2011 50th IEEE Conference on Decision and Control and
European Control Conference (CDC-ECC)
Orlando, FL, USA, December 12-15, 2011

Revisiting Synthesis of Switching Controllers for Linear Hybrid Systems

Massimo Benerecetti Marco Faella Stefano Minopoli

Abstract— **We study the problem of automatically generating switching controllers for the class of Linear Hybrid Automata, with respect to safety objectives. We identify and solve inaccuracies contained in previous characterizations of the problem, providing a sound and complete symbolic fixpoint procedure, based on polyhedral abstractions of the state space. We also prove the termination of each iteration of the procedure. Some promising experimental results are presented, based on an implementation of the semi-algorithm on top of the tool PHAVer.**

I. Introduction

Hybrid systems are an established formalism for modeling physical systems which interact with a digital controller. From an abstract point of view, a hybrid system is a dynamic system whose state variables are partitioned into discrete and continuous ones. Typically, continuous variables represent physical quantities like temperature, speed, etc., while discrete ones represent *control modes*, i.e., states of the controller.

Hybrid automata [9] are the most common syntactic variety of hybrid system: a finite set of locations, similar to the states of a finite automaton, represents the value of the discrete variables. The current location, together with the current value of the (continuous) variables, form the instantaneous description of the system. Change of location happens via discrete transitions, and the evolution of the variables is governed by differential equations attached to each location. In a Linear Hybrid Automaton (LHA), the allowed differential equations are in fact differential inclusions of the type $\dot{\mathbf{x}} \in P$, where $\dot{\mathbf{x}}$ is the vector of the first derivatives of all variables and $P \subseteq \mathbb{R}^n$ is a convex polyhedron. Notice that differential inclusions are non-deterministic, allowing for infinitely many solutions.

The most studied problem for hybrid systems is *reachability*: computing the set of states that are reachable from the initial states, in any amount of time. The reachability problem for LHAs was proved undecidable in [11], indicating that no exact discrete abstraction exists. The complexity standing of the problem was further refined to *semi-decidable* in [15], whose results imply that it is possible to exactly compute the set of states that are reachable within a bounded number of discrete transitions (*bounded-horizon reachability*).

We study LHAs whose discrete transitions are partitioned into controllable and uncontrollable ones, and we wish to compute a strategy for the controller to satisfy a given goal, regardless of the evolution of the continuous variables and of the uncontrollable transitions. Hence, the problem can be viewed as a *two player game* [14]: on one side the controller, who can only issue controllable transitions, on the other side the environment, who can choose the trajectory of the variables and can take uncontrollable transitions whenever they are enabled.

As control goal, we consider safety, i.e., the objective of keeping the system within a given region of safe states. This problem has been considered several times in the literature. Here, we fix some inaccuracies in previous presentations, propose a sound and complete procedure for the problem[1], and we present a publicly available implementation of the procedure, in a tool called PHAVer+. In particular, we present a novel algorithm for computing the set of states that may reach a given polyhedral region while avoiding another one, a problem that is at the heart of the synthesis procedure.

Contrary to most recent literature on the subject [1], [2], [8], we focus on exact algorithms. Although it is established that exact analysis and synthesis of realistic hybrid systems is computationally demanding, we believe that the ongoing research effort on approximate techniques should be based on the solid grounds provided by the exact approach. For instance, a tool implementing an exact algorithm (like our PHAVer+) may serve as a benchmark to evaluate the performance and the precision of an approximate tool.

Related work: The idea of automatically synthesizing controllers for dynamic systems arose in connection with discrete systems [13], and was later applied to real-time systems modeled by timed automata [12], and finally to hybrid systems [15], [10], and in particular to Linear Hybrid Automata, the very model that we analyze in this paper. Wong-Toi proposed the first symbolic procedure to compute the controllable region of a LHA w.r.t. a safety goal [15]. The heart of the procedure lies in the operator *flow_avoid*(U, V), which computes the set of system configurations from which a continuous trajectory may reach the set U while avoiding the set V (hence, in this paper we call this operator RWA, for *Reach While Avoiding*). Tomlin et al. [14] and Balluchi et al. [3] analyze much more expressive models, with generality in mind rather than automatic synthesis. Their *Reach* and *Unavoid_Pre* operators, respectively, again correspond to *flow_avoid*.

As explained in Section III-D, the algorithm provided in [15] for *flow_avoid*(U, V) does not work for non-convex V, a case which is very likely to occur in practice, even if the original safety goal is convex. A slightly different algorithm for *flow_avoid* is reported to have been implemented in the

All authors are with the Department of Physics, Università di Napoli "Federico II", Italy. {mfaella,bene,minopoli}@na.infn.it

[1]In other words, an algorithm that may or may not terminate, and that provides the correct answer whenever it terminates.

tool HONEYTECH [7], and we compare it with ours in Section III-D.

Asarin et al. [1] investigate the synthesis problem for hybrid systems where all discrete transitions are controllable and the trajectories satisfy given linear differential equations of the type $\dot{\mathbf{x}} = A\mathbf{x}$. The expressive power of these constraints is incomparable with the one offered by the differential inclusions occurring in LHAs. In particular, linear differential equations give rise to deterministic trajectories, while differential inclusions are non-deterministic. In control theory terms, differential inclusions can represent the presence of environmental *disturbances*. The tool `d/dt` [2], by the same authors, is reported to support controller synthesis for safety objectives, but the publicly available version in fact does not.

The rest of the paper is organized as follows. Section II introduces and motivates the model. In Section III, we present the procedure which solves the synthesis problem. Section IV reports some experiments performed on our implementation of the procedure, while Section V draws some conclusions.

II. LINEAR HYBRID AUTOMATA

A *convex polyhedron* is a subset of $\mathbb{R}^n$ that is the intersection of a finite number of half-spaces. A *polyhedron* is a subset of $\mathbb{R}^n$ that is the union of a finite number of convex polyhedra. For a general (i.e., not necessarily convex) polyhedron $G \subseteq \mathbb{R}^n$, we denote by $[\![G]\!] \subseteq 2^{\mathbb{R}^n}$ its representation as a finite set of convex polyhedra.

Given an ordered set $X = \{x_1, \dots, x_n\}$ of variables, a *valuation* is a function $v : X \to \mathbb{R}$. Let $\mathit{Val}(X)$ denote the set of valuations over X. There is an obvious bijection between $\mathit{Val}(X)$ and $\mathbb{R}^n$, allowing us to extend the notion of (convex) polyhedron to sets of valuations. We denote by $\mathit{CPoly}(X)$ (resp., $\mathit{Poly}(X)$) the set of convex polyhedra (resp., polyhedra) on $\mathit{Val}(X)$.

We use $\dot{X}$ to denote the set $\{\dot{x}_1, \dots, \dot{x}_n\}$ of dotted variables, used to represent the first derivatives, and X' to denote the set $\{x'_1, \dots, x'_n\}$ of primed variables, used to represent the new values of variables after a transition. Arithmetic operations on valuations are defined in the straightforward way. An *activity* over X is a differentiable function $f : \mathbb{R}^{\geq 0} \to \mathit{Val}(X)$. Let $\mathit{Acts}(X)$ denote the set of activities over X. The *derivative* $\dot{f}$ of an activity f is defined in the standard way and it is an activity over $\dot{X}$. A *Linear Hybrid Automaton* (LHA) $H = (\mathit{Loc}, X, \mathit{Edg}_{\mathrm{c}}, \mathit{Edg}_{\mathrm{u}}, \mathit{Flow}, \mathit{Inv}, \mathit{Init})$ consists of the following:

- A finite set Loc of *locations*.
- A finite set $X = \{x_1, \dots, x_n\}$ of continuous, real-valued *variables*. A *state* is a pair $\langle l, v\rangle$ of a location l and a valuation $v \in \mathit{Val}(X)$.
- Two sets $\mathit{Edg}_{\mathrm{c}}$ and $\mathit{Edg}_{\mathrm{u}}$ of *controllable* and *uncontrollable transitions*, respectively. They describe instantaneous changes of locations, in the course of which variables may change their value. Each transition $(l, \mu, l') \in \mathit{Edg}_{\mathrm{c}} \cup \mathit{Edg}_{\mathrm{u}}$ consists of a *source location* l, a *target location* l', and a *jump relation* $\mu \in \mathit{Poly}(X \cup X')$, that specifies how the variables may change their value during the transition. The projection of μ on X describes the valuations for which the transition is enabled; this is often referred to as a *guard*.
- A mapping $\mathit{Flow} : \mathit{Loc} \to \mathit{CPoly}(\dot{X})$ attributes to each location a set of valuations over the first derivatives of the variables, which determines how variables can change over time.
- A mapping $\mathit{Inv} : \mathit{Loc} \to \mathit{Poly}(X)$, called the *invariant*.
- A mapping $\mathit{Init} : \mathit{Loc} \to \mathit{Poly}(X)$, contained in the invariant, defining the *initial states* from which all behaviors of the automaton originate.

We use the abbreviations $S = \mathit{Loc} \times \mathit{Val}(X)$ for the set of states and $\mathit{Edg} = \mathit{Edg}_{\mathrm{c}} \cup \mathit{Edg}_{\mathrm{u}}$ for the set of all transitions. Moreover, we let $\mathit{InvS} = \bigcup_{l \in \mathit{Loc}} \{l\} \times \mathit{Inv}(l)$ and $\mathit{InitS} = \bigcup_{l \in \mathit{Loc}} \{l\} \times \mathit{Init}(l)$. Notice that InvS and InitS are sets of states. For a polyhedron or a set of states P, we denote by $\overline{P}$ its complement.

A. Semantics

The behavior of a LHA is based on two types of transitions: *discrete* transitions correspond to the Edg component, and produce an instantaneous change in both the location and the variable valuation; *timed* transitions describe the change of the variables over time in accordance with the Flow component.

Given a state $s = \langle l, v\rangle$, we set $\mathit{loc}(s) = l$ and $\mathit{val}(s) = v$. An activity $f \in \mathit{Acts}(X)$ is called *admissible from* s if *(i)* $f(0) = v$ and *(ii)* for all $\delta \geq 0$ it holds $\dot{f}(\delta) \in \mathit{Flow}(l)$. We denote by $\mathit{Adm}(s)$ the set of activities that are admissible from s. Additionally, for $f \in \mathit{Adm}(s)$, the *span* of f in l, denoted by $\mathit{span}(f, l)$ is the set of all values $\delta \geq 0$ such that $\langle l, f(\delta')\rangle \in \mathit{InvS}$ for all $0 \leq \delta' \leq \delta$. Intuitively, δ is in the span of f iff f never leaves the invariant in the first δ time units. If all non-negative reals belong to $\mathit{span}(f, l)$, we write $\infty \in \mathit{span}(f, l)$.

a) Runs: Given two states s, s', and a transition $e \in \mathit{Edg}$, there is a *discrete transition* $s \xrightarrow{e} s'$ with *source* s and *target* s' iff: *(i)* $s, s' \in \mathit{InvS}$, *(ii)* $e = (\mathit{loc}(s), \mu, \mathit{loc}(s'))$, and *(iii)* $(\mathit{val}(s), \mathit{val}(s')) \in \mu$. There is a *timed transition* $s \xrightarrow{\delta, f} s'$ with *duration* $\delta \in \mathbb{R}^{\geq 0}$ and activity $f \in \mathit{Adm}(s)$ iff: *(i)* $s \in \mathit{InvS}$, *(ii)* $\delta \in \mathit{span}(f, \mathit{loc}(s))$, and *(iii)* $s' = \langle \mathit{loc}(s), f(\delta)\rangle$. For technical convenience, we admit timed transitions of duration zero[2]. A special timed transition is denoted $s \xrightarrow{\infty, f}$ and represents the case when the system follows an activity forever; this is only allowed if $\infty \in \mathit{span}(f, \mathit{loc}(s))$. Finally, a *joint transition* $s \xrightarrow{\delta, f, e} s'$ represents the timed transition $s \xrightarrow{\delta, f} \langle \mathit{loc}(s), f(\delta)\rangle$ followed by the discrete transition $\langle \mathit{loc}(s), f(\delta)\rangle \xrightarrow{e} s'$.

A *run* is a sequence

$$r = s_0 \xrightarrow{\delta_0, f_0} s'_0 \xrightarrow{e_0} s_1 \xrightarrow{\delta_1, f_1} s'_1 \xrightarrow{e_1} s_2 \cdots s_n \cdots \qquad (1)$$

[2]Timed transitions of duration zero can be disabled by adding a clock variable t to the automaton and requesting that each discrete transition happens when $t > 0$ and resets t to 0 when taken.

of alternating timed and discrete transitions, such that either the sequence is infinite, or it ends with a timed transition of the type $s_n \xrightarrow{\infty, f}$. If the run r is finite, we define $len(r) = n$ to be the length of the run, otherwise we set $len(r) = \infty$. The above run is *non-Zeno* if for all $\delta \geq 0$ there exists $i \geq 0$ such that $\sum_{j=0}^{i} \delta_j > \delta$. We denote by $States(r)$ the set of all states visited by r. Formally, $States(r)$ is the smallest set containing all states $\langle loc(s_i), f_i(\delta) \rangle$, for all $0 \leq i \leq len(r)$ and all $0 \leq \delta \leq \delta_i$. Notice that the states from which discrete transitions start (states s'_i in (1)) appear in $States(r)$. Moreover, if r contains a sequence of one or more zero-time timed transitions, all intervening states appear in $States(r)$.

b) Zenoness and well-formedness: A well-known problem of real-time and hybrid systems is that definitions like the above admit runs that take infinitely many discrete transitions in a finite amount of time (i.e., *Zeno* runs), even if such behaviors are physically meaningless. In this paper, we assume that the hybrid automaton under consideration generates no such runs. This is easily achieved by using an extra variable, representing a clock, to ensure that the delay between any two transitions is bounded from below by a constant. We leave it to future work to combine our results with more sophisticated approaches to Zenoness known in the literature [3], [6].

Moreover, we assume that the hybrid automaton under consideration is *non-blocking*, i.e., whenever the automaton is about to leave the invariant there must be an uncontrollable transition enabled. Formally, for all states s in the invariant, if all activities $f \in Adm(s)$ eventually leave the invariant, then there exists one such activity f and a time $\delta \in span(f, loc(s))$ such that $s' = \langle loc(s), f(\delta) \rangle$ is in the invariant and there is an uncontrollable transition $e \in Edg_{\mathrm{u}}$ such that $s' \xrightarrow{e} s''$. The well-formedness condition ensures that the system can always evolve in some way, be it a timed step or an uncontrollable transition. If a hybrid automaton is non-Zeno and non-blocking, we say that it is *well-formed*. In the following, all hybrid automata are assumed to be well-formed.

c) Strategies: A *strategy* is a function $\sigma : S \to 2^{Edg_{\mathrm{c}} \cup \{\perp\}} \setminus \emptyset$, where $\perp$ denotes the null action. As customary in this context, our strategies are *non-deterministic* and *memoryless* (or *positional*). A strategy can only choose a transition which is allowed by the automaton. Formally, for all $s \in S$, if $e \in \sigma(s) \cap Edg_{\mathrm{c}}$, then there exists $s' \in S$ such that $s \xrightarrow{e} s'$. Moreover, when the strategy chooses the null action, it should continue to do so for a positive amount of time, along each activity that remains in the invariant. If all activities immediately exit the invariant, the above condition is vacuously satisfied. This ensures that the null action is enabled in right-open regions, so that there is an earliest instant in which a controllable transition becomes mandatory.

Notice that a strategy can always choose the null action, even if the system is on the boundary of the invariant, because, in our interpretation, it is not the responsibility of the controller to ensure that the invariant is not violated.

We say that a run like (1) is *consistent* with a strategy σ if for all $0 \leq i < len(r)$ the following conditions hold:

- for all $\delta \geq 0$ such that $\sum_{j=0}^{i-1} \delta_j \leq \delta < \sum_{j=0}^{i} \delta_j$, we have $\perp \in \sigma(r(\delta))$;
- if $e_i \in Edg_{\mathrm{c}}$ then $e_i \in \sigma(s'_i)$.

We denote by $Runs(s, \sigma)$ the set of runs starting from the state s and consistent with the strategy σ.

d) Safety control problem: Given a hybrid automaton and a set of states $T \subseteq InvS$, the *safety control problem* asks whether there exists a strategy σ such that, for all initial states $s \in InitS$ and all runs $r \in Runs(s, \sigma)$, it holds $States(r) \subseteq T$. We call the above σ a *winning strategy*.

III. Safety Control

In this section, we consider a fixed LHA and we present a sound and complete semi-algorithm to solve the safety control problem.

A. The Synthesis Algorithm

We start by defining some preliminary operators. For a set of states A and $x \in \{\mathrm{u}, \mathrm{c}\}$, let $Pre_x^{\mathrm{m}}(A)$ (for *may predecessors*) be the set of states where some discrete transition belonging to Edg_x is enabled, which leads to A. Analogously, let $Pre_x^{\mathrm{M}}(A) = Pre_x^{\mathrm{m}}(A) \setminus Pre_x^{\mathrm{m}}(\overline{A})$ (the *must predecessors*) be the set of states where all enabled discrete transitions belonging to Edg_x lead to A, and there is at least one such transition enabled.

The following theorem provides a fixed-point characterization for the safety control problem, in terms of the *controllable predecessor operator*, defined below.

Theorem 1: The answer to the safety control problem for safe set $T \subseteq InvS$ is positive if and only if

$$InitS \subseteq \nu W \,.\, T \cap CPre(W),$$

where ν denotes the largest fixed point and $CPre$ is the controllable predecessor operator below.

Controllable predecessor operator: For a set of states A, the operator $CPre(A)$ returns the set of states from which the controller can ensure that the system remains in A during the next joint transition. This happens if for all activities chosen by the environment and all delays δ, one of two situations occurs:

- the systems stays in A up to time δ, while all uncontrollable transitions enabled up to time δ (included) also lead to A, or
- there exists a time $\delta' < \delta$, such that the system stays in A up to time δ', all uncontrollable transitions enabled up to time δ' (included) also lead to A, and the controller can issue a transition at time δ' leading to A.

To improve readability, for a set of states A, an activity f, and a time delay $\delta \geq 0$ (including infinity), we denote by $While(A, f, \delta)$ the set of states from which following the activity f for δ time units keeps the system in A all the time, and any uncontrollable transition taken meanwhile also leads into A. Formally,

$$While(A, f, \delta) = \Big\{ s \in S \,\Big|\, \forall 0 \leq \delta' \leq \delta : \langle loc(s), f(\delta') \rangle \in A \setminus Pre_{\mathrm{u}}^{\mathrm{m}}(\overline{A}) \Big\}.$$

We can now formally define the *CPre* operator and prove Theorem 1.

$$\begin{aligned}CPre(A) = \Big\{ s \in S \,\Big|\, & \forall f \in Adm(s), \delta \in span(f, loc(s)) : \\ & s \in While(A, f, \delta) \text{ or } \exists 0 \le \delta' < \delta : \\ & s \in While(A, f, \delta') \text{ and } \langle loc(s), f(\delta')\rangle \in Pre^{\mathrm{m}}_{\mathrm{c}}(A) \Big\}.\end{aligned}$$

B. Computing the Predecessor Operator

In this section, we show how to compute the value of the predecessor operator on a given set of states A, assuming that we can compute the following basic operations on arbitrary polyhedra G and G': the Boolean operations $G \cup G$, $G \cap G$, and $\overline{G}$; the topological closure $cl(G)$ of G; finally, for a given location $l \in Loc$, the *pre-flow* of G in l:

$$G{\swarrow_l} = \{ u \in Val(X) \mid \exists \delta \ge 0, c \in Flow(l) : u + \delta \cdot c \in G \}.$$

Notice that, for two polyhedra P and P', if $P \subseteq P'$ then $P \swarrow_l \subseteq P' \swarrow_l$ (monotonicity), and $(P{\swarrow_l}) \swarrow_l = P \swarrow_l$ (idempotence). All of the above operations are provided by standard polyhedra libraries, except for the pre-flow operation. An algorithm for exactly computing the pre-flow of general polyhedra is presented in [5].

In the following, we proceed from the basic components of *CPre* to the full operator. Given a set of states A and a location l, we denote by $A\!\downharpoonright_l$ the projection of A on l, i.e. $\{v \in Val(X) \mid \langle l, v\rangle \in A\}$. For all $A \subseteq InvS$ and $x \in \{\mathrm{u}, \mathrm{c}\}$, it holds:

$$Pre^{\mathrm{m}}_{\mathrm{x}}(A) = InvS \cap \bigcup_{(l,\mu,l') \in Edg_x} \mu^{-1}(A\!\downharpoonright_{l'}),$$

where $\mu^{-1}(Z)$ is the pre-image of Z w.r.t. μ. We also introduce the auxiliary operator RWA^{m} (*may reach while avoiding*). Given a location l and two sets of variable valuations U and V, $RWA^{\mathrm{m}}_l(U, V)$ contains the set of valuations from which the flow of the system *may* reach U while avoiding $V \cap \overline{U}$. Notice that on a dense time domain this is not equivalent to reaching U while avoiding V: If an activity avoids V in a right-closed interval, and then enters $U \cap V$, the first property holds, while the latter does not. Formally, we have:

$$\begin{aligned}RWA^{\mathrm{m}}_l(U, V) = \Big\{ u \in Val(X) \,\Big|\, & \exists f \in Adm(\langle l, u\rangle),\ \delta \ge 0 : \\ & f(\delta) \in U \text{ and } \forall\, 0 \le \delta' < \delta : f(\delta') \in \overline{V} \cup U \Big\}.\end{aligned}$$

An algorithm for effectively computing RWA^{m} is presented in the next section, while the following lemma states the relationship between *CPre* and RWA^{m}. Intuitively, consider the set B_l of valuations u such that from state $\langle l, u\rangle$ the environment can take a discrete transition leading outside A, and the set C_l of valuations u such that from $\langle l, u\rangle$ the controller can take a discrete transition into A. We use the RWA^{m} operator to compute the set of valuations from which there exists an activity that either leaves A or enters B_l, while staying in the invariant and avoiding C_l. These valuations do not belong to $CPre(A)$, as the environment can violate the safety goal within (at most) one discrete transition.

We say that a set of states $A \subseteq S$ is *polyhedral* if for all $l \in Loc$, the projection $A\!\downharpoonright_l$ is a polyhedron. We can now state the following lemma, whose proof is omitted and can be found in [4].

Lemma 1: For all polyhedral sets of states $A \subseteq InvS$, we have $CPre(A) =$

$$\bigcup_{l \in Loc} \{l\} \times \Big(A\!\downharpoonright_l \setminus RWA^{\mathrm{m}}_l\big(Inv(l) \cap (\overline{A\!\downharpoonright_l} \cup B_l), C_l \cup \overline{Inv(l)}\big) \Big)$$

where $B_l = Pre^{\mathrm{m}}_{\mathrm{u}}(\overline{A})\!\downharpoonright_l$ and $C_l = Pre^{\mathrm{m}}_{\mathrm{c}}(A)\!\downharpoonright_l$.

C. Computing the RWA^{m} Operator

In this section, we consider a fixed location l. Given two polyhedra G and G', we define their *boundary* to be

$$bndry(G, G') = (cl(G) \cap G') \cup (G \cap cl(G')).$$

We can compute RWA^{m} by the following fixpoint characterization.

Theorem 2: For all locations l and sets of valuations U, V, and W, let $\tau(U, V, W) =$

$$U \cup \bigcup_{P \in [\![\overline{V}]\!]} \bigcup_{P' \in [\![W]\!]} \Big(P \cap \big(bndry(P, P') \cap P'{\swarrow_l} \big){\swarrow_l} \Big). \quad (2)$$

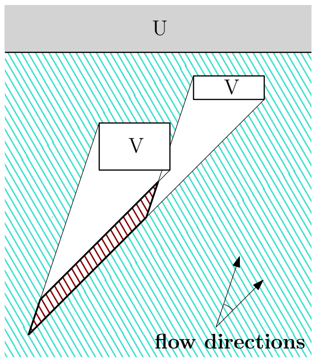
We have $RWA^{\mathrm{m}}_l(U, V) = \mu W \,.\, \tau(U, V, W)$.

Roughly speaking, $\tau(U, V, W)$ represents the set of points which either belong to U or do not belong to V and can reach W along a straight line which does not cross V. We can interpret the fixpoint expression $\mu W \,.\, \tau(U, V, W)$ as an incremental refinement of an under-approximation to the desired result. The process starts with the initial approximation $W_0 = U$. One can easily verify that $U \subseteq RWA^{\mathrm{m}}_l(U, V)$. Additionally, notice that $RWA^{\mathrm{m}}_l(U, V) \subseteq U \cup \overline{V}$. The equation refines the under-approximation by identifying its *entry regions*, i.e., the boundaries between the area which *may* belong to the result (i.e., $\overline{V}$), and the area which already belongs to it (i.e., W). That is, let $P \in [\![\overline{V}]\!]$ and $P' \in [\![W]\!]$, let $b = bndry(P, P')$, we call $R = b \cap P'{\swarrow_l}$ an entry region *from P to P'*, and also an entry region *of W*. The set R contains the points of b that may reach P' by following the flow of the system. Hence, the system may move from P to P' through R. Moreover, the set $R' = P \cap R{\swarrow_l}$ contains the points of P that can move to P' through R. Any point in $\overline{V}$ that may reach an entry region (without reaching V first) must be added to the under-approximation, since it belongs to $RWA^{\mathrm{m}}_l(U, V)$.

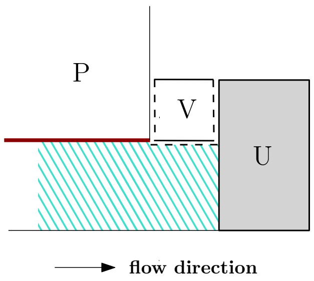
The following theorem states the termination of the fixpoint procedure defined in Theorem 2.

Theorem 3: The fixpoint procedure for RWA^{m} defined in Theorem 2 terminates in a finite number of steps.

Notice that $[\![\overline{V}]\!]$ and $[\![U]\!]$ are finite sets of convex polyhedra, therefore so is the number of initial entry regions of $[\![U]\!]$. Intuitively, the fixpoint procedure of Theorem 2 applies the refinement steps according to a breadth-first policy, starting from the initial entry regions. In other words, at every iteration each entry region discovered so far is employed in a refinement step. It can be proved (see [4]) that the number of



(a) Wong-Toi



(b) HoneyTech

Fig. 1. Mistakes in previous fixpoint characterizations.

different entry regions is finite and that every possible entry region is discovered after a number of iterations bounded by the size of $[\![\overline{V}]\!]$. Therefore, after at most $|[\![\overline{V}]\!]|+1$ iterations of the procedure, the fixpoint is reached.

D. Previous Algorithms

In the literature, the standard reference for safety control of linear hybrid systems is [15]. The model and the abstract algorithm are essentially the same as ours. As to the computation of *CPre*, they introduce an operator *flow_avoid*, which corresponds to our RWA^{m} operator. They propose to compute $RWA^{\mathrm{m}}_l(U,V)$ using the following fixpoint formula (see the proof of Lemma 3.3 in [15]):

$$\bigcup_{U'\in[\![U]\!]}\ \bigcap_{V'\in[\![V]\!]} \Big(\mu W.U' \cup \bigcup_{P\in[\![\overline{V'}]\!]} \big(cl(P)\cap\overline{V'}\cap(W\cap P)\swarrow_l\big)\Big) \tag{3}$$

A simple example, however, shows that (3) is different from (in particular, larger than) $RWA^{\mathrm{m}}_l(U,V)$ when V is non-convex. Consider the example in Figure 1(a), where U is the gray area on top and V is the union of the two white boxes. Formula (3) treats the two convex parts of V separately. As a consequence, the result is the area covered by stripes. However, the correct results should not include the area within the thick border (in red-colored stripes), because any point in that region cannot prevent hitting one of the two convex parts of V.

Notice that it is virtually impossible to avoid non-convex V's. Even if all guards and invariants in the input automaton are convex, a non-convex V may arise as soon as more than one controllable transition is enabled from the same location.

In [7], Deshpande et al. report about an implementation of Wong-Toi's algorithm in the tool HONEYTECH, obtained as an extension of HyTech. The fixpoint formula that is meant to capture $RWA^{\mathrm{m}}_l(U,V)$ is the following:

$$\mu W.U \cup \bigcup_{P\in[\![\overline{V}]\!]} \Big(P\cap\big(cl(W)\cap cl(P)\cap\overline{V}\cap W\swarrow_l\big)\swarrow_l\Big) \tag{4}$$

Compared to (3), formula (4) correctly treats the case of non-convex V. However, it suffers from another issue, pertaining the distinction between topologically open and closed polyhedra. Consider the example in Figure 1(b), where U is the gray box, V is the white box, and dashed lines represent topologically open sides of polyhedra. The result of applying formula (4) is the area covered by stripes. This area includes the thick solid line that starts from a corner of V. Indeed, if W is the union of U and the striped region and $P \in [\![\overline{V}]\!]$, the thick line is exactly $cl(W)\cap cl(P)\cap\overline{V}\cap W\swarrow_l$. However, this line does not belong to $RWA^{\mathrm{m}}_l(U,V)$, because all its points cannot avoid hitting V before eventually reaching U.

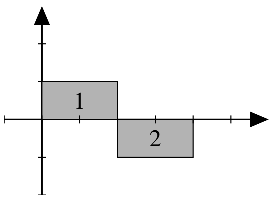
IV. EXPERIMENTS WITH PHAVER+

We implemented the procedure showed in the previous section on the top of the open-source tool PHAVer [8]. A binary pre-release of our implementation, that we call PHAVer+, can be downloaded at `http://people.na.infn.it/mfaella/phaverplus`. The experiments were performed on an Intel Xeon (2.80GHz) PC.

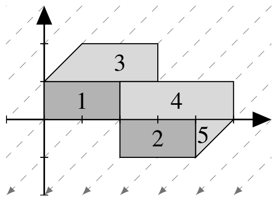
Truck Navigation Control (TNC): This example is derived from [7]. Consider an autonomous toy truck, which is responsible for avoiding some 2 by 1 rectangular pits. The truck can take 90-degree left or right turns: the possible directions are North-East (NE), North-West (NW), South-East (SE) and South-West (SW). One time unit must pass between two changes of direction. The control goal consists in avoiding the pits. Figure 3(a) shows the hybrid automaton that models the system: there is one location for each direction, where the derivative of the position variables (x and y) are set according to the corresponding direction. The variable t represents a clock ($\dot{t}=1$) that enforces a one-time-unit wait between turns.

Figure 2 shows the three iterations needed to compute the fixpoint in Theorem 1, in the case of two pits. The safe set is the white area, while the gray region contains the points wherefrom it is not possible to avoid the pits.

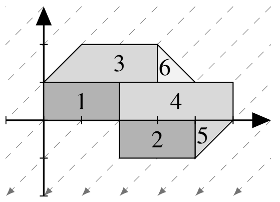
The input safe region T is the area outside the gray boxes 1 and 2 in Figure 2(a). The first iteration (Figure 2(b)) computes $CPre(T)$ and extends the unsafe set to those points (areas 3, 4, and 5) that will inevitably flow into the pits, before the system reaches $t=1$ and the truck can turn. The second iteration (Figure 2(c)) computes $CPre(CPre(T))$ and extends the unsafe set by adding the area 6: those points may turn before reaching the pits, but after the turn they end up in $CPre(T)$ anyway (for instance, if turning left, they end up in area 4 of Figure 2(d)). The third iteration reaches the fixpoint.



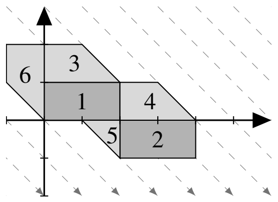
(a) The pits to avoid (i.e., $\overline{T}$).



(b) $CPre(T)$, SW direction.



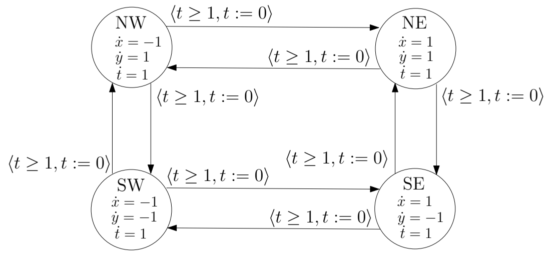
(c) $CPre(CPre(T))$, SW direction.



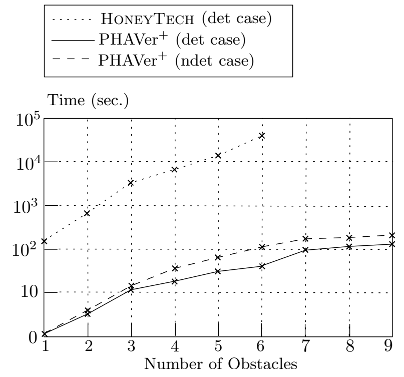
(d) $CPre(T)$, SE direction.

Fig. 2. Evolution of the fixpoint in the case of two pits. All figures are cross-sections for $t = 0$. Dashed arrows represent flow direction.

We tested our implementation on progressively larger versions of the truck model, by increasing the number of pits. We also considered a version of TNC with non-deterministic continuous flow, allowing some uncertainty on the exact direction taken by the vehicle. Using an exponential scale, Figure 3(b) compares the performance of our tool (solid line for deterministic model, dashed line for non-deterministic) to the performance reported in [7] (dotted line). We were not able to replicate the experiments in [7], since HONEYTECH is not publicly available.



(a) Hybrid Automaton for TNC.



(b) Computation time as a function of the number of pits.

Fig. 3. Hybrid Automaton and performance for TNC.

Because of the different hardware used, only a qualitative comparison can be made: going from 1 to 6 pits (as the case study in [7]), the run time of HONEYTECH shows an exponential behavior, while our tool exhibits an approximately linear growth, as shown in Figure 3(b), where the performance of PHAVer+ is plotted up to 9 pits.

V. CONCLUSIONS

We revisited the problem of automatically synthesizing a switching controller for an LHA w.r.t. safety objectives. The synthesis procedure is based on the RWA^{m} operator, for which we presented a novel fixpoint characterization and formally proved its termination.

To the best of our knowledge, this represents the first sound and complete procedure for the task in the literature. We extended the tool PHAVer with our synthesis procedure and performed a series of promising experiments. An account of the challenges involved in the implementation is presented in [5].

REFERENCES

[1] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proceedings of the IEEE*, 88(7):1011–1025, 2000.

[2] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification*, volume 2404 of *Lect. Notes in Comp. Sci.*, pages 746–770. Springer, 2002.

[3] A. Balluchi, L. Benvenuti, T. Villa, H. Wong-Toi, and A. Sangiovanni-Vincentelli. Controller synthesis for hybrid systems with a lower bound on event separation. *Int. J. of Control*, 76(12):1171–1200, 2003.

[4] M. Benerecetti, M. Faella, and S. Minopoli. Automatic synthesis of switching controllers for linear hybrid automata. Technical report, Università di Napoli "Federico II", 2011. Available on arXiv.

[5] M. Benerecetti, M. Faella, and S. Minopoli. Towards efficient exact synthesis for linear hybrid systems. In *GandALF 11: 2nd Int. Symp. on Games, Automata, Logics and Formal Verification*, volume 54 of *Elec. Proc. in Theor. Comp. Sci.*, 2011.

[6] L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *CONCUR 03: Concurrency Theory. 14th Int. Conf.*, volume 2761 of *Lect. Notes in Comp. Sci.*, pages 144–158. Springer, 2003.

[7] R.G. Deshpande, D.J. Musliner, J.E. Tierno, S.G. Pratt, and R.P. Goldman. Modifying HYTECH to automatically synthesize hybrid controllers. In *Proc. of 40th IEEE Conf. on Decision and Control*, pages 1223–1228. IEEE Computer Society Press, 2001.

[8] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past hyTech. In *Proc. of Hybrid Systems: Computation and Control (HSCC), 8th International Workshop*, volume 3414 of *Lect. Notes in Comp. Sci.*, pages 258–273. Springer, 2005.

[9] T.A. Henzinger. The theory of hybrid automata. In *11th IEEE Symp. Logic in Comp. Sci.*, pages 278–292, 1996.

[10] T.A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *CONCUR 99: Concurrency Theory. 10th Int. Conf.*, volume 1664 of *Lect. Notes in Comp. Sci.*, pages 320–335. Springer, 1999.

[11] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proc. of the 27th annual ACM symposium on Theory of computing*, STOC '95, pages 373–382. ACM, 1995.

[12] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *12th Annual Symp. on Theor. Asp. of Comp. Sci.*, volume 900 of *Lect. Notes in Comp. Sci.* Springer, 1995.

[13] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25:206–230, 1987.

[14] C.J. Tomlin, J. Lygeros, and S. Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proc. of the IEEE*, 88(7):949–970, 2000.

[15] H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *36th IEEE Conf. on Decision and Control*, pages 4607 – 4612, San Diego, CA, 1997. IEEE Computer Society Press.