# Accounting for communication delays in hybrid systems

Guruprerana Shabadi, Sergio Mover
Department of Computer Science (LIX), Ecole Polytechnique

**Abstract**

Cyber-physical systems use communication protocols to gather information on their environments in order to react to changes around them. However, in practice these protocols are far from perfect and can have non-negligible delays in relaying state information, i.e., there might be a delay in observing the current value of a state variable. To further motivate the study of these delays, we explore the case study of a collision avoidance protocol with a group of robots navigating in space while avoiding collisions. Here, we cannot ensure that each robot has complete information about the state of the other robots at every instant of time. To model these delays, we revisit the class of hybrid automata called lazy hybrid automata in the continuous, instead of discrete, semantic. Lazy hybrid automata model non-deterministic delays in observing the value of state variables used in invariants and guard conditions. We then discuss the results we have obtained on the expressiveness of some restricted classes of (continuous) lazy hybrid automata by providing their translations into classical hybrid automata.

## 1 Introduction

As cyber-physical systems (CPS) automate increasingly important tasks in our daily lives, they are no longer restricted to local domains of functioning. In order to complete objectives, they need to interact with multiple agents around them, which include other autonomous systems, humans, and even the environment. Safe coordination between these distributed agents necessitates the use of communication protocols to exchange information about individual states. Practical implementations of communication protocols in CPS inevitably lead to non-negligible and non-deterministic delays in the transmission of crucial information. Consider the example of a system of multiple aerial drones which need to avoid collisions during flight by coordinating with each other. In a real world setting, we cannot ensure that the controller of each drone has access to perfect information about the states of its surrounding drones. This motivates the study of possible communication delays and encourages us to consider the task of modeling delays within existing frameworks for representing CPS.

A hybrid system is a popular model for CPS that can capture a combination of continuous and discrete time behaviors. In [10], the authors present a simple decentralized control policy for the collision-free navigation of drones in space. This policy consists of a single hybrid automaton plant model for each drone controlled by a constant signal in each discrete location (Fig. 1). However, to facilitate the discrete location switching, the hybrid automaton model of the drone requires perfect information of the positions, velocities, and heading directions of the other drones around it. As pointed out earlier, in realistic implementations, we would need to account for potential observation delays which may arise in such a multi-agent setting. In this work, we capture observation delays within the hybrid automaton model by through a well constructed extension of the semantic.

With the aim of modeling delays in the observation of state variables within hybrid systems, we extend the semantic of a hybrid automaton (HA), to formulate a new class of lazy hybrid automata (LHA). This new extention was heavily inspired by [1] where a similar model (of the same name) is described. However, the key difference is that we consider a continuous time model whereas in [1], they adopt discrete time hybrid model.

This new class of lazy hybrid automata (LHA) capture bounded non-deterministic delays in the observation of state variables. Here, the state variables represent a physical quantity which may evolve within the system

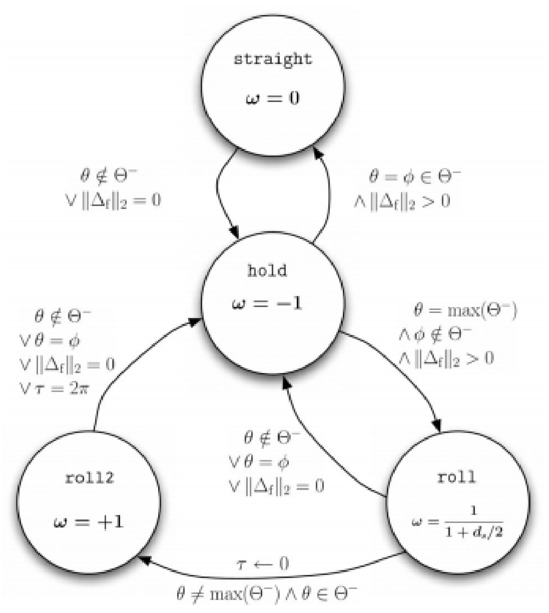
straight
$\omega = 0$
$\theta \notin \Theta^-$
$\vee \|\Delta_f\|_2 = 0$
$\theta = \phi \in \Theta^-$
$\wedge \|\Delta_f\|_2 > 0$
hold
$\omega = -1$
$\theta \notin \Theta^-$
$\vee \theta = \phi$
$\vee \|\Delta_f\|_2 = 0$
$\vee \tau = 2\pi$
$\theta = \max(\Theta^-)$
$\wedge \phi \notin \Theta^-$
$\wedge \|\Delta_f\|_2 > 0$
$\theta \notin \Theta^-$
$\vee \theta = \phi$
$\vee \|\Delta_f\|_2 = 0$
roll2
$\omega = +1$
roll
$\omega = \frac{1}{1 + d_s/2}$
$\tau \leftarrow 0$
$\theta \neq \max(\Theta^-) \wedge \theta \in \Theta^-$

Figure 1: Hybrid automaton figure taken from [10]

itself or may be observed as inputs from a neighboring system. For example, in the case of a drone, a system variable which evolves within the system is the velocity of the drone. On the other hand, the velocities of neighboring drones appear as inputs within a given drone. In this work, we do not consider delays within the continuous dynamics of the system. Rather, we restrict ourselves to considering non-deterministic delays within the discrete switching between locations of the hybrid automaton and also within the invariants at each location.

## 1.1 Contributions

After discussing the reasoning behind the usefulness and validity of our definition of LHA, we explore the expressive power of LHA compared to classical HA. To this end, we present translations of two restricted subclasses of LHA into classical HA. A translation provides a bisimulation and hence proves the equivalence (in terms of expressive power) between these subclasses of LHA and HA. However, as we will explain towards the end of the paper, we have sufficient reason to believe that LHA are strictly more expressive than classical HA, although a proof of this fact has not been sought.

## 1.2 Related work

Our work was heavily inspired by the semantic of lazy rectangular hybrid automata presented in [1]. However, a fundamental difference is that we consider a continuous time behavior compared the discrete time behavior considered in [1]. A discrete time approach is appropriate when considering delays with sensors where we have a minimum sampling rate. However, when considering communication protocols, we do not have a defined sampling rate in the most general case. This is the primary motivation behind considering the continuous time case.

Delays have been extensively studied in the context of networked control systems (NCS). In particular, a lot of work has focused on studying stability of NCS under network delays ([3], [8]). A more recent work is [12], where they try to minimize delays in the network by reducing the number of required transmissions which in turn decreases network load. Another work is [11] which adopts a stochastic approach to handling state estimator delays in closed loop systems. In contrast, with our approach, we consider big delays to be just as likely as small delays which is a potential drawback.

Another recent research thrust has been to study hybrid systems with dynamics described by differential delay equations (DDEs). DDEs are differential equations where the derivative is given in terms of the values of the function at previous times. Reachability analysis ([9], [13]) and unbounded verification ([6]) of DDE systems have been studied. A similar work is [2], in which they consider both delays within the dynamics of a hybrid system, as well as within the discrete switching. However, a key difference is that, all of these works consider constant delays in the observation of the variables, which might not be realistic. Nevertheless, we hope to explore these further to study variable delays within the dynamics of our lazy hybrid automaton semantic.

## 1.3 Preliminaries

We define a hybrid automaton (HA) which we refer to as a classical HA in the rest of the paper to distinguish it from the extended semantic defined in the following section.

**Definition 1** (Hybrid automaton)**.** Consider a vector of system variables $\mathbf{x} = (x_1, \ldots, x_n) \in X$, a vector of *input system variables* $\mathbf{y} = (y_1, \ldots, y_n) \in Y$, control inputs $\mathbf{u} \in U$ and disturbances $\mathbf{w} \in W$. Then, a lazy hybrid automaton (LHA) is a tuple $\mathscr{H} : (L, E, I, F, G, R)$ consisting of:

1. A finite set of *locations* $Q = \{q_1, \ldots, q_k\}$ and *transitions* that form edges between states $E \subseteq Q \times Q$.
2. A map $I$ that associates with each state $q \in Q$ an *invariant* $I(q)$ which is a predicate over the variables in $\{\mathbf{x}, \mathbf{y}\}$.
3. A map $F$ that associates each mode $q \in Q$ with a *vector field* $F_q : X \times U \times W \to (\mathscr{T} X)$ that forms the RHS of the ODE $\dot{x}(t) = F_q(\mathbf{x}, \mathbf{u}, \mathbf{w})$. $F_q$ is assumed to be Lipschitz continous over $\mathbf{x}$ and continous over the remaining variables.
4. A *guard map* $G$ that associates a *guard* condition $G(q_i, q_j)$ with each transition, which is also a predicate over $\{\mathbf{x}, \mathbf{y}\}$ like the invariant.
5. A *reset map* $R$ that associates each transition with an *update function* $R_{(q_i, q_j)} : X \to X$.

The set of initial conditions for a HA is given by the initial states of the variables $(\mathbf{x}_0, \mathbf{y}_0) \in X \times Y$, an initial location $q_0 \in Q$, the control inputs $\mathbf{u} : [0, T] \to U$, and the disturbance inputs $\mathbf{w} : [0, T] \to W$, where $T > 0$ is a time horizon for an execution of the HA.

This definition of hybrid automaton was adapted from the definition presented in [5], which is also found across literature. Although, a slight pecularity of our definition is the introduction of the vector of *input system variables* $\mathbf{y}$. We explicitly distinguish these from the other system variables because these are variables which are observed as *inputs* within our system and do not explicitly evolve in our system. In particular, the vector field at each location do not describe the dynamics of the input variables and the reset functions do not reset these variables. These are called I/O hybrid automata in some literature, but we simply refer to them as hybrid automata for the sake of brevity.

Next, we also need to define the set of execution traces of a hybrid automata which captures the set of evolutions of a system modeled by a hybrid automaton. The following definition is again inspired from [5].

**Definition 2** (Set of execution traces in a hybrid automaton)**.** Let $\mathscr{H} : (L, E, I, F, G, R)$ be a hybrid automaton and let $T > 0$ be a time horizon of execution. Further, let $\mathbf{u} : [0, T] \to U$ be a control signal input and $\mathbf{w} : [0, T] \to W$ be a disturbance input. The rest of the initial conditions of the LHA are given by discrete location of the automaton $q_i \in Q$ and an initial state of the variables $(\mathbf{x}_0, \mathbf{y}_0) \in X \times Y$. We can then define the semantic of an execution trace on a LHA as a sequence of jumps and flows:

- A flow $(q, \mathbf{x}, \mathbf{y}, t) \rightsquigarrow (q, \mathbf{x}', \mathbf{y}', t + \delta)$ for some $\delta \geq 0$ is a solution to the ODE $\dot{\mathbf{x}} = F_q(\mathbf{x}, \mathbf{u}, \mathbf{w})$ starting from the initial condition $t_0 = t$ and $\mathbf{x}(t)$. Since $F_q$ is Lipschitz continous over $\mathbf{x}$, the trajectory is uniquely defined. The invariant is also expected to hold throughout the flow period, i.e., for all $\lambda \in [t, t + \delta]$,

$$(x_1(\lambda), \ldots, x_n(\lambda), y_1(\lambda), \ldots, y_m(\lambda)) \models I(q)$$

- A jump
$$(q, \mathbf{x}, \mathbf{y}, t) \rightarrow (q', \mathbf{x}', \mathbf{y}', t)$$
is an instantaneous transition from the discrete location $q$ to $q'$, where $(q, q') \in E$ and
$$(x_1(t), \ldots, x_n(t), y_1(t), \ldots, y_m(t)) \models G(q, q')$$
where $G(q, q')$ is the guard set of the transition. Lastly, we obtain $\mathbf{x}' = R_{(q,q')}(\mathbf{x})$, by applying the reset map of the transition.

A sequence of these jumps and flows over a time horizon $T > 0$ defines an execution trace of the given LHA. For a given time horizon $T > 0$, and for a given set of initial conditions of the LHA, we can define the set of execution traces accepted by the HA $\mathscr{H}$ and we denote this set by $\Gamma_{\mathscr{H}}$.

Lastly, we give define timed automata which represent a subclass of hybrid automata where all the system variables are *clocks*. This means that the system variables all evolve at the same constant rate.

**Definition 3** (Timed automaton)**.** A hybrid automaton $\mathscr{T} : (L, E, I, F, G, R)$ is said to be a timed automaton if at each location of the automaton, the dynamics are defined by $\dot{\mathbf{x}} = 1, \dot{\mathbf{y}} = 1$. Also, for each $(q_i, q_j) \in L$ and for each $x \in \{\mathbf{x}\}$, $R_{(q_i,q_j)}(x) = x$ or 0, i.e., all the resets are restricted to resetting the clocks (variables) to 0 or not having any effect on them.

## 2 Lazy hybrid automata

We now have everything required to extend the classical definition of a hybrid automaton to capture non-deterministic bounded delays on the observation of state variables.

**Definition 4** (Lazy hybrid automaton in the continuous semantic[1])**.** Consider a vector of system variables $\mathbf{x} = (x_1, \ldots, x_n) \in X$, a vector of *input system variables* $\mathbf{y} = (y_1, \ldots, y_n) \in Y$, control inputs $\mathbf{u} \in U$ and disturbances $\mathbf{w} \in W$. Then, a lazy hybrid automaton (LHA) is a tuple $\mathscr{H} : (L, E, I, F, G, B, R)$ consisting of:

1. A finite set of *locations* $Q = \{q_1, \ldots, q_k\}$ and *transitions* that form edges between states $E \subseteq Q \times Q$.
2. A map $I$ that associates with each state $q \in Q$ an *invariant* $I(q)$ which is a predicate over the variables in $\{\mathbf{x}, \mathbf{y}\}$.
3. A map $F$ that associates each mode $q \in Q$ with a *vector field* $F_q : X \times U \times W \rightarrow (\mathscr{T}X)$ that forms the RHS of the ODE $\dot{x}(t) = F_q(\mathbf{x}, \mathbf{u}, \mathbf{w})$. $F_q$ is assumed to be Lipschitz continous over $\mathbf{x}$ and continous over the remaining variables.
4. A *guard map* $G$ that associates a *guard* condition $G(q_i, q_j)$ with each transition, which is also a predicate over $\{\mathbf{x}, \mathbf{y}\}$ like the invariant.
5. A map of *time bounds* which associates each system variable in $\{\mathbf{x}, \mathbf{y}\}$ to a pair of non-negative reals $(l_{x_i}, u_{x_i})$ or $(l_{y_i}, u_{y_i})$ respectively. These bounds define the time interval that we are allowed to look backward with respect to each system variable in $\{\mathbf{x}, \mathbf{y}\}$ to evaluate its value.
6. A *reset map* $R$ that associates each transition with an *update function* $R_{(q_i,q_j)} : X \rightarrow X$.

The set of initial conditions for a LHA is given by the initial states of the variables $(\mathbf{x}_0, \mathbf{y}_0) \in X \times Y$, an initial location $q_0 \in Q$, the control inputs $\mathbf{u} : [0, T] \rightarrow U$, and the disturbance inputs $\mathbf{w} : [0, T] \rightarrow W$, where $T > 0$ is a time horizon for an execution of the LHA.

[1]We refer to our definition as lazy hybrid automaton in the continuous semantic to distinguish it from the lazy hybrid automaton in the discrete semantic defined in [1] as explained in the introduction.

Note that the vector of *input* variables are system variables which evolve in other systems and appear to us as inputs. Also, we introduce delays in the observation of the system variables to represent communication delays in the observation of these variables.

Corresponding to the delays, we also introduce **new semantics for guard and invariant evaluation** along with this new definition of a lazy hybrid automaton (LHA) which extends the definition of a standard hybrid automaton. Before introducing these new semantics, we define the *timings map*.

**Definition 5** (Timings map)**.** Given a LHA $\mathscr{H}$, and for a given time $t \geq 0$, we can define a *timings map* $\tau_t$ which associates every system variable in $s \in \{\mathbf{x}, \mathbf{y}\} = \{x_1, \ldots, x_n, y_1, \ldots, y_m\}$ with a time in the interval $\tau(s) \in [\max(t - l_s, 0), \max(t - u_s, 0)]$.

Note that we consider a maximum with zero here within the intervals because we only want to consider positive times.

## 2.1 Execution traces in a lazy hybrid automaton

Next, we also need to extend the semantic of the execution traces by making use of the new addition of the time bounds within the time bounds. With this, we also define the new *delayed* semantics of guard and invariant evaluation.

**Definition 6** (Set of execution traces in a lazy hybrid automaton)**.** Let $\mathscr{H} : (L, E, I, F, G, B, R)$ be a lazy hybrid automaton and let $T > 0$ be a time horizon of execution. Let $\mathbf{u} : [0, T] \to U$ be a control signal input and $\mathbf{w} : [0, T] \to W$ be a disturbance input. We need an initial discrete location of the automaton $q_i \in Q$ and an initial state of the variables $(\mathbf{x}_0, \mathbf{y}_0) \in X \times Y$. An execution trace of a LHA is a sequence of jumps and flows:

- A flow $(q, \mathbf{x}, \mathbf{y}, t) \rightsquigarrow (q, \mathbf{x}', \mathbf{y}', t + \delta)$ for some $\delta \geq 0$ is a solution to the ODE $\dot{\mathbf{x}} = F_q(\mathbf{x}, \mathbf{u}, \mathbf{w})$ starting from the initial condition $t_0 = t$ and $\mathbf{x}(t)$. Since $F_q$ is Lipschitz continous over $\mathbf{x}$, the trajectory is uniquely defined. The invariant is also expected to hold throughout the flow period, i.e., for all $\lambda \in [t, t + \delta]$, there exists a timings map $\tau_\lambda$ at time $\lambda$, such that

$$(x_1(\tau_\lambda(x_1)), \ldots, x_n(\tau_\lambda(x_n)), y_1(\tau_\lambda(y_1)), \ldots, y_m(\tau_\lambda(y_m))) \models I(q)$$

- A jump

$$(q, \mathbf{x}, \mathbf{y}, t) \xrightarrow{\tau_t} (q', \mathbf{x}', \mathbf{y}', t)$$

  is an instantaneous transition from the discrete location $q$ to $q'$, where $(q, q') \in E$ and the timings map $\tau_t$ satisfies

$$(x_1(\tau_t(x_1)), \ldots, x_n(\tau_t(x_n)), y_1(\tau_t(y_1)), \ldots, y_m(\tau_t(y_m))) \models G(q, q')$$

  where $G(q, q')$ is the guard set of the transition. Lastly, we have $\mathbf{x}' = R_{(q,q')}(\mathbf{x})$.

A sequence of these jumps and flows over a time horizon $T > 0$ defines an execution trace of the given LHA. For a given time horizon $T > 0$, and for a given set of initial conditions of the LHA, we can define the set of execution traces accepted by a LHA $\mathscr{H}$ and we denote this set by $\Gamma_{\mathscr{H}}$.

## 2.2 Understanding lazy hybrid automata

When we *extended* the semantic of a hybrid automaton into a lazy hybrid automaton, the only addition are the time bounds associated to each system variable. Essentially, these time bounds indicate the bounds on the delays in observation of the state variable. We present an example LHA here to better understand how these time bounds work.

**Example 1** (Oversimplified cruise control)**.** Consider two cars on a straight road with an orange car following a green car. We are trying to automatically adjust the speed of the orange car to avoid collision with the green car. Let us assume that the green car is communicating its position and velocity to the orange car which has an associated communication delay. We model the green car using a lazy hybrid automaton (Fig.

2). Firstly, we have two system variables $\mathbf{x} = (x_1, v_1)$ which represent the position and velocity of the orange car. Along with this, we have two input variables $\mathbf{y} = (x_2, v_2)$ which represent the leading green car, whose evolution we do not know. We only observe their values as inputs. The lazy hybrid automaton then has the following features:

1. Two locations $\{\texttt{accelerate}, \texttt{decelerate}\}$ and two transition edges between the locations, forward and backward.
2. The invariant at `accelerate` is given by $v_2 \geq 10$ and at `decelerate` by $v_2 < 10$.
3. The vector field at `accelerate` is $\dot{x}_1 = v_1, \dot{v}_1 = 3$, and at `decelerate` is $\dot{x}_1 = v_1, \dot{v}_1 = 0$.
4. The guard on the transition from `accelerate` to `decelerate` is $v_2 < 10$ and that from `decelerate` to `accelerate` is $v_2 \geq 10$.
5. We have upto a 2s delay in observation of $v_2$, the velocity of the leading car, i.e., the time bound associated to $v_2$ is $(2, 0)$.
6. There are no resets.

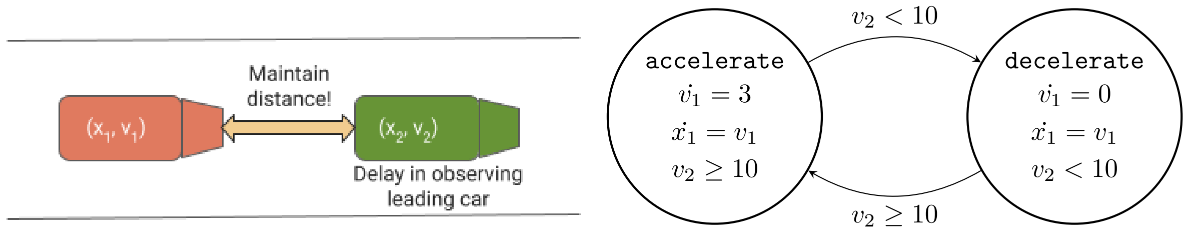

Figure 2: LHA for an oversimplified cruise control problem

Making use of this simple LHA, let us try to build an intuition for how the time bounds work and help us capture communication delays. Firstly, it is important to convince oneself that a delay in observing a variable corresponds to observing a *past* value of a variable. In this example, the 2 seconds observational delay on $v_2$ means that at any given point in time, there exists at least one value of $v_2$ successfully transmitted to the green car within the past 2 seconds. In other words, this means that the green car *knows* at least one value held by $v_2$ in the past 2 seconds. This represents the non-determinism of the delay that we are trying to model: we can have upto a 2 second delay in the observation of $v_2$, but we do not know the exact delay involved.

It is then evident why the semantic of the guard and invariant evaluation have to be adapted. Since we are not assured of knowing the *current* value of a variable, we allow the guard and invariant predicates to be evaluated on past values of a variable. This is a weaker invariant than with a classical hybrid automaton where the invariant is expected to be true at a given location for the current values of the variables. We cannot do better than this because the hybrid system (here, the green car) only has one observed value of $v_2$ in the past 2 seconds. When we have more variables in the invariants, we cannot expect all of the variables to have been recorded at the same time instant. Thus, we also allow variables to be evaluated at different instants of time. For example, if we had an invariant depending on $v_2$ and $x_2$ at one of the locations, then the invariant would be satisfied at some time $t$ if there exist $t_1 \in [t-2, t], t_2 \in [t-2, t]$ such that $(x_2(t_1), v_2(t_2)) \models I$. This is exactly the notion captured by the timings map.

Let us make this more clear by looking an execution trace of the LHA in Example 1. To do this, we need to define some initial conditions. Suppose we are initially at the `accelerate` location with $x_1 = v_1 = 0$. We also fix the dynamics of $v_2$ to be described by $v_2(t) = 15 - t$ for $t \in [0, 10]$. With this, we have the following execution trace which is accepted by the LHA.

$$\begin{pmatrix} t=0 \\ loc=\texttt{accelerate} \\ v_1=0 \\ x_1=0 \\ v_2=15 \end{pmatrix} \rightsquigarrow \begin{pmatrix} t=6 \\ loc=\texttt{accelerate} \\ v_1=18 \\ x_1=54 \\ v_2=9 \end{pmatrix} \xrightarrow{v_2(6-0.5)=9.5<10} \begin{pmatrix} t=6 \\ loc=\texttt{decelerate} \\ v_1=18 \\ x_1=54 \\ v_2=9 \end{pmatrix}$$

To check that this is indeed an execution trace of our LHA, we need to check that the invariants and guard conditions are indeed satisfied. Within the flow from $t = 0$ to $t = 6$, we need to check that the invariant at `accelerate` is satisfied for each $\lambda \in [0, 6]$. For each $\lambda \in [0, 5]$, we already have that $v_2(\lambda) = 15 - \lambda \geq 10$. So the construction of the timings map is simple: we can just assign $\tau_\lambda(v_2) := \lambda$ which indeed satisfies the time bounds of $v_2$. However, for each $\lambda' \in [5, 6]$, we have that $v_2(\lambda') < 10$. But since we are allowed to have a delayed evaluation of $v_2$ upto 2 seconds, we can define $\tau_{\lambda'}(v_2) := \lambda' - 1 \in [\lambda' - 2, \lambda']$ because $v_2(\lambda' - 1) \geq 10$. The invariant continues to stay true at `accelerate` even after the value of $v_2$ goes below 10.

Lastly, we need to check that there exists a timings map which can satisfy the discrete location switch at $t = 6$. Although it is not stricly required, we can again make use of the delayed evaluation of $v_2$ to define the timings map for the transition to be $\tau_6(v_2) = 6 - 0.5$. Indeed, $v_2(6 - 0.5) = 9.5 < 10$ which satisfies both the time bounds on $v_2$ and the guard condition. We deduce that this is an execution trace of the LHA in Example 1.

## 2.3 Parallel composition of lazy hybrid automata

Given two lazy hybrid automata which might share input variables with each other we would like to be able to define the parallel composition of these two automata resulting in one single LHA. *Sharing input variables* amounts to the fact that there might be some system variables which evolve in one LHA that might appear as inputs in the other LHA. It is important to define this parallel composition because we are trying to model multi-agent systems as hybrid systems. When we combine parallelly running hybrid systems communicating with each other, we would like to be able to view the composition as one single hybrid system. The parallel composition procedure has been described in the appendix.

# 3 Reduction of lazy hybrid automata

In this section we explore some subclasses of lazy hybrid automata and show how they can be translated into regular hybrid automata.

## 3.1 Translation of lazy hybrid automata without invariants

The first subclass we consider are referred to as LHA*.

**Definition 7** (LHA*)**.** A LHA $\mathscr{H}^* : (L, E, I, F, G, B, R)$ is a LHA* if it satisfies the following conditions:

- *Invariants*: The invariants of $\mathscr{H}^*$ at every location given by $I(q)$ is **only** a predicate over the variables in $\mathbf{x}$ and $\mathbf{y}$ which do not have any associated delays, i.e., the set of state variables $s$ such that $B(s) = (0, 0)$.
- *Strongly non-zeno*: There exists an upper bound $K \in \mathbb{N}$ on the number of transitions that can be taken in any given time period of length $\max_{i \in \{s \in \{\mathbf{x}, \mathbf{y}\}\}}(l_s - u_s)$ where $(l_s, u_s)$ are the time bounds associated to the input variables. We refer to the length of this time period as the *maximum delay period.*

To define the translation of LHA*, we define a function $\Phi$ which takes a LHA* (as described above) and produces a translated standard hybrid automaton. We define the contruction of $\Phi$ first and then proceed to prove that the translation indeed preserves the set of execution traces accepted by the two automata.

Given a LHA* $\mathscr{H} : (L, E, I, F, G, B, R)$, $\Phi(\mathscr{H})$ is a standard hybrid automaton given by

$$\Phi(\mathscr{H}) = \mathscr{H}' \Big\|_{s\in S} \mathscr{S}_s$$

where

$$S = \{s|B(s) \neq (0,0), s \in \{\mathbf{x}, \mathbf{y}\}\}$$

which is the set of system variables which do have an associated delay. Here we have a parallel composition of multiple standard hybrid automata which are described here.

- For each system variable $s$ such that $B(s) \neq 0$, we construct a standard hybrid automaton $\mathscr{S}_s$ which has the following properties:
  - It has $2K$ system variables

    $$\{\sigma_{(s,1)}, \theta_{(s,1)}, \ldots, \sigma_{(s,K)}, \theta_{(s,K)}\}$$

    where we recall that $K$ is the upper bound on the number of transitions within the maximum delay period, a property of a LHA*.
  - It has $K$ locations $\{\lambda_{(s,1)}, \ldots, \lambda_{(s,K)}\}$ and $K$ edges

    $$\{(\lambda_{(s,1)}, \lambda_{(s,2)}), (\lambda_{(s,2)}, \lambda_{(s,3)}), \ldots, (\lambda_{(s,K)}, \lambda_{(s,1)})\}$$
  - There are no invariants in $\mathscr{S}_s$.
  - At each location, each of the variables have the same dynamics given by,

    $$\begin{aligned}\dot{\sigma}_{(s,i)} &= 0, i \in \{1, \ldots K\},\\ \dot{\theta}_{(s,i)} &= 0, i \in \{1, \ldots K\}\end{aligned}$$

    From this, we see that each $\sigma_{(s,i)}$ and $\theta_{(s,i)}$ are simply static variables.
  - There are no guard conditions on the transition edges.
  - At each edge $(\lambda_{(s,i)}, \lambda_{(s,i+1)})$, we have the following reset on the variables

    $$\begin{aligned}\sigma_{(s,i)} &:= s(t)\\ \theta_{(s,i)} &:= t\end{aligned}$$

  With the resets here, we see that $\mathscr{S}_s$ acts like an automaton which takes transitions independently of the rest of the system and records the value of the system variable $s$ at random points of time. It also stores the time at which the value was recorded. It can record upto $K$ values of $s$ which is sufficient for us since this is the upper bound on the number of transitions in the maximum delay period.
- The standard hybrid automaton $\mathscr{H}'$ has the following properties:
  - The set of system variables and input variables remains the same as in $\mathscr{H}$.
  - The set of locations and edges remains the same.
  - Again, the dynamics within each location remains the same as well.

– The invariants remain the same as the LHA* which is a predicate over the variables without delays.

– For the guard predicate at a given edge $G(q_i, q_j)$, we need to modify all the occurences of the system variables which have a delay in observation. For each predicate which contains a system variable with a delay, we need to check if any of the recorded values satisfy the guard. For each variable $s$ which has a delay, we need to replace all its occurences within the guard with

$$(\theta_{(s,1)} \in [t - l_s, t - u_s], \sigma_{(s,1)}) \vee \cdots \vee (\theta_{(s,K)} \in [t - l_s, t - u_s], \sigma_{(s,K)})$$

For example, if we had a clause of the form $s \leq 2$, then we would translate it into

$$(\theta_{(s,1)} \in [t - l_s, t - u_s], \sigma_{(s,1)} \leq 2) \vee \cdots \vee (\theta_{(s,K)} \in [t - l_s, t - u_s], \sigma_{(s,K)} \leq 2)$$

It is easy to see how this can be generalized to larger clauses with many delayed variables, where we would need to similarly check if any combination of the recorded variables satisfy the guard condition.

This concludes the construction of

$$\Phi(\mathscr{H}) = \mathscr{H}' \Big\|_{s \in S} \mathscr{S}_s$$

Next, before moving on to the proof of the fact that the function $\Phi$ does indeed construct a valid translation, we need to have a way to compare the execution traces of $\mathscr{H}$ and $\Phi(\mathscr{H})$. To this end, we define a projection function which *projects* an execution trace of $\Phi(\mathscr{H})$ onto an execution trace of $\mathscr{H}$. The projection is defined in the appendix along with a proof of the following theorem.

**Theorem 1** (Translation from LHA* to HA)**.** *For every LHA* $\mathscr{H} : (L, E, I, F, G, B, R)$ *and a given set of initial conditions for* $\mathscr{H}$*, we have*

$$\Gamma_{\mathscr{H}} = \Pi\left(\Gamma_{\Phi(\mathscr{H})}\right)$$

*or in other words, the translation given by* $\Phi(\mathscr{H})$ *represents a bisimulation of* $\mathscr{H}$ *and they both accept the same set of execution traces (which is the notion of equivalence that we have considered).*

### 3.1.1 Example of a translation from LHA* to HA

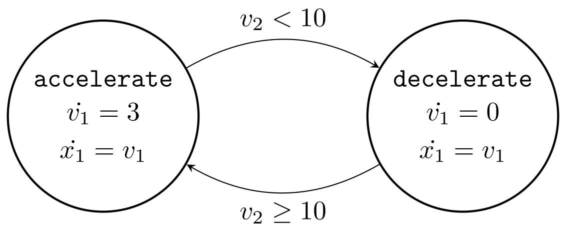

Figure 3: LHA* of the oversimplified cruise control model

**Example 2** (Translation of LHA*)**.** We consider the same LHA as in Example 1, but we remove the invariants because they cannot be translated. We also impose the strongly non-zeno condition by setting an upper bound on the number of transitions to be $K = 3$ within the maximum delay period of $2s$. The LHA without invariants can be seen in Fig. 3. The translation is presented in Fig. 4. It consists of a parallel composition of two hybrid automata. The translated guard conditions are given by

$$G_1 = (t - \theta_{(v_2,1)} < 2 \wedge \sigma_{(v_2,1)} < 10) \vee (t - \theta_{(v_2,2)} < 2 \wedge \sigma_{(v_2,2)} < 10) \vee (t - \theta_{(v_2,3)} < 2 \wedge \sigma_{(v_2,3)} < 10)$$

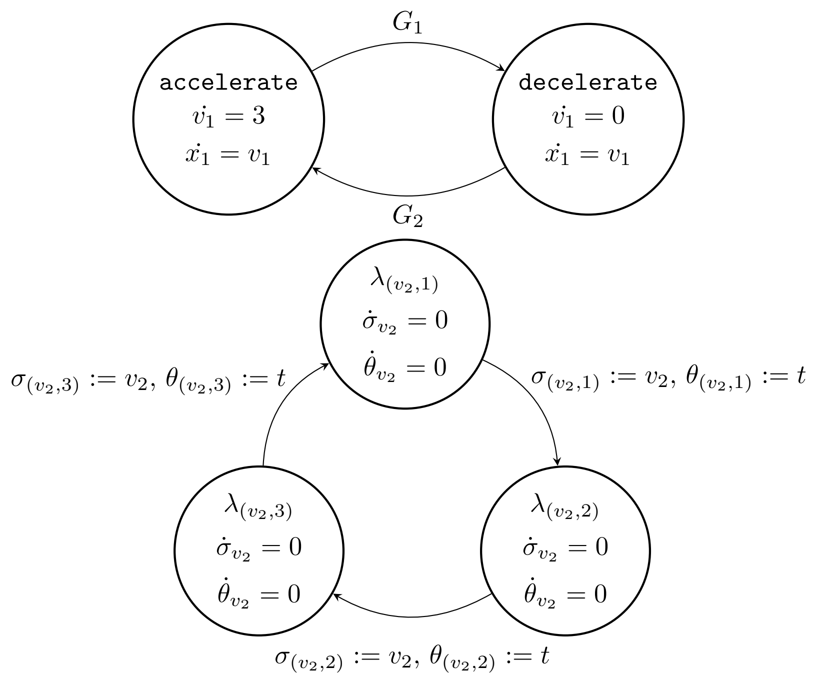

Figure 4: Translation of the LHA* consisting of a parallel composition of two automata

and similarly for $G_2$. It is easy to understand why this translation works: the parallely running *saving* automaton saves the state of $v_2$ at some points in time. Then, these saved values are used to evaluate the guard conditions on the past values of $v_2$. We never need more than 3 saved values due to the upper bound on the number of transitions.

## 3.2 Translation of lazy timed automata *with* invariants

Timed automata are hybrid automata with the exception that all the state variables are *clocks* which means that each of them evolve at the same constant rate of 1. We can extend the lazy definition to this simpler class of timed automata as well to allow for delays in the observation of these *clock* variables. As we will show in this section, this restriction on the evolution on the variables is sufficient to allow us to also maintain invariants within the translation into regular timed automata.

We define the new subclass of LHA which we call lazy timed automata (LTA*).

**Definition 8** (LTA*)**.** A LHA $\mathscr{T}* : (L, E, I, G, B, R)$ is a LTA* if it satisfies the following conditions:

- *Input variables:* There are no input variables within the automaton, or $\mathbf{y}$ is empty. All variables are system variables and evolve in the same automaton.
- *Vector field:* We remove the vector fields at each location since each variable is only a clock and it evolves at a constant rate of 1. Or, this equivalently means that at each location, the differential equation is simply given by

$$\dot{\mathbf{x}} = 1$$

- *Reset functions:* We follow the convention that all the resets of clocks happen to zero. In other words, if there is a reset of a variable $s$ on a given edge, it has the form $s := 0$.
- *Strongly non-zeno:* Similar to LHA*, there exists an upper bound $K \in \mathbb{N}$ on the number of transitions that can be taken in any given time period of length $\max_{i \in \{s \in \{\mathbf{x},\mathbf{y}\}\}}(l_s - u_s)$ where $(l_s, u_s)$ are the time

bounds associated to the input variables. We refer to the length of this time period as the *maximum delay period.*

The key difference with LTA* is that we retain invariants on variables with delays as compared to LHA* where it is the contrary.

In a similar spirit as before, we define a translation function $\Phi$ which takes a LTA*, and produces a standard timed automaton. Given a LTA* denoted by $\mathscr{T} : (L, E, I, G, B, R)$, the translated timed automaton $\Phi(\mathscr{T})$ is given by

$$\Phi(\mathscr{T}) = \mathscr{T}' \Big\|_{s \in \mathbf{S}} \mathscr{S}_s$$

where

$$S = \{s | B(s) \neq (0,0), s \in \{\mathbf{x}\}\}$$

which is the set of system variables which do have an associated delay. Here we have a parallel composition of multiple standard timed automata. We describe each of these here

- For each system variable $s$ such that there is a non-zero associated delay, we have a timed automaton $\mathscr{S}_s$ which has a similar function as the previous translation: to store the value of $s$ at different points. However, since $s$ is just a clock, it is sufficient to only store the times at which there was a reset on $s$. $\mathscr{S}_s$ has the following properties:
  - It has $K+1$ system variables (or clocks in this case)

    $$\{s_1, \ldots, s_{K+1}\}$$

  - It has $K+1$ locations $\{\lambda_{(s,1)}, \ldots, \lambda_{(s,K+1)}\}$ and $K+1$ *named* edges

    $$\{(\lambda_{(s,1)}, \lambda_{(s,2)}, \texttt{save}_s), (\lambda_{(s,2)}, \lambda_{(s,3)}, \texttt{save}_s), \ldots, (\lambda_{(s,K+1)}, \lambda_{(s,1)}, \texttt{save}_s)\}$$

    We use named edges here in order to synchronize transitions within a parallel composition of timed automata.
  - There are no invariants in $\mathscr{S}_s$.
  - There are no guard conditions on the transition edges.
  - At each edge $(s_i, s_{i+1})$, we have the following reset on the variables

    $$s_i := 0$$

- Next, we need to describe $\mathscr{T}' : (L', E', I', G', R')$ in $\Phi(\mathscr{T})$:
  - The set of clocks $\mathbf{x}$ is the same as $\mathscr{T}$.
  - The set of locations is also the same, or $L' = L$. With the edges, we add zero or more names depending on the clocks which are reset on this edge given by the reset function on this edge. For each edge $(q_i, q_j) \in E$, we add the corresponding named edge $(q_i, q_j, \{\texttt{save}_{s_1}, \ldots, \texttt{save}_{s_r}\})$ to $E'$, where $\{s_1, \ldots, s_r\}$ are the clocks which are reset by the reset function $R$ of this edge.

    These names allow synchronizations to take place with the parallely running timed automata so that a transition takes place in $\mathscr{S}_s$ if $s$ is reset on an edge. Note that this would never force a transition in $\mathscr{H}'$ because of the absence of guards or invariants in $\mathscr{S}_s$.

– The invariant at each location $I(q)$ has to be transformed to make use of the saved variables. At a given location $q$, the invariant $I(q)$ is composed of conjunctions of clauses of the form

$$\sum_{s\in\{\mathbf{x}\}} c_s s \bowtie C, \quad \bowtie\in \{=,\geq,>,\leq,<\}, c_s, C \in \mathbf{R}$$

We translate each clause to be

$$\sum_{s\in\{\mathbf{x}\}} V_s \bowtie C$$

where $V_s \subseteq \mathbf{R}$ is the set of values traversed by the clock $s$ in its associated delay period $B(s) = (l_s, u_s)$, given by

$$\begin{aligned} V_s =&[\max(s-l_s,0), s-u_s] \\ &\cup [\max(s_1-l_s,0), \min(s_1-s_2, s_1-u_s)] \\ &\cup [\max(s_2-l_s,0), \min(s_2-s_3, s_2-u_s)] \\ &\vdots \\ &\cup [\max(s_{K+1}-l_s,0), \min(s_{K+1}-s_1, s_{K+1}-u_s)] \end{aligned}$$

Since there can be at most $K$ resets of $s$ within the maximum delay period, it is clear that $V_s$ represents the set of values taken on by $s$ in the time interval $[t-l_s, t-u_s]$.

Notice that within the translated clause, we are considering sums and comparisons of sets. We endow the subsets of $\mathbf{R}$ with the usual addition of sets on the reals. For the comparison, we say that a subset $V \subseteq \mathbf{R}$ satisfies $V \bowtie C$ for $C \in \mathbf{R}$ if there exists $v \in V$ such that $c \bowtie C$, where $\bowtie\in \{=,\geq,>,\leq,<\}$.

We thus define the translated invariant $I'(q)$ to be the conjunction of the translations of each the clauses within $I(q)$ as shown above.

It is also easy to see that since each $V_s$ is a union of sums of intervals, the translated clause $\sum_{s\in\{\mathbf{x}\}} V_s \bowtie C$ can be rewritten as a simple predicate over the system variables with disjunctions and conjunctions present within it.

– We apply a similar translation as the invariants to each of the guard predicates associated to edges to obtain $G'(q_i, q_j)$ for each edge $(q_i, q_j) \in E$.

– The resets remain the same $R' = R$.

Now that we have constructed the translation $\Phi(\mathscr{T})$, we once again need a way to compare its execution traces with the original LTA* $\mathscr{T}$. We construct the projection map $\Pi : \Gamma_{\Phi(\mathscr{T})} \to \Gamma_{\mathscr{T}}$ which takes an execution trace in $\Phi(\mathscr{T})$ and projects it onto an execution trace of $\mathscr{T}$. The construction of this map as well as the proof of the following theorem can be found in the appendix.

**Theorem 2** (Translation from LTA* to TA)**.** *For every LTA* $\mathscr{T} : (L, E, I, G, B, R)$, and a given set of initial conditions for $\mathscr{T}$, we have*

$$\Gamma_{\mathscr{T}} = \Pi(\Gamma_{\Phi(\mathscr{T})})$$

*or, the translation given by $\Phi(\mathscr{T})$ represents a bisimulation of $\mathscr{T}$ and they both accept the same set of execution traces, upto a projection.*

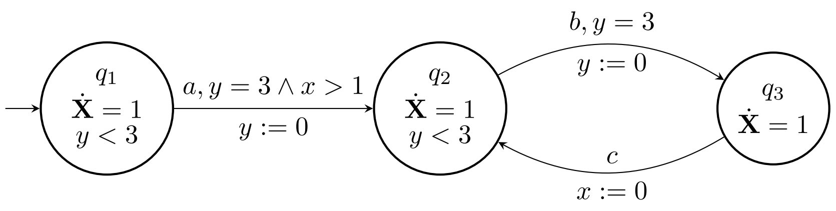

Figure 5: Lazy timed automaton example

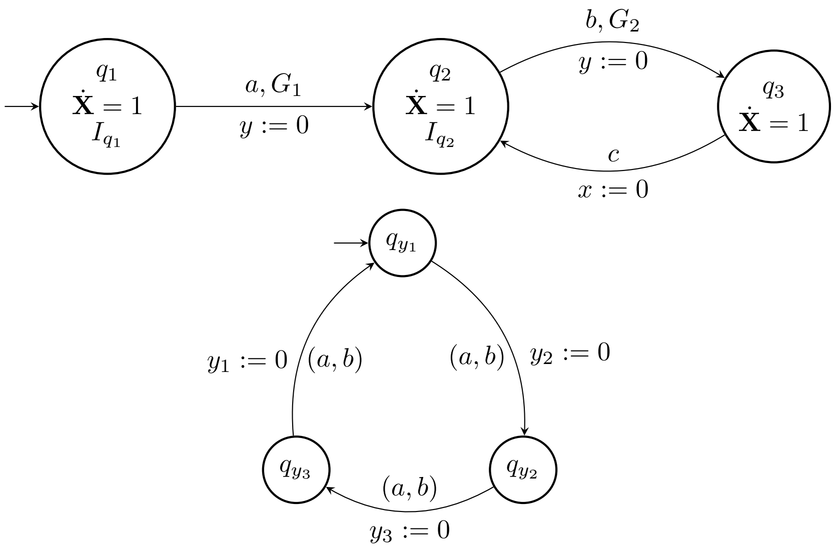

Figure 6: LTA* translation consisting of parallel composition of two timed automata with label synchronized transitions

### 3.2.1 Example of a translation from LTA* to TA

**Example 3** (Translation of LTA)**.** Consider the lazy timed automaton with labelled transitions in Fig. 5. We consider a 10s delay on the observation of the value of the clock $y$. We also impose an upper bound of $K = 2$ transitions within the maximum delay period of 10s. Its translation is presented in Fig. 6 with the new variables to save the states of $y$ when it is reset. Here, we have two timed automata parallely composed with each other with label synchronized transitions, i.e., whenever the main automaton makes a reset on the clock $y$, we record the time of reset in the save automaton by making a synchronized transition. With the help of these saved values, we are able to determine every value taken by the clock $y$ in the past 10s. Also with timed automata, both the guards and the invariants can be preserved. For example, in the translated guard condition $G_2$, we would need to translate the predicate $y = 3$. To do this, we need to check whether $y = 3$ at some point in the past 10s. By using the described translation of predicates, we obtain that

$$\begin{aligned} G_2 = \quad & (y_1 > y_2) \wedge (3 \in [\max(y_1 - 10, 0), y_1 - y_2]) \\ & \vee (y_2 > y_3) \wedge (3 \in [\max(y_2 - 10, 0), y_2 - y_3]) \\ & \vee (y_3 > y_1) \wedge (3 \in [\max(y_3 - 10, 0), y_3 - y_1]) \\ & \vee (3 \in [\max(y - 10, 0), y]) \end{aligned}$$

## 4 Experimentation

To test out a practical implementation of our translation of lazy timed automata, we chose the Generalized Rail Crossing (GRC) timed automaton benchmark presented in [7]. We introduced delays on the observation of the clock of the train by the controller of the gate. Following this, we constructed the translation using our procedure for lazy timed automata and then used the Hycomp model checker [4]. However, since we introduce disjunctions within the invariants, it made them non-convex which could not be handled by the model checker. To handle this problem, we further replicated each location with a disjunctive invariant and distributed the disjunctive predicates among the replicated locations.

We then verified under which values of delays, the system continued to remain safe. In this case, safety means that the gate is always closed when a train is closing. Confirming our expectations, we were able to observe that for low values of observation delays, the system continued to be safe. However, as soon as the delay was raised beyond a threshold, the system was not safe anymore[2], i.e., the gate could not be closed in time due to the delays.

## 5 Conclusions

In this paper, we introduced the semantic of lazy hybrid automata in order to account for bounded non-deterministic delays on the observation of state variables. This can be particularly effective while dealing with communication delays which are inevitable in multi-agent settings. We also presented translations of restricted subclasses of these lazy hybrid automata into classical hybrid automata. A key idea which made the translations possible was the introduction of parallelly running automata which saved the value of the delayed variables a finite number of times. We remark that an important restriction is the upper bound on the number of transitions given by the strongly non-zeno condition. Lifting this assumptions immediately renders the translation impossible because we cannot *save* an infinite number of values of a variable within a hybrid automaton. This is an indication that lazy hybrid automata represent a strict superset of hybrid automata in terms of expressivity. However, this upper bound is not restrictive in reality. As we saw with the generalized rail crossing example, an upper bound can often be deduced easily and it is not unrealistic. In the future, we hope to study the effects of such delays within the dynamics of each location which would lead us towards differential delay equations (DDEs).

## References

[1] Agrawal, M. and Thiagarajan, P. S. 2004. Lazy rectangular hybrid automata. In Alur, R. and Pappas, G. J., editors, *Hybrid Systems: Computation and Control*, pages 1–15, Berlin, Heidelberg. Springer Berlin Heidelberg.

[2] Bai, Y., Gan, T., Jiao, L., Xia, B., Xue, B., and Zhan, N. 2021. *Switching Controller Synthesis for Delay Hybrid Systems under Perturbations.* Association for Computing Machinery, New York, NY, USA.

[3] Branicky, M., Phillips, S., and Zhang, W. 2000. Stability of networked control systems: explicit analysis of delay. In *Proceedings of the 2000 American Control Conference. ACC (IEEE Cat. No.00CH36334)*, volume 4, pages 2352–2357 vol.4.

[4] Cimatti, A., Griggio, A., Mover, S., and Tonetta, S. 2015. Hycomp: An smt-based model checker for hybrid systems. In Baier, C. and Tinelli, C., editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 52–67. Springer.

[5] Deshmukh, J. V. and Sankaranarayanan, S. 2019. *Formal Techniques for Verification and Testing of Cyber-Physical Systems.*

[2] `https://github.com/guruprerana/lazy-hybrid-automaton/blob/master/train_crossing_delays.hydi`

[6] Feng, S., Chen, M., Zhan, N., Fränzle, M., and Xue, B. 2019. Taming delays in dynamical systems. In Dillig, I. and Tasiran, S., editors, *Computer Aided Verification*, pages 650–669, Cham. Springer International Publishing.

[7] Fontana, P. and Cleaveland, R. 2020. Timed automata benchmark description. *CoRR*, abs/2005.13151.

[8] Gao, H., Meng, X., and Chen, T. 2008. Stabilization of networked control systems with a new delay characterization. *IEEE Trans. Autom. Control.*, 53(9):2142–2148.

[9] Goubault, E., Putot, S., and Sahlmann, L. 2018. Inner and outer approximating flowpipes for delay differential equations. In Chockler, H. and Weissenbacher, G., editors, *Computer Aided Verification*, pages 523–541, Cham. Springer International Publishing.

[10] Pallottino, L., Scordio, V. G., Bicchi, A., and Frazzoli, E. 2007. Decentralized cooperative policy for conflict resolution in multivehicle systems. *IEEE Transactions on Robotics*, 23(6):1170–1183.

[11] Pant, Y. V., Abbas, H., Mohta, K., Quaye, R. A., Nghiem, T. X., Devietti, J., and Mangharam, R. 2021. Anytime computation and control for autonomous systems. *IEEE Transactions on Control Systems Technology*, 29(2):768–779.

[12] Segata, M., Dressler, F., and Cigno, R. L. 2015. Jerk beaconing: A dynamic approach to platooning. In *2015 IEEE Vehicular Networking Conference, VNC 2015, Kyoto, Japan, December 16-18, 2015*, pages 135–142. IEEE.

[13] Xue, B., Mosaad, P., Fränzle, M., Chen, M., Li, Y., and Zhan, N. 2017. Safe over- and under-approximation of reachable sets for delay differential equations.

# Appendices

## A Parallel composition of lazy hybrid automata

Given two lazy hybrid automata which might share input variables with each other we would like to be able to define the parallel composition of these two automata resulting in one single LHA. *Sharing input variables* amounts to the fact that there might be some system variables which evolve in one LHA that might appear as inputs in the other LHA. It is important to define this parallel composition because we are trying to model multi-agent systems as hybrid systems. When we combine parallelly running hybrid systems communicating with each other, we would like to be able to view the composition as one single hybrid system.

In order to formally define this parallel composition, consider two LHA given by

- A vector of system variables for the first LHA $\mathbf{x} = (x_1, \ldots, x_k, s_1, \ldots, s_p)$, where $(s_1, \ldots, s_p)$ is the vector of system variables evolving in the first LHA which appear as inputs in the second LHA.

  The input variables for this LHA are $\mathbf{y} = (y_1, \ldots, y_m, s'_1, \ldots, s'_{p'})$. Again, $(s'_1, \ldots, s'_{p'})$ is the vector of system variables of the second LHA which appear as input variables in this first LHA.

  Lastly, the tuple describing the first LHA is $\mathscr{H} : (L, E, I, F, G, B, R)$.

- Similarly, the vector of system variables for the second LHA would be $\mathbf{x}' = (x'_1, \ldots, x'_{k'}, s'_1, \ldots, s'_{p'})$, and the vector of input variables, $\mathbf{y}' = (y'_1, \ldots, y'_{m'}, s_1, \ldots, s_p)$.

  The tuple is $\mathscr{H}' : (L', E', I', F', G', B', R')$.

The first step towards defining the parallel composition is to introduce copy variables for $(s_1, \ldots, s_p)$ and $s'_1, \ldots, s'_{p'}$ which we denote by $(c_1, \ldots, c_p)$ and $(c'_1, \ldots, c'_{p'})$ respectively. When we say *copy* variables, we mean that the new variables have the exact same dynamics as the original variables. Now, we can define the parallel composition of $\mathscr{H} || \mathscr{H}'$ as the LHA defined over the system variables

$$\mathbf{x}_c = (x_1, \ldots, x_k, s_1, \ldots, s_p, c_1, \ldots, c_p, x'_1, \ldots, x'_{k'}, s'_1, \ldots, s'_{p'}, c'_1, \ldots, c'_{p'})$$

along with the tuple of the parallel composition $\mathscr{H}_c : (L_c, E_c, I_c, F_c, G_c, B_c, R_c)$ where

- The new vector input variables is given by

  $$\mathbf{y}_c = (y_1, \ldots, y_m, y'_1, \ldots, y'_{m'})$$

  Here, we remove the shared system variables which appeared previously as inputs. However, they are no longer inputs when we compose the two automata since they evolve in the composed LHA.
- The new set of locations is just $L_c = L \times L'$ and there exists an edge from $(q_a, q'_a) \in L_c$ to $(q_b, q'_b) \in L_c$ in $E_c$ if there is an edge $(q_a, q_b) \in E$ and $(q'_a, q'_b) \in E'$.
- For each location $(q, q') \in L_c$, we associate the invariant $I_c((q, q')) = I(q) \wedge I'(q')$. However, within $I(q)$, since there might be variables of the form $(s'_1, \ldots, s'_{p'})$, we replace these by their copies $(c'_1, \ldots, c'_{p'})$. Within $I'(q')$ we similarly replace all the variables of the form $(s_1, \ldots, s_p)$ with their copies $(c_1, \ldots, c_p)$. Note that these are the system variables of the LHA which appear as input variables to the other LHA. However, observe that we might still have variables of the form $(s_1, \ldots, s_p)$ in $I(q)$, but we do not replace these with their copies because these are system variables within $\mathscr{H}_1$ and there are no delays associated with these within $\mathscr{H}_1$.
- For the vector field at each location $(q, q')$, we have $\dot{\mathbf{x}}_c = (\dot{\mathbf{x}}, \dot{\mathbf{x}}') = (F(\mathbf{x}, \mathbf{u}, \mathbf{w}), F'(\mathbf{x}', \mathbf{u}', \mathbf{w}'))$. The only change required would be to ensure to make appropriate changes within the dynamics to ensure that the copies of the variables obtain the same dynamics as their counterparts, i.e., $(c_1, \ldots, c_p)$ all share the same differential equation as $s_1, \ldots, s_p$ and similarly with $(c'_1, \ldots, c'_{p'})$.
- The new guard conditions associated to each transition $((q_a, q'_a), (q_b, q'_b)) \in E_c$ is given by $G_c((q_a, q'_a), (q_b, q'_b)) = G(q_a, q_b) \wedge G'(q'_a, q'_b)$. However, similar to the invariants, we need to replace the variables $(s'_1, \ldots, s'_{p'})$ appearing in $G(q_a, q_b)$ with their copies and similarly with $G'(q'_a, q'_b)$.
- With respect to the new time bounds $B_c$, we need to have time bounds associated to each of the system and input variables.
  - For the variables $(x_1, \ldots, x_k, s_1, \ldots, s_p)$ and $(y_1, \ldots, y_m)$ we retain the same time bounds from $B$ in $\mathscr{H}$.
  - For the variables $(c'_1, \ldots, c'_{p'})$, we associate the time bounds associated to $(s'_1, \ldots, s'_{p'})$ in $B$ since these are input variables in $\mathscr{H}$.
  - For the variables $(x'_1, \ldots, x'_{k'}, s'_1, \ldots, s'_{p'})$ and $(y'_1, \ldots, y'_{m'})$ we retain the same time bounds associated with these variables in $B'$.
  - Lastly, for the variables $(c_1, \ldots, c_p)$, we associate the time bounds associated with $(s_1, \ldots, s_p)$ in $B'$ since these are input variables in $\mathscr{H}'$.
- Finally, for the reset maps, we would have for each transition $((q_a, q'_a), (q_b, q'_b)) \in E_c$, the associated reset map defined by

  $$R_c((q_a, q'_a), (q_b, q'_b))(\mathbf{x}, \mathbf{x}') = (R(q_a, q_b)(\mathbf{x}), R'(q'_a, q'_b)(\mathbf{x}'))$$

  However, we should also keep in mind to modify this reset function appropriately so as to also reset the copied variables to the same values as their originals.

The main reason why we need to introduce copy variables of our system variables within this parallel composition is because the shared system variables between the two LHA might have different time bounds (delays) associated with each of them. We need to retain both these different delays on the same system variables within the invariants and the guard conditions.

## B Proof of translation from LHA* to HA

We define a projection function which *projects* an execution trace of $\Phi(\mathscr{H})$ onto an execution trace of $\mathscr{H}$. The projection is defined in the appendix along with a proof of the following theorem.

The projection function is given by $\Pi : \Gamma_{\Phi(\mathscr{H})} \to \Gamma_{\mathscr{H}}$. Since we know that any execution trace $\gamma \in \Gamma_{\Phi(\mathscr{H})}$ is composed of a sequence of flows and jumps, it is sufficient to describe the projection of each of these components. Also, observe that $\Phi(\mathscr{H})$ is a parallel composition of many hybrid automata, we have that $\gamma$ would be composed of multiple parallel execution traces. However, we only need the execution trace of $\mathscr{H}'$ within $\Phi(\mathscr{H})$, which we denote by $\gamma_{\mathscr{H}'}$.

- A flow in $\gamma_{\mathscr{H}'}$ is of the form $(q, \mathbf{x}, \mathbf{y}, t) \rightsquigarrow (q, \mathbf{x}', \mathbf{y}', t + \delta)$. The corresponding flow in $\mathscr{H}$ is the same. Also, the invariant remains satisfied because the invariant at the location $q$ is only a predicate over the variables without delays (which is a property of LHA*).
- A jump in $\gamma_{\mathscr{H}'}$ is of the form $(q_i, \mathbf{x}, \mathbf{y}, t) \to (q_j, \mathbf{x}', \mathbf{y}, t)$. To obtain the corresponding jump in $\mathscr{H}$, we first need to construct a timings map $\tau_t$ which can satisfy the guard condition $G(q_i, q_j)$ at time $t$ with the delays on the variables. For each variable $s$ such that there is no associated delay, we set $\tau_t(s) := t$, i.e., we evaluate $s$ at the same time as the transition.

  Consequently, for each variable $s$ such that there is an associated delay, we know there exists $i \in \{1, \ldots, K\}$ such that $\theta_{(s,i)} \in [t - l_s, t - u_s]$ and $\sigma_{(s,i)}$ satisfies the guard predicate because of the appropriate change we had made to the guard predicate within the translation. We can then set $\tau_t(s) := i$.

  Equipped with the timings map, we can write the corresponding jump in $\mathscr{H}$ as $(q_i, \mathbf{x}, \mathbf{y}, t) \xrightarrow{\tau_t} (q_j, \mathbf{x}', \mathbf{y}', t)$.

With the above two constructions, we can project any execution trace in $\Phi(\mathscr{H})$ into an execution trace in $\mathscr{H}$.

Now, equipped with the translation function $\Phi$ and the projection function $\Gamma$, we have the following result.

*Proof. Proof of Theorem 1*

**STEP 1:** $\Pi\left(\Gamma_{\Phi(\mathscr{H})}\right) \subseteq \Gamma_{\mathscr{H}}$

First consider a projected execution trace $\gamma \in \Pi\left(\Gamma_{\Phi(\mathscr{H})}\right)$. From the construction of $\Pi$, it is clear that we only need to verify that each flow and jump appearing in the execution trace does indeed match the LHA* semantic of $\mathscr{H}$.

- For a flow, from the construction it is clear that it matches the semantic of a flow in a LHA* because the invariants are only a predicate over the variables without a delay.
- For a jump in $\gamma$, given by $(q_i, \mathbf{x}, \mathbf{y}, t) \xrightarrow{\tau_t} (q_j, \mathbf{x}', \mathbf{y}', t)$, it is also clear from the construction of the timings map $\tau_t$ that

$$(x_1(\tau_t(x_1)), \ldots, x_n(\tau_t(x_n)), y_1(\tau_t(y_1)), \ldots, y_m(\tau_t(y_m))) \models G(q, q')$$

  because $\tau_t$ maps every variable with delays to a correct recorded value of the variable in one of the parallely running automata. Recall that these parallely running automata evolve independently and simply record the value of the variable with delays at certain points in time.

With this, we can conclude that indeed $\gamma$ matches the semantic of a LHA* and it is an accepted execution trace in $\mathscr{H}$, i.e., $\gamma \in \Gamma_{\mathscr{H}}$, allowing us to deduce that $\Pi\left(\Gamma_{\Phi(\mathscr{H})}\right) \subseteq \Gamma_{\mathscr{H}}$.

**STEP 2:** $\Gamma_{\mathscr{H}} \subseteq \Pi\left(\Gamma_{\Phi(\mathscr{H})}\right)$

For this step of the proof, we need to show that for every execution trace $\gamma \in \Gamma_{\mathscr{H}}$, there exists an execution trace $\gamma' \in \Gamma_{\Phi(\mathscr{H})}$, such that $\gamma = \Pi(\gamma')$. Notice that since $\gamma'$ is an execution trace of $\Phi(\mathscr{H})$, we need to construct parallely running execution traces for each of the parallely composed hybrid automata within $\Phi(\mathscr{H})$.

Firstly, let us construct the execution trace of $\mathscr{S}_s$ for each state variable $s \in \{\mathbf{x}, \mathbf{y}\}$ such that it has an associated delay $B(s) \neq (0,0)$. Now consider the set of all times at which there exists a jump within $\gamma$ which we denote by $\{t_1, \ldots, t_l\}$. We know that associated to each of these times, there exists a timings map $\tau_{t_i}$ in $\gamma$. Let the array $[t_{(s,1)}, \ldots, t_{(s,l)}]$ be the increasing order of the set of times given by $\{\tau_{t_1}(s), \ldots, \tau_{t_l}(s)\}$.

We now construct by induction on $l$, an execution trace of $\mathscr{S}_s$. When $l = 0$, we have a simple execution trace where it just remains static in the first state $\lambda_{(s,1)}$ throughout the time horizon of execution. Then, assume we can construct an execution trace for $l \geq 0$ times. When we have $l+1$ times in the set, we can simply add a flow until $t_{(s,l+1)}$ and then add a jump at this time, followed by another flow until the end of the time horizon of execution. Note that at any given location, there is only edge available in $\mathscr{S}_s$. There are no associated guards or invariants, so we are able to perform jumps at any point to save the value of $s$ when required.

The last step is to construct the execution trace for the automaton $\mathscr{H}'$ in $\Phi(\mathscr{H})$. But this is easy because the location, edges, and state variables are the same as $\mathscr{H}$. This is why all we need to do to construct the execution trace of $\mathscr{H}'$ is to remove the timings maps from the jumps in $\gamma$. Note that this execution trace is accepted because we have saved the values required to satisfy the invariants within the parallely running hybrid automata. Also, since we have an upper bound $K$ on the number of transitions within the maximum delay period, we are assured that all the values required by the timings map are saved by these parallel automata.

We have thus constructed an execution trace $\gamma'$ of $\Phi(\mathscr{H})$ by constructing an execution trace for each of the parallely running hybrid automata within the translation provided by $\Phi$. Indeed, it is trivial to verify that $\Pi(\gamma') = \gamma$, allowing us to finally conclude that

$$\Gamma_{\mathscr{H}} = \Pi\left(\Gamma_{\Phi(\mathscr{H})}\right).$$

□

## C Proof of translation from LTA* to TA

We construct the projection map $\Pi : \Gamma_{\Phi(\mathscr{T})} \to \Gamma_{\mathscr{T}}$ which takes an execution trace in $\Phi(\mathscr{T})$ and projects it onto an execution trace of $\mathscr{T}$.

Given a set of initial conditions, let $\gamma \in \Gamma_{\Phi(\mathscr{T})}$ be an execution trace in the translated LTA*. Since $\Phi(\mathscr{T})$ is a parallel composition of timed automata, the execution trace can also be decomposed into parallel execution traces associated to each of the automata. Here, we only consider the part of $\gamma$ which forms the execution trace of $\mathscr{T}'$ in $\Phi(\mathscr{T})$, which we denote by $\gamma_{\mathscr{T}'}$. We then construct the execution trace $\gamma' \in \Gamma_{\mathscr{T}}$ in the original automata by projecting the flows and jumps.

- For each flow in $\gamma$, we retain the same flows in $\gamma'$. One can see that if the invariant at the location of the flow was satisfied in $\mathscr{T}'$, then it must also hold in $\mathscr{T}$, because at each instant of time, the invariant in $\mathscr{T}'$ looks at all the values taken on by each clock during its associated delay period and checks if at least one of them satisfies the invariant. This is exactly the corresponding notion of the invariant in the lazy timed automaton $\mathscr{T}$.
- For each jump in $\gamma$, we again retain the same jump in $\gamma'$ except for the addition of the timings map associated to the jump. But this is trivial to construct because the guard conditions also account for all the values taken on by each clock within its delay period, so there must exist a time within this delay period for each clock such that they satisfy the guard.

With the translation and projection functions, we have the following result regarding LTA*.

*Proof. Proof of Theorem 2*

**STEP 1:** $\Pi(\Gamma_{\Phi(\mathscr{T})}) \subseteq \Gamma_{\mathscr{T}}$

Consider a projected execution trace of the translation $\gamma \in \Pi(\Gamma_{\Phi(\mathscr{T})})$. It remains to verify that $\gamma$ is indeed accepted by $\mathscr{T}$, or that $\gamma$ satisfies the execution semantic of a LTA*. But as explained during the construction of the projection, if the invariant was satisfied within the translation during a flow, it remains satisfied within the original LTA* because the translated invariants account for all the values taken on by each of the clocks in this associated delay period. We are able to compute all the values of each clock in the delay period thanks to the upper bound on the number of transitions and resets within the maximum delay period, which is a property of the LTA*. A similar argument applies to see that the guard conditions are also satisfied during a jump.

**STEP 2:** $\Gamma_{\mathscr{T}} \subseteq \Pi(\Gamma_{\Phi(\mathscr{T})})$

In this step, given an execution trace in the original LTA $\gamma \in \Gamma_{\mathscr{T}}$, we need to show that there exists an execution trace $\gamma' \in \Phi(\mathscr{T})$ such that $\Pi(\gamma') = \gamma$. To construct the execution trace $\gamma'$, we need to describe the components of this execution along each of the parallely composed timed automata within $\Phi(\mathscr{T})$.

Firstly, for the execution trace of $\mathscr{T}'$ in $\Phi(\mathscr{T})$, we simply consider the original execution trace $\gamma$ but with the timings maps removed on the jumps. Once we impose this execution on $\mathscr{T}'$, the execution traces for each $\mathscr{S}_s$ is automatically constructed due to the synchronizations on the transitions with resets. We do not need to modify this. This completes the construction of $\gamma'$ based on $\gamma$. Indeed, the fact that $\gamma'$ is accepted by $\Phi(\mathscr{T})$ is immediately evident. We also have $\Pi(\gamma') = \gamma$ which follows from the construction of $\gamma'$.

We can finally conclude that $\Gamma_{\mathscr{T}} \subseteq \Pi(\Gamma_{\Phi(\mathscr{T})})$ and consequently,

$$\Gamma_{\mathscr{T}} = \Pi(\Gamma_{\Phi(\mathscr{T})}).$$

□