Automatic Synthesis of Switching Controllers for Linear Hybrid Systems: Safety Control

Massimo Benerecetti, Marco Faella, Stefano Minopoli

Università di Napoli “Federico II”

Abstract

In this paper we study the problem of automatically generating switching controllers for the class of Linear Hybrid Automata, with respect to safety objectives. While the same problem has been already considered in the literature, no sound and complete solution has been provided so far. We identify and solve inaccuracies contained in previous characterizations of the problem, providing a sound and complete symbolic fixpoint procedure to compute the set of states from which a controller can keep the system in a given set of desired states. While the overall procedure may not terminate, we prove the termination of each iteration, thus paving the way to an effective implementation.

The techniques needed to effectively and efficiently implement the proposed solution procedure, based on polyhedral abstractions of the state space, are thoroughly illustrated and discussed. Finally, some supporting and promising experimental results, based on the implementation of the proposed techniques on top of the tool PHAVer, are presented.

Keywords: hybrid automata, controller synthesis, safety games

1. 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 [13] 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) [1], the allowed differential equations are in fact differential inclusions of the type $\dot{x} \in P$, where $\dot{x}$ 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 [16], indicating that no exact discrete abstraction exists. 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 [21]: 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 semi-algorithm for the problem\footnote{In other words, an algorithm that may or may not terminate, and that provides the correct answer whenever it terminates.} and we present a publicly available implementation, obtained by extending the open-source tool Phaver [9].

In particular, two operators on polyhedra need non-trivial new developments to be exactly and efficiently computed. Both operators pertain to intra-location behavior, and therefore assume that trajectories are subject to a fixed polyhedral differential inclusion of the type $\dot{x} \in P$.

- The pre-flow operator. Given a polyhedron $U \subseteq \mathbb{R}^n$, we wish to compute the set of all points that may reach $U$ via an admissible trajectory. This apparently easy task becomes non-trivial when the convex polyhedron $P$ is not (necessarily) topologically closed. This is the topic of Section 4.2.

- The may reach while avoiding operator, denoted by $\text{RWA}^m$. Given two polyhedra $U$ and $V$, the operator computes the set of points that may reach $U$ while avoiding $V$, via an admissible trajectory. We provide an algorithm for this operator and we extensively discuss the techniques required to efficiently implement it. The experiments performed show that the most efficient implementation is orders of magnitude faster than the naive one.

Contrary to most recent literature on the subject, 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 re-
search 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 per-
formance and the precision of an approximate tool.

Related work. The idea of automatically synthesizing controllers for dynamic
systems arose in connection with discrete systems [20]. Then, the same idea
was applied to real-time systems modeled by timed automata [19], thus coming
one step closer to the continuous systems that control theory usually deals with.
Finally, it was the turn of hybrid systems [23, 15], and in particular of Linear
Hybrid Automata, the very model that we analyze in this paper. Wong-Toi
proposed the first symbolic semi-algorithm to compute the controllable region
of a LHA w.r.t. a safety goal [23]. The heart of the algorithm 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. [21] and Balluchi et al. [5] analyze more expressive models, with generality in
mind rather than automatic synthesis. Their $Reach$ and $Unavoid\_Pre$ operators,
respectively, again correspond to $flow\_avoid$. Habets et al. [10] provide sufficient
conditions for controlling an affine hybrid system using continuous input.

As explained in Section 3.4, the algorithm provided in [23] for $flow\_avoid$
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 tool HONEYTech [8],
and we compare it with ours in Section 3.4.

Asarin et al. [2] investigate the synthesis problem for hybrid systems where
all discrete transitions are controllable and the trajectories satisfy given lin-
ear differential equations of the type $\dot{x} = Ax$. 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 de-
terministic trajectories, while differential inclusions are non-deterministic. In
control theory terms, differential inclusions can represent the presence of en-
vironmental disturbances. The tool $d/dt$ [3], 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 2 introduces and
motivates the model. In Section 3, we present the semi-algorithm which solves
the synthesis problem and in Section 4 we discuss the techniques required to
efficiently implement it. Section 5 reports some experiments performed on our
implementation of the semi-algorithm, while Section 6 draws some conclusions.

2. Linear Hybrid Automata

A convex polyhedron is a subset of $\mathbb{R}^n$ that is the intersection of a finite
number of strict and non-strict affine half-spaces. A polyhedron is a subset of
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 \( \text{cl}(G) \) its topological closure, and by \( [G] \subseteq 2^{\mathbb{R}^n} \) its representation as a finite set of convex polyhedra.

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

We use \( \dot{X} \) to denote the set \( \{\dot{x}_1, \ldots, \dot{x}_n\} \) of dotted variables, used to represent the first derivatives, and \( X' \) to denote the set \( \{x'_1, \ldots, x'_n\} \) of primed variables, used to represent the new values of variables after a discrete transition. Arithmetic operations on valuations are defined in the straightforward way. An \textit{activity} over \( X \) is a function \( f : \mathbb{R}^+ \rightarrow \text{Val}(X) \) that is continuous on its domain and differentiable except for a finite set of points.

\textit{Comparison with other models I.} Admitting a finite set of non-differentiable points gives trajectories the opportunity to take a finite number of sharp turns. This property is needed for our controllability algorithm to be exact. On the other hand, most papers on LHAs assume that trajectories are differentiable everywhere \([23, 1]\) or even smooth (i.e., in \( C^\infty \)) \([17]\). It can be shown that the algorithms presented in those papers also need our relaxed notion of trajectory in order to be exact in presence of non-convex invariants or uncontrollable transitions. We leave it as future work to solve the controllability problem for trajectories that are everywhere differentiable.

Let \( \text{Acts}(X) \) denote the set of activities over \( X \). The \textit{derivative} \( \dot{f} \) of an activity \( f \) is defined in the standard way and it is a partial function \( \dot{f} : \mathbb{R}^+ \rightarrow \text{Val}(\dot{X}) \). A \textit{Linear Hybrid Automaton (LHA)} \( H = (\text{Loc}, X, \text{Edg}_c, \text{Edg}_u, \text{Flow}, \text{Inv}, \text{Init}) \) consists of the following:

- A finite set \( \text{Loc} \) of \textit{locations}.
- A finite set \( X = \{x_1, \ldots, x_n\} \) of real-valued \textit{variables}. A \textit{state} is a pair \( \langle l, v \rangle \) of a location \( l \) and a valuation \( v \in \text{Val}(X) \).
- Two sets \( \text{Edg}_c \) and \( \text{Edg}_u \) of \textit{controllable} and \textit{uncontrollable transitions}, respectively. Each transition \( \langle l, \eta, l' \rangle \in \text{Edg}_c \cup \text{Edg}_u \) consists of a \textit{source location} \( l \), a \textit{target location} \( l' \), and a \textit{jump relation} \( \eta \in \text{Poly}(X \cup X') \), that specifies how the variables may change their value during the transition. The projection of \( \eta \) on \( X \) describes the valuations for which the transition is enabled; this is often referred to as a \textit{guard}.
- A mapping \( \text{Flow} : \text{Loc} \rightarrow \text{CPoly}(\dot{X}) \) 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 \( \text{Inv} : \text{Loc} \rightarrow \text{Poly}(X) \), called the \textit{invariant}.  

4
We use the abbreviations $S = \text{Loc} \times \text{Val}(X)$ for the set of states and $\text{Edg} = \text{Edg}_d \cup \text{Edg}_u$ for the set of all transitions. Moreover, we let $\text{InvS} = \bigcup_{l \in \text{Loc}} \{l\} \times \text{Inv}(l)$ and $\text{InitS} = \bigcup_{l \in \text{Loc}} \{l\} \times \text{Init}(l)$. Notice that $\text{InvS}$ and $\text{InitS}$ are sets of states. Given a set of states $A$ and a location $l$, we denote by $A|_l$ the projection of $A$ on $l$, i.e. $\{v \in \text{Val}(X) \mid (l, v) \in A\}$.

2.1. Semantics

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

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

**Runs.** Given two states $s, s'$, and a transition $e \in \text{Edg}$, there is a discrete step $s \xrightarrow{e} s'$ with source $s$ and target $s'$ iff (i) $s, s' \in \text{InvS}$, (ii) $e = (\text{loc}(s), \eta, \text{loc}(s'))$, and (iii) $(\text{val}(s), \text{val}(s')[X'/X]) \in \eta$, where $\text{val}(s')[X'/X]$ is the valuation in $\text{Val}(X')$ obtained from $s'$ by renaming each variable in $X$ with the corresponding primed variable in $X'$. Whenever this is the case, we say that $e$ is enabled in $s$.

There is a timed step $s \xrightarrow{\delta,f} s'$ with duration $\delta \in \mathbb{R}_{\geq 0}$ and activity $f \in \text{Adm}(s)$ iff (i) $s \in \text{InvS}$, (ii) $\delta \in \text{span}(f, \text{loc}(s))$, and (iii) $s' = (\text{loc}(s), f(\delta))$. For technical convenience, we admit timed steps of duration zero.

**Comparison with other models II.** Some authors prohibit such timed steps [2, 5]. We can disable them by adding a clock variable $t$ to the automaton and requesting that each discrete transition is enabled when $t > 0$ and resets $t$ to 0 when taken.

A special timed step is denoted $s \xrightarrow{\infty,f} s_n$ and represents the case when the system follows the activity $f$ forever. This is only allowed if $\infty \in \text{span}(f, \text{loc}(s))$.

A run is a sequence

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

of alternating timed and discrete steps, such that either the sequence is infinite, or it ends with a timed step of the type $s_n \xrightarrow{\infty,f} s_n$. If the run $r$ is finite, we define
len(r) = n to be the length of the run, otherwise we set len(r) = ∞. We denote by States(r) the set of all states visited by r. Formally, States(r) is the smallest set containing all states (loc(s_i), f_i(δ)), for all 0 ≤ i ≤ len(r) and all 0 ≤ δ ≤ δ_i. Notice that the states from which discrete steps start (states s'_i in (1)) appear in States(r). Moreover, if r contains a sequence of one or more zero-time timed steps, all intervening states appear in States(r).

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 steps in a finite amount of time, even if such behaviors are physically meaningless. Such runs are called Zeno runs. On the other hand, a run of the form (1) is called non-Zeno if, for all δ ≥ 0, there exists i ≥ 0 such that \( \sum_{j=0}^{i} \delta_j > \delta \). In other words, a run is non-Zeno if time diverges along the run.

In this paper, we assume that the hybrid automaton under consideration does not generate Zeno runs. The simplest non-Zeno systems are those that employ an extra variable, representing a clock, to ensure that the delay between any two discrete transitions is bounded from below by a constant. More precisely, these systems exhibit a variable t whose differential constraint is ˙t = 1 everywhere and all discrete transitions can only be taken when t ≥ 1 and reset t to zero. This is the case for the models used in our experiments in Section 5.

We leave it to future work to combine our results with more sophisticated approaches to Zenoness known in the literature [5, 7].

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 ∈ Adm(s) eventually leave the invariant, there exists one such activity f and a time δ ∈ span(f, loc(s)) such that there is an uncontrollable transition enabled in \( \langle \text{loc}(s), f(\delta) \rangle \). 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.

**Example 1.** Consider the LHAs in Figure 1. The fragment in Figure 1(a) is non-blocking, because the system may choose derivative ˙x = 0 and remain indefinitely in location l. The fragment in Figure 1(b) is also non-blocking, because the system cannot remain in l forever, but an uncontrollable transition leading outside is always enabled. Finally, the fragment in Figure 1(c) is blocking, because the system cannot remain in l forever, and no uncontrollable transition is enabled.

**Comparison with other models III.** In the work of Wong-Toi [23], the property called “control δ-positivity” plays the role of our non-blocking condition. Such property states that there is a (global) δ > 0 such that if a controllable transition is enabled in state s, it is also possible to let δ time unit pass from s, without leaving the invariant. Essentially, this property constrains the guards of the controllable transitions to be detached from the boundary of the invariant by
the equivalent of at least $\delta$ time units. The objective is to enable the controller to choose the null action even when the system is on the invariant boundary. Our approach achieves the same effect by restricting the guards of the uncontrollable transitions. This is in line with the standard interpretation of the invariant: since it is an internal system constraint (and not the control goal), system transitions alone should be able to enforce it.

**Strategies.** A strategy is a function $\sigma : S \rightarrow 2^{Edg_c \cup \{\bot\} \setminus \emptyset}$, where $\bot$ denotes the null action. Notice that 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 $e \in \sigma(s) \cap Edg_c$, 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 (persistence property). Formally, if $\sigma(s) = \{\bot\}$, for all $f \in \text{Adm}(s)$ there exists $\delta > 0$ such that for all $0 < \delta' < \delta$ if $\delta' \in \text{span}(f, \text{loc}(s))$ then $\bot \in \sigma(\langle \text{loc}(s), f(\delta') \rangle)$. If all activities starting from $s$ immediately leave the invariant, the above condition is vacuously satisfied.

**Comparison with other models IV.** The strategies considered by Wong-Toi [23] are deterministic, i.e., of the type $\sigma : S \rightarrow Edg_c \cup \{\bot\}$, and subject to the following condition: the set of states from which any given discrete transition is chosen is a closed polyhedron. As a consequence, the set of states from which the null action $\bot$ is chosen is an open polyhedron and therefore those strategies satisfy our persistence property. However, our condition on strategies is less restrictive, putting no restriction on the shape of the set of states where a given discrete transition is chosen.

Notice that a strategy can always choose the null action. The well-formedness condition ensures that the system can always evolve in some way, be it a timed step or an uncontrollable transition. In particular, even if we are on the boundary of the invariant we allow the controller to choose the null action, 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 $\sigma$ if for all $0 \leq i < \text{len}(r)$ the following conditions hold:

![Figure 1: Three LHA fragments. Locations contain the invariant (first line) and the flow constraint (second line). Solid (resp., dashed) edges represent controllable (resp., uncontrollable) transitions. Guards are true.](image-url)
• for all $\delta \geq 0$ such that $\sum_{j=0}^{i-1} \delta_j \leq \delta < \sum_{j=0}^{i} \delta_j$, we have $\bot \in \sigma(s)$, where $s = (\text{loc}(s_i), f_i(\delta - \sum_{j=0}^{i-1} \delta_j))$;

• if $e_i \in \text{Edg}_q$, then $e_i \in \sigma(s'_i)$.

We denote by $\text{Runs}(s, \sigma)$ the set of runs starting from the state $s$ and consistent with the strategy $\sigma$. The following result ensures that each strategy has at least one run that is consistent with it, otherwise the controller may surreptitiously satisfy the safety objective by blocking the system. The result can be proved by induction by considering the following. As long as the strategy chooses the null action, the system may continue along one of the activities that remain within the invariant; if a state is reached from which all activities immediately leave the invariant, the non-blocking assumption ensures that there exists an uncontrollable transition that is enabled; finally, if the strategy chooses a discrete transition, that transition is enabled.

**Theorem 1.** Given a non-blocking hybrid automaton, for all strategies $\sigma$ and states $s \in \text{InvS}$, there exists a run that starts from $s$ and is consistent with $\sigma$.

**Proof.** Let

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

be a finite prefix of a run that starts in $s_0 = s$ and is consistent with $\sigma$ (as base case, consider $r = s$). We show that $r$ can be extended with either an infinite timed step, or a timed step followed by a discrete step. Let $l = \text{loc}(s_n)$.

If there exists an activity $f \in \text{Adm}(s_n)$ that never leaves the invariant (i.e., such that $\infty \in \text{span}(f,l)$), then $r$ can be completed by the infinite timed transition $s_n \xrightarrow{\infty}$. Otherwise, since the automaton is non-blocking, there exists an activity $f \in \text{Adm}(s_n)$, a time $\delta \in \text{span}(f,l)$, and an uncontrollable transition $e_a \in \text{Edg}_q$ such that $e_a$ is enabled in the state $\langle l, f(\delta) \rangle$. If, for all $0 < \delta' < \delta$, we have $\bot \in \sigma(\langle l, f(\delta') \rangle)$, then $r$ can be extended with the steps $s_n \xrightarrow{\delta} s' \xrightarrow{e_a} s''$.

Otherwise, let $\delta = \inf\{\delta' \geq 0 | \delta' < \delta \text{ and } \bot \notin \sigma(\langle l, f(\delta') \rangle)\}$. Notice that $\delta \in \text{span}(f,l)$. If there exists $e_c \in \sigma(\langle l, f(\delta) \rangle) \cap \text{Edg}_q$, it is easy to verify that $r$ can be extended with the steps $s_n \xrightarrow{\delta} s' \xrightarrow{e_a} s''$. Otherwise, it holds $\sigma(\langle l, f(\delta) \rangle) = \{\bot\}$ and we obtain the following contradiction: by the persistence property of strategies, there exists $\delta^* > \delta$ such that for all $\delta'$ with $\delta < \delta' < \delta^*$ it holds $\bot \in \sigma(\langle l, f(\delta') \rangle)$, against the definition of $\delta$.

**Safety control problem.** Given a hybrid automaton $H$ and a set of states $T \subseteq \text{InvS}$, the safety control problem asks whether there exists a strategy $\sigma$ such that, for all initial states $s \in \text{InitS}$, all runs $r \in \text{Runs}(s, \sigma)$ it holds $\text{States}(r) \subseteq T$.

We call the above $\sigma$ a winning strategy.

### 3. Safety Control

In this section, we consider a fixed hybrid automaton and we present a sound and complete procedure to solve the safety control problem.
3.1. The Abstract Algorithm

We start by defining some preliminary operators. For a set of states $A$ and $x \in \{u, c\}$, we define $\text{Pre}_x(A)$ (for predecessors) as:

$$\text{Pre}_x(A) = \{ s \in S \mid s \xrightarrow{e} s', \text{ with } s' \in A \text{ and } e \in \text{Edg}_x \},$$

denoting the set of states where some discrete transition belonging to $\text{Edg}_x$ is enabled and leads to $A$. Let $\overline{A}$ be the complement of $A$.

Example 2. Assume that $\text{Edg}_c = \{(l, \eta, l')\}$, where $\eta$ is the polyhedron $(x \leq 0 \land x' \geq 0) \lor (0 < x \leq 2 \land x' = x + 1)$. In Figure 2, $\eta$ is the union of the gray box and the gray segment. In words, if $x \leq 0$ the new value of $x$ will be a non-deterministically chosen non-negative number. If instead $x \in (0, 2]$, the value of $x$ will be incremented by 1. The transition is not enabled for $x > 2$. Let $A$ be the set of states $\langle l', 0 \leq x \leq 2 \rangle$, we have $\text{Pre}_c(A) = \langle l, x \leq 1 \rangle$, because all states of the type $\langle l, x \leq 0 \rangle$ may end up in $A$ after the transition and all states of the type $\langle l, 0 < x \leq 1 \rangle$ will definitely end up in $A$ after the transition. All other states will not end up in $A$. Graphically, rather than $A$ it is convenient to draw $A' = A[X'/X]$, i.e., the polyhedron obtained from $A$ by renaming each variable from $X$ as the corresponding primed variable. In this way, $\text{Pre}_c(A)$ is the set of values of $x$ for which $A'$ and $\eta$ intersect and it corresponds to the highlighted part of the $x$-axis.

Controllable predecessor operator. For a set of states $A$, the operator $\text{CPre}(A)$ returns the set of states from which the controller can ensure that the system remains in $A$ during the next pair of timed and discrete steps. This happens if for all activities chosen by the environment and all delays $\delta$, one of two situations occurs:

- either the systems stays in $A$ up to time $\delta$, while all uncontrollable transitions enabled up to time $\delta$ (included) also lead to $A$, or
there exists a time $\delta' < \delta$, such that the system stays in $A$ up to time $\delta'$, all uncontrollable transitions enabled up to time $\delta'$ (included) also lead to $A$, and the controller can issue a transition at time $\delta'$ 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 $\text{While}(A,f,\delta)$ the set of states from where following the activity $f$ for $\delta$ time units keeps the system in $A$ all the time, and any uncontrollable transition taken meanwhile also leads into $A$. Formally,

$$\text{While}(A,f,\delta) = \{ s \in S \mid \forall 0 \leq \delta' \leq \delta : (\text{loc}(s), f(\delta')) \in A \setminus \text{Pre}_u(A) \}.$$

We can now formally define the $C\text{Pre}$ operator.

$$C\text{Pre}(A) = \{ s \in S \mid \forall f \in \text{Adm}(s), \delta \in \text{span}(f, \text{loc}(s)) : s \in \text{While}(A,f,\delta) \}
\quad \text{or } \exists 0 \leq \delta' < \delta : s \in \text{While}(A,f,\delta') \text{ and } (\text{loc}(s), f(\delta')) \in \text{Pre}_c(A) \}.$$

The following theorem shows that the controllable predecessor operator can be used to characterize the safety control problem. This result is classical for all game-theoretic approaches to safety control [19, 2, 18]. Being classical, the details of its proof are often taken for granted in the literature, although they strongly depend on the precise definitions of the game model, its semantics and the notion of strategy. Hence, we provide a detailed proof here.

**Theorem 2.** The answer to the safety control problem for safe set $T \subseteq \text{InvS}$ is positive if and only if

$$\text{InitS} \subseteq \nu W \cdot T \cap C\text{Pre}(W),$$

where $\nu$ denotes the greatest fixed point operator.

**Proof.** [if] We shall first build a winning strategy in two steps. Let $W^\ast = \nu W \cdot T \cap C\text{Pre}(W)$ and let $\sigma$ be a strategy defined as follows, for all states $s$:

- $\bot \in \sigma(s)$ and
- if $s \xrightarrow{e} s'$, $s, s' \in W^\ast$ and $e \in \text{Edg}_c$, then $e \in \sigma(s)$.

While $\sigma$ is clearly a strategy, it is not necessarily a winning strategy, as it may admit runs which delay controllable actions either beyond the safety set $W^\ast$ or beyond their availability. We can, however, recover a winning strategy by restricting $\sigma$ in appropriate ways. For all states $s \in S$ and activities $f \in \text{Adm}(s)$, let

$$D_{f,s} = \{ \delta > 0 \mid \forall 0 \leq \delta' \leq \delta : (\text{loc}(s), f(\delta')) \in W^\ast \text{ and } \sigma((\text{loc}(s), f(\delta'))) \cap \text{Edg}_c \neq \emptyset \}.$$
denote the set of positive time units for which the system can follow activity
f, starting from s, always remaining in W* with some controllable transition
enabled and available to the controller.

Starting from σ, we can define a new strategy σ' which coincides with σ
on all the states, except for the states s ∈ W* with Edg_e ∩ σ(s) ≠ ∅, where it
satisfies σ'(s) ⊆ σ(s) and the following condition:

a) If there is f ∈ Adm(s) such that Df,s = ∅, then ⊥ ∉ σ'(s);

b) otherwise, for all f ∈ Adm(s), there exists a δ ∈ Df,s with ⊥ ∉
σ'((loc(s), f(δ))) and ⊥ ∈ σ'((loc(s), f(δ′))), for all 0 ≤ δ′ < δ.

Intuitively, the new strategy σ' ensures that following any activity from a state
s ∈ W* in which some controllable action is enabled, a controllable action will
always be taken before none of them is available and before leaving W*.

To prove that σ' is winning, we must show that for every s ∈ InitS and every
r ∈ Runs(σ', s), States(r) ⊆ T. Let

\[ r = s_0  \xrightarrow{δs_0 f_0} s'_0  \xrightarrow{c_0} s_1  \xrightarrow{δ_1 f_1} s'_1  \xrightarrow{c_1} s_2 \ldots s_n \ldots \]

be a run consistent with σ'. The following properties can be proved:

1. if \( s_i  \xrightarrow{δ_i f_i} s'_i \) occurs in r, with δ_i > 0 and s_i ∈ W*, then for all 0 ≤ δ' ≤ δ_i,
   it holds \( \langle loc(s_i), f_i(δ') \rangle \in W^* \);

2. if \( s_i  \xrightarrow{∞ f_i} \) occurs in r and s_i ∈ W*, then for all δ' ≥ 0, it holds \( \langle loc(s_i), f_i(δ') \rangle \in
   W^* \);

3. if \( s_i  \xrightarrow{c_i} \) occurs in r and s_i ∈ W*, then s'_i ∈ W*.

We shall prove property (1), as (2) can be proved similarly. Since δ_i > 0, by
the consistency of r with σ', we have ⊥ ∈ σ'(s_i). Assume, by contradiction, that
\( \langle loc(s_i), f_i(δ) \rangle \notin W^* \) for some 0 < δ' < δ_i. Since s_i ∈ W* = CPre(W*), then
s_i ∈ While(W*, f_i, δ) for some δ ∈ \( \mathbb{R}^≥ 0 \cup \{∞ \} \), and either δ = ∞ or s_i  \xrightarrow{δ f_i} s
and s ∈ CPre(W*).

If δ ≥ δ', we have an immediate contradiction, since it would imply s_i ∈
While(W*, f_i, δ') and, therefore, \( \langle loc(s_i), f_i(δ') \rangle \in W^* \).

Assume, then, δ < δ'. Then \( \langle loc(s_i), f_i(δ) \rangle \in Pre_e(W^*), \) i.e., \( \langle loc(s_i), f_i(δ) \rangle \xrightarrow{e}
s' \) for some e ∈ Edg_e, and s' ∈ W*. Therefore, both e ∈ σ'((loc(s_i), f_i(δ)))
and, by the consistency of r with σ', ⊥ ∈ σ'((loc(s_i), f_i(δ))). Since ⊥ ∈
σ'((loc(s_i), f_i(δ))), by definition of σ' the premise of property a) cannot hold.
Therefore, by property b), there must be a \( δ < δ' < δ' \) with ⊥ ∉ σ'((loc(s_i), f_i(δ'))).

On the other hand, the consistency of r requires that ⊥ ∈ σ'((loc(s_i), f_i(δ)))
for all 0 ≤ δ < δ_i, which is a contradiction. Therefore, for all 0 ≤ δ' < δ_i,
\( \langle loc(s_i), f_i(δ') \rangle \in W^* \).

Finally, to prove that s'_i ∈ W* we can proceed again by contradiction, assuming
s'_i ∉ W*. Let 0 < δ' < δ_i, then \( \langle loc(s_i), f_i(δ') \rangle \in W^* \). Therefore,
\( \langle loc(s_i), f_i(δ') \rangle \in CPre(W^*) \) and there exists δ′ ≤ δ* < δ_i with \( \langle loc(s_i), f_i(δ') \rangle \in

While($W^*, f_i, \delta^*$) and $\langle \text{loc}(s_i), f_i(\delta^*) \rangle \in \text{Pre}_c(W^*)$. Hence, there is a controllable transition $e \in \text{Edg}_c$ enabled in $\langle \text{loc}(s_i), f_i(\delta^*) \rangle$ and leading to $W^*$. As a consequence, $\{e, 1\} \subseteq \sigma(\langle \text{loc}(s_i), f_i(\delta^*) \rangle)$ and, by condition b), $\perp \notin \sigma'(\langle \text{loc}(s_i), f_i(\delta) \rangle)$, for some $\delta^* < \delta_i$, which contradicts consistency of $r$ with $\sigma'$, hence $s'_i \in W^*$.

Let us consider property (3). We have two cases. If $e \in \text{Edg}_c$, then the consistency of $r$ ensures that $e \in \sigma'(s_i)$ which, by definition of $\sigma'$, requires that $s_{i+1} \in W^*$. Assume then that $e \in \text{Edg}_c$. Then $\perp \in \sigma'(s_i)$. Since $s_i \in W^* = \text{CPre}(W^*)$, it must hold $s_i \in \text{While}(W^*, f, 0)$, for every $f \in \text{Adm}(s_i)$. This, in turn, ensures that $s_i \in W^* \setminus \text{Pre}_u(W^*)$, therefore, all the uncontrollable transitions enabled in $s_i$ lead to $W^*$. Hence the thesis.

To complete the proof, notice that $W^* \subseteq T$ and $s_0 \in \text{InitS} \subseteq W^*$. An easy induction on the length of $r$, using properties (1), (2) and (3), gives the result.

[only if] Let $s \notin W^*$, we prove that for all strategies there is a run that starts in $s$, is consistent with the strategy and leaves $T$. Let

- $W_0 = T$,
- $W_\alpha = T \cap \text{CPre}(W_{\alpha-1})$, for a successor ordinal $\alpha$, and
- $W_\alpha = \bigcap_{\beta < \alpha} W_\beta$ for a limit ordinal $\alpha$.

We proceed by induction on the smallest ordinal $\lambda$ such that $s \notin W_\lambda$. If $\lambda = 0$, it holds $s \notin T$ and the thesis is immediate.

We will show that if $\lambda > 0$ then $\lambda$ cannot be a limit ordinal. Assume by contradiction that $\lambda$ is a limit ordinal. Since $\lambda$ is the smallest ordinal such that $s \notin W_\lambda$, we have $s \in W_\alpha$, for all $\alpha < \lambda$: this means that $s \in \bigcap_{\alpha < \lambda} W_\alpha$. But, since $\lambda$ is a limit ordinal, $W_\lambda = \bigcap_{\alpha < \lambda} W_\alpha$ and we have that $s \in W_\lambda$, obtaining a contradiction.

Otherwise, if $\lambda > 0$ is a successor ordinal, we have $s \in W_{\lambda-1} \setminus W_\lambda$ and $s \notin \text{CPre}(W_{\lambda-1})$. According to the definition of $\text{CPre}$, there exists an activity $f \in \text{Adm}(s)$ and $\delta \in \text{span}(s, f)$ such that $s \notin \text{While}(W_{\lambda-1}, f, \delta)$ and for all $0 \leq \delta' < \delta$ either $s \notin \text{While}(W_{\lambda-1}, f, \delta')$ or $\langle \text{loc}(s), f(\delta') \rangle \notin \text{Pre}_c(W_{\lambda-1})$.

Let $\delta^*$ be the infimum of those $\delta'$ such that $s \notin \text{While}(W_{\lambda-1}, f, \delta')$, i.e.,

$$\delta^* = \inf \{ \delta \mid s \notin \text{While}(W_{\lambda-1}, f, \delta) \}.$$  (2)

Clearly $0 \leq \delta^* \leq \delta$ and, for all $0 \leq \delta < \delta^*$, $\langle \text{loc}(s), f(\delta) \rangle \notin \text{Pre}_c(W_{\lambda-1})$. Hence, any controllable transition enabled in $\langle \text{loc}(s), f(\delta) \rangle$, for any such $\delta$, leads outside $W_{\lambda-1}$. Therefore, any strategy choosing a controllable transition in some of the states $\langle \text{loc}(s), f(\delta) \rangle$ has a consistent run leading outside $W_{\lambda-1}$. By inductive hypothesis, we obtain the thesis.

If, on the other hand, the strategy allows the controller to stay inactive in all those states, there is a consistent run that reaches $\delta^*$. Then we have two cases. If $\delta^*$ is in fact the minimum of the above set, according to the definition of $\text{While}$, there exists $\delta_1 < \delta^*$ such that $\langle \text{loc}(s), f(\delta_1) \rangle \in W_{\lambda-1} \cup \text{Pre}_u(W_{\lambda-1})$. Therefore, since the controller may not act before $\delta^*$ along this strategy, there is
Given a location \( \eta \) and two sets of variable valuations \( V \) and \( W \), for all \( 0 \leq \delta \leq \delta^* \), consider the choice of \( \sigma \) in state \( \langle \eta, s \rangle \). If \( \bot \notin \sigma((\langle \eta, s \rangle, f(\delta^*))) \), the controller issues a discrete move which leads into \( W_{\lambda-1} \). If, instead, \( \bot \in \sigma((\langle \eta, s \rangle, f(\delta^*))) \), \( \delta^* < \delta \in \text{span}(s, f) \), by the definition of strategy \( \sigma \) will keep choosing \( \bot \) for a non-zero amount of time \( \gamma \). By (2), there exists \( \delta^* < \tilde{\delta} < \delta^* + \gamma \) such that \( s \notin \text{While}(W_{\lambda-1}, f, \tilde{\delta}) \). As a consequence, there is a consistent run that reaches a state which either is in \( W_{\lambda-1} \) or reaches it after an uncontrollable transition. Once again, the thesis follows from the inductive hypothesis.

\[ \square \]

3.2. Computing the Predecessor Operator on LHA

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

\[ G_{\prec} = \{ u \in \text{Val}(X) \mid \exists \delta \geq 0, c \in \text{Flow}(l) : u + \delta \cdot c \in G \} \]

containing the set of valuation from which a linear activity can reach \( G \). Notice that, for two convex polyhedra \( P \) and \( P' \), if \( P \subseteq P' \) then \( P_{\prec} \subseteq P'_{\prec} \) (monotonicity), and \( (P_{\prec})_{\prec} = P_{\prec} \) (idempotence). Moreover, it immediately follows from the definition that \((P_1 \cup P_2)_{\prec} = P_1_{\prec} \cup P_2_{\prec}\) (distribution over union).

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

\[ \text{Pre}_x(A) = \text{InvS} \cap \bigcup_{(l, \eta, u) \in \text{Edg}_x} \eta^{-1}(A \mid_l), \]

where \( \eta^{-1}(Z) = \{ v \in \text{Val}(X) \mid \exists v' \in Z : \langle v, v'[X'/X] \rangle \in \eta \} \) denotes the pre-image of \( Z \) w.r.t. \( \eta \).

We can now introduce the operator \( \text{RWA}^m \) (may reach while avoiding). Given a location \( l \) and two sets of variable valuations \( U \) and \( V \), \( \text{RWA}^m(U, V) \) contains the set of valuations from which there exists an activity that reaches \( U \) while avoiding \( V \). Notice that on a dense time domain this is not equivalent to reaching \( U \) while avoiding \( V \); if an activity avoids \( U \) in a right-closed

\[ ^2 \text{In Computation Tree Logic (CTL), we have } \text{RWA}^m(U, V) \equiv \exists (\overline{U} \cup U) \cup U. \]
time interval, and then enters $U \cap V$, the first property holds, while the latter does not. Formally, we have:

$$RWA^m_l(U, V) = \left\{ u \in \text{Val}(X) \mid \exists f \in \text{Adm}(l, u), \delta \geq 0 : f(\delta) \in U \text{ and } \forall 0 \leq \delta' < \delta : f(\delta') \in \overline{V} \cup U \right\}.$$  

Notice that $RWA^m_l(U, V) = RWA^m_l(U, V \setminus U)$ (i.e., the activities are allowed to reach $V$, provided they simultaneously reach $U$). Therefore, in the following we can assume w.l.o.g. that $U \cap V = \emptyset$. An algorithm for effectively computing $RWA^m$ on polyhedral arguments 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 $(l, u)$ the environment can take a discrete transition leading outside $A$, and the set $C_l$ of valuations $u$ such that from $(l, u)$ 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 $CPre$, 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 $l \in \text{Loc}$, the projection $A \mid_l$ is a polyhedron. First, we recall the following result, stating that if an activity goes from a point of a convex polyhedron $P$ to another point of $P$, there is also a linear activity that does the same.

**Lemma 1.** [1] For all states $s = (l, u)$ and convex polyhedra $P$ such that $u \in P$, if there is an activity $f \in \text{Adm}(s)$ and a time $\delta \in \text{span}(f, l)$ such that $f(\delta) = u' \in P$, then there is a linear activity $f' \in \text{Adm}(s)$ such that $f'(\delta) = u'$ and $f'(\delta') \in P$ for all $\delta' \in [0, \delta]$.

**Lemma 2.** For all polyhedral sets of states $A \subseteq \text{InvS}$, we have

$$CPre(A) = \bigcup_{l \in \text{Loc}} \{l\} \times \left( A \mid_l \setminus RWA^m_l(\text{InvS} \mid_l \cap (\overline{A} \mid_l \cup B_l), C_l \cup \text{InvS} \mid_l) \right), \quad (3)$$

where $B_l = \text{Pre}_u(\overline{A}) \mid_l$ and $C_l = \text{Pre}_c(A) \mid_l$.

**Proof.** In the following, let $I_l = \text{InvS} \mid_l$.

[\subseteq] Let $s = (l, u) \in CPre(A)$ and let $f \in \text{Adm}(s)$. By definition, $0 \in \text{span}(f, l)$ and hence $s \in \text{While}(A, f, 0)$. In particular, this implies that $s \in A$ and $u \in A \mid_l$.

Assume by contradiction that $s$ does not belong to the r.h.s. of (3). Since $u \in A \mid_l$, it must be

$$u \in RWA^m_l(I_l \cap (\overline{A} \mid_l \cup B_l), C_l \cup I_l).$$

Then, by definition there exists $f^* \in \text{Adm}(s)$ and $\delta^* \geq 0$ such that: (i) $f^*(\delta^*) \in I_l \cap (\overline{A} \mid_l \cup B_l)$, and (ii) for all $0 \leq \delta < \delta^*$ it holds $f^*(\delta) \in I_l \cap (\overline{C} \mid_l \cup \overline{A} \mid_l \cup B_l)$.  

14
In particular, this implies that $\delta^*$ belongs to $\text{span}(f^*, l)$. On the other hand, if we apply the definition of $\text{CPre}(A)$ to the activity $f^*$, we obtain that for all $\delta \in \text{span}(f^*, l)$ either $s \in \text{While}(A, f^*, \delta)$ or there exists $\delta' < \delta$ such that $s \in \text{While}(A, f^*, \delta')$ and $(l, f^*(\delta')) \in \text{Pre}_c(A)$. This implies that either $f^*(\delta^*) \in A \setminus \overline{B_l}$ or there exists $\delta' < \delta^*$ such that $f^*(\delta') \in A \setminus \overline{B_l} \cap C_l$, which is a contradiction.

We prove the counter-positive, i.e., let $s = (l, u) \notin \text{CPre}(A)$, we prove that either $s \notin A$ or $u \in \text{RWA}_m(I_l \cap (\overline{A_l} \cup B_l), C_l \cup \overline{T_l})$. By definition of $\text{CPre}$, there exists an activity $f \in \text{Adm}(s)$ and a time $\delta \in \text{span}(f, l)$ such that $s \notin \text{While}(A, f, \delta)$ and for all $0 \leq \delta' < \delta$ it holds $s \notin \text{While}(A, f, \delta')$ or $f(\delta') \notin C_l$. Let $\delta^* = \inf\{\delta \in \text{span}(f, l) \mid s \notin \text{While}(A, f, \delta)\}$. If $s \notin \text{While}(A, f, \delta^*)$ (i.e., $\delta^*$ is the minimum of the above set), it holds $f(\delta^*) \in \overline{A_l} \cup B_l$; moreover, since $\delta^* \in \text{span}(f, l)$, it also holds $f(\delta^*) \in I_l$. For all $0 \leq \delta' < \delta^*$, it holds $s \in \text{While}(A, f, \delta')$ and also $f(\delta') \notin C_l$. Hence, $f(\delta') \in I_l \setminus C_l$. We conclude that $f$ is a witness for $u \in \text{RWA}_m(I_l \cap (\overline{A_l} \cup B_l), C_l \cup \overline{T_l})$.

Finally, assume that $s \in \text{While}(A, f, \delta^*)$ and consequently $f(\delta^*) \in A \setminus \overline{B_l}$. For all $\delta' > \delta^*$, it holds $s \notin \text{While}(A, f, \delta')$. Hence, by definition of $\text{While}$ there exists $\delta \in (\delta^*, \delta']$ such that $f(\delta) \in \overline{A_l} \cup B_l$. In other words, in every interval $(\delta^*, \delta']$ there exists a point for which $f$ exits from the safe region or hits the enabling region of a "bad" uncontrollable transition. Since $\overline{A_l} \cup B_l$ is a polyhedron, there is a convex polyhedron $P \in [\overline{A_l} \cup B_l]$ such that in every interval $(\delta^*, \delta']$ there is a point $\delta$ such that $f(\delta) \in P$. Since $f$ is continuous, we obtain that $f(\delta^*) \in \text{cl}(P)$. Let $\delta > \delta^*$ be any point such that $f(\delta) \in P$ and let $f'$ be the activity that coincides with $f$ until time $\delta^*$, and for all $\delta > \delta^*$

$$f'(\delta) = f(\delta^*) + \frac{\delta - \delta^*}{\delta - \delta^*} \cdot (f(\delta) - f(\delta^*)),$$

i.e., from $\delta^*$ to $\delta'$ the activity $f'$ follows a straight line at constant speed from $f(\delta^*)$ to $f(\delta)$. Lemma 1 ensures that $f' \in \text{Adm}(s)$. Notice that $f'$ is continuous, but it may not be differentiable in $\delta^*$. Since $f'(\delta^*) = f(\delta^*) \in \text{cl}(P)$ and $P$ is convex, it holds $f'(\delta) \in P$ for all $\delta \in (\delta^*, \delta]$, and therefore $f'$ is a witness for $u \in \text{RWA}_m(I_l \cap (\overline{A_l} \cup B_l), C_l \cup \overline{T_l})$. $\square$

### 3.3. Computing the $\text{RWA}_m$ Operator on Polyhedra

In this section, we consider a fixed location $l$. Lemma 2 implies that one can compute $\text{CPre}(A)$ by computing $\text{RWA}_m(U, V)$ for general polyhedra $U$ and $V$. We start this section by checking whether $\text{RWA}_m$ for non-convex arguments can be expressed in terms of $\text{RWA}_m$ for convex arguments. It is easy to verify that the first argument of $\text{RWA}_m$ distributes over union, i.e., for all polyhedra $U_1, U_2$ and $V$ it holds

$$\text{RWA}_m(U_1 \cup U_2, V) = \text{RWA}_m(U_1, V) \cup \text{RWA}_m(U_2, V).$$

Hence, in the following we can assume that the first argument of $\text{RWA}_m$ is a convex polyhedron.
Avoiding the non-convex region \( V_1 \cup V_2 \) cannot be reduced to separately avoiding \( V_1 \) and \( V_2 \).

Linear activities are not sufficient to avoid \( V_1 \cup V_2 \).

Regarding the second argument (the set to be avoided), a run avoids \( V_1 \cup V_2 \) if and only if it avoids both \( V_1 \) and \( V_2 \). On the other hand, if there exists a run that avoids \( V_1 \) (i.e., \( s \in RWA^m_U(\cdot, V_1) \)) and a (possibly different) run avoiding \( V_2 \) (i.e., \( s \in RWA^m_U(\cdot, V_2) \)), it does not mean that there exists a run that avoids both (i.e., \( s \in RWA^m_U(\cdot, V_1 \cup V_2) \)). For instance, in the case pictured in Figure 3(a), the runs starting from the dotted area can avoid either \( V_1 \) or \( V_2 \) and reach \( U \), but they cannot avoid both. Hence, the dotted area does not belong to \( RWA^m_U(V_1 \cup V_2) \), while it belongs to \( RWA^m_U(V_1) \cap RWA^m_U(V_2) \).

Additionally, it is not possible to restrict the analysis from arbitrary activities (i.e., any differentiable function which stays in the invariant and whose slope belongs to \( \text{Flow}(l) \)) to linear activities. In Figure 3(b), the dotted area contains the set of points that cannot avoid \( V_1 \cup V_2 \) following linear activities. On the other hand, \( RWA^m_U(V_1 \cup V_2) = V_1 \cap V_2 \), because all other points (including those in the dotted area) can avoid \( V_1 \cup V_2 \) by passing through the gap between \( V_1 \) and \( V_2 \). In fact, those points can avoid \( V_1 \cup V_2 \) via a sequence of two linear activities. The forthcoming Lemma 5 shows that this observation can be generalized: if the system can move from point \( u \) to point \( v \) while avoiding a given polyhedral region, it can also do so via a finite sequence of linear activities.

We can now proceed to presenting the algorithm for computing \( RWA^m \). Given two polyhedra \( G \) and \( G' \), recalling that the topological closure of \( G \) (resp., \( G' \)) is denoted by \( \text{cl}(G) \) (resp., \( \text{cl}(G') \)), we define their \textit{boundary} to be

\[
\text{bndry}(G, G') = (\text{cl}(G) \cap G') \cup (G \cap \text{cl}(G')).
\]

Clearly, \( \text{bndry}(G, G') \) is not empty only if \( G \) and \( G' \) are adjacent to one another or if they overlap; it is empty, otherwise. Notice also that, for any polyhedra \( G, G_1, \) and \( G_2 \), \( \text{bndry}(G, G_1 \cup G_2) = \text{bndry}(G, G_1) \cup \text{bndry}(G, G_2) \). The following property also holds.

\textbf{Lemma 3.} For all convex polyhedra \( P \) and \( P' \), \( \text{bndry}(P, P') \) is a convex polyhedron.
PROOF. Let \( x \) and \( y \) be two points in \( \text{bndry}(P, P') \), and \( z_a \) be a convex combination of \( x \) and \( y \), i.e., \( z_a = ax + (1-a)y \), with \( 0 < a < 1 \). We prove that \( z_a \) belongs to \( \text{bndry}(P, P') \).

Let \( L = (P \cap \text{cl}(P')) \) and \( R = (\text{cl}(P) \cap P') \), so that \( \text{bndry}(P, P') = L \cup R \). Notice that \( L \) and \( R \) are convex polyhedra. Moreover, each point in \( P \cap P' \) belongs to both \( L \) and \( R \), and therefore to \( \text{bndry}(P, P') \). Now, we distinguish two cases:

1. If both \( x \) and \( y \) belong to \( L \) (resp., \( R \)), the thesis is a consequence of the convexity of \( L \) (resp., \( R \)).
2. Otherwise, assume w.l.o.g. that \( x \in L \) and \( y \in R \). By definition, we have that:
   (i) \( x \in P \) and \( y \in \text{cl}(P) \), hence \( z_a \in P \), and
   (ii) \( x \in \text{cl}(P') \) and \( y \in P' \), hence \( z_a \in P' \). Therefore, \( z_a \in (P \cap P') \subseteq \text{bndry}(P, P') \), which concludes the proof. \( \square \)

Given a location \( l \) and two polyhedra \( G \) and \( G' \), let \( \text{entry}(G, G') \), the entry region from \( G \) to \( G' \), denote the set of points of the boundary between \( G \) and \( G' \), which can reach \( G' \) by following some linear activity in location \( l \), while always remaining in \( G \cup G' \). Formally:

\[
\text{entry}(G, G') = \{ p \in \text{bndry}(G, G') \mid p + \delta \cdot c \in G', \text{ for some } c \in \text{Flow}(l) \}
\quad \text{ and } \delta \geq 0, \text{ and for all } 0 \leq \delta' < \delta, \ p + \delta' \cdot c \in G \cup G' \}.
\] (4)

For two convex polyhedra \( P \) and \( P' \), \( \text{entry}(P, P') \) can easily be computed as follows:

\[
\text{entry}(P, P') = \text{bndry}(P, P') \cap P'_l.
\] (5)

Indeed \( \text{bndry}(P, P') \subseteq P \cup P' \) is a convex polyhedron and, by definition of \( P'_l \), \( \text{bndry}(P, P') \cap P'_l \) is the set of points which can reach \( P' \) along a linear activity, while always remaining in \( \text{bndry}(P, P') \subseteq P \cup P' \), as required by equation (4).

Notice, however, that equation (5) does not lift to general polyhedra. Indeed, while equation (5) holds even when \( P \) is not convex, it may not hold if the second argument is a non-convex polyhedron as demonstrated by the following example.

![Figure 4: Example showing that \( \text{entry}(P, G) \neq \text{bndry}(P, G) \cap G'_l \), for non-convex \( G \). Flow is deterministic and horizontal.](image-url)

17
Example 3. Consider Figure 4 where \( U \) and \( Z \) are two convex polyhedra represented by gray boxes. Taking \( G = U \cup Z \) and applying equation (5) to compute \( \text{entry}(P,G) \) would result in the thick solid line between \( P \) and \( G \). However, this line does not belong to \( \text{entry}(P,G) \) (in fact, \( \text{entry}(P,G) = \emptyset \)) in this case, since all of its points cannot avoid exiting from both \( P \) and \( G \) before eventually reaching \( U \subseteq G \). Therefore, \( \text{entry}(P,G) \neq \text{bdry}(P,G) \cap G_\ell^L \).

On the other hand, the second argument of \( \text{entry}() \) distributes over union.

Lemma 4. For all polyhedra \( G, G_1 \) and \( G_2 \), it holds:

\[
\text{entry}(G, G_1 \cup G_2) = \text{entry}(G, G_1) \cup \text{entry}(G, G_2).
\]

Proof. By monotonicity of \( \text{entry}() \) w.r.t. the second argument, it follows immediately that \( \text{entry}(G, G_1) \cup \text{entry}(G, G_2) \subseteq \text{entry}(G, G_1 \cup G_2) \).

As to the other direction, let \( p \in \text{entry}(G, G_1 \cup G_2) \). We have \( p \in \text{bdry}(G, G_1 \cup G_2) \) and there exist \( c \in \text{Flow}(l) \) and \( \delta \geq 0 \), such that \( p + \delta \cdot c \in G_1 \cup G_2 \), and for all \( 0 \leq \delta' < \delta \), it holds \( p + \delta' \cdot c \in G_1 \cup G_2 \). Clearly, \( p \in \text{bdry}(G, G_1) \) or \( p \in \text{bdry}(G, G_2) \). Now, for any polyhedron \( G' \), let us define:

\[
\Delta_p^c(G') = \{ \delta \geq 0 \mid p + \delta \cdot c \in G' \}.
\]

Intuitively, \( \Delta_p^c(G') \) is the set of delays during which the linear activity with slope \( c \) that starts in \( p \) lies in \( G' \). It is immediate from its definition that \( \Delta_p^c(G_1 \cup G_2) = \Delta_p^c(G_1) \cup \Delta_p^c(G_2) \). Let now \( \delta^* = \inf \Delta_p^c(G_1 \cup G_2) \). Since \( \delta \in \Delta_p^c(G_1 \cup G_2) \), it holds \( \delta^* \leq \delta \). Moreover, since \( \Delta_p^c(G_1 \cup G_2) = \Delta_p^c(G_1) \cup \Delta_p^c(G_2) \) and \( \inf(A \cup B) = \min\{\inf(A), \inf(B)\} \) for arbitrary sets \( A \) and \( B \), it holds that either \( \delta^* = \inf \Delta_p^c(G_1) \) or \( \delta^* = \inf \Delta_p^c(G_2) \). We can then have two cases:

i. if \( \delta^* \) is the minimum of \( \Delta_p^c(G_1 \cup G_2) \), then either we have \( p + \delta^* \cdot c \in G_1 \) or \( p + \delta^* \cdot c \in G_2 \), and for all \( 0 \leq \delta' < \delta^* \), \( p + \delta' \cdot c \in G \). Moreover, if \( p + \delta^* \cdot c \in G_1 \) (resp., \( p + \delta^* \cdot c \in G_2 \)) then \( p \in \text{bdry}(G, G_1) \) (resp., \( p \in \text{bdry}(G, G_2) \)). In either case, \( p \in \text{entry}(G, G_1) \cup \text{entry}(G, G_2) \);

ii. if \( \delta^* \notin \Delta_p^c(G_1 \cup G_2) \), then for all \( 0 \leq \delta' \leq \delta^* \), \( p + \delta' \cdot c \in G \). Assume \( \delta^* = \inf \Delta_p^c(G_1) \) (the case where \( \delta^* = \inf \Delta_p^c(G_2) \) is similar), then \( \delta^* \notin \Delta_p^c(G_1) \) either. Since, however, \( \delta^* \) is the infimum of the set, in any neighborhood of \( \delta^* \) there must be \( \delta > \delta^* \) such that for all \( \delta^* < \delta' \leq \delta \) we have \( p + \delta' \cdot c \in G_1 \). From the above reasoning, we conclude that \( p + \delta \cdot c \in G_1 \) and, for all \( 0 \leq \delta' < \delta \), \( p + \delta' \cdot c \in G \cup G_1 \). By assumption, we have that \( p + \delta^* \cdot c \in \text{bdry}(G, G_1) \). Moreover, since for all \( 0 \leq \delta'' \leq \delta^* \) it holds \( p + \delta'' \cdot c \notin G_2 \), it follows immediately that also \( p \in \text{bdry}(G, G_1) \). Therefore, \( p \) is a point in \( \text{bdry}(G, G_1) \) that can reach \( G_1 \) following a linear activity, while always remaining in \( G \cup G_1 \). As a consequence, we obtain \( p \in \text{entry}(G, G_1) \subseteq \text{entry}(G, G_1) \cup \text{entry}(G, G_2) \). \( \square \)

As a consequence of Lemma 4, for any convex \( P \) and any general polyhedron \( G \), we can compute \( \text{entry}(P,G) \) by collecting the entry regions from \( P \) to each
convex polyhedron in $[G]$, as follows:

$$entry(P,G) = \bigcup_{P' \in [G]} entry(P,P'),$$

(6)

where $entry(P,P')$ is computed according to equation (5). We can now compute $RWA^m_l$ by the following fixpoint characterization.

**Theorem 3.** For all locations $l$ and sets of valuations $U$, $V$, and $W$, let

$$\tau(U,V,W) = U \cup \bigcup_{P \in [V]} (P \cap entry(P,W) \downarrow l).$$

(7)

We have $RWA^m_l(U,V) = \mu W \cdot \tau(U,V,W)$, where $\mu$ denotes the least fixed point operator. Moreover, the above fixed point expression converges in a finite number of steps.

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 $\mu W \cdot \tau(U,V,W)$ 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 $U \subseteq RWA^m_l(U,V)$. Additionally, notice that $RWA^m_l(U,V) \subseteq U \cup \overline{V}$. 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., $W$).

**Example 4.** Figure 5 shows a single step in the computation of equation (7), for a fixed pair of convex polyhedra $P$ in $\overline{V}$ and $P'$ in $W$, assuming that $Flow(l)$ is the polyhedron $F$ displayed in Figure 5(e). In all the figures, dashed lines represent topologically open sides. In Figure 5(a), the thick segment between $P$ and $P'$ represents $bndry(P,P')$, which, in the example, is contained in $P$. Since $P'$ is topologically open (denoted by the dashed contour), the rightmost point of $bndry(P,P')$ (the point surrounded by the small circle in Figures 5(a) and 5(b)) cannot reach $P'$ along any linear activity. Being $P'$ open, so is $P' \downarrow l$, and its intersection with $P$, namely $entry(P,P')$, does not contain the rightmost point of the boundary (see Figure 5(c)).

Now, any point of $P$ that can reach $entry(P,P')$ (displayed in Figure 5(c)) following some activity can also reach $P'$. The set $Cut = P \cap entry(P,P')\downarrow l$ contains precisely those points (see Figure 5(d)). All these points must then be added to $W$, as they all belong to $RWA^m_l(U,V)$.

### 3.4. Previous Algorithms

In the literature, the standard reference for safety control of linear hybrid systems is [23]. 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^m(U, V) using the following fixpoint formula:

\[
\text{flow\_avoid}(U, V) = \bigcup_{U' \in [U]} \bigcap_{V' \in [V]} \left( \mu W . U' \cup \bigcup_{P \in [V']} (cl(P) \cap cl(P) \cap V \cap W \cap (W \cap P) \cap V') \right)
\]

(8)

The example in Figure 3(a), already discussed in Section 3.3, shows that (8) is different from (in particular, larger than) RWA^m(U, V) when V is non-convex. The problem lies in the fact that formula (8) treats each convex part of V separately, and then takes the intersection of the resulting sets. Considering the situation in Figure 3(a) and taking \( V = V_1 \cup V_2 \), formula (8) would reduce to \( \text{flow\_avoid}(U, V_1) \cap \text{flow\_avoid}(U, V_2) \), and would include the dotted area depicted in that figure. As explained in Section 3.3, this is an incorrect result.

In [8], Deshpande et al. report about an implementation of Wong-Toi’s algorithm in the tool HONEYTECH, obtained as an extension of HyTech [14]. The fixpoint formula that is meant to capture RWA^m(U, V) is the following:

\[
\mu W . U \cup \bigcup_{P \in [V]} \left( P \cap (cl(W) \cap P) \cap W \cap (W \cap P) \cap V \right)
\]

(9)

Differently from (8), formula (9) correctly treats the case of non-convex V. However, it suffers from another issue, related to the computation of the entry.
regions between polyhedra. First, notice that formula (9) can be compared to (7), once we observe that \((cl(W) \cap cl(P) \cap V \cap \bigwedge_i W_i)\) is meant to correspond to our definition of entry regions. In particular, \((cl(W) \cap cl(P) \cap V)\) essentially corresponds to \(\text{bdry}(P,W)\), except for subtle, though non crucial, differences.

Once computed the boundaries between \(P\) and \(W\), formula (9) computes the entry regions from \(P\) to the polyhedron \(W\), by intersecting them with \(W_i\), the pre-flow of \(W\). This corresponds to using equation (5) with polyhedron \(W\) as the second argument. However, Example 3 shows that this may lead to errors when \(W\) is not convex. Indeed, consider again Figure 4, and assume that \(W\) is the union of \(U\) and \(Z\). The result of applying \((cl(W) \cap cl(P) \cap V \cap \bigwedge_i W_i)\) is, in this case, the thick solid line between \(P\) and \(Z\). This is precisely the wrong entry region computed by \(\text{bdry}(P,W) \cap W\), in Example 3. Since this thick line is contained in \(P\), formula (9) ends up adding it to \(RWA_m(U,V)\). However, this line does not belong to \(RWA_m(U,V)\), since all its points cannot avoid hitting \(V\) before eventually reaching \(U\).

3.5. Soundness, Completeness, and Termination of the Fixpoint Procedure of Theorem 3

In order to prove Theorem 3, we characterize the set of points obtained by repeated application of the \(\tau\) operator. For a polyhedron \(V\) and two valuations \(u,v\), assume that there exists an admissible activity that starts in \(u\) and reaches \(v\) while always avoiding \(V\). Clearly, at any given time this activity lies in one of the convex polyhedra in \([V]\). We denote by \(d(u,v,V)\) the minimum number of convex polyhedra in \([V]\) that any admissible activity must cross when going from \(u\) to \(v\) while avoiding \(V\). Formally,

\[
d(u,v,V) = \min\left\{ n > 0 \mid \exists f \in \text{Adm}(f,l), \delta \in \text{span}(f,l), P_0, \ldots, P_{n-1} \in [V] : f(\delta) = v \text{ and } \forall 0 \leq \delta' \leq \delta \exists i \in \{0, \ldots, n-1\} : f(\delta') \in P_i \right\},
\]

and we set \(d(u,v,V) = \infty\) if no activity goes from \(u\) to \(v\) while avoiding \(V\). Clearly, we have either \(d(u,v,V) = \infty\) or \(d(u,v,V) \leq \|V\|\). Then, we define a version of \(RWA_m\) that only takes into account activities that cross a given number of convex polyhedra in \([V]\). Let \(RWA_m(U,V,i) = \{u \in RWA_m(U,V) \mid \exists v \in U : d(u,v,V) \leq i\}\). Moreover, let \(\tau^i\) denote the result of applying the \(\tau\) operator \(i\) times to its third argument. Formally,

\[
\tau^i(U,V,W) = \tau(U,V,W)
\]
\[
\tau^{i+1}(U,V,W) = \tau(U,V,\tau^i(U,V,W)).
\]

The following lemma states that if there is an activity that goes from \(u\) to \(v\) while avoiding \(V\), there is also a sequence of linear activities that does the same. Moreover, each linear activity is contained within one convex polyhedron of \([V]\) and hence the connecting points between any two consecutive linear activities lie on the boundary between two polyhedra in \([V]\).

**Lemma 5.** Let \(u\) and \(v\) be two valuations, and \(V\) a polyhedron. If \(d(u,v,V) = k < \infty\) then there is a finite sequence \(f_0, \ldots, f_{k-1}\) of linear activities, delays \(\delta_0, \ldots, \delta_{k-1}\), and convex polyhedra \(P_0, \ldots, P_{k-1} \in [V]\) such that (i) \(f_0 \in
For all locations \( l \), polyhedra \( U \) and \( V \), and \( i \geq 1 \), it holds \( \text{RWA}^\text{in}_i(U,V,i) \subseteq \tau^i(U,V,\emptyset) \).

**Proof.** Let \( u \in \text{RWA}^\text{in}_i(U,V,i) \). We proceed by induction on \( i \geq 1 \). If \( i = 1 \) then \( u \in \text{RWA}^\text{in}_1(U,V,1) \) and for some \( v \in U \), \( d(u,v,V) = 1 \). As a consequence, \( u \) and \( v \) must belong to the same convex polyhedron \( P_0 \subseteq U = \tau^1(U,V,\emptyset) \), and the thesis follows.

If \( i \geq 2 \), then there must be an activity that starts in \( u \) and reaches a point \( u' \in U \) without visiting \( V \) and \( d(u,u',V) \leq i < \infty \). If \( d(u,u',V) < i \), the thesis immediately follows by the induction hypothesis. If \( d(u,u',V) = i \), then, by Lemma 5, there is a finite sequence \( f_0, \ldots, f_{i-1} \) of linear activities, delays \( \delta_0, \ldots, \delta_{i-1} \), and convex polyhedra \( P_0, \ldots, P_{i-1} \in [\hat{V}] \) such that (i) \( f_0 \in \text{Adm}(\langle l, u \rangle) \), (ii) \( f_{i-1}(\delta_{i-1}) = u' \), (iii) for all \( j = 0, \ldots, i-2 \) it holds \( f_j(\delta_j) \in \text{cl}(P_j) \cap \text{cl}(P_{j+1}) \) and \( f_{j+1} \in \text{Adm}(\langle l, f_j(\delta_j) \rangle) \), and (iv) for all \( j = 0, \ldots, i-1 \) and \( 0 < \delta' < \delta_j \) it holds \( f_j(\delta') \in P_j \).

Then, for all \( 0 < \delta < \delta_1 \), \( f_1(\delta) \in P_1 \) and, therefore, \( d(f_1(\delta), u', V) = i - 1 \). This implies that, for all \( 0 < \delta < \delta_1 \), \( f_1(\delta) \in \text{RWA}^\text{in}_1(U,V,i-1) \) and, by induction hypothesis, \( f_1(\delta) \in \tau^{i-1}(U,V,\emptyset) \). Hence, we have that, for all \( 0 < \delta < \delta_1 \), \( f_1(\delta) \in \tau^{i-1}(U,V,\emptyset) \cap P_1 \).

Let now \( v = f_0(\delta_0) = f_1(0) \). By the above reasoning, the convexity of \( P_1 \) and, therefore, the convexity of \( \tau^{i-1}(U,V,\emptyset) \cap P_1 \), we conclude that \( v \in \text{cl}(\tau^{i-1}(U,V,\emptyset) \cap P_1) \). In addition, \( f_1 \in \text{Flow}(l) \), since \( f_1 \) is a linear activity in \( \text{Adm}(l,f_1(0)) \). As a consequence, we obtain that, for all \( 0 < \delta < \delta_1 \), \( v \in f_1(\delta) \cap \tau^{i-1}(U,V,\emptyset) \cap P_1 \).

From the above observations and from \( v = f_0(\delta_0) \in \text{cl}(P_0) \cap \text{cl}(P_1) \), it follows that \( v \in \text{cl}(P_0) \cap \text{cl}(\tau^{i-1}(U,V,\emptyset) \cap P_1) \subseteq \text{bdry}(P_0,\tau^{i-1}(U,V,\emptyset) \cap P_1) \subseteq \text{bdry}(P_0,\tau^{i-1}(U,V,\emptyset)) \). Hence, we have \( v \in \text{entry}(P_0,\tau^{i-1}(U,V,\emptyset)) \) and, since also \( f_0 \in \text{Flow}(l) \), \( u \in v \cap \text{entry}(P_0,\tau^{i-1}(U,V,\emptyset)) \). From that and \( u = f_0(0) \in P_0 \) we conclude that \( u \in P_0 \cap \text{entry}(P_0,\tau^{i-1}(U,V,\emptyset)) \). Hence the thesis.

**Lemma 7.** For all locations \( l \), polyhedra \( U \) and \( V \), and \( i \geq 1 \), it holds \( \text{RWA}^\text{in}_i(U,V,i) \supseteq \tau^i(U,V,\emptyset) \).
Proof. Let \( u \in \tau^i(U, V, \emptyset) \), we proceed by induction on \( i \). If \( i = 1 \) then \( u \in U \) and the thesis is obvious. Otherwise, let \( T = \tau^{i-1}(U, V, \emptyset) \), there exists \( P \in [\mathcal{V}] \) such that \( u \in P \cap \text{entry}(P, T) \). Hence, there is a linear activity \( f_1 \in \text{Adm}(\langle l, u \rangle) \) that reaches a point \( v \in \text{entry}(P, T) \). By definition, there exists \( P' \in [T] \) such that \( v \in \text{entry}(P, P') \cap P' \). Since \( T \subseteq \mathcal{V} \), assume w.l.o.g. that \( P' \) is entirely contained in one convex polyhedron \( P'' \) in \([\mathcal{V}]\). Since \( v \in P' \), there is another linear activity \( f_2 \) from \( v \) to a point \( w \in P'' \). Both activities \( f_1 \) and \( f_2 \) are entirely contained in \( V \). By concatenating \( f_1 \) and \( f_2 \), we obtain an activity that goes from \( u \) to \( T \) while avoiding \( V \). This activity spans at most two elements of \([\mathcal{V}]\): \( P \) and \( P'' \), which may coincide. Moreover, the activity may be non-differentiable in the connecting point. By inductive hypothesis, \( T \subseteq \text{RWA}^{m_l}(U, V, i - 1) \) and hence from \( w \) there is an activity that reaches \( U \) while avoiding \( V \) and crossing at most \( i - 1 \) elements of \([\mathcal{V}]\), including \( P'' \). We conclude that \( d(u, v, V) \leq i \) and \( u \in \text{RWA}^{m_l}(U, V, i) \).

Proof of Theorem 3. First, notice that \( \text{RWA}^{m_l}(U, V) = \text{RWA}^{m_l}(U, V, n) \), where \( n = |[\mathcal{V}]| \). Hence, by Lemmas 6 and 7 we have \( \text{RWA}^{m_l}(U, V) = \tau^k(U, V, \emptyset) \) for all \( k \geq n \), i.e., the sequence converges to the fixed point after at most \( n \) iterations. \( \square \)

4. Efficient Computation of the Predecessor Operator

We first provide an equivalent but finer-grained formulation of the \( \tau \) operator. Observe that the distributivity of \( \langle l \rangle \) over union ensures that, for any convex polyhedron \( P \) and general polyhedron \( W \), the following holds:

\[
P \cap \text{entry}(P, W) \langle l \rangle = P \cap \left( \bigcup_{P' \in [W]} \text{entry}(P, P') \langle l \rangle \right)
\]

\[
= P \cap \bigcup_{P' \in [W]} \left( \text{entry}(P, P') \langle l \rangle \right)
\]

\[
= \bigcup_{P' \in [W]} \left( P \cap \text{entry}(P, P') \langle l \rangle \right).
\]

As a consequence, we can equivalently reformulate equation (7) as follows:

\[
\tau(U, V, W) = U \cup \bigcup_{P \in [W]} \bigcup_{P' \in [W]} \left( P \cap \text{entry}(P, P') \langle l \rangle \right).
\]

In our implementation of the \( \text{CPre} \) operator, instead of computing the operator \( \text{RWA}^{m_l} \), we compute the dual operator \( \text{SOR}^{M}(Z, V) \) (for \text{must stay or reach} \), containing the points which either remain in \( Z \) forever or reach \( V \) before leaving \( Z \). The operator \( \text{SOR}^{M} \) can be defined as follows:

\[
\text{SOR}^{M}(Z, V) = \text{RWA}^{m_l}(Z, V).
\]
Since the arguments of $RWA^m$ can be assumed to be disjoint, we can assume w.l.o.g. that $V \subseteq Z$. As a consequence, we can compute $CPre(A)$ as
\[
\bigcup_{l \in Loc} \{l\} \times (\mathcal{A}_l \cap SOR^M(ZV) \cup (\mathcal{A}_l \setminus B_l) \cup C_l \cup \overline{Inv_l})).
\]
From (10), we obtain a fixpoint characterization of the operator $SOR^M_l$:
\[
SOR^M_l(Z,V) = RWA^m(Z,V) = \mu W . Z \cup \bigcup_{P \in [V]} \bigcup_{P' \in [W]} (P \cap entry(P,P') \setminus_l).
\]
\[
\nu W . Z \setminus \bigcup_{P \in [V]} \bigcup_{P' \in [W]} (P \cap entry(P,P') \setminus_l).
\]
(11)

The following two sections show how to effectively and efficiently compute fixpoint (11).

4.1. Computing $SOR^M$

In this section, we show how to efficiently compute $SOR^M_l(Z,V)$, given two polyhedra $Z$ and $V$. Fixpoint equation (11) can easily be converted into an iterative algorithm, consisting in generating a sequence of polyhedra $(W_n)_{n \in \mathbb{N}}$, where $W_0 = Z$ and
\[
W_{i+1} = W_i \setminus \bigcup_{P \in [V]} \bigcup_{P' \in [W]} (P \cap entry(P,P') \setminus_l).
\]
(12)

Theorem 3 ensures that such sequence converges to a fixpoint within a finite number of steps. The naive implementation of the algorithm is done by an outer loop over the polyhedra $P \in [V]$ and an inner loop over $P' \in [W]$. As a first improvement, we notice that each iteration of the outer loop removes from $W_i$ a portion of $P \in [V]$. Hence, the portion of $P$ that is not contained in $W_i$ is irrelevant, and we may replace (12) with:
\[
W_{i+1} = W_i \setminus \bigcup_{P \in [W_i \cap [V]]} \bigcup_{P' \in [W]} (P \cap entry(P,P') \setminus_l).
\]
(13)
Moreover, we can avoid the need to intersect $W_i$ with $V$ at each iteration, by starting with $W_0 = Z \setminus V$, setting:
\[
W_{i+1} = W_i' \setminus \bigcup_{P \in [W_i']} \bigcup_{P' \in [W]} (P \cap entry(P,P') \setminus_l).
\]
(14)

\(^3\)Recall that the relation between the least and the greatest fixed point operators is the following: $\mu W . \Phi(W) = \nu W . \Phi(W)$, where $\Phi(W)$ is obtained by substituting the free occurrences of $W$ in $\Phi$ with $\overline{W}$. 

24
and noticing that \( W_i = W_i' \cup V \) for all \( i \geq 0 \). As a consequence, \( SOR_i^M(Z, V) = \lim_{i \to \infty} W_i = V \cup \lim_{i \to \infty} W_i' \). The implementation described so far is called the basic approach in the following.

**Introducing Adjacency Relations.** Given two disjoint convex polyhedra \( P \) and \( P' \), we say that they are adjacent if \( \text{bndry}(P, P') \neq \emptyset \). In the basic approach, the inner loop is repeated for each \( P' \in [\mathbb{W}_i] \), even if convex polyhedra \( P' \) that are not adjacent to \( P \) result in an empty \( \text{entry}(P, P') \) and are therefore irrelevant. Hence, we define the binary relation of external adjacency \( \text{Ext}_i \), which associates a polyhedron \( P \in [W_i] \) with its entry regions \( \text{entry}(P, P') \neq \emptyset \), for all \( P' \in [\mathbb{W}_i] \). Formally,

\[
\text{Ext}_i = \{ (P, \text{entry}(P, P')) \mid P \in [W_i], P' \in [\mathbb{W}_i], \text{ and } \text{entry}(P, P') \neq \emptyset \}. \tag{15}
\]

Once \( \text{Ext}_i \) is introduced and properly maintained, it also enables to optimize the outer loop. Rather than \( P \in [W_i] \), it is enough to consider all \( P \) which are associated with at least one entry region in \( \text{Ext}_i \), i.e., all \( P \) such that \( (P, R) \in \text{Ext}_i \) for some \( R \). Summarizing, using \( \text{Ext}_i \) we can replace (14) with

\[
W_{i+1} = W_i \setminus \bigcup_{(P, R) \in \text{Ext}_i} (P \cap R/\{i\}). \tag{16}
\]

Clearly, some extra effort is required to initialize and maintain \( \text{Ext}_i \). Initialization is performed by simply applying (15). Regarding maintenance, we briefly discuss how to efficiently compute \( \text{Ext}_{i+1} \).

During the \( i \)-th iteration, certain convex polyhedra \( P \in [W_i] \) are cut by removing the points that may directly reach a convex polyhedron \( P' \in [\mathbb{W}_i] \). These cuts may expose other convex polyhedra in \( [W_i] \), that were previously covered by \( P \). These exposed polyhedra will be the only ones to have associated entry regions in \( \text{Ext}_{i+1} \). In order to be exposed by a cut made to \( P \), a convex polyhedron must be adjacent to \( P \). Hence, in order to compute \( \text{Ext}_{i+1} \) it is useful to have information about the adjacency among the polyhedra in \( [W_i] \). To this aim, we also introduce the binary relation of internal adjacency \( \text{Int}_i \) between polyhedra in \( [W_i] \):

\[
\text{Int}_i = \{ (P_1, P_2) \mid P_1, P_2 \in [W_i], P_1 \neq P_2 \text{ and } \text{bndry}(P_1, P_2) \neq \emptyset \}. \tag{17}
\]

The computation of \( \text{Int}_0 \) requires the complete scan of all \( P_1, P_2 \in [W_0] \), while \( \text{Int}_{i+1} \) is obtained incrementally from \( \text{Int}_i \) and \( \text{Ext}_i \). Given \( (P, R) \in \text{Ext}_i \), let \( \text{Cut} = P \cap (R/\{i\}) \) and \( P_{\text{new}} = P \setminus \text{Cut} \). Notice that \( P_{\text{new}} \) may be non-convex, being the result of a set-theoretical difference between two convex polyhedra. To obtain \( \text{Int}_{i+1} \), we add to \( \text{Int}_i \) the pairs of adjacent convex polyhedra \( (P_1, P_2) \) such that either (i) both \( P_1 \) and \( P_2 \) belong to \( [P_{\text{new}}] \), or (ii) one of them belongs to \( [P_{\text{new}}] \) and the other is adjacent to \( P \) according to \( \text{Int}_i \). Moreover, once \( P_{\text{new}} \) replaces \( P \) in \( W_{i+1} \), it is necessary to remove all the pairs \( (P, P') \) from \( \text{Ext}_i \) and \( \text{Int}_i \).

Algorithms 1-3, reported in Figures 6 and 7, represent a concrete implementation of the technique described so far. In Algorithm 1, \( \text{Ext}_{i\text{old}} \) and \( \text{Int}_{i\text{old}} \)
Algorithm 1: $SOR^M(Z,V,F)$

| Input: Poly $Z$, $V$, CPoly $F$ |
| Output: Poly $SOR^M(Z,V,F)$ |

foreach CPoly $P \in [Z]$ do
  $Int_{new} \gets UpdInt(Int_{old}, P, Z)$;
  $E \gets PotentialEntry(Z, P, Int_{new}, F)$;
  $Ext_{new} \gets UpdExt(Ext_{old}, P, E, F, V)$;
while $Ext_{new} \neq \emptyset$ do
  $Ext_{old} \gets Ext_{new}$;
  $Int_{old} \gets Int_{new}$;
  $Ext_{new} \gets \emptyset$;
  foreach $P$ s.t. $\langle P, R \rangle \in Ext_{old}$ do
    $B \gets \bigcup \{ R \mid \langle P, R \rangle \in Ext_{old} \}$;
    $Cut \gets P \cap (B_{\mathcal{C}})$;
    if $Cut \neq \emptyset$ then
      $P_{new} \gets P \setminus Cut$;
      foreach $P' \in [P_{new}]$ do
        $Int_{new} \gets UpdInt(Int_{new}, P', P_{new})$;
      endforeach
      foreach $P' s.t. \langle P, P' \rangle \in Int_{old}$ do
        $Int_{new} \gets UpdInt(Int_{new}, P', P_{new})$;
        $Ext_{new} \gets UpdExt(Ext_{old}, P', Cut, F, V)$;
      endforeach
    $Int_{new} \gets Int_{new} \setminus \{ \langle P, Q \rangle \in Int_{old} \}$;
  endforeach
  $Int_{new} \gets Int_{new} \{ \langle P, Q \rangle \in Int_{old} \}$;
return $\{ P \mid \langle P, P' \rangle \in Int_{new} \}$;

Figure 6: The algorithm computing $SOR^M$.

represent the old adjacency relations, while $Ext_{new}$ and $Int_{new}$ the new ones. The first “for each” loop initializes both relations, followed by a “while” loop that iterates until the external adjacency relation is empty. Maintenance of the adjacency relations is delegated to Algorithms 2 and 3, that receive as input the relation they have to update, the convex polyhedron $P$ whose adjacencies need to be examined, and a general polyhedron $Cand$ containing the convex polyhedra that may be adjacent to $P$. Additionally, Algorithm 3 also needs to know the input set $V$ (region to be avoided) and the location flow $F = Flow(l)$.

The auxiliary function $PotentialEntry$ returns the potential entry region for $P$, i.e., a set of convex polyhedra contained in $\bar{Z}$ that may have a non-empty entry region from $P$. In this version, we simply have

$$PotentialEntry(Z, P, Int_{old}, F) = \bar{Z}. \tag{18}$$

The reason why $PotentialEntry$ features four arguments will become clear in the next paragraph, where a different version of this function is introduced, leading to substantially improved overall performances.

Further Improving the Performance. It is often the case that the portion of $\bar{Z}$ which is relevant to computing the entry regions of a given convex polyhedron
Algorithm 2: \texttt{UpdInt}($\text{Int}, P, \text{Cand}$)

**Input:** Set of $\text{CPoly}$ pairs $\text{Int}$; $\text{CPoly}$ $P$; Poly $\text{Cand}$;  

**Output:** Set of $\text{CPoly}$ pairs $\text{Int}$;  

$\text{Int} \leftarrow \text{Int} \cup \{(P, \emptyset)\}$; \quad \textbf{foreach} $\text{CPoly} \; P' \in \{\text{Cand}\}$, with $P' \neq P$ \textbf{do} \quad \textbf{if} $\text{bndry}(P, P') \neq \emptyset$ \textbf{then} \quad $\text{Int} \leftarrow \text{Int} \cup \{(P, P')\}$; \quad \textbf{return} $\text{Int}$;  

Algorithm 3: \texttt{UpdExt}($\text{Ext}, P, \text{Cand}, F, V$)

**Input:** Set of $\text{CPoly}$ pairs $\text{Ext}$; $\text{CPoly}$ $P$, $F$; Poly $\text{Cand}$, $V$;  

**Output:** Set of $\text{CPoly}$ pairs $\text{Ext}$;  

\textbf{if} $P \nsubseteq V$ \textbf{then} \quad \textbf{foreach} $\text{CPoly} \; P' \in \{\text{Cand}\}$ \textbf{do} \quad $R \leftarrow \text{entry}(P, P')$; \quad \textbf{if} $R \neq \emptyset$ \textbf{then} \quad $\text{Ext} \leftarrow \text{Ext} \cup \{(P, R)\}$; \quad \textbf{return} $\text{Ext}$;  

Figure 7: The algorithms updating the adjacency relations $\text{Int}$ and $\text{Ext}$.  

$P$ is much smaller than the whole set $\bar{Z}$. This often leads to a large number of attempts to compute entry regions which end up empty. To avoid this issue, for each $P$ in $[Z]$ we proceed as follows. We first collect $P$ and all convex polyhedra in $[Z]$ that are adjacent to it: $P_{\text{adj}} = \{P\} \cup \{(P', P') \in \text{Int} \}$). Then, we set

$$
\text{PotentialEntry}(Z, P, \text{Int} \text{0}, F) = (P \uparrow F) \setminus P_{\text{adj}}.
$$

(19)

Notice that each convex polyhedron in $\bar{Z} \setminus \text{PotentialEntry}(Z, P, \text{Int} \text{0}, F)$ has an empty entry region from $P$, since it cannot be reached from $P$ following a straight-line admissible activity. Moreover, each convex polyhedron $P'$ in $\text{PotentialEntry}(Z, P, \text{Int} \text{0}, F) \cap Z$ is not adjacent to $P$, since, otherwise, it would be contained in $P_{\text{adj}}$ as well. Hence, the entry region from $P$ to $P'$ is empty. Therefore, the polyhedron $\text{PotentialEntry}(Z, P, \text{Int} \text{0}, F)$ contains all and only the convex polyhedra which, if adjacent to $P$, are contained in $Z$ and have a non-empty entry region from $P$.

4.2. Exact Computation of Pre-Flow

As seen in the previous section, one of the basic operations on polyhedra that are needed to compute $\text{SOR}^M$ is the pre-flow operator $\downarrow$. It is sufficient to compute $P \downarrow F$ for convex $P$ and $F$, for two reasons: First, we always have $F = \text{Flow}(l)$, for a given location $l$, and $\text{Flow}(l)$ is a convex polyhedron by assumption. Second, $(P_1 \cup P_2) \downarrow F = P_1 \downarrow F \cup P_2 \downarrow F$, so the pre-flow of a general polyhedron is the union of the pre-flows of its convex polyhedra. The pre-flow of $P$ w.r.t. $F$ is equivalent to the post-flow of $P$ w.r.t. $-F$, defined as:

$$
P \downarrow_{-F} = \{x + \delta \cdot y \mid x \in P, y \in -F, \delta \geq 0\}.
$$

The post-flow operation coincides with the time-elapse operation introduced in [11] for topologically closed convex polyhedra. Notice that for convex polyhedra $P$ and $F$, the post-flow of $P$ w.r.t. $F$ may not be a convex polyhedron:
following [2], let $P \subseteq \mathbb{R}^2$ be the polyhedron containing only the origin $(0,0)$ and let $F$ be defined by the constraint $y > 0$. We have $P \uparrow F = \{(0,0)\} \cup \{(x,y) \in \mathbb{R}^2 \mid y > 0\}$, which is not a convex polyhedron (although it is a convex subset of $\mathbb{R}^2$). The Parma Polyhedral Library (PPL, see [4]), for instance, only provides an over-approximation of the post-flow operator, that we denote by $\uparrow_{\text{PPL}}$. Precisely, $P \uparrow_{\text{PPL}} F$ is the smallest convex polyhedron containing $P \uparrow F$.

On the other hand, the post-flow of a convex polyhedron is always the union of two convex polyhedra, according to the equation

$$P \uparrow F = P \cup (P \uparrow_{>0} F),$$

where $P \uparrow_{>0} F$ is the positive post-flow of $P$, i.e., the set of valuations that can be reached from $P$ via a straight line of non-zero length whose slope belongs to $F$. Formally,

$$P \uparrow_{>0} F = \{x + \delta \cdot y \mid x \in P, y \in F, \delta > 0\}.$$

Hence, in order to exactly compute the post-flow of a convex polyhedron, we show how to compute the positive post-flow.

Convex polyhedra admit two finite representations, in terms of constraints or generators. Libraries like PPL maintain both representations for each convex polyhedron and efficient algorithms exist for keeping them synchronized [6, 22]. The constraint representation refers to the set of linear inequalities whose solutions are the points of the polyhedron. The generator representation consists in three finite sets of points, closure points, and rays, that generate all points in the polyhedron by linear combination. More precisely, for each convex polyhedron $P \subseteq \mathbb{R}^n$ there exists a triple $(V, C, R)$ such that $V$, $C$, and $R$ are finite sets of points in $\mathbb{R}^n$, and $x \in P$ if and only if it can be written as

$$\sum_{v \in V} \alpha_v \cdot v + \sum_{c \in C} \beta_c \cdot c + \sum_{r \in R} \gamma_r \cdot r, \quad (20)$$

where all coefficients $\alpha_v$, $\beta_c$, and $\gamma_r$ are non-negative reals, $\sum_{v \in V} \alpha_v + \sum_{c \in C} \beta_c = 1$, and there exists $v \in V$ such that $\alpha_v > 0$. We call the triple $(V, C, R)$ a generator for $P$.

Intuitively, the elements of $V$ are the proper vertices of the polyhedron $P$, the elements of $C$ are vertices of the topological closure of $P$ that do not belong to $P$, and each element of $R$ represents a direction of unboundedness of $P$.

The following result shows how to efficiently compute the positive post-flow operator, using the generator representation.

**Theorem 4.** Given two convex polyhedra $P$ and $F$, let $(V_P, C_P, R_P)$ be a generator for $P$ and $(V_F, C_F, R_F)$ a generator for $F$. The triple $(V_P \oplus V_F, C_P \cup V_P, R_P \cup V_F \cup C_F \cup R_F)$ is a generator for $P \uparrow_{>0} F$, where $\oplus$ denotes Minkowski sum.

**Proof.** Let $z \in P \uparrow_{>0} F$, we show that there are coefficients $\alpha_v$, $\beta_c$ and $\gamma_r$ such that $z$ can be written as (20), for $V = V_P \oplus V_F$, $C = C_P \cup V_P$, and $R = R_P \cup V_F \cup C_F \cup R_F$. 

28
By definition, there exist \( x \in P, \ y \in F, \) and \( \delta > 0 \) such that \( z = x + \delta y. \) Hence, there are coefficients \( \alpha^y, \beta^y, \) and \( \gamma^y \) witnessing the fact that \( x \in P, \) and coefficients \( \alpha^y, \beta^y, \) and \( \gamma^y \) witnessing the fact that \( y \in F. \) Moreover, there is \( i \in V_P \) and \( j \in V_F \) such that \( \alpha^y_i > 0 \) and \( \alpha^y_j > 0. \) Let \( \varepsilon = \min\{\alpha^y_i, \delta \alpha^y_j\} \) and notice that \( \varepsilon > 0. \) It holds

\[
\alpha^y_i \cdot i + \delta \cdot \alpha^y_j \cdot j = (\alpha^y_i - \varepsilon) i + \varepsilon i + (\delta \cdot \alpha^y_j - \varepsilon) j + \varepsilon j = \varepsilon(i + j) + (\alpha^y_i - \varepsilon)i + (\delta \cdot \alpha^y_j - \varepsilon)j.
\]

Hence,

\[
z = \sum_{v \in V_P} \alpha^y_v \cdot v + \sum_{c \in C_P} \beta^y_c \cdot c + \sum_{r \in R_P} \gamma^y_r \cdot r + \\
+ \delta \left( \sum_{v \in V_P} \alpha^y_v \cdot v + \sum_{c \in C_P} \beta^y_c \cdot c + \sum_{r \in R_P} \gamma^y_r \cdot r \right) = \varepsilon(i + j) + \left( (\alpha^y_i - \varepsilon)i + \sum_{v \in V_P \setminus \{i\}} \alpha^y_v \cdot v + \sum_{c \in C_P} \beta^y_c \cdot c \right) + \\
\left( (\delta \cdot \alpha^y_j - \varepsilon)j + \sum_{r \in R_P} \gamma^y_r \cdot r + \delta \sum_{v \in V_P \setminus \{j\}} \alpha^y_v \cdot v + \delta \sum_{c \in C_P} \beta^y_c \cdot c + \delta \sum_{r \in R_P} \gamma^y_r \cdot r \right).
\]

One can easily verify that: \( (i) \) all coefficients are non-negative; \( (ii) \) the sum of the coefficients of the points in \( V \) and \( C \) is 1; \( (iii) \) there exists a point in \( V, \) namely \( i + j, \) whose coefficient is strictly positive.

Conversely, let \( z \) be a point that can be expressed as (20), for \( V = V_P \oplus V_F, \)

\( C = C_P \cup V_P, \) and \( R = R_P \cup V_F \cup C_F \cup R_F. \) We prove that \( z \in P \not\in F \) by identifying \( x \in P, \ y \in F \) and \( \delta > 0 \) such that \( z = x + \delta y. \)

Notice that \( (a) \) \( \sum_{v \in V_P \oplus V_F} \alpha_v + \sum_{c \in C_P \cup V_P} \beta_c + 1, \) and \( (b) \) there exists \( v^* \in V_P \oplus V_F \) such that \( \alpha_{v^*} \not\in 0. \) We set

\[
x = \sum_{v_1 \in V_P, v_2 \in V_F} \alpha_{v_1 + v_2} \cdot v_1 + \sum_{c \in C_P \cup V_P} \beta_c \cdot c + \sum_{r \in R_P} \gamma_r \cdot r.
\]

We claim that \( x \in P: \) first, \( x \) is expressed as a linear combination of points in \( (V_P, C_P, R_P); \) second, all coefficients are non-negative; third, the sum of the coefficients of the points in \( V_P \) and in \( C_P \) is 1, due to \( (a) \) above; finally, since \( \alpha_{v^*} > 0, \) there is a point in \( V_P \) whose coefficient is positive. Then, we set

\[
\delta = \sum_{v \in V_P \oplus V_F} \alpha_v + \sum_{r \in V_P \cup C_P} \gamma_r, \quad \text{and}
\]

\[
y = \frac{1}{\delta} \left( \sum_{v_1 \in V_P, v_2 \in V_F} \alpha_{v_1 + v_2} \cdot v_2 + \sum_{r \in V_P \cup C_P \cup R_P} \gamma_r \cdot r \right).
\]
Since $\alpha_v > 0$, we have $\delta > 0$. We claim that $y \in F$: first, $y$ is a linear combination of points in $(V_F, C_F, R_F)$; second, all coefficients are non-negative; third, the sum of the coefficients of the points in $V_F$ and in $C_F$ is 1, due to our choice of $\delta$; finally, since $\alpha_v > 0$, there is a point in $V_F$ whose coefficient is positive. \hfill \square

5. Experiments with PHAVer+

In this section we present some experiments on safety control performed on an implementation of the three algorithms described in the previous section.

The synthesis procedure has been implemented on the top of the open-source tool PHAVer [9]. In the remainder of this section, the basic approach, corresponding to the naive implementation of equation (14), will be denoted by Basic, the adjacency approach, corresponding to Algorithm 1 with $PotentialEntry$ computed by equation (18), by Adj, and the local adjacency approach corresponding to Algorithm 1 with $PotentialEntry$ computed by equation (19), by Local. A binary pre-release of our implementation, that we call PHAVer+, can be downloaded at http://people.na.infn.it/mfaella/phaverplus. All reported experiments have been performed on an Intel Xeon (2.80GHz) PC.

**Truck Navigation Control.** The following example, the Truck Navigation Control (TNC), is derived from the work [8]. Consider an autonomous toy truck, which is responsible for avoiding some 2 by 1 rectangular pits. The truck can take 90-degrees 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. Notice that the TNC proposed in [8] is limited to one turn only, while our analysis is extended to the complete case (an unlimited number of turns is allowed). Figure 8 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 is used to enforce a one-time-unit wait between turns.

Figure 9 shows the two iterations needed to compute the fixpoint in Theorem 2, in the case of two pits. The safe set is the white area, while the gray region contains the points from which 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 9(a). The first iteration (Figure 9(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 9(d)) computes $CPre(CPre(T))$ and extends the unsafe set by adding the area 6. Area 6 contains points from where the truck may turn before reaching the pits, but then it will hit the pits despite the turn. In detail, the truck reaches the area called $A$ after one time unit, and then it may choose one of two possible turns (namely, NW or SE). However, as shown in Figures 9(c) and 9(e), the area $A$ is unsafe both in the NW and in the SE direction.
Figure 8: TNC modeled as a LHA.

The algorithm performs three iterations: the final result is obtained after the second one (see Figure 9(d)), while the third iteration is needed to discover that the fixpoint is reached.

We tested our implementation on progressively larger versions of the truck model, by increasing the number of pits. Using an exponential scale, Figure 10 compares the performance of our tool (solid lines for the different implementations of our algorithm) to the performance reported in [8] (dotted line). We were not able to replicate the experiments in [8], since HONEYTECH is not publicly available. Because of the different hardware used, only a qualitative comparison can be made: going from 1 to 6 pits (as the case study in [8]), the runtime of HONEYTECH shows an exponential behavior, while our tool exhibits an
approximately linear growth, as shown in Figure 10, where the performance of PHAVer+ is plotted up to 10 pits.

Figure 10 and Table 1 further compare the performances of the three versions of the algorithm (Basic, Adj and Local) against the case study considered.

Figure 10: Computation time as a function of the number of pits.

<table>
<thead>
<tr>
<th># pits</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>2.5</td>
<td>14.1</td>
<td>57.9</td>
<td>129.6</td>
<td>519.8</td>
<td>1176.0</td>
<td>3013.4</td>
<td>6176.5</td>
<td>7584.0</td>
</tr>
<tr>
<td>Adj</td>
<td>2.5</td>
<td>8.7</td>
<td>30.6</td>
<td>46.0</td>
<td>86.0</td>
<td>197.3</td>
<td>333.6</td>
<td>340.9</td>
<td>499.1</td>
</tr>
<tr>
<td>Local</td>
<td>0.5</td>
<td>1.1</td>
<td>2.1</td>
<td>3.6</td>
<td>5.2</td>
<td>8.1</td>
<td>10.7</td>
<td>14.5</td>
<td>18.8</td>
</tr>
</tbody>
</table>

Table 1: Performance in seconds for TNC w.r.t. the number of pits.

The comparison shows that there is a significant gain in performances when moving from the Basic approach (represented by the circle-crossed line in Figure 10) to the Adj approach (represented by the square-crossed line), and from the Adj approach to the Local approach (represented by the plus-crossed line). The comparison provides experimental evidence that both the introduction of the adjacency relations and the reduction of PotentialEntry in the Local approach may have a dramatic impact on the running times.

The TNC example described so far features a deterministic flow, i.e., from each state there is only one admissible activity. A non-deterministic version of the TNC, where environmental disturbances are allowed, can be obtained by, e.g., replacing the differential equation for position $y$ in locations SW and SE with the differential constraint $-1.5 \leq \dot{y} \leq -0.5$, and in locations NW and NE with the differential constraint $0.5 \leq \dot{y} \leq 1.5$. The resulting flow in each location allows for some uncertainty on the exact direction taken by the vehicle, as shown in Figure 11(b) for the SW direction. The safe area $T$ is the same
(a) The pits to avoid (i.e., T).
(b) Non-deterministic flow (SW direction).
(c) CPre(T), SW direction.
(d) CPre(CPre(T)), SW direction.

Figure 11: Evolution of the fixpoint in the case of two pits and non-deterministic flow. All figures are cross-sections for $t = 0$. Dashed arrows represent flow direction.

as in the deterministic version, e.g., the area outside the gray boxes 1 and 2 in Figure 11(a) in the case of two pits. The first iteration (Figure 11(c)) computes $CPre(T)$ and extends the unsafe set to those points (areas 3, 4, 5, and 6) from where the truck may flow into the pits, before being able to change direction. The second iteration (Figure 11(d)) computes $CPre(CPre(T))$ and extends the unsafe set by adding the area 7, for similar reasons as the deterministic example. Indeed, in all those points the truck may turn before reaching the pits; however, after any allowed turn, it still may end up hitting the pits. As in the deterministic case, the final result is computed after two iterations. Table 5 reports the running times of the Local algorithm on different instances with up to 10 pits.

<table>
<thead>
<tr>
<th># pits</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local</td>
<td>0.1</td>
<td>0.7</td>
<td>1.8</td>
<td>3.5</td>
<td>6.2</td>
<td>9.5</td>
<td>14.4</td>
<td>22.1</td>
<td>26.82</td>
<td>34.64</td>
</tr>
</tbody>
</table>

Table 2: Performance (secs.) for non-deterministic TNC w.r.t. the number of pits.

Flight Navigation Control. The three dimensional version of the TNC can be seen as an abstract Flight Navigation Control (FNC) system. Here the obstacles are 3D boxes, whose width, depth, and height are set to 2, 2 and 1, respectively. The 90-degrees left and right turns are extended by also allowing 90-degrees turns along the vertical direction (up and down). Hence, the possible directions of the vehicle are Up-North-East (UNE), Up-North-West (UNW), Up-South-East (USE), Up-South-West (USW), Down-North-East (DNE), Down-North-
West (DNW), Down-South-East (DSE), and Down-South-West (DSW).

\[
\begin{align*}
\dot{x} &= 1 & \dot{x} &= -1 & \dot{x} &= 1 & \dot{x} &= 1 & \dot{x} &= 1 & \dot{x} &= -1 \\
\dot{y} &= -1 & \dot{y} &= -1 & \dot{y} &= 1 & \dot{y} &= -1 & \dot{y} &= -1 & \dot{y} &= 1 \\
\dot{z} &= -1 & \dot{z} &= -1 & \dot{z} &= -1 & \dot{z} &= 1 & \dot{z} &= 1 & \dot{z} &= 1 & \dot{z} &= 1 \\
\dot{t} &= 1 & \dot{t} &= 1 & \dot{t} &= 1 & \dot{t} &= 1 & \dot{t} &= 1 & \dot{t} &= 1 & \dot{t} &= 1
\end{align*}
\]

Table 3: Flows associated to each location in the deterministic FNC.

The LHA modeling the system has one continuous variable for each coordinate position of the vehicle (namely \(x\), \(y\), and \(z\)), and an additional clock variable \(t\) (with \(\dot{t} = 1\)), used to enforce a one-time-unit wait between successive turns. There are 8 locations, one for each possible direction, whose associated flows are shown in Table 3, assigning a deterministic flow to each location.

Similarly to the TNC example, also a non-deterministic version of the FNC has been tested, where disturbances are allowed along the \(z\) dimension. In particular, the differential equation for \(z\) in Table 3 for the locations DSE, DSW, DNE and DNW is replaced by \(-1.5 \leq \dot{z} \leq -0.5\), and for all the other locations it is replaced by \(-0.5 \leq \dot{z} \leq 1.5\).

Table 4 shows the running times of the Local algorithm on instances of the case study with up to 7 obstacles, both for the deterministic and non-deterministic case. Not surprisingly, the results show that increasing the dimensionality of the activities may have a very strong impact on the performances, mainly due to the exponential worst-case complexity of exact polyhedral operations with respect to their number of dimensions.

\[
\begin{array}{cccccccc}
\text{# pits} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
\text{Det} & 2.0 & 29.1 & 70.7 & 168.0 & 295.5 & 498.7 & 664.6 \\
\text{Non-Det} & 2.9 & 56.2 & 225.1 & 529.0 & 1115.5 & 2638.2 & 3259.0 \\
\end{array}
\]

Table 4: Performance (secs.) for FNC w.r.t. the number of pits.

Water Tanks Control. The following example, the Water Tanks Control (WTC), is inspired by a similar example reported in [12]. Let us consider a system with two tanks — A and B — connected by a one-directional valve \(\text{mid}\) (from A to B) and with two additional valves, \(\text{in}\) and \(\text{out}\), placed on A and on B respectively. Valve \(\text{in}\) is used to pour water into A, valve \(\text{out}\) is used to drain water from B, and valve \(\text{mid}\) transfers water from A into B. Valves can be either in an open or in a closed state, and can be operated by a controller. Moreover, when at least two valves are open at the same time for more of two time units, an overload can occur. In this case, the environment takes the system in a special “blocked” mode, where all valves are closed and the controller can not operate. After a delay of at least one and at most two time units, the environment takes the system in a state where all valves still closed, but the controller can resume its role.
The two tanks are placed outdoors, so that the level of the water within is also affected by rain and evaporation. It is possible to change the state of only one valve at a time, and at least one time unit must pass between any two successive operations on the valves. Clearly, the controller may operate on the state of the valves, while the weather is governed by the environment. Figure 12(a) provides a schematic view of the system.

The corresponding hybrid automaton has nine locations: one for each combination of the state (open/closed) of the three valves, plus one location representing the “blocked” mode. In particular, the name of each location indicates the state (O for open and C for closed) of each valve. For instance, location COC models the configuration where the in and out valves are closed and mid valve is open. Location CCCB represents the blocked mode.

The LHA includes three continuous variables: \( x \) and \( y \), modeling the water level in the tanks, and \( t \) modeling the clock used to enforce a one-time-unit wait between consecutive valve operations. Each controllable transition of the LHA can be taken only if \( t \geq 1 \), and resets \( t \) to 0. Instead, the uncontrollable transition that models the overload may be taken only if \( t \geq 2 \), while the uncontrollable transition that models the recovery from the blocked mode can be taken only if \( 1 \leq t \leq 2 \). Since the tanks are in the same geographic location, rain and evaporation are assumed to have the same rate for both tanks.

We set the in (resp., mid, out) flow rate \( p_{\text{in}} \) (resp., \( p_{\text{mid}}, p_{\text{out}} \)) to 2 (resp., 3, 4), the maximum evaporation rate \( e \) to 0.5 and the maximum rain rate \( r \) to 1. Table 5 shows the flow constraints associated to each location. For instance, when all valves are open (location OOO), the evaporation and the valve mid contribute to decreasing the water level \( x \) in tank A, while the rain and the valve in tend to increase it. Considering all these parameters together, the water level \( x \) in the location OOO is subject to the differential constraint

\[-e + p_{\text{in}} - p_{\text{mid}} \leq \dot{x} \leq r + p_{\text{in}} - p_{\text{mid}}.\]

To account for the fact that rain and the evaporation have the same rate for both tanks, the differential constraint for the water level \( y \) in \( B \) depends on \( \dot{x} \). For instance, the differential constraint for the water level \( y \) in location OOO is \( \dot{y} = \dot{x} + 2p_{\text{mid}} - p_{\text{in}} - p_{\text{out}} \). Notice that the resulting differential constraints describe an LHA which is not a rectangular hybrid automaton [15].

Figure 12: The Water Tanks Control.
\[
\begin{array}{|c|c|}
\hline
CCC and CCCB & CCO \\
\hline
-e \leq \dot{x} \leq r & -e \leq \dot{x} \leq r \\
\dot{y} = \dot{x} & \dot{y} = \dot{x} - p_{out} \\
\hline
COC & COO \\
\hline
-e - p_{mid} \leq \dot{x} \leq r - p_{mid} & -e - p_{mid} \leq \dot{x} \leq r - p_{mid} \\
\dot{y} = \dot{x} + 2p_{mid} & \dot{y} = \dot{x} + 2p_{mid} - p_{out} \\
\hline
OCC & COO \\
\hline
-e + p_{in} \leq \dot{x} \leq r + p_{in} & -e + p_{in} \leq \dot{x} \leq r + p_{in} \\
\dot{y} = \dot{x} - p_{in} & \dot{y} = \dot{x} - p_{in} - p_{out} \\
\hline
OOC & OOO \\
\hline
-e + p_{in} - p_{mid} \leq \dot{x} \leq r + p_{in} - p_{mid} & -e + p_{in} - p_{mid} \leq \dot{x} \leq r + p_{in} - p_{mid} \\
\dot{y} = \dot{x} + 2p_{mid} - p_{in} & \dot{y} = \dot{x} + 2p_{mid} - p_{in} - p_{out} \\
\hline
\end{array}
\]

Table 5: Flows associated to each location of WTC. In all locations it holds $\dot{t} = 1$.

As control goal, we require that the water levels be always maintained between 2 and 12 in both tanks.

Figures 13(a) and 13(b) show the resulting safe regions in the case of all valves closed, and in the case of valves in and mid open and valve out closed. Due to the constraint of a one-time-unit delay between controllable transitions, in Figure 13(a) both $x$ and $y$ must initially be in the interval $[2.5, 11]$. Indeed, if the water level was greater than 11 and assuming maximum rain, after one time unit the level will exceed the limit. Similarly, if the water level was less than 2.5 and assuming maximum evaporation, after one unit the level would go below the lower bound. Notice that the region with $x \in [2.5, 6.5]$ and $y \in [2.5, 3]$ is not safe. Indeed, assuming maximal evaporation, the controller needs to open both the in and the mid valves to keep both levels above the lower limit. However, two time units are needed to open both valves and, in the meanwhile, at least one level could go below the lower bound. For a similar reason, the region where $x \in (10, 11]$ and $y \in (7, 11]$ is not safe, assuming maximum rain.

In the case depicted in Figure 13(b) (location OOC), for reasons similar to the previous case, $x$ must initially be in the interval $[3.5, 12]$ and $y$ in the interval $[2, 8]$. Notice that the region identified by the constraints $3.5 \leq x \leq 4.5$, $7 \leq y \leq 8$ and $2x \leq 2y - 7$ is not safe. Indeed, considering this region and assuming maximum evaporation, the controller first needs to close the mid valve, in order to keep $x$ above the lower bound and than it must open the out valve to make sure that $y$ does not exceed the upper bound. However, two time units are needed to perform these operations and, in the meanwhile, tank B could exceed the upper bound. The slope of the line $2x = 2y - 7$ depends on the fact that, when $y$ approaches 8, the controller has less time available to perform the two operations described above. As a consequence, $x$ must be proportionally higher. Vice versa, for similar reasons, when $x$ approaches 3.5, the controller can ensure to remain in the safe region only if $y$ is proportionally lower.

The final result for all the locations is computed by the Local algorithm after
6. Conclusions

In this paper we considered the problem of automatically synthesizing a switching controller for Linear Hybrid Automata with respect to safety objectives. We revisited previous attempts to solve the problem and pinpointed some inaccuracies preventing soundness and completeness of those approaches. The synthesis procedure we propose is based on the $RWA^m$ operator, for which we provided a novel fixpoint characterization and we formally proved termination. To the best of our knowledge, the solution proposed represents the first sound and complete procedure for the task in the literature.

Being the synthesis problem for LHAs at least as hard as the standard reachability problem, the overall synthesis procedure may not terminate. On the other hand, we proved termination of each iteration of the procedure (i.e., termination of $RWA^m$).

We extended the tool PHAVer with our synthesis procedure and discussed the results of a series of experiments, showing that the procedure converges in non trivial cases and that the approach is practical, at least for relatively small case studies. The challenges involved in the symbolic implementation, based on a polyhedral abstraction, are thoroughly discussed. In particular, we propose and compare different approaches to compute the $RWA^m$ operator in practice, and provide a novel algorithm to compute the exact time-elapse operator.

The work presented in this paper paves the way for some interesting future work. For instance, we are currently investigating the problem of automatically constructing a concrete control strategy, which, coupled with the hybrid system,
would result in a closed system, amenable to automatic verification by state-of-the-art analysis tools. The obtained closed system might be verified w.r.t. the safety goal, in order to validate the implementation of the synthesis procedure, and w.r.t. other properties of interest (stability, performance, etc.). Moreover, we are studying the synthesis problem for LHAs w.r.t. reachability objectives, a problem which has never been considered in the literature to date.

References


38


