PDDL+ Planning with Hybrid Automata: Foundations of Translating Must Behavior

Sergiy Bogomolov  
IST Austria, Austria  
University of Freiburg, Germany  
sergiy.bogomolov@ist.ac.at

Daniele Magazzeni  
King’s College London  
United Kingdom  
daniele.magazzeni@kcl.ac.uk

Stefano Minopoli  
UJF - Lab. VERIMAG  
Grenoble - France  
stefano.minopoli@imag.fr

Martin Wehrle  
University of Basel  
Switzerland  
martin.wehrle@unibas.ch

Abstract

Planning in hybrid domains poses a special challenge due to the involved mixed discrete-continuous dynamics. A recent solving approach for such domains is based on applying model checking techniques on a translation of PDDL+ planning problems to hybrid automata. However, the proposed translation is limited because must behavior is only over-approximated, and hence, processes and events are not reflected exactly. In this paper, we present the theoretical foundation of an exact PDDL+ translation. We propose a schema to convert a hybrid automaton with must transitions into an equivalent hybrid automaton featuring only may transitions.

1 Introduction

Planning in hybrid domains considers the problem of finding plans in domains with mixed discrete-continuous behavior. Such behavior often occurs in practical applications (like, e.g., in robotics, space applications, or embedded systems), hence planning in such hybrid domains has found increasing attention in the planning community. The continuous behavior of hybrid domains is modelled with continuous variables that evolve over time, where the evolution is described by differential equations. In addition, in many real-world applications, exogenous events may happen. Hybrid domains in planning are modelled with PDDL+ (Fox and Long 2006) that provides continuous processes and exogenous events.

From a computational point of view, planning in hybrid domains is challenging because in addition to the “discrete” state explosion problem, the continuous behavior causes the reachability problem generally even to be un-decidable (Alur et al. 1995). However, despite the undecidability result, various techniques and tools have been proposed in the past to solve (a subclass of) such problems that are practically relevant (Penberthy and Weld 1994; McDermott 2003; Li and Williams 2008; Coles et al. 2012; Shin and Davis 2005; Della Penna et al. 2009; Bryce and Gao 2015). A recent approach in this direction has been proposed by Bogomolov et al. (2014), who exploit the close relationship of hybrid planning domains and hybrid automata. More precisely, Bogomolov et al. provide a general framework to translate PDDL+ to the formalism of standard hybrid automata. The translation guarantees that traces in the obtained hybrid automata correspond to operator sequences in the original planning domain, which basically allows one to apply model checking tools for hybrid automata to solve hybrid planning problems. As standard hybrid automata are well-studied in the model checking community, various model checking tools for this formalism exist.

Bogomolov et al.’s framework provides a first step in bridging the gap between the hybrid planning and the model checking world. However, their approach suffers from the fact that must transitions, i.e., transitions that must fire as soon as they become enabled, cannot be handled precisely, but only as an approximation. Hence, processes and events in PDDL+ (which feature must transitions) cannot be handled precisely by their translation either. This is a quite significant restriction, as processes and events represent an essential ingredient of many realistic hybrid planning domains. While the approximation is safe in the sense that plan non-existence can be proven, it does not guarantee to yield valid plans in domains where processes and events exist.

In this paper, we present the theoretical foundations for extending Bogomolov et al.’s approach to precisely handle must behavior. In more details, we provide a translation from a given hybrid automaton with must transitions to an equivalent hybrid automaton with may transitions. Our translation yields equal reachable state spaces for linear hybrid automata, and can handle hybrid automata with affine dynamics with an over-approximation that can be made arbitrarily precise. Overall, translating must behavior precisely opens a way towards an exact PDDL+ translation into the formalism of standard hybrid automata because processes and events can be handled precisely as well.

2 Preliminaries

In this section, we introduce the PDDL+ language and define hybrid automata (HA) and their semantics.

The PDDL+ Language PDDL+ is particularly suited for modelling planning domains with a mixed discrete-continuous dynamics. This formalism provides an expressive language to define hybrid planning domains. In particular, a designer can define function and relation symbols, instantaneous and durative actions, events and processes. In this work, we focus on modelling must transitions, i.e., is-
sues relevant for processes and events. For example, consider
the following event formalized in PDDL+:

```plaintext
(:event tankEmpty
 :parameters (?g - generator ?t - tank)
 :precondition (and (using ?t ?g)
  (= (fuelInTank ?t) 0))
 :effect (and (not (using ?t ?g))))
```

This event is triggered if the tank is in use and the fuel
level is smaller or equal to 0. In other words, assuming
we can model transitions which must fire as soon as
the guard is enabled, we can use them as building blocks for
events. We can reason in a similar way also for processes.

**Hybrid automata** We first provide some auxiliary nota-
tions. A **convex polyhedron** is a subset of $\mathbb{R}^n$ that
be represented as the intersection of a finite number of strict
and non-strict affine half-spaces. A **polyhedron** is a subset
of $\mathbb{R}^n$ that can be represented as the union of a finite num-
ber of convex polyhedra. Given a polyhedron $G \subseteq \mathbb{R}^n$, we
denote its topological closure by $\mathcal{C}(G)$. We denote its repre-
sentation as a finite set of disjoint convex polyhedra by $[P]$.

Given an ordered set $X = \{x_1, \ldots, x_n\}$ of variables, a
**valuation** is a function $v : X \to \mathbb{R}$. Let $Y \subseteq X$ a set of
variables, we denote by $v|_Y$ the projection of $v$ onto $Y$. Let
$Val(X)$ denote the set of valuations over $X$. There is an
obvious bijection between $Val(X)$ and $\mathbb{R}^n$, allowing us to
extend the notion of a polyhedron to the sets of valuations.
We denote by $CPoly(X)$ and $Poly(X)$ the set of convex
and general polyhedra on $X$, respectively. The set $X' =
\{\hat{x}_1, \ldots, \hat{x}_n\}$ stands for the set of dotted variables which
represent the first derivatives. The set $X'' = \{x'_1, \ldots, x'_n\}$
denotes the set of primed variables which represent the new
values of variables after a discrete transition. A continuous
**activity** over $X$ is a function $f : \mathbb{R}^{2n} \to Val(X)$ that is
continuous on its domain and differentiable except for a
finite set of points. Let $Acts(X)$ denote the set of activities
over $X$. The **derivative** of an activity $f$ is defined in the
standard way and is a partial function $f' : \mathbb{R}^{2n} \to Val(X)$.

A hybrid automaton $\mathcal{H} = (Loc, X, Edg, Flow, Inv, Init)$
consists of the following components:

- **Loc** is a finite set of locations, and $X = \{x_1, \ldots, x_n\}$ is
  a finite set of real-valued variables. A **state** is a pair $(l, v)$
of a location $l$ in Loc and a valuation $v \in Val(X)$.

- **Edg** is a finite set of **discrete transitions** that describes
  instantaneous location changes. Each transition $(l, \eta, l') \in
  Edg$ consists of a **source location** $l$, a **target location** $l'$ and
  a **jump relation** $\eta \in Poly(X \cup X')$ that specifies how the
  variables may change their value during the transition. The
  **guard** is the projection of $\eta$ on $X$ and describes the
  valuations for which the transition is enabled.

- **Flow** is a mapping $Flow : Loc \to CPoly(X \cup X)$ that
  attributes to each location a set of valuations over the variables
  and over their first derivatives. This set determines how vari-
  ables can change over time. We refer to a HA with a flow of the form $Flow : Loc \to CPoly(X)$, i.e., with a flow
  that reasons only about the first derivatives, as a **linear
  hybrid automaton (LHA)**. We denote a HA with a flow
  that constrains both the variables and their derivatives an
  **affine hybrid automaton (AHA)**.

- **Inv** is a mapping $Inv : Loc \to CPoly(X)$ that is a mapping that
  defines the location **invariants**. Finally, a mapping $Init : Loc \to
  CPoly(X)$ defines the initial states of the automaton.

The set of states of $\mathcal{H}$ is $S = Loc \times Val(X)$. Moreover, we
use the shorthand notations $InvS = \bigcup_{l \in Loc} (\{l\} \times Inv(l))$. Given a set of states $A \subseteq S$, a set of locations $L \subseteq Loc$
and a set of variables $Y \subseteq X$, we denote by the projection
of $A$ onto $L$ and $Y$ the set of valuations $A \parallel_{L, Y} = \{v | v \in
Val(Y) \mid (l, v) \in A \land l \in L \}$.

**May Semantics** The behavior of a HA is based on two
types of steps: discrete steps correspond to the Edg com-
ponent, and produce an instantaneous change in both the
location and the variable valuation; continuous steps de-
scribe the change of the variables over time in accordance
with the Flow component. Given a state $s = (l, v)$, we set
$Loc(s) = l$ and $val(s) = v$. An activity $f \in Acts(X)$ is called
admissible from $s$ if (i) $f(0) = v$ and (ii) for all $\delta \geq 0$, if $f(\delta)$ is defined then $f(\delta) \in Flow(l)$. We denote by
$Adm(s)$ the set of activities that are admissible from $s$.

Given two states $s, s'$, and a transition $e \in Edg$, there is
a **discrete step** $s \xrightarrow{e} s'$ with **source** $s$ and **target** $s'$ if
(i) $s, s' \in InvS$, (ii) $e = (Loc(s), \eta, Loc(s'))$, and (iii)
$\forall l \in {\text{loc}}'(X' \cup X)$ defines $\eta$, where $\forall l \in {\text{loc}}'(X' \cup X)$ is the
valuation in $Val(X')$ obtained from $s'$ by renaming each variable
in $X$ with the corresponding primed variable in $X'$. Whenever
condition (iii) holds, we say that $e$ is **enabled in $s$**.

There is a **continuous step** $s \xrightarrow{f \Delta} s'$ with **duration** $\Delta \in \mathbb{R}_{\geq 0}$
and activity $f \in Adm(s)$ if (i) $s \in InvS$, (ii) for all $0 < \Delta' \leq \Delta$, $\forall l \in {\text{loc}}'(X' \cup X)$ defines $\eta$, and (iii)
$\forall l \in {\text{loc}}'(X' \cup X)$ defines $\eta$, where $\forall l \in {\text{loc}}'(X' \cup X)$ is the
valuation in $Val(X')$ obtained from $s'$ by renaming each variable
in $X$ with the corresponding primed variable in $X'$. Whenever
condition (iii) holds, we say that $e$ is **enabled in $s$**.

A run is a sequence $r = s_0 \xrightarrow{f_0} s_1 \xrightarrow{f_1} \cdots \xrightarrow{f_{n}} s_n$ of alternating timed and discrete steps. Given the automaton
$\mathcal{H}$, the set of all states and valuations reachable by runs is
denoted by $Reach(\mathcal{H})$ and $CReach(\mathcal{H})$, respectively. Note
that in the default (may) semantics of a discrete step, the
guard only provides the information on when an automaton
may make a discrete step. Hence, even if a guard of an out-
going transition is satisfied, the automaton may proceed by
following an activity as long as the invariant is not violated.
This semantics is often referred to as **may semantics**.

**Example 1.** Consider the HA in Figure 1(a). This HA has
two locations $l$ and $l'$ and two continuous variables $x$ and
$y$. Furthermore, there is a transition from $l$ to $l'$ with the
**guard** $G = x \geq 3 \land y \geq 2$. The initial state is given by
$s_0 = (l, x = 1 \land y = 1)$. The blue regions in Figure 1(b)
and Figure 1(c) show the reachable valuations from $s_0$ by
following linear and affine dynamics, respectively. We ob-
serve that the HA may stay in $l$ also when $G$ is satisfied.

**Must semantics** We consider a further class of discrete transitions which we call **must transitions**. Informally, a HA
is enforced to take a discrete must transition as soon as its
guard is satisfied. We formally define a continuous step for a
must transition with the guard $G$ in the following way. Given
two states $s$ and $s'$, there is a continuous step $s \xrightarrow{\delta_f} s'$ with duration $\delta \in \mathbb{R}^{\geq 0}$ and activity $f \in \text{Adm}(s)$ iff (i) there exists a continuous step $s \xrightarrow{\delta_f} s'$ in the corresponding HA with may semantics, and (ii) for all $0 \leq \delta' < \delta$, $f(\delta') \notin G$ and $f(\delta) \in G$. In other words, the $\delta$ represents the first time moment when the guard is satisfied. Assuming a discrete transition in Figure 1(a) to honor the must semantics, then the reachable region in Figure 1(b) is reduced to the closure of the difference between the blue region and the guard.

### 3 Translation of Must Semantics

In this section, we describe a translation of a given hybrid automaton featuring must transitions to an equivalent hybrid automaton with only may transitions (a corresponding theorem is given at the end of the section). We first provide the intuition behind our construction based on the HA $H_M$ in Figure 1(a). Assume that the transition from $l$ to $l'$ with the guard $G$ honors the must semantics. Our translation leads to the may automaton $H_m$ in Figure 2. We will discuss our construction step-by-step based on this example.

For our translation, we first consider a negation of the guard $G$ and partition this set into a finite number of disjoint sets. Intuitively, in this way, we define the regions from which the guard can be reached. In our example, we have two sets $Q_1 = x < 3$ and $Q_2 = x \geq 3 \land y < 2$. For those sets we introduce auxiliary locations $l_1$ and $l_2$ with the invariants $Q_1$ and $Q_2$, respectively. We say that the guard $G$ induces those two locations. Furthermore, those locations exhibit the same flow as the original location $l$. In this way, the behavior of a HA in those locations mimics the behavior in the location $l$. However, the guard $G$ can be reached from neither of them. We observe that the state of the original HA may generally evolve from $Q_1$ to $Q_2$ or vice versa. In the current version of the translation, this is, however, prohibited as $Q_1 \cap Q_2 = \emptyset$. In order to account for this issue, we add a further auxiliary location $l_{12}$ with the invariant $x \leq 3 \land y < 2$ and connect it to both $l_1$ and $l_2$. Note that this invariant does not allow to enable the guard $G$, however, makes the transition between $l_1$ and $l_2$ possible.

We move on by adding the locations $l_1'$ and $l_2'$. We use these locations to properly reflect the situation when the automaton $H_M$, by following an activity, reaches the border of the guard $G$. For this purpose, we add as invariants the closures of the sets $Q_1$ and $Q_2$, respectively. Furthermore, we add a clock $t$ to measure the dwelling time in those locations and the invariant of the form $t \leq \epsilon$. In other words, the HA might stay at the locations $l_1$ and $l_2$ at most $\epsilon$ time units. We will discuss this design choice in more details below.

Finally, we add an urgent location $l_u$, i.e. we leave it for the target location $l'$ of the must transition immediately after entering it. This location features the invariant equal to the guard $G$. We use this location to collect all options which enable the must transition.

Now we discuss the idea behind the $\epsilon$-invariant of the locations $l_1'$ and $l_2'$. The main challenge in reflecting the must semantics lies in capturing the first time moment when the guard is enabled. This becomes problematic if a run does not proceed to the interior of the guard after touching its border. We illustrate this problem on the following example (see Figure 3(a)). Let us assume that the location $l$ has the flow of the form $x \in [-1, 1]$. In this setting, a HA might reach the border of the guard using the activity $f$ and switch afterwards to the activity $f_0$ which is constant in the dimension $x$. Therefore, our translation schema would allow the HA state to evolve according to activity $f_0$ for $\epsilon$ time units in either location $l_1$ or $l_2$, whereas the must transition is required to fire immediately. However, at the same moment, we observe that all the reachable valuations within those $\epsilon$ time units are anyway reachable. To see this, we refer to the following fundamental property of the class of LHA (Wong-Toi 1997): for the reachable states within one continuous step, it holds that, if a valuation is reachable by a sequence of activities, then it is also reachable by a single activity. In other words, we can always replace a zigzag run with one in the form of a line. Hence, for the class of LHA, we can conclude that the set of reachable valuations of the translation coincides with the one of the original automaton.

Considering the size of the translation, we observe that for every must transition (given the negation of its guard can be partitioned into $n$ disjoint convex polyhedra), our construction induces $O(n^2)$ locations. In common planning benchmarks, $n$ is typically small. In contrast, the method introduced in Bogomolov et al. does not introduce auxiliary locations as it over-approximates the must semantics.

Now we proceed to the case of AHA (see Figure 3(b)). We observe that the run touches the guard twice. Furthermore, we assume that the time progresses for $\delta$ time units in between. We distinguish two cases. If $\epsilon \geq \delta$, then the HA can dwell in the locations $l_1$ or $l_2$ long enough to touch
Given two convex polyhedra $A$ and $B$, their boundary is defined as follows:

$$\text{bndry}(A, B) = (\text{cl}(A) \cap B) \cup (A \cap \text{cl}(B)).$$

Clearly, $\text{bndry}(A, B)$ is nonempty only if $A$ and $B$ are adjacent to one another or they overlap; otherwise, it is empty.

Now formally describe our construction. Let $\mathcal{H}_M = (\text{Loc}, X, \text{Lab}, \text{Edg}, \text{Flow}, \text{Inv}, \text{Init})$ be an automaton with must semantics, consisting of two locations $l$ and $l'$ and a single must transition from $l$ to $l'$. The transition guard is provided by a closed convex polyhedron $G$. Assumption that $[G] = Q_1 \cup \ldots \cup Q_n$, the may automaton $\mathcal{H}_M = (\text{Loc}', X', \text{Lab}', \text{Edg}', \text{Flow}', \text{Inv}', \text{Init}')$ is defined by:

- $\text{Loc}' = \{l'\} \cup \text{Loc}_1 \cup \text{Loc}_2 \cup \{l_0\}$. Here, $l'$ is the target location of the must transition in $\mathcal{H}_M$, $l_0$ is a further auxiliary location, $\text{Loc}_i = \bigcup_{i \in [1..n]} \{l_i, l_i\}$, and $\text{Loc}_2 = \bigcup_{i,j \in [1..n]} \{l_{ij}\}$.

Intuitively, the set $\text{Loc}_2$ stores a set of auxiliary locations which are used to ensure that the transition from the polyhedron $Q_i$ to $Q_j$ is possible. In this way, we enable the transitions of the form $Q_i \rightarrow B \rightarrow Q_j$. Furthermore, we ensure that $B \cap G = \emptyset$, i.e., it is impossible to reach the guard from $B$.

- $X' = X \cup \{t\}$, where $t$ is an auxiliary clock.

- $\text{Edg}'$ is defined as follows: For all $l_i \in \text{Loc}'$, it holds that $\langle l_i, \mu, l_i \rangle \in \text{Edg}'$, where $\mu$ defines an update function to reset the variable $t \in X$.

4 Conclusions

We have presented the theoretical foundations for translating hybrid automata with must transitions to hybrid automata with may transitions. Our construction results in the same reachable state space for linear hybrid automata. For hybrid automata with affine dynamics, the resulting reachable state space is over-approximated in an arbitrarily precise way. Overall, our construction provides the foundation for exactly translating PDDL+ problems in their full generality (including processes and events) into standard hybrid automata.
Acknowledgments
This work was partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center “Automatic Verification and Analysis of Complex Systems” (SFB/TR 14 AVACS, http://www.avacs.org/), by the European Research Council (ERC) under grant 267989 (QUAREM), by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award), and by the Swiss National Science Foundation (SNSF) as part of the project “Automated Reformulation and Pruning in Factored State Spaces (ARAP)”.

References


