Non-convex Invariants and Urgency Conditions on Linear Hybrid Automata

Stefano Minopoli and Goran Frehse

VERIMAG, Centre Équation - 2, avenue de Vignate, 38610 GIÉRES, France
{stefano.minopoli,goran.frehse}@imag.fr

Abstract. Linear hybrid automata (LHAs) are of particular interest to formal verification because sets of successor states can be computed exactly, which is not the case in general for more complex dynamics. Enhanced with urgency, LHA can be used to model complex systems from a variety of application domains in a modular fashion. Existing algorithms are limited to convex invariants and urgency conditions that consist of a single constraint. Such restrictions can be a major limitation when the LHA is intended to serve as an abstraction of a model with urgent transitions. This includes deterministic modeling languages such as Matlab-Simulink, Modelica, and Ptolemy, since all their transitions are urgent. The goal of this paper is to remove these limitations, making LHA more directly and easily applicable in practice. We propose an algorithm for successor computation with non-convex invariants and closed, linear urgency conditions. The algorithm is implemented in the open-source tool PHAVer, and illustrated with an example.

1 Introduction

Linear Hybrid Automata (LHA) are discrete automata enhanced with real-valued variables and linear constraints [12]. Despite their syntactical simplicity, they admit a rich variety of behaviors. In LHA, the evolution of the variables over time is governed by differential inclusions, called *flows*, which can be simple intervals such as $\dot{x} \in [1, 2]$, or more complex linear constraints over the derivatives such as the conservation law $\dot{x} + \dot{y} = 0$. Changes of the discrete state admit arbitrary linear updates of the variables. For example, LHA can model discrete-time affine systems, a widely used class of control systems, by using discrete updates of the form $x^+ = Ax + b$.

Linear Hybrid Automata belong to the very few classes of hybrid systems for which set-based successor computations can be carried out exactly [1]. This makes them prime candidates for formal verification. LHA can serve as abstractions of systems that require not only timed behavior but quantitative information, e.g., to capture accumulation effects. The LHA abstraction can then be verified using model checkers such as HyTech [11] or PHAVer [9]. If the abstraction is conservative, verifying it implies that the real system satisfies the specification; if the abstraction is an approximation that is not entirely conservative, its verification helps to find bugs and identify pertinent test cases.

In model-based design, the basis for building LHA abstractions is often an existing model, given in formats like Matlab-Simulink [14] or Modelica [15], which are the de-facto standard in many industries. Like the academic formalism Ptolemy [7], the semantics of these models are deterministic. In particular, a discrete transition is taken as soon as it is enabled, which is also referred to as urgent or as-soon-as-possible (ASAP) semantics. This can pose a problem when trying to build a corresponding LHA model, since LHA transitions do not force the system to change state when they are enabled. In particular, if the derivatives of the system happen to be zero when the guard is enabled, the system may remain forever at that state. One way to circumvent this problem is to add a clock to the controller model and periodically test (with a self-loop transition) whether the constraint is satisfied or not. This is a formally correct and conservative way to model such a system, and it even corresponds quite closely to actual behavior of process controllers, which periodically sample the sensors and set actuators. But it can tremendously increase the computational complexity of the verification task: the clock ticks introduce discrete state changes at a rate much higher than the time constants of the system, multiplying the number of sets of states that need to be computed. Another way is to build a LHA with extra locations whose invariants depend on the geometry of the urgency and flow conditions. But this requires several operations on polyhedra and one needs to disregard the reachable states in the extra locations. Our approach is to add *urgency* conditions to the LHA formalism and use a corresponding post-operator. Declaring certain states of the controller as urgent prevents time from elapsing, and one can now construct an LHA abstraction (or approximation) of deterministic transitions.

Existing algorithms for set-based successor computations of LHA require urgency conditions to either be independent of the continuous variables [11] or consist of a single constraint [9], which can be quite restrictive in practice. In this paper, we propose an algorithm to compute successor states for arbitrary, non-convex, closed urgency conditions. To be able to do so, we also propose an algorithm for computing successor states for general non-convex invariants, for which so far no algorithm is available. Related work is discussed in more detail for non-convex invariants in Sect. 2.3 and for urgency in Sect. 3.4.

The proposed algorithms are implemented in the open-source tool PHAVer on the SpaceEx tool platform [8]. The tool as well as all examples from this paper are available for download at `spaceex.imag.fr`. Detailed proofs are available in a technical report [16].

In the next section, we recall the basics on LHA and then propose our post operator for non-convex invariants. In Sect. 3, we propose our post operator for urgency conditions and make the connection to urgent transitions. The computation of reachable states with these operators is illustrated by an example in Sect. 4.

2 Linear Hybrid Automata with Non-convex Invariants

In this section, we give the syntax and the semantics description of a particular case of *Linear Hybrid Automata (LHA)*, where it is possible to define, for each location, a non-convex invariant.

2.1 Definition and Semantics

We first need to define some notation. 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 $\mathbb{R}^n$ that is the union of a finite number of convex polyhedra. For clarity, we write $\widehat{P}$ if P is convex. The topological closure of P is denoted by $cl(P)$. Given an ordered set $X = \{x_1, \ldots, x_n\}$ of variables, a *valuation* is a function $v : X \to \mathbb{R}$. Let $Val(X)$ denote the set of valuations over X. There is an obvious bijection between $Val(X)$ and $\mathbb{R}^n$, allowing us to extend the notion of (convex) polyhedron to sets of valuations. We denote by $CPoly(X)$ (resp., $Poly(X)$) 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 *activity* over X is a function $f : \mathbb{R}^{\geq 0} \to Val(X)$ that is continuous on its domain and differentiable except for a finite set of points. Let $Acts(X)$ denote the set of activities over X. The *derivative* $\dot{f}$ of an activity f is defined in the standard way and it is a partial function $\dot{f} : \mathbb{R}^{\geq 0} \to Val(\dot{X})$.

A *Linear Hybrid Automaton* is a tuple $H = (Loc, X, Lab, Edg, Flow, Inv, Init)$ with

- a finite set *Loc* of *locations*; a finite set $X = \{x_1, \ldots, x_n\}$ of real-valued *variables*; a *state* is a pair $\langle l, v\rangle$ of a location l and a valuation $v \in Val(X)$; a finite set of labels *Lab*;
- a finite set *Edg* of *discrete transitions* that describes instantaneous changes of locations, in the course of which variables may change their value. Each transition $(l, \alpha, \eta, l') \in Edg$ consists of a *source location* l, a *target location* l', a label $\alpha \in Lab$, and a *jump relation* $\eta \in Poly(X \cup X')$, that specifies how the variables may change their value during the transition. The *guard* is the projection of η on X;
- 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, defining the *initial states* of the automaton.

The set of states of H is $S = Loc \times Val(X)$. Moreover, we use the shorthand notations $InvS = \bigcup_{l \in Loc}\{l\} \times Inv(l)$ and $InitS = \bigcup_{l \in Loc}\{l\} \times Init(l)$. Given a set of states A and a location ℓ, we denote by $A\lfloor_\ell$ the projection of A on ℓ, i.e. $A\lfloor_\ell = \{v \in \mathit{Val}(X) \mid \langle \ell, v\rangle \in A\}$.

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

Given a state $s = \langle l, v\rangle$, we set $\mathit{loc}(s) = l$ and $\mathit{val}(s) = v$. An activity $f \in \mathit{Acts}(X)$ is called *admissible from* s if *(i)* $f(0) = v$ and *(ii)* for all $\delta \geq 0$, if $\dot{f}(\delta)$ is defined then $\dot{f}(\delta) \in \mathit{Flow}(l)$. An activity is *linear* if there exists a constant slope $c \in \mathit{Flow}(l)$ such that, for all $\delta \geq 0$, $\dot{f}(\delta) = c$. We denote by $\mathit{Adm}(s)$ the set of activities that are admissible from s.

Runs. Given two states s, s', and a transition $e \in \mathit{Edg}$, there is a *discrete step* $s \xrightarrow{e} s'$ with *source* s and *target* s' iff *(i)* $s, s' \in \mathit{InvS}$, *(ii)* $e = (\mathit{loc}(s), \alpha, \eta, \mathit{loc}(s'))$, and *(iii)* $(\mathit{val}(s), \mathit{val}(s')[X'/X]) \in \eta$, where $\mathit{val}(s')[X'/X]$ is the valuation in $\mathit{Val}(X')$ obtained from s' by renaming each variable in X with the corresponding primed variable in X'. Whenever condition *(iii)* holds, we say that e is *enabled* in s. There is a *timed step* $s \xrightarrow{\delta, f} s'$ with *duration* $\delta \in \mathbb{R}^{\geq 0}$ and activity $f \in \mathit{Adm}(s)$ iff *(i)* $s \in \mathit{InvS}$, *(ii)* for all $0 < \delta' \leq \delta$, $(\langle l, f(\delta')\rangle) \in \mathit{InvS}$, and *(iii)* $s' = \langle \mathit{loc}(s), f(\delta)\rangle$. Given a state $s \in S$ and a hybrid automaton H with initial set of states *Init*, s is said to be *reachable* in H if there exists a finite *run* $r = s_0 \xrightarrow{\delta_0, f_0} s'_0 \xrightarrow{e_0} s_1 \xrightarrow{\delta_1, f_1} s'_1 \xrightarrow{e_1} s_2 \cdots s_n$, such that $s_0 \in \mathit{Init}$ and $s_n = s$. We denote the set of reachable states by $\mathit{Reach}(H)$.

Classically, the algorithm that computes the set $\mathit{Reach}(H)$ is a fixed-point procedure, over all the locations $l \in \mathit{Loc}$, based on the *continuous post operator* and on the *discrete post operator*: given a set of states $S' \subseteq S$, the first one operator is used to compute the set of states reachable from S' by following an admissible trajectory, while the second one operator is used to compute the set of states reachable from S' via discrete transitions. Notice that the computation of the discrete post operator is not affected by the nature of the invariants, so we focus on the continuous post operator. The formal definitions are as follows:

Definition 1 (Post operators). *Given an hybrid automaton H, a location $\ell \in \mathit{Loc}$, a set of valuations $P, I \subseteq \mathit{Inv}(\ell)$, the* continuous post operator $\mathit{Post}_\ell(P, I)$ *contains the set of all valuations $v \in \mathit{Val}(X)$ reachable from some $u \in P$ without leaving I:*

$$\mathit{Post}_\ell(P, I) = \big\{v \in \mathit{val}(X) \big| \, \exists u \in P, f \in \mathit{Adm}(\langle l, u\rangle) \textit{ and } \delta \geq 0 : \\ \forall 0 < \delta' \leq \delta, f(\delta') \in I \quad \textit{and } f(\delta) = v\big\}. \quad (1)$$

The discrete post operator $\mathit{Post}_\varepsilon(P)$ *contains the set of all valuations $v \in \mathit{Val}(X)$ reachable from some $u \in P$ by taking the discrete transition $\varepsilon = (\ell, \eta, \ell')$:*

$$\mathit{Post}_\varepsilon(P) = \big\{v \in \mathit{val}(X) \big| \, \exists u \in P, (u, v[X'/X]) \in \eta \textit{ and } v \in \mathit{Inv}(\ell')\big\}.$$

From these operators on valuations we obtain the continuous and discrete post operators for a set of states S by iterating over all locations and transitions:

$$\mathit{Post}_c(S) = \bigcup_{\ell \in \mathit{Loc}} \{\ell\} \times \mathit{Post}_\ell(S\downharpoonright_\ell, \mathit{Inv}(\ell)), \quad \mathit{Post}_d(S) = \bigcup_{(\ell,\alpha,\eta,\ell') \in \mathit{Edg}} \{\ell'\} \times \mathit{Post}_\varepsilon(S\downharpoonright_\ell).$$

Note that definition (1) is valid regardless whether I is convex or not. It differs slightly from the classic definition in that we do not require that $P \subseteq I$. This trick is used in the next section to apply the operator iteratively to convex partitions of a non-convex invariant. In this case, I is a convex subset of the invariant but P is not necessarily a subset of I. For the sake of clarity, we will denote by $\mathit{Post}_\ell(P, I)$ the continuous post operator when I is convex and by $\mathit{ncPost}_\ell(P, I)$ when I is non-convex.

The reachable states $\mathit{Reach}(H)$ are computed as the smallest fixed point of the sequence $S_0 = \mathit{Post}_c(\mathit{InitS})$, and $S_{k+1} = S_k \cup \mathit{Post}_c(\mathit{Post}_d(S_k))$.

2.2 Computing the Continuous Post Operator with Nonconvex Invariants

In this section, after recalling how the continuous post operator is computed when the invariant is a convex polyhedron, we give a sound and complete procedure that, given a non-convex invariant I and an initial set of valuations $P \subseteq I$, computes the continuous post operator $\mathit{ncPost}_\ell(P, I)$. Given a linear hybrid automaton H, it is well known that the continuous post operator, on a location $l \in \mathit{Loc}$, a convex invariant $I = \mathit{Inv}(\ell)$, a flow $F = \mathit{Flow}(\ell)$ and a set of initial valuations $P \subseteq \mathit{Inv}(\ell)$, is given by:

$$\mathit{Post}_\ell(P, I) = (P\nearrow_F) \cap I, \tag{2}$$

where $P\nearrow_F$ are valuations on straight line trajectories starting in P with constant derivative $\dot{x} = c$ for any $c \in F$:

$$P\nearrow_F = \{x' \mid x \in P, c \in F, t \in \mathbb{R}^{\geq 0}, x' = x + ct\}. \tag{3}$$

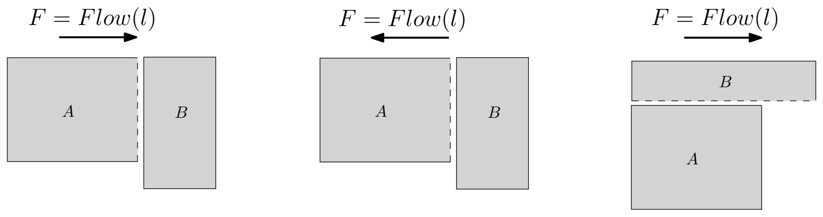
The operator (3) is straightforward to compute for polyhedral sets, and is available in computational geometry libraries such as the Parma Polyhedra Library (PPL) [2].

Before giving the fixed point characterization of ncPost_ℓ, we need to introduce some extra notation (some of them similar to operators defined in [6]). Given polyhedra A and B, their *boundary* is

$$\mathit{bndry}(A, B) = \big(\mathit{cl}(A) \cap B\big) \cup \big(A \cap \mathit{cl}(B)\big). \tag{4}$$

Clearly, $\mathit{bndry}(A, B)$ is nonempty only if A and B are adjacent to one another or they overlap; otherwise, it is empty.

Definition 2 (Potential entry). *Given a location ℓ and convex polyhedra A and B, the potential entry region from A to B denotes the set of points on the*



(a) Case 1: the flow allows to reach B from $bndry(A, B)$.

(b) Case 2: the flow does not allow to reach B from $bndry(A, B)$.

(c) Case 3: the flow does not allow to reach B from $bndry(A, B)$.

Fig. 1. The computation of potential entry from A to B involves computing the boundary of A and B, and identifying states reachable on that boundary

boundary between A and B that may reach B by following some linear activity in location ℓ, while always remaining in $A \cup B$:

$$pentry_\ell(A, B) = \big\{p \in bndry(A, B) \mid \exists q \in A, \delta \geq 0 \text{ and } c \in Flow(\ell) : \\ p = q + \delta \cdot c \text{ and for all } 0 \leq \delta' < \delta,\ q + \delta' \cdot c \in A\big\}. \quad (5)$$

We call the above set the "potential" entry because it may happen that, even though $pentry_\ell(A, B)$ is not empty, the system is not able to reach valuations in B starting from a valuation in A (see Example 1, Fig. 1(c)). The following Lemma gives us a way to effectively compute the potential entry region.

Lemma 1. *Given a location ℓ and convex polyhedra A and B, let $F = Flow(\ell)$, the potential entry region from A to B can be computed by:*

$$pentry_\ell(A, B) = bndry(A, B) \cap A\nearrow_F .$$

From Lemma 1 follows the following Corollary:

Corollary 1. *If $A \subseteq B$, then $A \subseteq pentry_\ell(A, B) \subseteq cl(A)$.*

Example 1. Figure 1 shows two convex polyhedra A and B whose boundary is non empty (A and B are adjacent), where flow is represented by an arrow. Considering Figure 1(a), it is easy to check that the flow allows to reach valuations on the boundary between A and B starting from a valuation belongs to A, and then $pentry_\ell(A, B) \neq \emptyset$. Considering instead the case depicted in Figure 1(b) (same as the previous one except for the flow), there is no way to reach valuations belong to $bndry(A, B)$ starting from a valuation $u \in A$, and then $pentry_\ell(A, B) = \emptyset$. Figure 1(c) shows a case where the polyhedron B is not closed and then by following the flow, the system can never reach B, even if the starting valuation is on the top border of A. Notice that, even if B is not reachable from A, we have that $pentry_\ell(A, B) \neq \emptyset$: this clearifies why we denote this set as "potential".

Now we are ready to give a way to correctly compute the continuous post operator when the invariants could be non-convex. Given a LHA H and let $l \in$

Loc, $I = \mathit{Inv}(\ell)$, $F = \mathit{Flow}(\ell)$ and $P \subseteq I$. The idea is to build incrementally the sets of reachable valuations by considering each time a single convex component $\widehat{I'} \in [\![I]\!]$ instead of considering the entire invariant I. The procedure starts by finding, for all $\widehat{I'} \in [\![I]\!]$ and $\widehat{P'} \in [\![P]\!]$, the potential entry from $\widehat{P'}$ to $\widehat{I'}$.

Once obtained the set $\mathit{pentry}_\ell(\widehat{P'}, \widehat{I'})$, the procedure computes the classical continuous post operator on $\mathit{pentry}_\ell(\widehat{P'}, \widehat{I'})$ and $\widehat{I'}$. The procedure is applied recursively by building the sequence $W_0 \subseteq W_1 \subseteq \dots W_{i-1} = W_i$ of the sets of the reachable valuations, with $W_0 = P$, and ends when no new valuation can be added to a set. When this happens, we have that $\mathit{ncPost}_\ell(P, I) = W_i$.

The formal relationship between the fixed-point procedure described above and the computation of the continuous post operator, when the invariant is non-convex, is given by the following theorem:

Theorem 1. *Given a location $\ell \in \mathit{Loc}$ and sets $P \subseteq \mathit{Inv}(\ell)$, $I = \mathit{Inv}(\ell)$, $\mathit{ncPost}_\ell(P, I)$ is the smallest fixed point of the sequence $W_0 = P$,*

$$W_k = \bigcup_{\widehat{W'} \in [\![W_{k-1}]\!]} \bigcup_{\widehat{I'} \in [\![I]\!]} \mathit{Post}_\ell(\mathit{pentry}_\ell(\widehat{W'}, \widehat{I'}), \widehat{I'}).$$

Moreover, the above sequence reaches the fixed point in at most $n = \big|[\![I]\!]\big|$ steps, that is $W_{n+1} = W_n$.

Notice that the role of $\mathit{pentry}_\ell(\widehat{W'}, \widehat{I'})$ is crucial in order to compute all and only those valuations that can be reached in $\widehat{I'}$ by always remaining in the global invariant I. This condition can not be ensured by applying the post operator directly on $\widehat{W'}$ instead of $\mathit{pentry}_\ell(\widehat{W'}, \widehat{I'})$ (see Section 2.3 in [16] for more details).

We prove Theorem 1 by induction on the number of the convex components of the invariant in which the system remains during a run. We define this number as follows. Given a polyhedron I and two valuations u and v, assuming that v is reachable from u via an admissible activity that always remains in I (i.e. always avoids $\overline{I}$), we denote by $d(u, v, I)$ the minimum number of convex polyhedra in $[\![I]\!]$ in which the system must remain in order to reach v from u via any admissible activity f:

$$\begin{aligned} d(u, v, I) = \min\{n > 0 \mid \exists f \in \mathit{Adm}(\langle \ell, u\rangle), \delta \geq 0, \widehat{I_1}, \dots \widehat{I_n} \in [\![I]\!] : \\ f(\delta) = v \text{ and } \forall 0 \leq \delta' \leq \delta\, \exists j \in \{1, \dots, n\} : f(\delta') \in \widehat{I_j}\}. \end{aligned}$$

When there is no activity that can reach v from u avoiding $\overline{I}$, we write $d(u, v, I) = \infty$. Hence either $d(u, v, I) \leq \big|[\![I]\!]\big|$ or $d(u, v, I) = \infty$. For the induction proof, we define a version of ncPost_ℓ that takes into account only valuations v such that the system, in order to reach v, always remains in a fixed number of convex polyhedra in the invariant. Given a location ℓ and sets $P, I \subseteq \mathit{Inv}(\ell)$ and $i \leq |[\![I]\!]|$,

$$\mathit{ncPost}_\ell(P, I, i) = \big\{v \in \mathit{ncPost}_\ell(P, I) | \exists u \in P : d(u, v, I) \leq i\big\}.$$

Note that for all $i \leq j$, $\mathit{ncPost}_\ell(P, I, i) \subseteq \mathit{ncPost}_\ell(P, I, j)$.

We exploit the following, fundamental property of LHA: if there is an activity that goes from u to v inside the invariant, there is also a sequence of linear activities that does the same. Moreover, each linear activity is contained within one convex polyhedron of $[\![I]\!]$ and hence the connecting points between any two consecutive linear activities lie on the boundary between two polyhedra in $[\![I]\!]$. The following formalization is a reformulation of Lemma 2.2 in [18] given as Lemma 5 in [6]:

Lemma 2. [6] *Let u and v be valuations, and I a polyhedron. If $d(u, v, I) = i < \infty$, then there is a sequence of linear activities $f_1, \dots, f_i$, delays $\delta_0, \dots, \delta_i$, and convex polyhedra $\widehat{I}_1, \dots, \widehat{I}_i \in [\![I]\!]$ such that* (i) *$f_1 \in Adm(\langle l, u\rangle)$,* (ii) *$f_{i-1}(\delta_{i-1}) = v$,* (iii) *for all $j < i$ it holds $f_j(\delta_j) \in bndry(\widehat{I}_j, \widehat{I}_{j+1})$ and $f_{j+1} \in Adm(\langle l, f_j(\delta_j)\rangle)$, and* (iv) *for all $j \le i$ and $0 < \delta' < \delta_j$ it holds $f_j(\delta') \in \widehat{I}_j$.*

For lack of space, we only give a proof sketch of Theorem 1. The complete proof can be found in the appendix and in [16].

Proof of Theorem 1. (Sketch) Let $n = \left|[\![I]\!]\right|$, first notice that for two valuation u and v, where $u, v \in ncPost_\ell(P, I)$, by definition of post operator $d(u, v, I) \le \left|[\![I]\!]\right|$. Then trivially holds that $ncPost_\ell(P, I) = ncPost_\ell(P, I, n)$.

We show by induction that for all locations ℓ, polyhedra $P, I \subseteq Inv(\ell)$, and $i \ge 1$, $ncPost_\ell(P, I, i) = W_i$. The base case is straightforward, since it corresponds to a convex invariant.

We first discuss $ncPost_\ell(P, I, i) \subseteq W_i$. Consider a run that goes from some $u \in W_1$ through some $u' \in W_{i-1}$ to some $v \in ncPost_\ell(P, I, i)$. We need to show that $v \in W_i$. Let $\hat{I}_{i-1}$ be the $i-1$th invariant visited on the run. If $u' \in \hat{I}_{i-1}$, the proof is straightforward, since all reachable states in $\hat{I}_{i-1}$ are in W_{i-1}. Otherwise, $u' \in \hat{I}_i$. With Lemma 2, there is some $u^* \in \hat{I}_{i-1}$ such that a straight line activity can be extended from u^* to u'. These states are contained in the potential entry set, and by definition also in W_i.

To show $W_i \subseteq ncPost_\ell(P, I, i)$, we need to show that W_i does not contain more states than $ncPost_\ell(P, I, i)$. Using case distinctions similar to the previous paragraph, we can show that W_i consists of states reachable using the convex post operator inside a convex invariant, plus the boundary states reachable by straight line trajectories. From Lemma 2, it follows that these boundary states are also in $ncPost_\ell(P, I, i)$, which concludes the proof. □

2.3 Related Work

In [13], the author shows a different approach in order to tackle non-convex invariants. The proposed algorithm to compute the reachable set is built only for closed convex invariants, but this is not a restriction because (closed) non-convex invariants can be modeled by splitting locations. This means that starting from an automaton A with non-convex invariants, it is necessary to build an equivalent automaton B whose locations have only convex invariants: this is done by taking, for each location of A, the exact convex covering Q of the corresponding invariant and then, for each convex component $\widehat{Q} \in Q$, by adding a location to B whose associated (convex and closed) invariant is $\widehat{Q}$. Therefore, this approach does not work with non-closed invariants and needs a postprocessing phase in order to build the automaton B. Our approach tries to overcome these limitations: the reachability analysis is directly done by using the $ncPost_\ell$ operator, allowing the usage of non-closed invariants and avoiding the hidden process of building a new automaton.

3 Linear Hybrid Automata with Urgency

In this section, we extend LHA by allowing the possibility to attach to each location a so-called *urgency condition.* The urgency condition impedes time elapse, i.e., no continuous activities continue from a valuation that satisfies the condition. As we will see later, there is a connection between urgency conditions on locations and urgent semantics on transitions.

3.1 Definition and Semantics

We denote by $SPoly(X)$ the subset of $\mathbb{R}^X$ that can be obtained by finite disjunction of closed convex polyhedra. A *Linear Hybrid Automaton with Urgency (LHAU)* $H = (Loc, X, Lab, Edg, Flow, Inv, Urg, Init)$ consists of a LHA defined in Sect. 2 and a mapping $Urg : Loc \to SPoly(X)$, called *urgency condition.* To designate the urgent states, we use the shorthand $UrgS = \bigcup_{l \in Loc} \{\ell\} \times Urg(\ell)$.

Urgent transitions. In our definition, the urgency condition is defined for each location. An alternative approach, popular mainly because of its syntactical simplicity, is to designate each discrete transition as urgent or not. This is also referred to as *as-soon-as-possible (ASAP) transitions.* Urgent transitions can easily be translated to an urgency condition: Let $Edg_U \subseteq Edg$ be the set of urgent transitions. Then the equivalent urgency condition is the union of the outgoing guards, $Urg(\ell) = \{u \mid \exists (\ell, \eta, \ell') \in Edg_U : (u, v) \in \eta\}$.

Semantics. The urgency conditions affect only the timed steps, while the definition of discrete step remains the same as for LHA. Given a state $s = \langle l, v \rangle$, we define $loc(s) = l$ and $val(s) = v$. In order to give the semantics of timed-steps for *LHAU* we define, for an activity $f \in Adm(s)$, the *Switching Time* of f in l, denoted by $SwitchT(f, U)$, as the value $\delta \geq 0$ such that, for all $0 \leq \delta' < \delta$, $f(\delta') \notin U$ and $f(\delta) \in U$. When for all $\delta \geq 0$ it holds that $f(\delta) \notin U$, we write $SwitchT(f, U) = \infty$. Informally, the switching time of an activity f in the location l specifies the maximum amount of time δ such that the system, by following the activity f, is allowed to remain in the location l.

Given two states s, s', there is a *timed step* $s \xrightarrow{\delta, f} s'$ with *duration* $\delta \in \mathbb{R}^{\geq 0}$ and activity $f \in Adm(s)$ iff *(i)* there exists the timed step $s \xrightarrow{\delta, f} s'$ in the LHA without urgency conditions, and *(ii)* $\delta \leq SwitchT(f, Urg(loc(s)))$.

Parallel Composition. We give a brief formal definition of parallel composition with urgency for the case where both automata range over the same variables. The key here is that the urgency condition of the composition is the union of the urgency conditions of the operands.

Definition 3 (Parallel composition). *Given linear hybrid automata with urgency H_1, H_2 with $H_i = (Loc_i, X, Lab_i, Edg_i, Flow_i, Inv_i, Urg_i, Init_i)$, their* parallel composition *is the LHAU $H = (Loc_1 \times Loc_2, X, Lab_1 \cup Lab_2, Edg, Flow, Inv, Urg, Init)$, written as $H = H_1 \| H_2$, where*

- $((l_1, l_2), \alpha, \eta, (l_2', l_2')) \in Edg$ *iff*
  - $\alpha \in Lab_1 \cap Lab_2$, *for* $i = 1, 2$, $(l_i, \alpha, \eta_i, l_i') \in Edg_i$, *with* $\eta = \eta_1 \cap \eta_2$, *or*
  - $\alpha \notin Lab_1$, $l_2' = l_2$, *and* $(l_1, \alpha, \eta, l_1') \in Edg_1$, *or*
  - $\alpha \notin Lab_2$, $l_1' = l_1$, *and* $(l_2, \alpha, \eta, l_2') \in Edg_2$;
- $Flow(l_1, l_2) = Flow_1(l_1) \cap Flow_2(l_2)$; $Inv(l_1, l_2) = Inv_1(l_1) \cap Inv_2(l_2)$;
- $Urg(l_1, l_2) = Urg_1(l_1) \cup Urg_2(l_2)$; $Init(l_1, l_2) = Init_1(l_1) \cap Init_2(l_2)$.

3.2 Reachability

The discrete post operator for the class of *LHAU* is trivially the same of the classical one, while the continuous one, that we call *Urgent Continuous Post Operator*, changes due to the extra condition induced by the operator *SwitchT*:

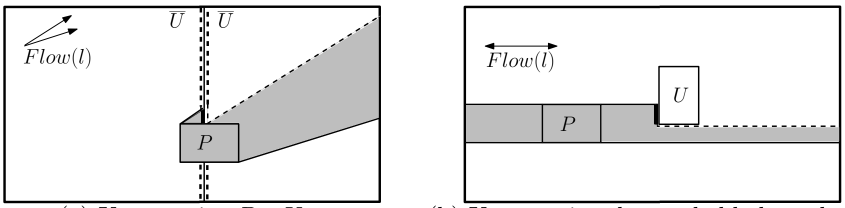
Definition 4 (Urgent continous post). *Given a linear hybrid automaton with urgency H, a location $\ell \in Loc$, and a set of valuations $P \subseteq Inv(\ell)$, let $I = Inv(\ell)$, and $U = Urg(\ell)$. The* urgent continuous post operator $UPost(P, I, U)$ *is defined as:*

$$UPost(P, I, U) = \Big\{ v \in val(X) \Big| \exists u \in P, f \in Adm(\langle \ell, u \rangle), \delta \geq 0 : \\ f(\delta) = v, \text{ for all } 0 < \delta' \leq \delta, f(\delta') \in I, \text{ and } \delta \leq SwitchT(f, U) \Big\}.$$

3.3 Computing the Urgent Continuous Post Operator

We now derive a construction of the urgent post operator, starting with the post operator for non-convex invariants and adding the states that are missing.

The urgent post operator has to compute the valuations that are reachable from some set P without passing through states in the urgent set U. This includes the states that are reachable within the complement of U, so $ncPost_\ell(P \cap \overline{U}, I \cap \overline{U})$ is an underapproximation of $UPost_\ell(P, I, U)$. In the following, let $\mathbb{V}_{nc} = ncPost_\ell(P \cap \overline{U}, \overline{U})$ and $\mathbb{V}_U = UPost_\ell(P, I, U)$. The set $\mathbb{V}_{nc}$ trivially does not contain valuations that belong to U (since $\bar{U}$ is used in the invariant), while $\mathbb{V}_U$ also contains those valuations that touch U for the first time on a run. As shown in the examples of Figure 2, the system is allowed to remain on the boundary of an invariant for any time as the invariant is satisfied, while the system can not remain on the boundary of an urgency condition. In the instant the urgency



(a) $\mathbb{V}_U$ contains $P \cap U$. (b) $\mathbb{V}_U$ contains the reachable boundary

Fig. 2. The urgent post states $\mathbb{V}_U = UPost_\ell(P, I, U)$ can be obtained from $\mathbb{V}_{nc} = ncPost_\ell(P \cap \overline{U}, I \cap \overline{U})$ plus the part (identified by the thick lines) of the boundary between $\mathbb{V}_{nc}$ and U that can be reached from $\mathbb{V}_{nc}$. The dashed lines identify the non-closed borders.

condition is met, the system can not evolve any more, i.e., it is forced either to stop the evolution of the continuous variables or to jump in another location. The thick lines in Figure 2(a) and Figure 2(b) identify the valuations on the boundary between $\mathbb{V}_{nc}$ and U that can be reached from $\mathbb{V}_{nc}$, and therefore they belong to $\mathbb{V}_U$.

In summary, we can compute $\mathbb{V}_U$ as the union of P, $\mathbb{V}_{nc}$ and the set of the valuations that belong to the boundary between $\mathbb{V}_{nc}$ and U from where it is possible to reach U by following some admissible activity. The latter set is obtained by using the potential entry operator. This is formalized as follows:

Theorem 2. *Given a location $\ell \in Loc$ and a set $P \subseteq Inv(\ell)$, let $I = Inv(\ell)$, $U = Urg(\ell)$, $\mathbb{V}_{nc} = ncPost_\ell(P \cap \overline{U}, I \cap \overline{U})$, and $B = \bigcup_{\widehat{A}' \in [\![\mathbb{V}_{nc}]\!]} \bigcup_{\widehat{U}' \in [\![U]\!]} pentry_\ell(\widehat{A}', \widehat{U}' \cap I)$. Then $UPost_\ell(P, I, U) = P \cup \mathbb{V}_{nc} \cup B$.*

3.4 Related Work

A general class of hybrid automata with urgency conditions is described in [17], but without giving the computation of the continuous post operator for urgency. In that work, the *Time Can Progress (tcp)* predicate specifies, for each location the maximum sojourn time, which may depend on the values of the variables when entering the location. This corresponds to the complement of our urgency condition. Notice that the semantics in [17] require the *tcp* to be satisfied when the location is entered. In our framework we relax this constraint by allowing to enter a location even if its urgency condition is already satisfied: in this case, the system must exit the location instantaneously. A similar urgency condition is described in the *Computational Interchange Format for Hybrid Systems (CIF)* (see [4]). For a more detailed and formal discussion of urgency, see [10] and references therein.

Urgent locations. In the classic LHA model checker HyTech, a transition can be designated as urgent by adding the keyword ASAP [13]. But this is restricted to transitions without guard constraints [11,13], which is equivalent to having *urgent locations*, i.e., locations in which time progress is not allowed. The real-time verification tool UPPAAL [5] similarly features urgent locations and urgent channels (synchronization labels) that can be used only on transitions without guard constraints. Urgent locations are semantically equivalent to adding an extra variable t, with dynamics $\dot{t} = 1$, that is set to zero when the location is entered and by attaching the invariant $t = 0$ to the location. In previous versions of our model checker PHAVER, transitions could be designated as urgent, but only if the guard consists of a single constraint, locally as well as in the composed model [9]. This restriction was imposed because it suffices to be able to compute the urgent post using the standard post operator for convex invariants.

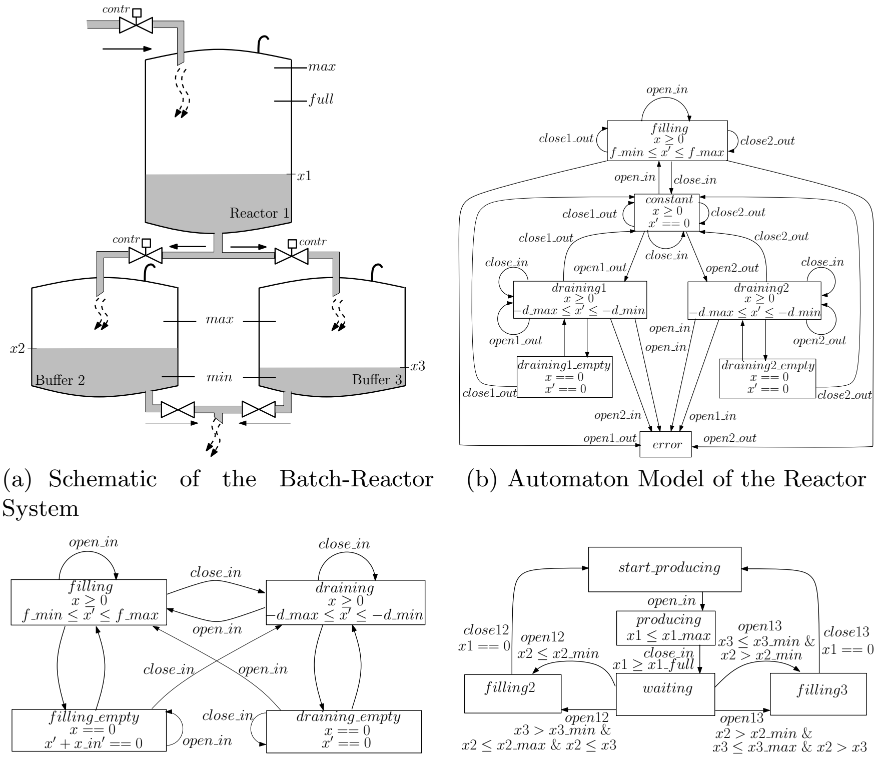
Almost ASAP. In [19] the authors propose a relaxed semantics on asap transition in the context of the timed automaton, for the so called *almost asap* by delay δ. In practice, they define the *guard enlargement*, that means that transitions can be taken also with δ time delay. The rationale behind this approach is that no hardware can guarantee that a transition will always be taken in the exact moment as defined in theory. We could define a similar approach, not only on clock variables, in a simple but opposite way: it is enough to define the urgency condition by narrowing all the constraints by a quantity that is equal to the maximum variation of the variable in the time δ.

4 Example: Batch Reactor

To showcase the algorithm an its implementation, we present a modular model of a batch-reactor system, which is a variation of the case study in [3]. It shall illustrate that non-urgent transitions as well as urgent transitions with more than one guard constraint arise naturally.

The batch reactor is comprised of a reactor R1 and two buffer tanks B2,B3 connected by pipes. The reactor is used to create a product that is then made available to a consumer in the two buffer tanks, see the schematic in Fig. 3(a). A controller measures the fill levels in the reactor and the buffers, and opens and closes valves connecting the reactor to the buffers in order to produce and deliver the product to the consumer. The specification is to verify that neither buffer ever becomes empty, and that none of the tanks overflows.

We now present the LHA models. The controller automaton is shown in Fig. 3(d). The opening and closing of valves is modeled by synchronization labels. In the production step, the reactor is filled with educts (raw materials) coming from the outside. Details on the filling and reaction process itself are omitted since they are irrelevant to this example, but it does take a certain amount of time and produces an uncertain amount of product. This is modeled by the fact that the controller ends the filling process when the reactor level $x_1 \in [x_{1,full}, x_{1,max}]$, which is accomplished with the invariant $x_1 \leq x_{1,max}$ and a non-urgent transition with label *close_in* and guard condition $x_1 \geq x_{1,full}$. When the product is ready, the controller decides whether to fill buffer B2, buffer B3, or wait. The controller decides which buffer to fill using the following simple criteria:



(a) Schematic of the Batch-Reactor System (b) Automaton Model of the Reactor

(c) Automaton Model of the Buffers (d) Automaton Model of the Controller

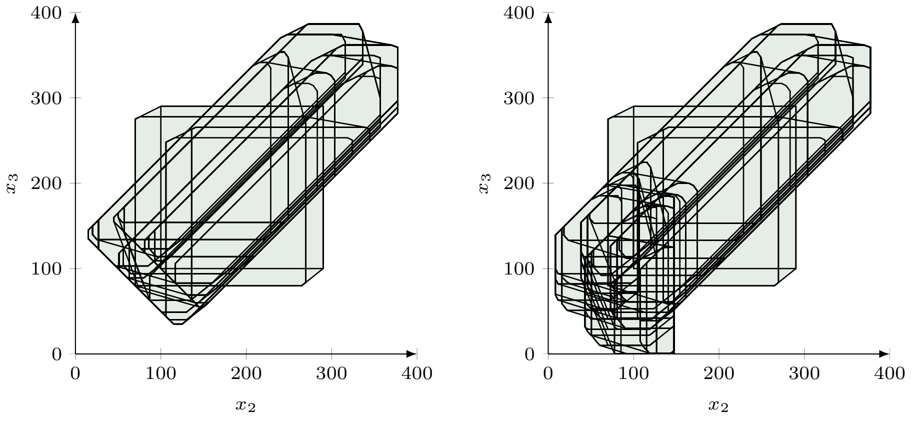
Fig. 3. Batch Reactor System: Schematic and Automata Models

- To avoid overflow, never start filling a buffer above a given maximum level.
- To avoid empty buffers, fill a buffer below a given minimum level.
- If the above is met, fill the buffer with the lower level.
- To be deterministic, prioritize B2.

All transitions for filling buffers B2 and B3 are urgent. The if-then-else structure of the criteria leads to guards with more than one constraint, some of which are strict inequalities. Thanks to the urgency, the controller model requires no clocks or while-loops.

The reactor automaton is shown in Fig. 3(b). The locations of the reactor correspond to the different combinations of open and closed valves. The model is simplified using the assumptions that the reactor must not be filled and drained at the same time (a common requirement in chemical engineering), and that only one of the buffers is filled at any given time. An error location is included so that violations of these assumptions can be detected. The transitions are not set as urgent in this automaton; the urgency in the composed system results from the controller.

The buffers are modeled each as an instantiation of the automaton shown in Fig 3(c). The outflow of the buffers is determined by the consumer, and therefore only known within the bounds (there is no valve to control outflow). The inflow



(a) reachable buffer levels x_2 and x_3 for safe parameter values

(b) decreasing the min. reactor outflow by 5% eventually leads to an empty buffer B3

Fig. 4. The evolution of the continuous variables of the batch reactor example, starting from location *start_producing* with initial values $x_1 = 0$, $x_2 = 100$ and $x_2 = 100$

is determined is equal to the outflow of the reactor. This leads to the dynamics $\dot{x}_i = [-d_{i,max}, -d_{i,min}] - \dot{x}_1$. Note that $\dot{x}_1$ is negative when the buffer is filling, so $\dot{x}_i$ is augmented by $-\dot{x}_1$ in this dynamics. Again, the transitions are not set as urgent; the urgency in the composed system results from the controller.

The specification was verified using SpaceEx/PHAVer (an implementation of the PHAVer reachability algorithm built on the SpaceEx platform). The inflow and outflow rates were set nondeterministically to be within intervals; the models incl. parameter values are available at `spaceex.imag.fr`. The computation of the complete reachable states shown in Fig. 4(a) takes 3.0 s and 24 MB of memory on a standard laptop. Finding the fixed point takes a total of 178 continuous post operations. Buffer 3 goes empty if the lower bound on the reactor outflow is reduced by 5%. The fixed point is found in 1153 post operations, which takes 18.7 s and consumes 24 MB of memory.

5 Conclusions

Linear Hybrid Automata stand out in the hybrid systems domain because sets of successor states can be carried out exactly. Available algorithms require convex invariants and single-constraint urgency conditions. In this paper, we propose algorithms that can handle non-convex invariants and (closed) non-convex urgency conditions. The practical impact is that this extension can be used in order to model systems in which transitions have to be taken as soon as possible. This is a common feature in several commercial tools used as de-facto standard in industry (for example in the automotive context) such as Matlab/Simulink or Modelica. We formally proved the correctness and the termination of the proposed procedures, which are based on two operators: the first one is the classical continuous post operator for convex sets $Post_\ell$ and the other one (defined here) is the so-called potential entry operator $pentry_\ell$. To the best of our knowledge, the proposed solutions represent the first sound and complete procedures for the task in the literature.

References

1. Alur, R., Henzinger, T., Ho, P.H.: Automatic symbolic verification of embedded systems. IEEE Trans. Softw. Eng. 22, 181–201 (1996)
2. Bagnara, R., Hill, P.M., Zaffanella, E.: 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), 3–21 (2008)
3. Bauer, N., Kowalewski, S., Sand, G., Löhl, T.: A case study: Multi product batch plant for the demonstration of control and scheduling problems. In: Engell, S., Kowalewski, S., Zaytoon, J. (eds.) ADPM 2000, pp. 383–388. Shaker (2000)
4. van Beek, D.A., Reniers, M.A., Schiffelers, R.R.H., Rooda, J.E.: Foundations of a compositional interchange format for hybrid systems. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 587–600. Springer, Heidelberg (2007)
5. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
6. Benerecetti, M., Faella, M., Minopoli, S.: Automatic synthesis of switching controllers for linear hybrid systems: Safety control. TCS 493, 116–138 (2012)
7. Buck, J.T., Ha, S., Lee, E.A., Messerschmitt, D.G.: Ptolemy: A framework for simulating and prototyping heterogeneous systems. Ablex Publishing Corp. (1994)
8. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)
9. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. STTT 10(3), 263–279 (2008)
10. Gebremichael, B., Vaandrager, F.: Specifying urgency in timed i/o automata. In: SEFM 2005, pp. 64–74. IEEE Computer Society (2005)
11. Henzinger, T.A., Ho, P.H., Wong-Toi, H.: Hytech: the next generation. In: Proc. IEEE Real-Time Systems Symposium, p. 56. IEEE Computer Society (1995)
12. Henzinger, T.: The theory of hybrid automata. In: 11th IEEE Symp. Logic in Comp. Sci., pp. 278–292 (1996)
13. Ho, P.H.: Automatic Analysis of Hybrid Systems. Ph.D. thesis, Cornell University, technical Report CSD-TR95-1536 (August 1995)
14. MathWorks: Mathworks simulink: Simulation et model-based design (Mar 2014), `http://www.mathworks.fr/products/simulink`
15. Mattsson, S.E., Elmqvist, H., Otter, M.: Physical system modeling with Modelica. Control Engineering Practice 6(4), 501–510 (1998)
16. Minopoli, S., Frehse, G.: Non-convex invariants and urgency conditions on linear hybrid automata. Tech. Rep. TR-2014-4, Verimag (April 2014)
17. Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: An approach to the description and analysis of hybrid systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) HS 1991 and HS 1992. LNCS, vol. 736, pp. 149–178. Springer, Heidelberg (1993)
18. Wong-Toi, H.: The synthesis of controllers for linear hybrid automata. In: IEEE Conf. Decision and Control, pp. 4607–4612. IEEE (1997)
19. De Wulf, M., Doyen, L., Raskin, J.-F.: Almost ASAP semantics: From timed models to timed implementations. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 296–310. Springer, Heidelberg (2004)