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

Sergiy Bogomolov  
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 re-lected 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 undecidable (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.,
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)
 :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 that 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 notations. A *convex polyhedron* is a subset of $\mathbb{R}^n$ that can 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 number of convex polyhedra. Given a polyhedron $G \subseteq \mathbb{R}^n$, we denote its topological closure by $cl(G)$. We denote its representation 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 $v : X \rightarrow \mathbb{R}$. 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 $\hat{X} = \{\hat{x}_1, \ldots, \hat{x}_n\}$ 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 $X'$ over $\mathbb{R}^{\geq 0}$ 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 $\dot{f}$ of an activity $f$ is defined in the standard way and is a partial function $f : \mathbb{R}^{\geq 0} \rightarrow Val(\hat{X})$.

A hybrid automaton $\mathcal{H}$ is a pair $(X, Loc, Edg, Flow, Inv, Init)$ 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 $\eta \in Poly(\hat{X})$ 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** : $Loc \rightarrow CPoly(\hat{X})$ is a mapping that attributes to each location a set of valuations over the variables and over their first derivatives. This set determines how variables can change over time. We refer to a HA with a flow of the form $Flow : Loc \rightarrow CPoly(\hat{X})$, 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** : $Loc \rightarrow CPoly(X)$ is a mapping that defines the location invariants. Finally, a mapping $Init : Loc \rightarrow CPoly(X)$ 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 $A \parallel_{LY} = \{v | v \in Val(Y) \mid (l, v) \in A \land l \in L\}$.

### (May) Semantics

The behavior of a HA is based on two types of steps: discrete steps correspond to the Edg component, and produce an instantaneous change in both the location and the variable valuation; continuous steps describe 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* $s \rightarrow s'$ with source $s$ and target $s'$ iff (i) $s, s' \in InvS$, (ii) $e = (Loc(s), \eta, Loc(s'))$, and (iii) $(val(s), val(s')(\hat{X}/X)) \in \eta$, where $val(s')(\hat{X}/X)$ 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* $s \xrightarrow{\Delta} s'$ with duration $\Delta \in \mathbb{R}^{\geq 0}$ and activity $f \in Adm(s)$ iff (i) $s \in InvS$, (ii) for all $0 < \Delta' \leq \Delta$, $((l, f(\Delta'))) \in InvS$, and (iii) $s' = (Loc(s), f(\Delta))$.

A run is a sequence $r = s_0 \xrightarrow{\Delta_0} s_1 \xrightarrow{\Delta_1} \ldots \xrightarrow{\Delta_n} s_n$ of alternating timed and discrete steps. Given the automaton $\mathcal{H}$, the set of all states and valuations reachable by runs is denoted by $Reach(\mathcal{H})$. 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 outgoing 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 $G \equiv x \geq 3 \land y \geq 2$. 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 observe 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

```plaintext
(\cdot event tankEmpty
 :parameters (?g - generator ?t - tank)
 :precondition (and (using ?t ?g)
 :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 that 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.
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 may 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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants 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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.

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 automaton with only may transitions (a corresponding automaton featuring must transitions to an equivalent hybrid invariants of the difference between the blue region and the guard.
the guard border twice and thus violate the must semantics. Otherwise, the invariant $t \leq \varepsilon$ ensures that the run reaches the guard only once. Therefore, by picking a small enough $\varepsilon$ we can ensure that the guard is reached only once before the invariant $t \leq \varepsilon$ is violated.

Finally, we remark that the set $\text{Reach}(H_m)$ still contains the additional states reachable in the locations $I_l$ or $L_2$ within the $e$ time units. Moreover, the valuations of this set are defined for the extra variable $t$. In order to account for this issue, we compute the projection $\text{Reach}(H_m)[t_i, t_1, t_2, t_3, t_4, (x, y)]$.

To extend the technique to HA with multiple locations and transitions, we apply the construction to each location $l$ of $H_M$ such that $l$ is the source of a must transition. If $l$ has several outgoing must transitions, then we apply our construction with the guard $G$ equal the union of the individual guards. Finally, given a may transition from $l$ to another location $l''$, we just add transitions between either those locations or the corresponding induced locations.

Before presenting a general construction of our translation, we introduce an auxiliary definition of the boundary. Given two convex polyhedra $A$ and $B$, their boundary is

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

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 we formally describe our construction. Let $H_M = (L, X, \text{Lab}, \text{Edg}, \text{Flow}, \text{Inv}, \text{Init})$ 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 $G$. Assuming that $[G] = Q_1 \cup \ldots \cup Q_n$, the may automaton $H_m = (L', X', \text{Lab}', \text{Edg}', \text{Flow}', \text{Inv}', \text{Init}')$ is defined by

- $L' = \{l'\} \cup \text{Loc}_1 \cup \text{Loc}_2 \cup \{l_n\}$. Here, $l'$ is the target location of the must transition in $H_M$, $L_n$ is a further auxiliary location, $\text{Loc}_1 = \{i \in [1..n] | \{l_i, l_i\}\}$, and $\text{Loc}_2 = \{i \in [1..n] | B \subset \text{cl}(Q_i) \cap G \} \{l_{ij}\}$.

- 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 $l_i \in L'$, it holds that $\langle l_i, \mu, l_i' \rangle \in \text{Edg}'$, where $\mu$ defines an update function to reset the variable $t \in X$. For every $l_i \in L'$, if there exists a location $l_{ij} \in \text{Loc}_2$, then $\langle l_i, \mu, l_{ij} \rangle, \langle l_{ij}, \mu, l_i' \rangle \in \text{Edg}'$, where the guard is true and the update function is the identity. For every $\bar{l}_i \in L'$, it holds $\langle \bar{l}_i, \mu, t \rangle \in \text{Edg}'$, where $\mu$ defines an update function to reset the variable $t \in X$. Finally, there is a transition $\langle l_i, \mu, l_i' \rangle \in \text{Edg}'$, where the update function corresponds the original update function and the guard is true.

- $\text{Flow}'$ is defined as follows: For every location $l_i \in L'$, $\text{Flow}'(l_i) = \text{Flow}(l)$ holds, where $l$ is the source location of the must transition in $H_M$. For every location $\bar{l}_i \in L'$, it holds $\text{Flow}'(\bar{l}_i) = \text{Flow}(l) \cup \{t = 1\}$, where $l$ is the source location of the must transition in $H_M$.

- $\text{Inv}'$ is defined as follows: For every location $l_i \in L'$, $\text{Inv}'(l_i) = Q_i$. For every location $\bar{l}_i \in L'$ and some arbitrarily small $\varepsilon \in \mathbb{R}^+$, it holds that $\text{Inv}'(\bar{l}_i) = \text{cl}(Q_i) \cap \{t \geq \varepsilon\}$. For each location $\bar{l}_i \in L'$, we have that $\text{Inv}'(\bar{l}_{ij}) = B$, where $B = [\text{cl}(Q_i) \setminus G]$ and $\text{bndry}(B, Q_i) \neq \emptyset$. Finally, $\text{Inv}'(l_n) = G \cap \{t = 0\}$, and $\text{Inv}'(l') = \text{Inv}'(l_i)$.

- For each $v \in \text{Init}(l)$, if $v \notin Q_i$, then $v \in \text{Init}'(l_i)$. Otherwise $v \in G$, then $v \in \text{Init}'(l_i)$.

According to the description above, our translation offers the following properties. A formal proof is given in a separate technical report (Bogomolov et al. 2015).

**Theorem 1.** For a hybrid automaton (LHA or AHA) $H_M = (L, X, \text{Lab}, \text{Edg}, \text{Flow}, \text{Inv}, \text{Init})$ with must transitions featuring closed guards, there exists a hybrid automaton $H_m = (L', X', \text{Lab}', \text{Edg}', \text{Flow}', \text{Inv}', \text{Init}')$ with may transitions and a location set $\text{Loc}_S \subseteq L'$ such that:

1. $\text{CReach}(H_M) = \text{Reach}(H_m) \cup \text{Loc}_S \times X$ for LHA.
2. $\text{CReach}(H_M) \subseteq \text{Reach}(H_m) \cup \text{Loc}_S \times X$ for AHA, and the approximation can be made arbitrarily precise.

In Theorem 1, the set $\text{Loc}_S$ contains locations of the form $\bar{l}_i$ which are projected away as part of the construction. The theorem states the equivalence and inclusion relationships for the set of reachable valuations of the automata, which is the crucial part in the must behavior translation. The general result for the entire reachable state space (including locations) holds as well (Bogomolov et al. 2015).

**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