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 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., issues relevant for processes and events. For example, consider the following event formalized in PDDL+:

```
(: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 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 $v : X \to \mathbb{R}$. Let $Y \subseteq X$ a set of variables, we denote by $v\downarrow_Y$ 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 $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ 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 $f : \mathbb{R}^{\geq 0} \to Val(X)$ 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 $\dot{f} : \mathbb{R}^{\geq 0} \to Val(\dot{X})$.

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 $\langle l, v\rangle$ 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 η on X and describes the valuations for which the transition is enabled.
- $Flow : Loc \to CPoly(\dot{X} \cup X)$ 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 $Flow : Loc \to CPoly(\dot{X})$, 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).
- $Inv : Loc \to CPoly(X)$ is a mapping that defines the location *invariants*. Finally, a mapping $Init : Loc \to CPoly(X)$ defines the *initial states* of the automaton.

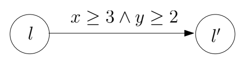
The set of states of $\mathcal{H}$ is $S = Loc \times Val(X)$. Moreover, we use the shorthand notations $InvS = \bigcup_{l \in Loc}(\{l\} \times Inv(l))$. Given a set of states $A \subseteq S$, a set of locations $L \subseteq Loc$ and a set of variables $Y \subseteq X$, we denote by the *projection*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

(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 $s = \langle l, v\rangle$, 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 $\dot{f}(\delta)$ is defined then $\dot{f}(\delta) \in Flow(l)$. We denote by $Adm(s)$ the set of activities that are admissible from s.

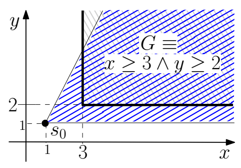
Given two states s, s', and a transition $e \in Edg$, there is a *discrete step* $s \xrightarrow{e} s'$ with *source* s and *target* s' iff *(i)* $s, s' \in InvS$, *(ii)* $e = (Loc(s), \eta, Loc(s'))$, and *(iii)* $(val(s), val(s')[X'/X]) \in \eta$, where $val(s')[X'/X]$ is the valuation in $Val(X')$ 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* $s \xrightarrow{\Delta, f} s'$ with *duration* $\Delta \in \mathbb{R}^{\geq 0}$ and activity $f \in Adm(s)$ iff *(i)* $s \in InvS$, *(ii)* for all $0 < \delta' \leq \Delta$, $(\langle l, f(\delta')\rangle) \in InvS$, and *(iii)* $s' = \langle Loc(s), f(\Delta)\rangle$. A run is a sequence $r = s_0 \xrightarrow{\Delta_0, f_0} s'_0 \xrightarrow{e_0} s_1 \to \cdots \to s_n$ 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 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 $G \equiv x \geq 3 \wedge y \geq 2$. The initial state is given by $s_0 = (\langle l, x = 1 \wedge y = 1\rangle)$. 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.*

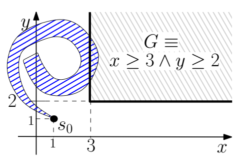
Must semantics We consider a further class of discrete transitions which we call *must* transitions. Informally, a HA is enforced to take a discrete must transition as soon as its guard is satisfied. We formally define a continuous step for a must transition with the guard G in the following way. Given



(a) Example HA



(b) LHA



(c) AHA

Figure 1: Set of reachable valuations for LHA and AHA (in location ℓ) which honors the may semantics.

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 $f \in Adm(s)$ 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 δ 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 $\mathcal{H}_M$ 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 $\mathcal{H}_m$ in Figure 2. We will discuss our construction step-by-step based on this example.

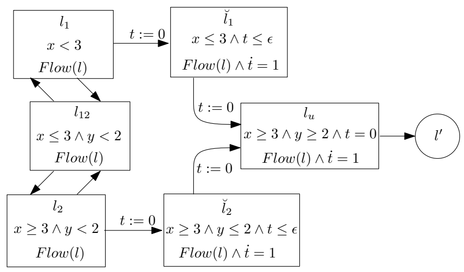

l_1
$x < 3$
$Flow(l)$
$t := 0$
$\breve{l}_1$
$x \leq 3 \wedge t \leq \epsilon$
$Flow(l) \wedge \dot{t} = 1$
l_{12}
$x \leq 3 \wedge y < 2$
$Flow(l)$
$t := 0$
l_u
$x \geq 3 \wedge y \geq 2 \wedge t = 0$
$Flow(l) \wedge \dot{t} = 1$
l'
$t := 0$
l_2
$x \geq 3 \wedge y < 2$
$Flow(l)$
$t := 0$
$\breve{l}_2$
$x \geq 3 \wedge y \leq 2 \wedge t \leq \epsilon$
$Flow(l) \wedge \dot{t} = 1$

Figure 2: The HA obtained by applying the construction.

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 $Q_2 = x \geq 3 \wedge y < 2$. 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 $x \leq 3 \wedge y < 2$ 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.

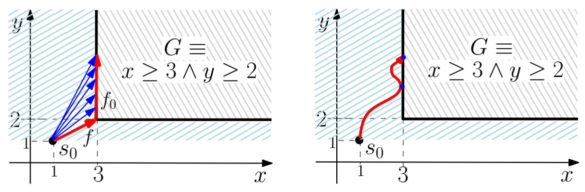
We move on by adding the locations $\breve{l}_1$ and $\breve{l}_2$. We use these locations to properly reflect the situation when the automaton $\mathcal{H}_M$, by following an activity, reaches the border of the guard G. For this purpose, we add as invariants the *closures* of the sets Q_1 and Q_2, respectively. Furthermore, we add a clock t to measure the dwelling time in those locations and the invariant of the form $t \leq \varepsilon$. In other words, the HA might stay at the locations $\breve{l}_1$ and $\breve{l}_2$ at most ε 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 ε-invariant of the locations $\breve{l}_1$ and $\breve{l}_2$. 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 $\dot{x} \in [-1, 1]$. 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 ε time units in either location $\breve{l}_1$ or $\breve{l}_2$, whereas the must transition is required to fire immediately. However, at the same moment, we observe that all the reachable valuations within those ε 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 $\mathcal{O}(n^2)$ 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 δ time units in between. We distinguish two cases. If $\varepsilon \geq \delta$, then the HA can dwell in the locations $\breve{l}_1$ or $\breve{l}_2$ long enough to touch



(a) LHA (b) AHA

Figure 3: Runs of HA with the may semantics (in location l).

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 ε we can ensure that the guard is reached only once before the invariant $t \leq \varepsilon$ is violated.

Finally, we remark that the set $Reach(\mathcal{H}_m)$ still contains the additional states reachable in the locations $\breve{l}_1$ or $\breve{l}_2$ within the ε 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 $Reach(\mathcal{H}_m) \downdownarrows_{\{l_1, l_{12}, l_2, l_u, l'\}, \{x,y\}}$.

To extend the technique to HA with multiple locations and transitions, we apply the construction to each location l of $\mathcal{H}_M$ 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 themselves 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

$$bndry(A, B) = (cl(A) \cap B) \cup (A \cap cl(B)). \quad (1)$$

Clearly, $bndry(A, B)$ 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 $\mathcal{H}_M = (Loc, X, Lab, Edg, Flow, Inv, Init)$ 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 $H_m = (Loc', X', Lab', Edg', Flow', Inv', Init')$ is defined by

- $Loc' = \{l'\} \bigcup Loc_i \bigcup Loc_b \bigcup \{l_u\}$. Here, l' is the target location of the must transition in $\mathcal{H}_M$, l_u is a further auxiliary location, $Loc_i = \bigcup_{i \in [1..n]} \{l_i, \breve{l}_i\}$, and $Loc_b = \bigcup_{\substack{i,j \in [1..n] \\ i \neq j}} \bigcup_{\substack{B \in [\![cl(Q_i) \setminus G]\!] \\ B \neq Q_i \\ bndry(B, Q_j) \neq \emptyset}} \{l_{ij}\}$. Intuitively, the set Loc_b 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 $Q_i \to B \to Q_j$. Furthermore, we ensure that $B \cap G = \emptyset$, i.e., it is impossible to reach the guard from B.
- $X' = X \bigcup \{t\}$, where t is an auxiliary clock.
- Edg' is defined as follows: For all $l_i \in Loc'$, it holds that $\langle l_i, \mu, \breve{l}_i \rangle \in Edg'$, where μ defines an update function to reset the variable $t \in X$. For every $l_i \in Loc'$, if there exists a location $l_{ij} \in Loc'$, then $\langle l_i, \mu, l_{ij} \rangle, \langle l_{ij}, \mu, l_i \rangle \in Edg'$, where the guard is true and the update function is the identity. For every $\breve{l}_i \in Loc'$, it holds $\langle \breve{l}_i, \mu, l_u \rangle \in Edg'$, where μ defines an update function to reset the variable $t \in X$. Finally, there is a transition $\langle l_u, \mu, l' \rangle \in Edg'$, where the update function corresponds the original update function and the guard is true.
- $Flow'$ is defined as follows: For every location $l_i \in Loc'$, $Flow'(l_i) = Flow(l)$ holds, where l is the source location of the must transition in $\mathcal{H}_M$. For every location $\breve{l}_i \in Loc'$, $Flow'(\breve{l}_i) = Flow(l) \cap \{\dot{t} = 1\}$ holds, where l is the source location of the must transition in $\mathcal{H}_M$, For every location $l_{ij} \in Loc'$, $Flow'(l_{ij}) = Flow(l)$ holds, where l is the source location of the must transition in $\mathcal{H}_M$, Finally, $Flow'(l_u) = Flow(l) \cap \{\dot{t} = 1\}$, where l is the source location of the must transition in $\mathcal{H}_M$, and $Flow'(l') = Flow(l')$.
- Inv' is defined as follows: For every location $l_i \in Loc'$, $Inv'(l_i) = Q_i$ holds. For every location $\breve{l}_i \in Loc'$ and some arbitrarily small $\varepsilon \in \mathbb{R}^{\geq 0}$, it holds that $Inv'(\breve{l}_i) = cl(Q_i) \cap \{t \geq \varepsilon\}$. For each location $l_{ij} \in Loc'$, we have that $Inv'(l_{ij}) = B$, where $B \in [\![cl(Q_i \setminus G)]\!]$ and $bndry(B, Q_j) \neq \emptyset$. Finally, $Inv'(l_u) = G \cap \{t = 0\}$, and $Inv'(l') = Inv(l')$.
- For each $v \in Init(l)$, if $v \in Q_i$ then $v \in Init'(l_i)$ holds. Otherwise if $v \in G$, then $v \in Init'(l')$.

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) $\mathcal{H}_M = (Loc, X, Edg, Flow, Inv, Init)$ with must transitions featuring closed guards, there exists a hybrid automaton $\mathcal{H}_m = (Loc', X', Edg', Flow', Inv', Init')$ with may transitions and a location set $Loc_\varepsilon \subset Loc'$ such that:*

1. *$CReach(\mathcal{H}_M) = Reach(\mathcal{H}_m) \downdownarrows_{Loc' \setminus Loc_\varepsilon, X}$ for LHA.*
2. *$CReach(\mathcal{H}_M) \subseteq Reach(\mathcal{H}_m) \downdownarrows_{Loc' \setminus Loc_\varepsilon, X}$ for AHA, and the approximation can be made arbitrarily precise.*

In Theorem 1, the set Loc_ε contains locations of the form $\breve{l}_i$ 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

Alur, R.; Courcoubetis, C.; Halbwachs, N.; Henzinger, T.; Ho, P.; Nicolin, X.; Olivero, A.; Sifakis, J.; and Yovine, S. 1995. The algorithmic analysis of hybrid systems. *Theoretical Computer Science* 138:3–34.

Bogomolov, S.; Magazzeni, D.; Podelski, A.; and Wehrle, M. 2014. Planning as model checking in hybrid domains. In *Proceedings of the Twenty Eighth Conference on Artificial Intelligence (AAAI-14)*. AAAI Press.

Bogomolov, S.; Magazzeni, D.; Minopoli, S.; and Wehrle, M. 2015. From PDDL+ to hybrid automata: Foundations of translating must behavior: Proof. Technical Report TR-2015-3, Verimag.

Bryce, D., and Gao, S. 2015. SMT-based nonlinear PDDL+ planning. In *Proceedings of the Twenty Nineth Conference on Artificial Intelligence (AAAI-15)*. AAAI Press.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)* 44:1–96.

Della Penna, G.; Magazzeni, D.; Mercorio, F.; and Intrigila, B. 2009. UPMurphi: A tool for universal planning on PDDL+ problems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS 2009)*. AAAI.

Fox, M., and Long, D. 2006. Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research (JAIR)* 27:235–297.

Li, H. X., and Williams, B. C. 2008. Generative planning for hybrid systems based on flow tubes. In *ICAPS*, 206–213.

McDermott, D. V. 2003. Reasoning about autonomous processes in an estimated-regression planner. In *ICAPS*, 143–152.

Penberthy, J. S., and Weld, D. S. 1994. Temporal planning with continuous change. In *AAAI*, 1010–1015.

Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a sat-based planner. *Artif. Intell.* 166(1-2):194–253.

Wong-Toi, H. 1997. The synthesis of controllers for linear hybrid automata. In *IEEE Conf. Decision and Control*, 4607 – 4612. IEEE Computer Society.