Using Model Checking for Pre-Planning Analysis

Maria Fox, Derek Long, Steven Bradley and James McKinna
University of Durham,UK
maria.fox@durham.ac.uk, d.p.long@durham.ac.uk
s.p.bradley@durham.ac.uk, j.h.mckinna@durham.ac.uk

Abstract

Planning domain descriptions contain many structural features, not made explicit by the domain designer but nevertheless present in the representation, that can be exploited by a planner to reduce search. We have explored the effect of a range of logical features, extracted during pre-planning analysis, on the search performance of both forward- and backward-searching planning systems and have confirmed that considerable planning-time benefits can result from the use of efficient pre-planning analysis techniques. Domains involving time and other numeric quantities contain non-logical features which are harder to extract from a domain model. We have explored the combination of our existing pre-planning technology with standard model-checking approaches in order to extract temporal features of a planning domain that can improve planning performance. In this paper we explain how such domain features are extracted and how they can be exploited by the search strategy of a planner.

Introduction

Static analysis of planning domains can yield information that shortcuts a planner's search and improves its planning performance. We have demonstrated the utility of static analysis of STRIPS domains using our pre-planning analysis system TIM(Fox & Long 1998; 2000b; Long & Fox 2000; Gerevini & Schubert 1996). In this paper we show how automatic static analysis can be applied to domains involving numeric constraints and properties including the passage of time. One possibility is to inform a planner by identifying when critical events will occur, which may or may not be desirable, so that the planner can focus on avoiding or facilitating these. Another possibility is to recognise the presence of deadlock states in a domain model. We are currently investigating the use of model checking technology to perform such analyses as a pre-planning phase. Model-checking has been identified as a planning paradigm (Cimatti *et al.* 1997) and there is a significant body of research on planning as model-checking. However, our work is concerned not with planning but with the analysis of domains prior to planning. We have a number of reasons for believing that model-checking for pre-planning can be efficient and can ease some of the important bottlenecks in planning. These issues are discussed further below.

It is necessary to begin by describing the language in which the planning domains we analyze are encoded. We will then briefly describe the processes by which timed automata are constructed from domains expressed in this language, and our reasons for believing that the automata which need to be model-checked are generally small with respect to the overall size of the domain. Finally we will indicate how the results of model-checking these automata might be exploited by a planner.

The Domain Representation Language: PDDL+

Our intention is to capture rich domains characterized by temporal and numeric aspects as well as purely logical ones. There are a number of planning systems in existence that can handle domains of this kind (IxTeT (Ghallab & Laruelle 1994), HSTS (Jo'nsson *et al.* 2000), Aspen (Smith *et al.* 1998)), though all of these use their own "in house" languages. A central feature of all of these languages is the use of temporal intervals and relationships between them, such as whether they meet, or overlap, to identify where state changes occur on one or more timelines.

The domain representation language we use is PDDL+, an extension to McDermott's PDDL language(McDermott 2000; McDermott & the AIPS'98 Planning Competition Committee 1998), currently being developed by Maria Fox and Derek Long (Fox & Long 2000a). PDDL+ uses a point-based representation, rather than an interval-based one, in which an action that does not have its effect until some time after its application is represented by a three-stage sequence consisting of a triggering action, with instantaneous effect, the *process* that this initiates, which continues to

be active whilst its preconditions are satisfied, and the instantaneous *event* or action that terminates the process. We refer to this as the *start-process-stop* model. The *start-process-stop* model allows the representation of the idea of durative action, but has a cleaner underlying semantics. Furthermore, it is as expressive as the interval-based approach used by existing planning systems as it is possible to map the *meets, met-by, contains* and other temporal interval relationships to point-based relationships between the actions, processes and events in the *start-process-stop* representation (Fox & Long 2001).

PDDL+ allows the modelling of instantaneous state transitions, or actions, as in PDDL. In PDDL+ actions can have numeric pre-and post-conditions. Actions are chosen for application by the planning agent and cannot model aspects of the physics of the domain that lie outside of the planner's control. If the only modelling components are actions, the application of which is under the planner's control, then it is impossible to properly model environmental changes that occur spontaneously, or because of unintended interactions between actions and environmental conditions. In order to model how action can bring about desirable consequences, in a potentially dynamic environment, it is necessary to be able to model things that happen outside of the planner's control.

An example of the kind of situation that one might want to model is the consequence of leaving a pan of water boiling on a lighted stove. If the pan is not removed from the stove when the water boils it will begin to boil away. Eventually the pan will boil dry and will burn. In order to successfully boil water for some purpose it is necessary to plan to remove the pan from the stove when the desired temperature has been reached. This requires the modelling of the change in temperature of water over time as well as the event of the pan burning, which is not something that the planning agent is likely to want to bring about.

The language therefore supports two additional modelling components to make this modelling possible. In addition to actions it provides *events* which look like actions but are applied by the environment and are outside the control of the planner. Whenever their preconditions are true they apply and result in state transitions. It further supplies *processes* which, like events, are applied by the environment and are not controllable by the planner. Unlike events they do not model state change but the extent of time spent in a state. They model change over time in some quantity so they always have at least one effect describing the change in some numeric value over time. These are termed *durative* numeric effects. Processes cannot have logical effects, since they do not model state transition, although they might have logical preconditions as well as non-durative numeric ones. Using a combination of events, processes and actions, domain features that might, alternatively, be modelled by actions with duration are modelled, in PDDL+, using the *start-process-stop* approach described above.

Processes can be initiated by actions or events and continue to execute while their preconditions are satisfied. As soon as their preconditions cease to hold they stop. This can be brought about by an event (something happens in the environment to cause the process to terminate) or by an action (the planner chooses to stop the process). Returning to the water example given above, the process of heating can be stopped by the event of the water reaching boiling point, or by the event of a clock timer timing out (assuming that it can be known in advance how long it takes for a certain amount of water to boil) or by the planning agent removing the pan from the stove. The start-process-stop model is very flexible, because it allows the representation of very detailed aspects of the processes being modelled - for example, the rate of water evaporation over time - but is also allows these details to be completely abstracted out and the passage of time only to be modelled. For example, *walking* is an action that is naturally seen as having duration, and walking between locations A and B can reasonably be modelled by a durative action that has the effect of the walker arriving at B within a certain time of leaving A. This kind of action is different from water-boiling because the planner does not have the option of going off to do something else while walking is in progress. In PDDL+, walking can be modelled as the planner initiating, by setting off on the walk, a process of time passing. This process is terminated by an arrive event which becomes applicable as soon as the amount of time required, to reach B from A, has passed.

A PDDL+ plan contains only time-stamped actions, where the time granularity is determined by the domain designer and specified as part of the domain description. For example, the domain designer might specify a granularity of 0.1 seconds, which means that things that happen at a finer granularity than tenths of seconds apart will not be able to be identified with distinct time stamps. This has the effect of making occurrences appear to be simultaneous. However, simultaneity of occurrences is always an abstraction of a finer granularity at which time passes between the two occurrences. At the given granularity level actions may only be simultaneous if their pre-and post-conditions are non-interacting (following a standard mutual-exclusion protocol).

Neither events nor processes are visible in a plan, but the validity of a plan depends on the actions in the plan being applicable in the states that result from any triggered events and/or processes. Processes update numeric values at a rate determined by the specified level of granularity of the domain, so it possible for such updates, and for any triggered events, to occur apparently simultaneously with actions. A plan will be considered invalid if conflicts occur between actions and natural occurrences (events and processes) at the specified granularity level, even if there would be no such conflicts at a lower level of granularity. This is sensible in a model that allows things to happen concurrently, because the coarser the granularity the more difficult it is to temporally separate possibly conflicting behaviours (whether selected by the planner or triggered within the environment).

Because of the mappings that exist, between PDDL+ and the interval-based languages of existing resource-compatible planners, the static analysis that we perform on PDDL+ models can equally be applied to models expressed in these interval-based languages, following the (meanining-preserving) conversion of the interval-based model into PDDL+.

Modelling Domains in PDDL+

We have undertaken a range of domain modelling exercises in PDDL+, resulting in the development of a number of small domains with temporal and other numeric features. These domains all involve exogenous events that can be either exploited or avoided, depending on context. For example, in the bath-filling domain (figure 1) the exogenous event of the bath flooding can be exploited to achieve a state in which the floor is wet if the goal is to have the floor washed and there are no other means by which water can be transferred to the floor. On the other hand, the flooding event can be avoided, by taking some action to avoid it, if the floor being wet is not a desirable state. The fact that the environment is dynamic, and that things change over time without the planner's direct intervention, means that reasoning about the effects of actions is a much more complex process than it is in a classical planning context. We are exploring the extent to which static analysis techniques can reveal information about the logical and causal structure of the domain, in a form that the planner can exploit, to help to reduce the planning-time reasoning burden on the planner.

We are investigating the use of a combination of two forms of static analysis. These two forms are the analysis of logical features of the domain, using the TIM technology, and the analysis of the temporal and numeric aspects of PDDL+ domains using model-

```
(define (domain bath)
  (:requirements :strips)
  (:predicates (off ?x ?y) (on ?t ?b) (plug_in ?b)
               (wet_floor ?b) (dry_floor ?b))
  (:functors (capacity ?b) (level ?b) (flow ?b)
             (flow-rate ?t))
(:action turn_on
        :parameters
                (?tap
                 ?b)
        :precondition
                (and (off ?tap ?b)
                     (plug_in ?b))
        :effect
                (and (not (off ?tap ?b))
                     (on ?tap ?b)
                     (flow ?b += flow-rate ?tap)))

(:action turn_off
        :parameters
                (?tap
                 ?b)
        :precondition
                (and (on ?tap ?b)
                     (plug_in ?b))
        :effect
                (and (not (on ?tap ?b))
                     (off ?tap ?b)
                     (flow ?b -= flow-rate ?tap)))

(:action put_plug_in
        :parameters (?b
                     ?tap)
        :precondition
                (and (off ?tap ?b)
                     (plug_out ?b))
        :effect
                (and (not (plug_out ?b))
                     (plug_in ?b)))

(:action pull_plug_out
        :parameters (?b
                     ?tap)
        :precondition
                (and (off ?tap ?b)
                     (plug_in ?b))
        :effect
                (and (not (plug_in ?b))
                     (plug_out ?b)
                     (level ?b := 0)))

(:process bath_filling
        :parameters
                (?b
                ?tap)
        :precondition
                (and (level ?b <= capacity ?b) (on ?tap ?b))
        :effect (and (level ?b += #t * flow ?b))

(:event flood
        :parameters
                (?b)
        :precondition
                (and (level ?b == capacity ?b)
                     (flow ?b > 0)
                     (dry_floor ?b))
        :effect
                (and (wet_floor ?b)
                     (not (dry_floor ?b))
                     (level ?b := capacity ?b)))
)
```

Figure 1: PDDL+ description of the bath domain.

checking techniques. TIM can be used to partition a domain into functionally distinct object classes together with the state transitions they can perform. This analysis results in the construction of a collection of FSMs, one for each identified object class. If TIM is analysing a domain specification that already contains type distinctions and explicit associations between types and behaviours, as a domain might when it has been obtained by conversion from an interval-based language, then TIM can exploit this specified information rather than rediscover it.

When analysing STRIPS domains TIM is able to identify a range of structural features that can be very effectively exploited by a planner to reduce its search (Fox & Long 1998; 2000b; Long & Fox 2000; 2001). For example, TIM can identify the presence of certain sub-problems that can be solved using specialised heuristics that can be automatically invoked. We first considered the identification of Travelling Salesman sub-problems and designed a way of integrating a specialised TSP strategy with a forward search based planner (Fox & Long 2000b). More recently we have used TIM to identify Multi-Processor Scheduling problems (MPS) in STRIPS domains (Long & Fox 2001). To exploit the presence of MPS sub-problems effectively it is necessary to know the task lengths. In a STRIPS domain the length of a task is encoded as a path through states in the state-space through which objects in the task class can move. TIM can identify the lengths of these paths from the FSMs it builds to represent the object classes and their behaviours. Having identified the task lengths we are able to make use of a good quality MPS heuristic to solve these problems.

However, STRIPS domains have no temporal or numeric features, so the exact techniques appropriate for analysing these domains are of less direct use when analysing non-STRIPS domains expressed in PDDL+. When domains are temporal the transitions that objects make between states have a temporal aspect to them as well as a logical one (the bath takes time to go from being empty to being full). This affects the way we analyse such domains for the presence of sub-problems. The task lengths in a MPS problem will be determined not just by the lengths of the paths that task objects take through state spaces, but also the duration of these paths. Furthermore, the integration of the purely logical structure of a domain with its temporal and other numeric features cannot be modelled using FSMs alone. In order to analyse PDDL+ domains we therefore extended TIM to construct timed automata instead of FSMs wherever processes and events affect the behaviour of any class of objects.

In a domain involving time there are a variety of useful hints that can be given to a planner to help it to avoid unnecessary search. For example, a planner can benefit from knowing the latest time at which it can afford to perform some action to enable a desired state to be reached at some specified time point. It can also benefit from knowing how long it will take for a certain event to occur following the application of some action. In terms of automatically identifying sub-problems, such as MPS, the computation of the task lengths will involve determining how much time it will take to get a task object along its path of necessary transitions. This is the kind of information that a model-checker can extract from a timed automaton model.

In the following section we describe how TIM can be used to extract, from a PDDL+ domain description, timed automata that can be analysed using standard model-checking techniques. We then describe how we have so far used model-checking to identify temporal properties of interest from a domain description.

Analysis of PDDL+ Domains

Our current strategy for analysing domains expressed in PDDL+ is to begin by analysing the logical model, in order to obtain the types and logical invariants that are present, and then to extend the logical model into a temporal one by supplementing it with numeric information.

The logical component of a domain is analysed using our static analysis system, TIM (Fox & Long 1998; 1999). This constructs FSM models of the behaviours associated with each of the object classes, or types, that can be inferred from a planning domain. These automata provide a basis for analysing the underlying functional structures of domains, and have proved a useful basis for automatic recognition of a variety of common sub-problems and representations. TIM was originally designed for analysis of STRIPS domains expressed in the PDDL domain description language but it remains appropriate for analysing the logical properties of the finite components of a non-STRIPS domain.

The FSMs constructed by TIM correspond to a decomposition of the behaviour of types of objects in the planning domain. A single type might be characterised by several FSMs, each describing a separate facet of the behaviour of the type. For example, doors that can be open or closed and locked or unlocked would have two separate FSMs to characterise this behaviour, illustrated in figure 2. Although this decomposition is highly useful and allows a compact representation of the behaviours of objects, it can make obscure certain

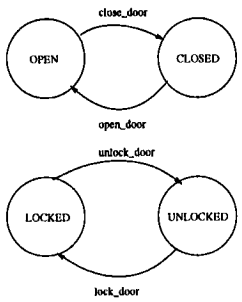

Figure 2: Two FSMs describe the behaviours of door objects.

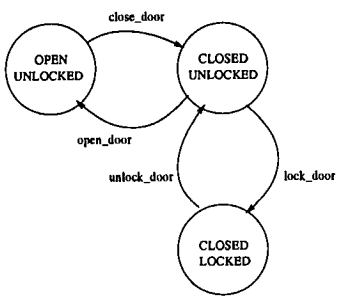

Figure 3: Single product FSM describes more precisely the behaviour of door objects.

aspects of the behaviours of objects. To continue the example of the doors, if doors can only be opened or closed when unlocked, and can only be locked when closed, then the state in which the door is both locked and open is unreachable. This is not clear in the decomposition, but becomes apparent when the product of the FSMs is formed, using the conditions satisfied in the states of one FSM that enable transitions between states in another FSM to filter the accessibility relations between compound states in the new product FSM. Since the transitions in the FSMs are guarded by the logical preconditions that must be satisfied by objects making these transitions, it is possible to form the product under the closure of these guards and hence eliminate predicates from the product that are not accessible to the objects that will traverse the product automaton. This is illustrated, for the doors example, in figure 3.

Once the TIM analysis has been extended by forming the product of two or more FSMs relating the behavour of objects in the same class, the behaviours they describe are amended with the numerical information extracted from the representation of the actions, events and processes in the model. Transitions in the product automata are supplemented with numeric guards (the numeric preconditions for objects making these transitions), assignments (the numeric effects of the transitions) and, in the cases of transitions triggered by actions, synchronizations with the execution of those actions. Transitions triggered by events and processes are not synchronized with any external event. The result of this process is the basis for the construction of a collection of timed automata that can be analysed to identify the temporal properties of the domain model.

PDDL+ does not support arbitrary numbers as being of equivalent status to other objects, but restricts them to be the values assigned to the measurement of properties of named objects in the domain. This has an important consquence, which is that the collection of first-class objects is finite, with a logical behaviour that is described by finite collections of states and hence supports the construction of FSMs. The numeric aspects of the domain are confined to being values associated with properties of these objects. These values can be affected by transitions in an FSM, be used in guards that constrain access to transitions and be used to model the effects of passing time. This restriction on the role of numeric values has the important consequences that choices of different instantiations of actions during planning remain finite, since they cannot range over arbitrary numeric values, and the structure of the behaviours of objects conform to *finite* state machines, making them amenable to various analyses, including some based on model-checking.

The product automata provide a complete picture of the possible states of a class of objects and the transitions, both logical and numeric, that are possible between these states. They still represent a decomposition of the entire state structure of the behaviour of the domain, since separate automata are constructed for each type of objects, and only one automaton is constructed for the entire class of objects of each type. The complete state space for the domain would be given by taking the product of the collection of automata formed by constructing a distinct instance of each base automaton for each object of the corresponding type. This would be, in general, far too large to construct. The automata for individual type classes already represent a significant expansion of the compact decomposition generated by TIM. However, the benefit we gain from allowing this expansion is a complete and analysable picture of the potentially accessible states for each class of objects, together with the transitions that link them (complete with temporal dimensions where appropriate).

Model-Checking the Automata

It is relatively simple to translate the product automata into timed hybrid automata, which can then be further analysed to provide information to the planner either before or during planning. First we describe the automaton model we use, then the translation, then analysis which we are able to perform and the possible uses of this in producing a plan.

Timed hybrid automata are an extension of timed automata (Alur & Dill 1990). The basic notion is that of a finite state machine extended with real-valued variables. These variables can be used to form enabling conditions for transitions by comparing with a constant value, and to form invariant conditions for states. Variables can be reset when transitions take place, but cannot be assigned a value other than 0, and cannot be compared with each other. In a standard timed automaton all variables increase their value in line with the increase in global time, and are known as clocks. In a hybrid automaton, the rate at which a variable changes may vary with the state. When this rate is restricted to be a natural number these automata are called *linear* hybrid automata. A (linear) hybrid automaton is defined by:

- a set of states S;
- a set of transitions between states $T \subset S \times S$;
- a set of variables V.

A *variable valuation* is a function $v : V \rightarrow Real$ which assigns a real value to each variable.

Associated with each state in S is:

- an invariant function which takes a variable valuation and specifies whether the state is allowed with that valuation;
- a rate of change for each variable.

Associated with each transition in T is:

- an enabling function which takes a variable valuation and specifies whether the transition is allowed with that valuation;
- a reset function which specifies which variables are to be reset to 0 when the transition is taken;
- a set of labels for sychronisation.

The product automata constructed by TIM have very nearly all of the information required in the form needed for their timed automaton representation, with states, transition guards and synchronisations mapping across directly. A little bit of work needs to be done with the assignments and transitions, because in the planning model changes to variables due to the passage of time are made explicit with a transition assignment of the form

$$level \quad += \quad \#t \quad * \quad flow$$

In the automaton model this transition is not listed explicitly, although there is a corresponding transition in the timed labelled transition system which arises from the timed automaton. Instead this is used to define the rate of change of the variable for the timed automaton state. This is well-defined because all such transitions in the planning model are made from a state to itself.

Finally, the invariants for the timed automaton states need to be produced. These are needed to govern transitions that *must* occur at a certain time, corresponding to processes in the planning domain. In such a case it is not enough to add a transition with a guard like $level >= capacity$ because this does not guarantee that such a transition will occur once the guard becomes true. Each such guard is therefore (non-strictly) negated to form an invariant on the source state (in this case: $level <= capacity$), so that it is not possible to remain in that state once the transition becomes enabled. Where there is more than one such outgoing transition from a state, the negated guards are conjoined to form the invariant.

All of the work required to transform a product automaton into a timed automaton can be done automatically. The necessary information is inferred by TIM – it is just a case of expressing this information in the appropriate form and this task is trivial.

During planning the planner will be searching for certain goal states which may be the final goal state or some intermediate goal state. One way in which model-checking can help the planner is to prune the search space by eliminating search paths which are never going to lead to the desired goal state. In a planning domain with resources, such as we have been discussing, the planner may try to reach a goal state from a state in which there is insufficient time, or not enough resources available, to make this possible. Eventually the planner will realise that it is not possible to reach the state, so will either have to abandon that goal state (if it is an intermediate state) or abandon the source state and backtrack. In either case, time will be wasted in trying to solve impossible problems. Using a hybrid model-checker such as HyTech (Henzinger, Ho, & Wong-Toi 1995) we can perform forward timed reachability analysis to settle the question statically where it pertains to the behaviour of an object traversing a timed automaton. One advantage of HyTech is the

availability of *parametric analysis*, allowing us to establish the conditions on a parameter under which a state is reachable from another. This means that we can use HyTech to calculate minimum and maximum times for reachability. Such information can assist a planner by identifying specific regions of a timeline in which certain activities must take place if the goal is to be achieved. Although this information can be computed dynamically, for example by a constraint propagation algorithm, pre-planning analysis allows it to be pre-determined and therefore separated from the reasoning burden on the planner. Once computed the information can be presented to the planner as part of the initial state specification (specific activities can be placed within bounded regions on a timeline). HyTech also offers backwards reachability analysis which could be used for the same purpose, or for suggesting intermediate goal states.

To perform such simple *a priori* analyses we simply throw away all synchronisation actions, so we represent no knowledge of how the other components or the planner will behave. A more sophisticated analysis would allow the behaviour of other types of objects to be taken into account in determining whether a given object can reach its goal state. This will certainly be necessary for extending our analysis to deal with goals involving interactions between types belonging to different product spaces. Finally, the analysis could be extended to take into account aspects of the dynamic planning process (for example, a representation of the current candidate plan) in the form of another automaton to be composed and synchronised for analysis during the planning process.

Clearly the main issue is that of performance. Model-checking can provide guarantees about the behaviour of portions of the planning domain, and hence serve to reduce planning time without compromising the quality of the final plan, but only if the model-checking process itself is computationally feasible. We expect this analysis to be feasible if the timed automata that must be analysed remain compact – clearly the approach breaks down in domains in which all significant behaviour takes place within the same automaton. However, our experiences with analysing different domains has convinced us that type differentiation tends to increase as domains increase in size and complexity. If this is the case (we do not yet have any formal argument to support it) then domains that cannot be feasibly model-checked in their entirety will decompose into small enough automata to make their model-checking feasible.

The performance of the model-checker itself may need further scrutiny, as HyTech is quite old and focuses on expressiveness of the model rather than brute performance. Other model-checkers, such as Uppaal (Yi, Larsen, & Pettersson 1997), have focused on performance at the expense of the expressiveness of the model and complexity of the properties that can be checked. However, recent experimental work has been done on extending Uppaal with stopwatches — hybrid variables with rates of 0 or 1 — which are in fact as expressive as more general linear hybrid automata (Cassez & Larsen 2000). This work may be useful in improving the performance of pre-planning analysis.

Examples of Analysis of Domains

We have applied our analysis to PDDL+ encodings of several domains, including a simple model of a bath (which can be filled, possibly to the point of overflowing), a version of a simple transportation domain in which vehicles consume fuel and drivers have restrictions on their working and resting hours and a trivial robot domain constructed by automatic translation from DDL (the language used in the HSTS planner) to PDDL+. This translation relies on a mapping, from the interval-based constructs of DDL to the point-based constructs of PDDL+, which we have constructed and are in the process of refining.

The bath domain is a simple model of what happens when the process of running water into a bath is initiated. Assuming that the bath is plugged, water will run into the bath and eventually the bath will become full. If the tap is not turned off before the bath becomes full then the bath will overflow. The flooding of the bathroom is probably not desired, so the planner must turn the tap on, wait for some moments and then turn the tap off in order to achieve a state in which the bath is full enough to serve its purpose. This is achieved using an *action-process-action* instantiation of the *start-process-stop* model. If the tap is not turned off the bath-filling process will be terminated by the event of flooding.

Analysis of the domain results in the construction of the timed automaton shown in Figure 4. This automaton models the behaviour of the bath, which is defined in terms of the tap and plug associated with that bath. It can be model-checked to obtain the latest time (relative to t) at which the tap can be turned off, having been turned on at time t, in order to avoid flooding. Model-checking will also give us the latest (in fact, exact) time at which the tap should be turned on if a full bath is required at time t. Our present, simple, model assumes that water flows into the bath at a constant rate and that there is only one tap per bath. Furthermore, this one tap can only be turned on

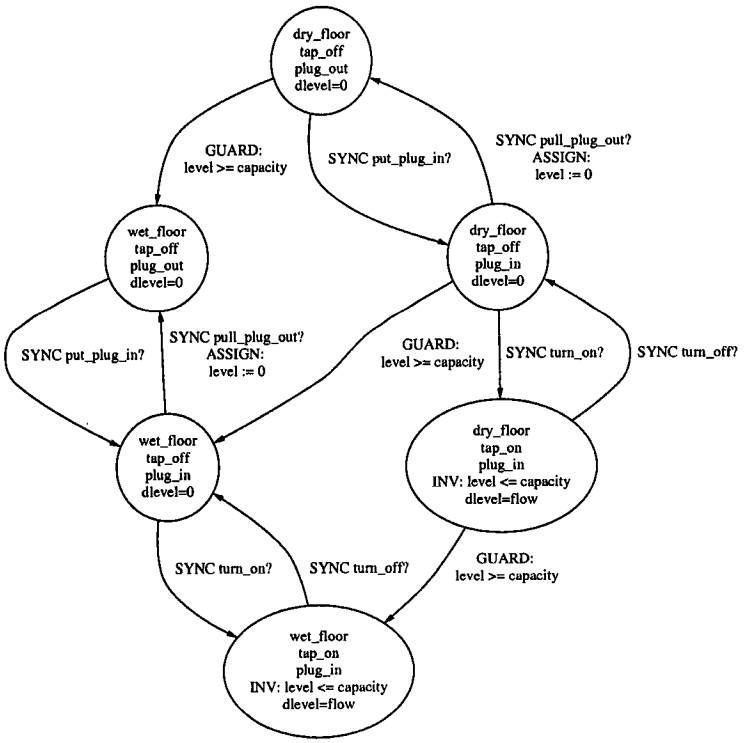

Figure 4: The Timed Automaton for the Bath.

when the plug is in (we have not tried to model how the level in the bath changes when water is running both in and out at different rates).

If we were to extend the model, so that there were different taps from which water flowed at different rates, then model-checking could compute the envelope of time and tap flow combinations required to achieve a full bath. A planner can use this information to most effectively deploy resources in a domain in which different actions (such as turning on taps at different times) use a resource (the agent) to initiate and control the rate at which a process (the bath filling) executes. Although such deployment choices can be made by search there is considerable potential for performance improvement when such choices are taken offline.

Conclusions

We have extended the static analysis techniques of TIM to handle a language expressive enough to capture time and numerically varying quantities. This has involved combining the types, invariants and subproblem-identification techniques of TIM with standard model-checking techniques. Our preliminary work in this area suggests that model-checking can provide information that can improve the search efficiency of a planner.

Although, in the worst case, the construction of the product spaces is exponential in the number of predicates in the domain description, we have observed that the sizes of the product spaces is strongly correlated with the degree of type differentiation present in the model. As domain models increase in complexity the degree of type differentiation that they exhibit tends to increase. This suggests that our strategy of constructing product automata is likely to be acceptably efficient in practice. Further work remains to establish whether the use of model-checking techniques during pre-planning can effectively reduce the planning-time reasoning burden. We intend to integrate our pre-planning strategies with a timeline-based planning strategy to determine the advantage obtained from using pre-computed constraint envelopes to constrain the choices available at planning time.

References

Alur, R., and Dill, D. 1990. Automata for modeling real-time systems. In *17th International Colloquium on Automata, Languages and Programming (ICALP'90) (LNICS 443)*. Springer-Verlag.

Cassez, F., and Larsen, K. 2000. The impressive power of stopwatches. In *11th International*

Conference on Concurrency Theory (CONCUR 2000) (LNICS 1877). Springer-Verlag.

Cimatti, A.; Giunchiglia, E.; Giunchiglia, F.; and Traverso, P. 1997. Planning via model checking: A decision procedure for AR. In *Proceedings of European Conference on Planning, (ECP97).*

Fox, M., and Long, D. 1998. The automatic inference of state invariants in TIM. *JAIR* 9.

Fox, M., and Long, D. 1999. The detection and exploitation of symmetry in planning problems. In *Proceedings of the International Joint Conference on Artificial Intelligence.*

Fox, M., and Long, D. 2000a. PDDL+: An extension of PDDL to handle time. Technical report, Durham University.

Fox, M., and Long, D. 2000b. Utilizing automatically inferred invariants in graph construction and search. In *Proceedings of the 5th Artificial Intelligence Planning Systems Conference, Colorado, US.*

Fox, M., and Long, D. 2001. Conversion of interval-based planning domain descriptions into pddl+. Technical report, In preparation.

Gerevini, A., and Schubert, L. 1996. Accelerating Partial Order Planners: Some Techniques for Effective Search Control and Pruning. *JAIR* 5:95–137.

Ghallab, M., and Laruelle, H. 1994. Representation and control in IxTeT, a temporal planner. In *Proceedings of the 2nd AIPS International Conference.*

Henzinger, T.; Ho, P.-H.; and Wong-Toi, H. 1995. A user guide to HYTECH. In *E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, Tool and Algorithms for the Construction and Analysis of Systems: (TACAS 95), volume 1019 of Lecture Notes in Computer Science*, 41–71.

Jo'nsson, A.; Morris, P.; Muscettola, N.; Rajan, K.; and Smith, B. 2000. Planning in interplanetary space: Theory and practice. In *Proceedings of the 2nd NASA International Workshop on Planning and Scheduling for Space.*

Long, D., and Fox, M. 2000. Automatic synthesis and use of generic types in planning. In *Proceedings of the 5th Artificial Intelligence Planning Systems Conference, Colorado, US.*

Long, D., and Fox, M. 2001. Multi-processor problems in planning. Technical report, Durham University.

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

Smith, B.; Sherwood, R.; Govindjee, A.; Yan, D.; Rabideau, G.; Chien, S.; and Fukunaga, A. 1998. Representing spacecraft mission planning knowledge in ASPEN. In *AIPS Workshop on Knowledge Acquisition.*

Yi, W.; Larsen, K.; and Pettersson, P. 1997. Uppaal in a nutshell. *International Journal of Software Tools for Technology Transfer* 1(1).