDDL2.1 is decidable even when actions contain duration inequalities. This is because the work in determining how the duration inequalities should be solved has already been completed in the finished plan so validation of the plan can proceed by simulation of its execution, as is the case for PDDL plans.

The situation is more complex for PDDL+ however. Our hybrid automaton-based semantics is very powerful and allows us to give meaning to far more plans than we can validate. In general, given a sequence of actions, P, the question of whether P is a valid plan is undecidable. Once events are introduced, it is possible to create domains in which entering a certain state can trigger a cascade of events so that the number of important happenings in the plan is no longer finite (even though the number of actions still is). Checking whether such a cascade terminates before the next action is executed can be undecidable. One solution would be to allow only a finite number of events to occur between any two time points (by insisting that they be separated by a minimum of some fixed amount of time). This results in a finite trace, and hence a decidable validation problem. However, this would place an arbitrary constraint on what domains can be correctly modelled.

Identifying the restrictions under which validation can be done efficiently is is a focus of our current research.

Conclusions

PDDL+ enables the representation of a class of deterministic mixed discrete-continuous domains as planning domains. This is important for the modelling of many realistic problem domains in which interacting processes arise resulting in continuous change that cannot be terminated under the direct control of the planner. We have given some examples of situations in which this arises and in which simplification to discrete models would result in failure to capture the true meaning of the domain. The semantics of the language is given by means of a mapping showing how hybrid automata can be constructed from PDDL+ domains. The power of hybrid automata means that we can interpret more plans than we can efficiently validate and we are currently working on identifying the conditions under which efficient validation is possible.

The development of the PDDL sequence towards greater expressive power is important to the planning community because the PDDL family of languages has provided a common foundation for much of the research effort over the past decade. The problems involved in modelling the behaviour of mixed discrete-continuous systems have been well explored but there have been no widely adopted models within the planning community. PDDL+ begins to bridge the gap between basic research and applications-oriented planning by providing the expressive power necessary to capture real problems.

References

Allen, J. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23:123–154.

Bacchus, F., and Kabanza, F. 2000. Using temporal logic to express search control knowledge for planning. *Artificial Intelligence* 116(1-2):123–191.

Bacchus, F.; Tenenberg, J.; and Koomen, J. 1991. A non-reified temporal logic for AI. *Artificial Intelligence* 52:87–108.

Fox, M., and Long, D. 2002. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Technical Report Department of Computer Science, 20/02, Durham University, UK. Available at `www.dur.ac.uk/d.p.long/competition.html`.

Gelfond, M.; Lifschitz, V.; and Rabinov, A. 1991. What are the limitations of the situation calculus? In Boyer, R., ed., *Essays in honor of Woody Bledsoe*. Kluwer Academic. 167–179.

Gupta, V.; Henziner, T.; and Jagadeesan, R. 1997. Robust timed automata. In *HART'97: Hybrid and Real-time Systems, LNCS 1201*, 331–345. Springer-Verlag.

Henzinger, T., and Raskin, J.-F. 2000. Robust undecidability of timed and hybrid systems. In *Proceedings of the 3rd International Workshop on Hybrid Systems: Computation and Control. LNCS 1790.*, 145–159. Springer-Verlag.

Henzinger, T. 1996. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on ogic in Computer Science. Invited tutorial.*, 278–292. IEEE Computer Society Press.

Kowalski, R., and Sergot, M. 1986. A logic-based calculus of events. *New Generation Computing* 4:67–95.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proc. of 14th International Joint Conference on AI*. Morgan Kaufmann.

McCarthy, J., and Hayes, P. 1969. Some philosophical problems from the standpoint of artificial intelligence. In Meltzer, B., and Michie, D., eds., *Machine Intelligence 4*. Edinburgh University Press. 463–502. reprinted in McC90.

McCarthy, J. 1980. Circumscription — a form of non-monotonic reasoning. *Artificial Intelligence* 13:27–39.

McDermott, D., and the AIPS'98 Planning Competition Committee. 1998. PDDL–the planning domain definition language. Technical report, Available at: `www.cs.yale.edu/homes/dvm`.

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

McDermott, D. 2000. The 1998 AI planning systems competition. *AI Magazine* 21(2).

Muscettola, N. 1994. HSTS: Integrating planning and scheduling. In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*. San Mateo, CA: Morgan Kaufmann. 169–212.

Reichgelt, H. 1989. A comparison of first order and modal theories of time. In Jackson, P.; Reichgelt, H.; and van Harmelen, F., eds., *Logic-based knowledge representation*. MIT Press. 143–176.

Reiter, R. 1980. A logic for default reasoning. *Artificial Intelligence* 13:81–132.

Sandewall, E. 1994. *Features and fluents: the representation of knowledge about dynamical systems, volume I*. Oxford University Press.

Shanahan, M. 1990. Representing continuous change in the event calculus. In *Proceedings of ECAI'90*, 598–603.

Shanahan, M. 1999. The event calculus explained. In Wooldridge, M., and M.Veloso., eds., *Artificial Intelligence Today*. Springer Lecture Notes in Artificial Intelligence no. 1600. 409–430.

Shoham, Y. 1985. Ten requirements for a theory of change. *New Generation Computing* 3:467–477.

Smith, D., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proceedings of IJCAI-99, Stockholm*, 326–337.

van Bentham, J. 1983. *The logic of time*. Kluwer Academic Press, Dordrecht.

Vila, L. 1994. A survey on temporal reasoning in artificial intelligence. *AI Communications* 7:4–28.