Towards Efficient Exact Synthesis for Linear Hybrid Systems

Massimo Benerecetti Marco Faella Stefano Minopoli
Università di Napoli
"Federico II", Italy
{mfaella,bene,minopoli}@na.infn.it

We study the problem of automatically computing the controllable region of a Linear Hybrid Automaton, with respect to a safety objective. We describe the techniques that are needed to effectively and efficiently implement a recently-proposed solution procedure, based on polyhedral abstractions of the state space. Supporting experimental results are presented, based on an implementation of the proposed techniques on top of the tool PHAVer.

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 both discrete and continuous. Typically, continuous variables represent physical quantities like temperature, speed, etc., while discrete ones represent *control modes*, i.e., states of the controller.

Hybrid automata [11] are the most common syntactic variety of hybrid system: a finite set of locations, similar to the states of a finite automaton, represents the value of the discrete variables. The current location, together with the current value of the (continuous) variables, form the instantaneous description of the system. Change of location happens via discrete transitions, and the evolution of the variables is governed by differential equations attached to each location. In a Linear Hybrid Automaton (LHA), the allowed differential equations are in fact *polyhedral differential inclusions* of the type $\dot{\mathbf{x}} \in P$, where $\dot{\mathbf{x}}$ is the vector of the first derivatives of all variables and P is a convex polyhedron. Notice that differential inclusions are non-deterministic, allowing for infinitely many solutions.

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*: 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 at any moment.

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. In [6], we fixed some inaccuracies in previous presentations, and proposed a sound and complete semi-procedure for the problem. Here, we discuss the techniques required to efficiently implement the algorithms in [6]. 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 $\dot{\mathbf{x}} \in P$.

- 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.

To appear in EPTCS.

- The *may reach while avoiding* operator, denoted by 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. A fixpoint algorithm for this operator was presented in [6]. Here, we introduce a number of efficiency improvements (Section 5), accompanied by a corresponding experimental evaluation (Section 6), carried out on our tool PHAVer+, based on the open-source tool PHAVer [9].

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 research effort on approximate techniques should be based on the solid grounds provided by the exact approach. For instance, a tool implementing an exact algorithm (like our PHAVer+) may serve as a benchmark to evaluate the performance and the precision of an approximate tool.

Related work. The idea of automatically synthesizing controllers for dynamic systems first arose in connection with discrete systems [17]. Then, the same idea was applied to real-time systems modeled by timed automata [16], thus coming one step closer to the continuous systems that control theory usually deals with. Finally, it was the turn of hybrid systems [20, 13], and in particular of LHA, the very model that we analyze in this paper. Wong-Toi proposed the first symbolic semi-procedure to compute the controllable region of a LHA w.r.t. a safety goal [20]. The heart of the procedure lies in the operator $\mathit{flow\_avoid}(U,V)$, which is analogous to our RWA^{m}. However, the algorithm provided in [20] 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 revised algorithm, correcting such flaw, was proposed in [6].

Tomlin et al. and Balluchi et al. analyze much more expressive models [18, 5], with generality in mind rather than automatic synthesis. Their *Reach* and *Unavoid_Pre* operators, respectively, again correspond to RWA^{m}.

Asarin et al. investigate the synthesis problem for hybrid systems where all discrete transitions are controllable and the trajectories satisfy given linear differential equations of the type $\dot{\mathbf{x}} = A\mathbf{x}$ [2]. The expressive power of these constraints is incomparable with the one offered by the differential inclusions occurring in LHAs. In particular, linear differential equations give rise to deterministic trajectories, while differential inclusions are non-deterministic. In control theory terms, differential inclusions can represent the presence of environmental *disturbances*. The tool `d/dt` [3], by the same authors, is reported to support controller synthesis for safety objectives, but the publicly available version in fact does not.

2 Linear Hybrid Automata

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

Given an ordered set $X = \{x_1, \ldots, x_n\}$ of variables, a *valuation* is a function $v : X \to \mathbb{R}$. Let $\mathit{Val}(X)$ denote the set of valuations over X. There is an obvious bijection between $\mathit{Val}(X)$ and $\mathbb{R}^n$, allowing us to extend the notion of (convex) polyhedron to sets of valuations. We denote by $\mathit{CPoly}(X)$ (resp., $\mathit{Poly}(X)$) the set of convex polyhedra (resp., polyhedra) on 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 transition. Arithmetic operations on valuations are defined in the straightforward way. An *activity* over X is a differentiable function $f : \mathbb{R}^{\geq 0} \to \mathit{Val}(X)$. Let $\mathit{Acts}(X)$ denote the set of activities over X. The

derivative $\dot{f}$ of an activity f is defined in the standard way and it is an activity over $\dot{X}$. A *Linear Hybrid Automaton* $H = (Loc, X, Edg_c, Edg_u, Flow, Inv, Init)$ consists of the following:

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

We use the abbreviations $S = Loc \times Val(X)$ for the set of states and $Edg = Edg_c \cup Edg_u$ for the set of all transitions. Moreover, we let $InvS = \bigcup_{l \in Loc} \{l\} \times Inv(l)$ and $InitS = \bigcup_{l \in Loc} \{l\} \times Init(l)$. Notice that $InvS$ and $InitS$ are sets of states.

2.1 Semantics

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

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

Runs. Given two states s, s', and a transition $e \in Edg$, there is a *discrete transition* $s \xrightarrow{e} s'$ with *source* s and *target* s' iff *(i)* $s, s' \in Invs$, *(ii)* $e = (loc(s), \mu, loc(s'))$, and *(iii)* $(val(s), val(s')') \in \mu$, where $val(s')'$ is the valuation over X' obtained from $val(s')$ by renaming each variable $x \in X$ onto the corresponding primed variable $x' \in X$. There is a *timed transition* $s \xrightarrow{\delta, f} s'$ with *duration* $\delta \in \mathbb{R}^{\geq 0}$ and activity $f \in Adm(s)$ iff *(i)* $s \in Invs$, *(ii)* $\delta \in span(f, loc(s))$, and *(iii)* $s' = \langle loc(s), f(\delta) \rangle$. For technical convenience, we admit timed transitions of duration zero[1]. A special timed transition is denoted $s \xrightarrow{\infty, f}$ and represents the case when the system follows an activity forever. This is only allowed if $\infty \in span(f, loc(s))$. Finally, a *joint transition* $s \xrightarrow{\delta, f, e} s'$ represents the timed transition $s \xrightarrow{\delta, f} \langle loc(s), f(\delta) \rangle$ followed by the discrete transition $\langle loc(s), f(\delta) \rangle \xrightarrow{e} s'$.

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

A *run* is a sequence

$$r = s_0 \xrightarrow{\delta_0, f_0} s'_0 \xrightarrow{e_0} s_1 \xrightarrow{\delta_1, f_1} s'_1 \xrightarrow{e_1} s_2 \dots s_n \dots \tag{1}$$

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

Zenoness and well-formedness. A well-known problem of real-time and hybrid systems is that definitions like the above admit runs that take infinitely many discrete transitions in a finite amount of time (i.e., *Zeno* runs), even if such behaviors are physically meaningless. In this paper, we assume that the hybrid automaton under consideration generates no such runs. This is easily achieved by using an extra variable, representing a clock, to ensure that the delay between any two transitions is bounded from below by a constant. We leave it to future work to combine our results with more sophisticated approaches to Zenoness known in the literature [5, 1].

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. 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.

Strategies. A *strategy* is a function $\sigma : S \to 2^{Edg_c \cup \{\perp\}} \setminus \emptyset$, where $\perp$ 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. If all activities immediately exit the invariant, the above condition is vacuously satisfied. This ensures that the null action is enabled in right-open regions, so that there is an earliest instant in which a controllable transition becomes mandatory.

Notice that a strategy can always choose the null action. 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 σ if for all $0 \leq i < len(r)$ the following conditions hold:

- for all $\delta \geq 0$ such that $\sum_{j=0}^{i-1} \delta_j \leq \delta < \sum_{j=0}^{i} \delta_j$, we have $\perp \in \sigma(\langle loc(s_i), f_i(\delta - \sum_{j=0}^{i-1} \delta_j) \rangle)$;
- if $e_i \in Edg_c$ then $e_i \in \sigma(s'_i)$.

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

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

3 Solving the Safety Control Problem

In this section, we recall the semi-procedure that solves the safety control problem for a given LHA and safe region. It is well known in the literature (see e.g. [15, 2]) that the answer to the safety control problem for safe set $T \subseteq Inv$ is positive if and only if

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

where $CPre$ is the *controllable predecessor operator*, defined below. Since the reachability problem for LHA was proved undecidable [14], the above fixpoint may not converge in a finite number of steps. On the other hand, it does converge in many cases of practical interest, as witnessed by the examples in Section 6.

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

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

In order to compute $CPre(A)$ on LHA, the auxiliary operator RWA^{m} (*may reach while avoiding*) was proposed [6]. Intuitively, given a location l and two sets of variable valuations U and V, $RWA^{\mathrm{m}}_l(U,V)$ contains the set of valuations from which the continuous evolution of the system *may* reach U while avoiding $V \cap \overline{U}$.

For a set of states A and $x \in \{u,c\}$, let $Pre^{\mathrm{m}}_{\mathrm{x}}(A)$ (for *may predecessors*) be the set of states where some discrete transition leading to A and belonging to Edg_x is enabled. We denote with $A\lfloor_l$ the projection of A on l, i.e. $\{v \in Val(X) \mid \langle l, v\rangle \in A\}$. As proved in [6], we then have that

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

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

Intuitively, the set B_l is the set of valuations u such that from state $\langle l,u\rangle$ the environment can take a discrete transition leading outside A, and C_l is the set of valuations u such that from $\langle l,u\rangle$ the controller can take a discrete transition into A. Then, using the RWA^{m} operator, we compute the set of valuations from which there exists an activity that either leaves A or enters B_l, while staying in the invariant and avoiding C_l. These valuations do not belong to $CPre(A)$, as the environment can violate the safety goal within (at most) one discrete transition.

Next, we show how to characterize RWA^{m} in terms of simple operations on polyhedra. Let $cl(P)$ denote the topological closure of a polyhedron P. Given two polyhedra P and F, the *pre-flow* of P w.r.t. F is:

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

For a given location $l \in Loc$, the pre-flow of P w.r.t. $Flow(l)$ is the set of points that can reach P via a straight-line activity whose slope is allowed in l. For notational convenience, we use the abbreviation $P \swarrow_l$ for $P \swarrow Flow(l)$, and for all polyhedra P and P' we define their *boundary* to be

$$bndry(P,P') = (cl(P) \cap P') \cup (P \cap cl(P')),$$

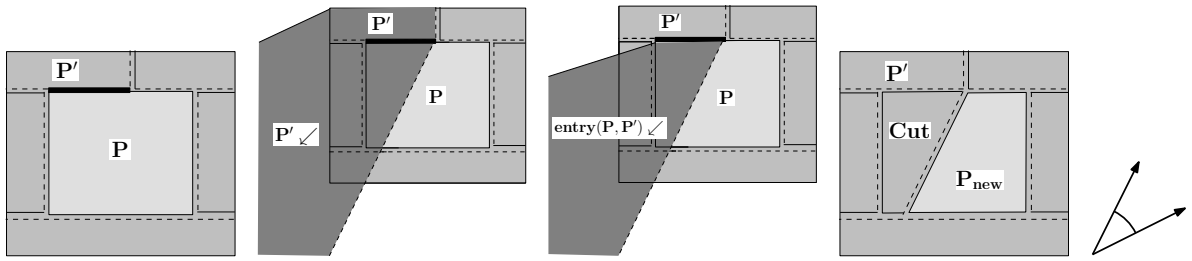
which identifies a boundary between two (not necessarily closed) convex polyhedra. Clearly, $bndry(P,P')$ is not empty only if P and P' are adjacent to one another or if they overlap; it is empty, otherwise. Moreover, given a location l, $entry(P,P')$, the *entry region* between P and P', denotes the set of points of the boundary between P and P' which can reach P' by following some straight-line activity in location l. In symbols: $entry(P,P') = bndry(P,P') \cap P'{\swarrow_l}$. The following theorem gives a fixpoint characterization of RWA^{m}.

Theorem 1 ([6]) *For all locations l and polyhedra U, V, it holds*

$$RWA^{\mathrm{m}}_l(U,V) = \mu W\,.\,U \cup \bigcup_{P\in[\![\overline{V}]\!]} \bigcup_{P'\in[\![W]\!]} \Big(P \cap entry(P,P'){\swarrow_l}\Big). \tag{2}$$

The equation refines the under-approximation U 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). Figure 1 shows a single step in the computation of equation 2, for a fixed pair of convex polyhedra P in $\overline{V}$ and P' in W. Dashed lines represent topologically open sides. The dark gray rectangles represent convex polyhedra in W, while the light gray one is P.

In Figure 1(a) the thick segment between P and P' represents $bndry(P,P')$ and, in the example, is contained in P. Since P' is topologically open (denoted by the dashed line), the rightmost point of $bndry(P,P')$ cannot reach P' along any straight-line activity. Being P' open, so is $P'{\swarrow_l}$, and its intersection with P, namely $entry(P,P')$, does not contain the rightmost point of the boundary (Figure 1(b)). Now, any point of P that can reach $entry(P,P')$ following some activity can also reach P', and the set $Cut = P \cap entry(P,P'){\swarrow_l}$ contains precisely those points (Figure 1(c) and Figure 1(d)). All these points must then be added to W, as they all belong to $RWA^{\mathrm{m}}_l(U,V)$.



(a) Initial input, with $bndry(P,P')$ highlighted.

(b) Pre-flow of P'.

(c) Entry region.

(d) P_{new}, Cut.

(e) $Flow(l)$.

Figure 1: Algorithm behavior.

In our implementation, instead of computing the operator RWA^{m}_l, we compute the dual operator $SOR^{\mathrm{M}}_l(Z,V)$ (for *must stay or reach*), containing the points which either remain in Z forever or reach V along a system trajectory that does not leave Z. The operator SOR^{M}_l can be defined as follows:

$$SOR^{\mathrm{M}}_l(Z,V) = \overline{RWA^{\mathrm{m}}_l(\overline{Z},V)}. \tag{3}$$

As a consequence, we can compute $CPre(A)$ as

$$\bigcup_{l\in Loc} \{l\} \times \Big(A\lfloor_l \cap SOR^{\mathrm{M}}_l\big(\overline{Inv}\lfloor_l \cup \big(A\lfloor_l \setminus B_l\big), C_l \cup \overline{Inv\lfloor_l}\big)\Big).$$

From (3), we obtain a fixpoint characterization of the operator SOR^{M}_l:

$$SOR^{\mathrm{M}}_l(Z,V) = \overline{RWA^{\mathrm{m}}_l(\overline{Z},V)} = \overline{\mu W\,.\,\overline{Z} \cup \bigcup_{P\in[\![\overline{V}]\!]} \bigcup_{P'\in[\![W]\!]} \left(P \cap entry(P,P')_{\swarrow_l}\right)} =$$

$$= \nu W\,.\,Z \cap \overline{\bigcup_{P\in[\![\overline{V}]\!]} \bigcup_{P'\in[\![\overline{W}]\!]} \left(P \cap entry(P,P')_{\swarrow_l}\right)} = \nu W\,.\,Z \setminus \bigcup_{P\in[\![\overline{V}]\!]} \bigcup_{P'\in[\![\overline{W}]\!]} \left(P \cap entry(P,P')_{\swarrow_l}\right). \quad (4)$$

The following two sections show how to effectively and efficiently compute fixpoint (4).

4 Exact Computation of Pre-Flow

As seen in the previous section, one of the basic operations on polyhedra that are needed to compute SOR^{M} is the pre-flow operator $\swarrow$. It is sufficient to compute $P\swarrow F$ for convex P and F, for two reasons: First, we always have $F = Flow(l)$, for a given location l, and $Flow(l)$ is a convex polyhedron by assumption. Second, $(P_1 \cup P_2)\swarrow F = (P_1\swarrow F) \cup (P_2\swarrow F)$, 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:

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

The post-flow operation coincides with the *time-elapse* operation introduced in [10] 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 $P\nearrow F = \{(0,0)\}\cup\{(x,y)\in\mathbb{R}^2 \mid y>0\}$, 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 $P\nearrow_{\mathrm{PPL}} F$ of the post-flow $P\nearrow F$, as the smallest convex polyhedron containing $P\nearrow F$.

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

$$P\nearrow F = P \cup \left(P\nearrow_{>0} F\right),$$

where $P\nearrow_{>0} F$ 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,

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

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 [7, 19]. 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

$$\sum_{v\in V}\alpha_v\cdot v + \sum_{c\in C}\beta_c\cdot c + \sum_{r\in R}\gamma_r\cdot r, \quad (5)$$

where all coefficients α_v, β_c and γ_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 2 *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 $P\nearrow_{>0}F$, where $\oplus$ denotes Minkowski sum.*

Proof Let $z\in P\nearrow_{>0}F$, we show that there are coefficients α_v, β_c and γ_r such that z can be written as (5), for $V=V_P\oplus V_F$, $C=C_P\cup V_P$, and $R=R_P\cup V_F\cup C_F\cup R_F$.

By definition, there exist $x\in P$, $y\in F$, and $\delta>0$ such that $z=x+\delta y$. Hence, there are coefficients α_v^x, β_c^x, and γ_r^x witnessing the fact that $x\in P$, and coefficients α_v^y, β_c^y, and γ_r^y witnessing the fact that $y\in F$. Moreover, there is $i\in V_P$ and $j\in V_F$ such that $\alpha_i^x>0$ and $\alpha_j^y>0$. Let $\varepsilon=\min\{\alpha_i^x,\delta\alpha_j^y\}$ and notice that $\varepsilon>0$. It holds

$$\alpha_i^x\cdot i+\delta\cdot\alpha_j^y\cdot j=(\alpha_i^x-\varepsilon)i+\varepsilon i+(\delta\cdot\alpha_j^y-\varepsilon)j+\varepsilon j=\varepsilon(i+j)+(\alpha_i^x-\varepsilon)i+(\delta\cdot\alpha_j^y-\varepsilon)j.$$

Hence,

$$\begin{aligned}z&=\sum_{v\in V_P}\alpha_v^x\cdot v+\sum_{c\in C_P}\beta_c^x\cdot c+\sum_{r\in R_P}\gamma_r^x\cdot r+\quad\delta\left(\sum_{v\in V_F}\alpha_v^y\cdot v+\sum_{c\in C_F}\beta_c^y\cdot c+\sum_{r\in R_F}\gamma_r^y\cdot r\right)\\&=\varepsilon(i+j)+\quad\left((\alpha_i^x-\varepsilon)i+\sum_{v\in V_P\setminus\{i\}}\alpha_v^x\cdot v+\sum_{c\in C_P}\beta_c^x\cdot c\right)+\\&\quad\left((\delta\cdot\alpha_j^y-\varepsilon)j+\sum_{r\in R_P}\gamma_r^x\cdot r+\delta\sum_{v\in V_F\setminus\{j\}}\alpha_v^y\cdot v+\delta\sum_{c\in C_F}\beta_c^y\cdot c+\delta\sum_{r\in R_F}\gamma_r^y\cdot r\right).\end{aligned}$$

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 $i+j$, such that its coefficient is strictly positive.

Conversely, let z be a point that can be expressed as (5), 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 $z\in P\nearrow_{>0}F$ 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 $\alpha_{v^*}>0$. We set

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

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

$$\delta=\sum_{v\in V_P\oplus V_F}\alpha_v+\sum_{r\in V_F\cup C_F}\gamma_r,\qquad\text{and}\qquad y=\frac{1}{\delta}\cdot\left(\sum_{\substack{v_1\in V_P\\v_2\in V_F}}\alpha_{v_1+v_2}\cdot v_2+\sum_{r\in V_F\cup C_F\cup R_F}\gamma_r\cdot r\right).$$

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 δ; finally, since $\alpha_{v^*} > 0$, there is a point in V_F whose coefficient is positive.

5 Computing SOR^{M}

In this section, we show how to efficiently compute $SOR^{\mathrm{M}}_l(Z,V)$, given two polyhedra Z and V. Fixpoint equation (4) can easily be converted into an iterative algorithm, consisting in generating a (potentially infinite) sequence of polyhedra $(W_n)_{n\in\mathbb{N}}$, where $W_0 = Z$ and

$$W_{i+1} = W_i \setminus \bigcup_{P\in[\![\overline{V}]\!]} \bigcup_{P'\in[\![\overline{W_i}]\!]} \Big(P \cap entry(P,P')_{\swarrow_l}\Big). \tag{6}$$

Theorem 4 in [6] proves 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 $P \in [\![\overline{V}]\!]$ and an inner loop over $P' \in [\![\overline{W_i}]\!]$. As a first improvement, we notice that each iteration of the outer loop removes from W_i a portion of $P \in [\![\overline{V}]\!]$. Hence, the portion of P that is not contained in W_i is irrelevant, and we may replace (6) with:

$$W_{i+1} = W_i \setminus \bigcup_{P\in[\![W_i\cap\overline{V}]\!]} \bigcup_{P'\in[\![\overline{W_i}]\!]} \Big(P \cap entry(P,P')_{\swarrow_l}\Big). \tag{7}$$

Moreover, we can avoid the need to intersect W_i with $\overline{V}$ at each iteration, by starting with $W'_0 = Z \setminus V$, setting:

$$W'_{i+1} = W'_i \setminus \bigcup_{P\in[\![W'_i]\!]} \bigcup_{P'\in[\![\overline{W'_i}]\!]} \Big(P \cap entry(P,P')_{\swarrow_l}\Big), \tag{8}$$

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

5.1 Introducing Adjacency Relations

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

$$Ext_i = \big\{\langle P, entry(P,P')\rangle \mid P \in [\![W_i]\!], P' \in [\![\overline{W}_i]\!], \text{ and } entry(P,P') \neq \emptyset\big\}. \tag{9}$$

Once 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 Ext_i, i.e., all P such that $\langle P,R\rangle \in Ext_i$ for some R. Summarizing, using Ext_i we can replace (8) with

$$W_{i+1} = W_i \setminus \bigcup_{\langle P,R\rangle\in Ext_i} \Big(P \cap R_{\swarrow_l}\Big). \tag{10}$$

Clearly, some extra effort is required to initialize and maintain Ext_i. Initialization is performed by simply applying (9). Regarding maintenance, we briefly discuss how to efficiently compute Ext_{i+1}.

Algorithm 1: $SOR^{\mathrm{M}}(Z,V,F)$

Input: *Poly* Z, V, *CPoly* F
Output: *Poly* $SOR^{\mathrm{M}}(Z,V,F)$
foreach *CPoly* $P \in [\![Z]\!]$ **do**
  $Int_{new} \leftarrow UpdInt(Int_{new}, P, Z)$;
  $E \leftarrow PotentialEntry(P, Int_{new}, F)$;
  $Ext_{new} \leftarrow UpdExt(Ext_{new}, P, E, F, V)$;
while $Ext_{new} \neq \emptyset$ **do**
  $Ext_{old} \leftarrow Ext_{new}$;
  $Int_{old} \leftarrow Int_{new}$;
  $Ext_{new} \leftarrow \emptyset$;
  foreach P *s.t.* $\langle P, R\rangle \in Ext_{old}$ **do**
    $B \leftarrow \bigcup \{R \mid \langle P, R\rangle \in Ext_i\}$;
    $Cut \leftarrow P \cap (B_{\swarrow_l})$;
    if $Cut \neq \emptyset$ **then**
      $P_{new} \leftarrow P \setminus Cut$;
      foreach $P' \in [\![P_{new}]\!]$ **do**
        $Int_{new} \leftarrow UpdInt(Int_{new}, P', P_{new})$;
      foreach P' *s.t.* $\langle P, P'\rangle \in Int_{old}$ **do**
        $Int_{new} \leftarrow UpdInt(Int_{new}, P', P_{new})$;
        $Ext_{new} \leftarrow UpdExt(Ext_{new}, P', Cut, F, V)$;
      $Int_{new} \leftarrow Int_{new} \setminus \{\langle P, Q\rangle \in Int_{old}\}$;
return $\{P \mid \langle P, P'\rangle \in Int_{new}\}$;

Algorithm 2: $UpdInt(Int, P, Candidates)$

Input: Set of *CPoly* pairs *Int*; *CPoly* P;
  Poly Candidates;
Output: Set of *CPoly* pairs *Int*;
$Int \leftarrow Int \cup \{\langle P, \emptyset\rangle\}$;
foreach *CPoly* $P' \in [\![Candidates]\!]$, *with* $P' \neq P$ **do**
  if $bndry(P, P') \neq \emptyset$ **then**
    $Int \leftarrow Int \cup \{\langle P, P'\rangle\}$;
return *Int*;

Algorithm 3: $UpdExt(Ext, P, Candidates, F, V)$

Input: Set of *CPoly* pairs *Ext*; *CPoly* P, F;
  Poly Candidates, V;
Output: Set of *CPoly* pairs *Ext*;
if $P \not\subseteq V$ **then**
  foreach *CPoly* $P' \in [\![Candidates]\!]$ **do**
    $R \leftarrow entry(P, P')$;
    if $R \neq \emptyset$ **then**
      $Ext \leftarrow Ext \cup \{\langle P, R\rangle\}$;
return *Ext*;

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 $P' \in [\![\overline{W}_i]\!]$. 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 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 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* Int_i between polyhedra in $[\![W_i]\!]$:

$$Int_i = \big\{\langle P_1, P_2\rangle \mid P_1, P_2 \in [\![W_i]\!], P_1 \neq P_2 \text{ and } bndry(P_1, P_2) \neq \emptyset\big\}. \tag{11}$$

The computation of Int_0 requires the complete scan of all $P_1, P_2 \in [\![W_0]\!]$, while Int_{i+1} is obtained incrementally from Int_i and Ext_i. Given $\langle P, R\rangle \in Ext_i$, let $Cut = P \cap \big(R_{\swarrow_l}\big)$ and $P_{new} = P \setminus Cut$. Notice that P_{new} may be non-convex, being the result of a set-theoretical difference between two convex polyhedra. To obtain Int_{i+1}, we add to Int_i the pairs of adjacent convex polyhedra (P_1, P_2) such that either *(i)* both P_1 and P_2 belong to $[\![P_{new}]\!]$, or *(ii)* one of them belongs to $[\![P_{new}]\!]$ and the other is adjacent to P according to Int_i. Moreover, once P_{new} replaces P in W_{i+1}, it is necessary to remove all the pairs $\langle P, P'\rangle$ from Ext_i and Int_i.

Algorithms 1-3 represent a concrete implementation of the technique described so far. In Algorithm 1, Ext_{old} and Int_{old} 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 *Candidates* 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. In this version, we simply have

$$PotentialEntry(P, Int_0, F) = \bar{Z}.$$

This will be improved in Section 5.2.

5.2 Further Improving the Performance

Recall that $PotentialEntry(P, Int_0, F)$ returns $\bar{Z}$, regardless of its inputs. Experimental evidence (see Section 6.2) shows that it is often the case that the portion of $\bar{Z}$ which is relevant to computing the entry regions of a given a convex polyhedron 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, for each P in $[\![Z]\!]$ we proceed as follows. We first collect P and all convex polyhedra in $[\![Z]\!]$ that are adjacent to it: $P_{adj} = \{P\} \cup \{P' \mid \langle P, P' \rangle \in Int_0\}$. Then, we compute

$$PotentialEntry(P, Int_0, F) = (P \nearrow F) \setminus P_{adj}.$$

The resulting polyhedron contains all and only the convex polyhedra of $\bar{Z}$ which, if adjacent to P, give rise to a non-empty entry region.

6 Experiments with PHAVer+

We implemented the three algorithms described in the previous section on the top of the open-source tool PHAVer [9]. In the following figures, the basic approach (Section 5) is denoted by *Basic*, the adjacency approach (Section 5.1) by *Adj*, and the local adjacency approach (Section 5.2) by *Local*. We show some results obtained by testing our package on two different examples: the Truck Navigation Control (TNC) and the Water Tanks Control (WTC). The experiments are divided into two distinct categories: the *macro* analysis shows the performance of the three implementations when solving safety control problems, while the *micro* analysis shows the performances of a single call to the $SOR^{M}_{l}(Z, V)$ operator. A binary pre-release of our implementation, that we call PHAVer+, can be downloaded at `http://people.na.infn.it/mfaella/phaverplus`. The experiments were performed on an Intel Xeon (2.80GHz) PC.

6.1 Macro Analysis

We now describe in detail the two examples used to evaluate the performance of our package.

Truck Navigation Control. This example is derived from [8], where the tool HONEYTECH is presented, as an extension of HYTECH [12] for the automatic synthesis of controllers. Consider an autonomous toy truck, which is responsible for avoiding some 2 by 1 rectangular pits. The truck can take 90-degree left or right turns: the possible directions are North-East (NE), North-West (NW), South-East (SE) and South-West (SW). One time unit must pass between two changes of direction. The control goal

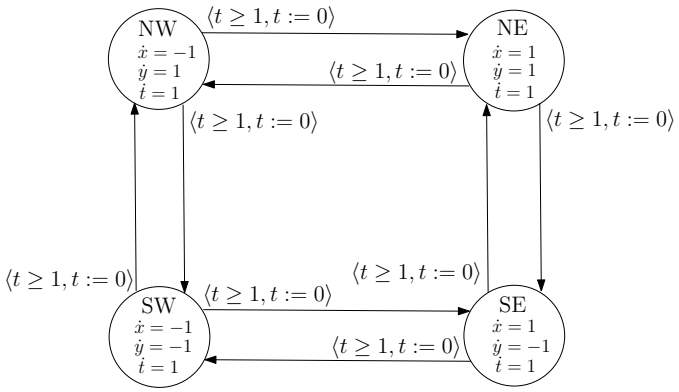
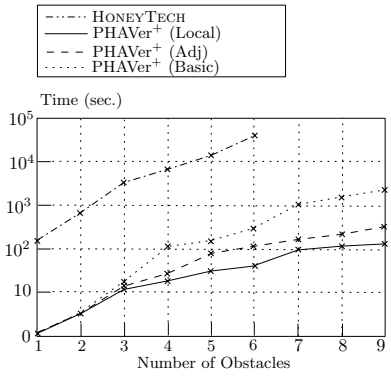


Figure 2: TNC modeled as a Hybrid Automaton.

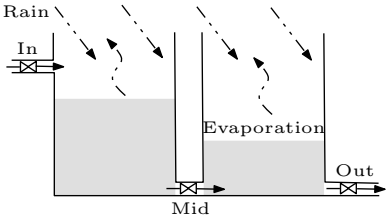
consists in avoiding the pits. Figure 2 shows the hybrid automaton modeling the system: there is one location for each direction, where the derivative of the position variables (x and y) are set according to the corresponding direction. The variable t represents a clock ($\dot{t} = 1$) that enforces a one-time-unit wait between turns.

We tested our implementations on progressively more complex control goals, by increasing the number of obstacles. Figure 3(a) compares the performance of the three implementations of the algorithm (solid line for local, dashed line for adjacency, dotted line for basic and dotted-dashed line for the performance reported in [8]). We were not able to replicate the experiments in [8], since HONEYTECH is not publicly available. Notice that the time axis is logarithmic.

Because of the different hardware used, only a qualitative comparison can be made between our implementations and HONEYTECH: going from 1 to 6 obstacles (as the case study in [8]), the run time of HONEYTECH shows an exponential behavior, while our best implementation exhibits an approximately linear growth, as shown in Figure 3(a), where the performance of PHAVer+ is plotted up to 9 obstacles.



(a) Performance for TNC.



(b) System schema for WTC.

Algorithm	Time (sec.)
Basic	21.0
Adj	16.2
Local	9.3

(c) Performance for WTC.

Figure 3: Schema and performance for the two examples.

Water Tank Control. Consider the system depicted in Figure 3(b), where two tanks — A and B — are linked by a one-directional valve *mid* (from A to B). There are two additional valves: the valve *in* to fill A and the valve *out* to drain B. The two tanks are open-air: the level of the water inside also depends on the potential rain and evaporation. It is possible to change the state of one valve only after one second since the last valve operation.

The corresponding hybrid automaton has eight locations, one for each combination of the state (open/closed) of the three valves, and three variables: x and y for the water level in the tanks, and t as the clock that enforces a one-time-unit wait between consecutive discrete transitions. Since the tanks are in the same geographic location, rain and evaporation are assumed to have the same rate in both tanks, thus leading to a proper LHA that is not rectangular [13].

We set the *in* and *mid* flow rate to 1, the *out* flow rate to 3, the maximum evaporation rate to 0.5 and maximum rain rate to 1, and solve the synthesis problem for the safety specification requiring the water levels to be between 0 and 8. Figure 3(c) shows the run time of the three versions of the algorithm on WTC.

6.2 Micro Analysis

In this subsection we show the behavior of individual calls to $SOR^{\mathrm{M}}_{l}(Z,V)$, implemented in the three different ways described in Section 5. The evaluation of the efficiency of the three versions is carried out based on the number of comparisons that the three algorithms perform in order to identify the boundaries between polyhedra in Z and polyhedra in *PotentialEntry*, with respect to the size of the input. We choose to highlight the number of computed boundaries because the idea that led us to the realization of the final version of the algorithm is precisely to avoid unnecessary adjacency checks.

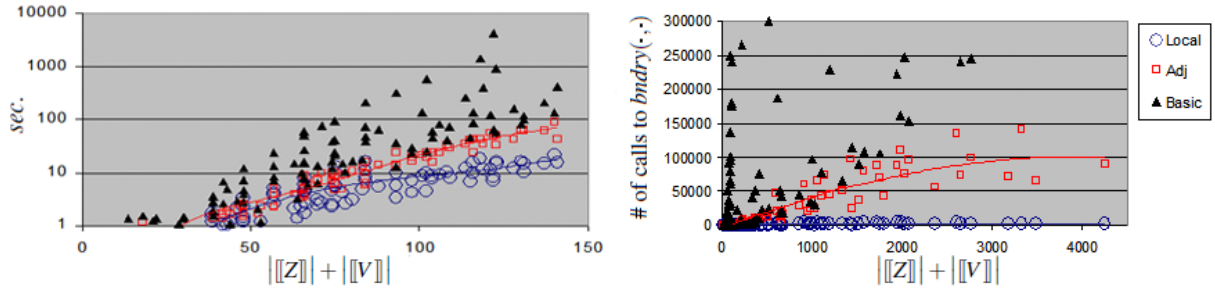


Figure 4: Run time (in sec.) and number of boundary checks of the three algorithms for SOR^{M} w.r.t. the size of the input.

Figure 4 shows the run time and the number of boundary computations made by the three approaches. As expected, the number of calls made by the basic algorithm is higher than those made by the adjacency approach, which in turn is higher then those made by the local adjacency algorithm. This is reflected in the execution times of the three procedures. One also notices a certain instability in the case of the basic algorithm, due to the fact that in some instances of the problem, even with small inputs, the algorithm can cut an individual polyhedron in many parts: this dramatically increases the size of the sets Z and $\bar{Z}$ in the next steps and consequently the number of comparisons required. This instability is held much more under control with the introduction of the adjacency relations. Note that in the local version the number of comparisons required is much lower: we can easily explain this fact, recalling that *PotentialEntry* in

the adjacency version returns the whole $\bar{Z}$, forcing Algorithm 3 to perform $|\bar{Z}|$ iterations of its "foreach" loop.

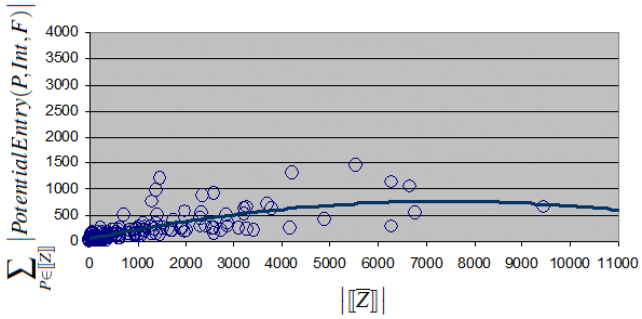


Figure 5: Size of *PotentialEntry* in the *Adj* and the *Local* algorithms.

Figure 5 shows, for the same inputs, the relationship between the size of *PotentialEntry* in the basic and in the adjacency versions (i.e., $\bar{Z}$) and in the local version: the ratio is 1 to 10, which reduces drastically the number of checks, and consequently the overall run time.

References

[1] L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar & M. Stoelinga. (2003): *The Element of Surprise in Timed Games*. In: *CONCUR 03: Concurrency Theory. 14th Int. Conf., Lect. Notes in Comp. Sci.* 2761, Springer, pp. 144–158, doi:10.1007/978-3-540-45187-7_9.

[2] E. Asarin, O. Bournez, T. Dang, O. Maler & A. Pnueli (2000): *Effective synthesis of switching controllers for linear systems*. *Proceedings of the IEEE* 88(7), pp. 1011 –1025, doi:10.1109/5.871306.

[3] E. Asarin, T. Dang & O. Maler (2002): *The d/dt Tool for Verification of Hybrid Systems*. In: *Computer Aided Verification, Lecture Notes in Computer Science* 2404, Springer, pp. 746–770, doi:10.1007/3-540-45657-0_30.

[4] R. Bagnara, P. M. Hill & E. Zaffanella (2008): *The Parma Polyhedra Library: Toward a Complete Set of Numerical Abstractions for the Analysis and Verification of Hardware and Software Systems*. *Science of Computer Programming* 72(1–2), pp. 3–21, doi:10.1016/j.scico.2007.08.001.

[5] A. Balluchi, L. Benvenuti, T. Villa, H. Wong-Toi & A. Sangiovanni-Vincentelli (2003): *Controller synthesis for hybrid systems with a lower bound on event separation*. *Int. J. of Control* 76(12), pp. 1171–1200, doi:10.1080/0020717031000123616.

[6] M. Benerecetti, M. Faella & S. Minopoli (2011): *Automatic Synthesis of Switching Controllers for Linear Hybrid Automata*. Technical Report, Università di Napoli "Federico II". Available on arXiv. Submitted for publication.

[7] N. V. Chernikova (1968): *Algorithm for discovering the set of all the solutions of a linear programming problem*. *USSR Computational Mathematics and Mathematical Physics* 8(6), pp. 282–293, doi:10.1016/0041-5553(68)90115-8.

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

[9] G. Frehse (2005): *PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech*. In: *Proc. of Hybrid Systems: Computation and Control (HSCC), 8th International Workshop*, *Lect. Notes in Comp. Sci.* 3414, Springer, pp. 258–273, doi:10.1007/978-3-540-31954-2_17.

[10] N. Halbwachs, Y.-E. Proy & P. Roumanoff (1997): *Verification of Real-Time Systems using Linear Relation Analysis*. *Formal Methods in System Design* 11, pp. 157–185, doi:10.1023/A:1008678014487.

[11] T.A. Henzinger (1996): *The Theory of Hybrid Automata*. In: *Proc. 11th IEEE Symp. Logic in Comp. Sci.*, pp. 278–292, doi:0.1109/LICS.1996.561342.

[12] T.A. Henzinger, P.-H. Ho & H. Wong-Toi (1997): *HyTech: A Model Checker for Hybrid Systems*. *Software Tools for Tech. Transfer* 1, pp. 110–122, doi:10.1007/s100090050008.

[13] T.A. Henzinger, B. Horowitz & R. Majumdar (1999): *Rectangular Hybrid Games*. In: *CONCUR 99: Concurrency Theory. 10th Int. Conf.*, *Lect. Notes in Comp. Sci.* 1664, Springer, pp. 320–335, doi:10.1007/3-540-48320-9_23.

[14] T.A. Henzinger, P.W. Kopke, A. Puri & P. Varaiya (1998): *What's Decidable about Hybrid Automata?* *J. of Computer and System Sciences* 57(1), pp. 94 – 124, doi:10.1006/jcss.1998.1581.

[15] O. Maler (2002): *Control from computer science*. *Annual Reviews in Control* 26(2), pp. 175–187, doi:10.1016/S1367-5788(02)00030-5.

[16] O. Maler, A. Pnueli & J. Sifakis (1995): *On the Synthesis of Discrete Controllers for Timed Systems*. In: *Proc. of 12th Annual Symp. on Theor. Asp. of Comp. Sci.*, *Lect. Notes in Comp. Sci.* 900, Springer, doi:10.1007/3-540-59042-0_76.

[17] P.J. Ramadge & W.M. Wonham (1987): *Supervisory Control of a Class of Discrete-Event Processes*. *SIAM Journal of Control and Optimization* 25, pp. 206–230, doi:10.1137/0325013.

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

[19] H. Le Verge (1992): *A note on Chernikova's Algorithm*. Technical Report 635, IRISA, Rennes.

[20] H. Wong-Toi (1997): *The synthesis of controllers for linear hybrid automata*. In: *Proc. of the 36th IEEE Conf. on Decision and Control*, IEEE Computer Society Press, San Diego, CA, pp. 4607 – 4612, doi:10.1109/CDC.1997.649708.