Almost ASAP Semantics: from Timed Models to Timed Implementations*

Martin De Wulf, Laurent Doyen**, and Jean-François Raskin

Computer Science Departement, Université Libre de Bruxelles, Belgium

Abstract. In this paper, we introduce a parametric semantics for timed controllers called the *Almost* ASAP *semantics*. This semantics is a relaxation of the usual ASAP semantics (also called the *maximal progress semantics*) which is a mathematical idealization that can not be implemented by any physical device no matter how fast it is. On the contrary, any correct Almost ASAP controller can be implemented by a program on a hardware if this hardware is fast enough. We study the properties of this semantics, show how it can be analyzed using the tool HyTech, and illustrate its practical use on examples.

1 Introduction

Timed and hybrid systems are dynamical systems with both discrete and continuous components. A paradigmatic example of a hybrid system is a digital embedded control program for an analog plant environment, like a furnace or an airplane: the controller state moves discretely between control modes, and in each control mode, the plant state evolves continuously according to physical laws. A natural model for hybrid systems is the *hybrid automaton*, which represents discrete components using finite-state machines and continuous components using real-numbered variables which evolution is governed by differential equations or differential inclusions [ACH+95]. Several verification and control problems have been studied for hybrid automata or interesting subclasses (see for example [HKPV98]). Tools like HyTech [HHWT95], or Uppaal [PL00], have proven useful to analyze high-level descriptions of embedded controllers in continuous environments.

When a high level description of a controller has been proven *correct* it would be valuable to ensure that an implementation of that design can be obtained in a systematic way in order to ensure the *conservation of correctness*. This is often called program refinement: given a high-level description P_1 of a program, refine that description into another description P_2 such that the "important" properties of P_1 are maintained. Usually, P_2 is obtained from P_1 by reducing nondeterminism. To reason about the correctness of P_2 w.r.t. P_1, we often use a notion of simulation [Mil80] which is powerful enough to ensure conservation of LTL properties for example.

In this paper, we show how to adapt this elegant schema in the context of real-time embedded controllers. To reach this goal, there are several difficulties to overcome. First, the notion of time used by hybrid automata is based on a dense set of values (usually the real numbers). This is unarguably an interesting notion of time at the modeling level but when implemented, a digital controller manipulates timers that are digital clocks. Digital clocks have finite precision and take their values in a discrete domain. As a consequence, any control strategy that requires clocks with infinite precision can not be implemented. Second, hybrid automata can be called "*instantaneous devices*" in that they are capable of instantaneously react to time-outs or incoming events by taking discrete transitions without any delay. Again, while this is a convenient way to see reactivity and synchronization at the modeling level, any control strategy that relies for its correctness on that instantaneity can not be implemented by any physical device no matter how fast it is. Those problems are known and have

* Supported by the FRFC project "Centre Fédéré en Vérification" funded by the Belgian National Science Fundation (FNRS) under grant nr 2.4530.02

** Research fellow supported by the Belgian National Science Fundation (FNRS)

already attracted some attention from our research community. For example, it is well-known that timed automata may describe controllers that control their environment by playing a so called zeno strategy, that is, by taking an infinite number of actions in a finite amount of time. This is widely considered as unacceptable even by authors making the synchrony hypothesis [Yi03]. But even if we prove our controller model non-zeno, that does not mean that it can be implemented. In fact, we recently shown in [CHR02] that there are (very simple) timed automata respecting a syntactic criterion that ensures nonzenoness (defined in [ABD+00]) but require faster and faster reactions, say at times $0, \frac{1}{2}, 1, 1\frac{1}{4}, 2, 2\frac{1}{8}, 3, 3\frac{1}{16}, \ldots$. So, timed automata may model control strategies that can not be implemented because the control strategy does not maintain a minimal bound between two control actions. A direct consequence is that we can not hope to define for the entire class of timed automata a notion of refinement such that if a model of a real-time controller has been proven correct then it can be systematically implemented in a way that preserves its correctness.

The infinite precision and instantaneity characteristics of the traditional semantics given to timed automata is very closely related to the *synchrony hypothesis* that is commonly adopted in the community of synchronous languages [Ber00]. Roughly speaking, the synchrony hypothesis can be stated as follows: "*the program reacts to inputs of the environment by emitting outputs instantaneously*". The rationale behind the synchrony hypothesis is that the speed at which a digital controller reacts is usually so high w.r.t. the speed of the environment that the reaction time of the controller can be neglected and considered as nil. This hypothesis *greatly simplify* the work of the designer of an embedded controller: he/she does not have to take into account the performances of the platform on which the system will be implemented. We agree with this view at the modeling level. But as any hypothesis, the synchrony hypothesis *should be validated* not only by informal arguments but formally if we want to transfer correctness properties from models to implementations. We show in this paper how this can be done *formally* and *elegantly* using a semantics called the Almost ASAP semantics (AASAP-semantics).

The AASAP-semantics is a parametric semantics that leaves as a parameter the *reaction time* of the controller. This semantics relaxes the synchrony hypothesis in that it does not impose the controller to react instantaneously but imposes on the controller to react *within Δ time* units when a synchronization or a control action has to take place (is urgent). The designer acts as if the synchrony hypothesis was true, i.e. he/she models the environment and the controller strategy without referring to the reaction delay. This reaction delay is taken into account during the *verification phase*: we compute the largest Δ for which the controller is still receptive w.r.t. to the environment in which it will be embedded and for which the controller is still correct w.r.t. to the properties that it has to enforce (to avoid the environment to enter bad states for example).

We show that the AASAP semantics has several important and interesting properties. First, the semantics is such that "faster is better". That is, if the controller is correct for a reaction delay bounded by Δ then it is correct for any smaller Δ'. Second, any controller which is correct for a reaction delay bounded by $\Delta > 0$ can be implemented by a program on a hardware provided that the hardware is *fast enough* and provides *sufficiently precise digital clocks*. Third, the semantics can be analyzed using existing tools like HyTech or Uppaal.

Structure of the paper. The paper is organized as follows. In section 2, we recall the notion of timed transition systems. We extend them with a well-chosen notion of structured alphabet and synchronization in order to formalize a notion of *receptiveness* and *safety control*. We also define a notion of *simulation* that will ensure the conservation of receptiveness and safety properties imposed by the controller. In section 3, we review the syntax and *classical semantics* of rectangular automata. In section 4, we define formally the AASAP semantics and study some of its properties. In section 5, we introduce a *very simple and naive notion of real-time program* to make clear that any correct real-time controller for the AASAP semantics can be implemented. In section 6, we explain how the AASAP semantics can be *analyzed* and *used in practice*. All the non trivial proofs are given in appendix as well as a larger example of the use of the AASAP-semantics in practice.

2 Preliminaries

In this section, we recall the definition of timed transition systems and extend them with structured sets of labels. We define a notion of freedom of receptiveness problem and a compatible notion of simulation. This notion of simulation will be the formal basis for our notion of refinement. Finally, we introduce the problem of safety control and show how our notion of simulation can be used in that context.

Definition 1 [TTS] A *timed transition system* $\mathcal{T}$ is a tuple $\langle S, \iota, \Sigma, \rightarrow\rangle$ where S is a (possibly infinite) set of states, $\iota \in S$ is the initial state, Σ is a finite set of labels, and $\rightarrow \subseteq S \times \Sigma \cup \mathbb{R}^{\geq 0} \times S$ is the transition relation where $\mathbb{R}^{\geq 0}$ is the set of positive real numbers.

Definition 2 [Reachable States of TTS] A state s of a TTS $\mathcal{T} = \langle S, \iota, \Sigma, \rightarrow\rangle$ is *reachable* if there exists a finite sequence $s_0 s_1 \dots s_n$ of states such that $s_0 = \iota$, $s_n = s$ and for any i, $0 \leq i < n$, there exists $\sigma \in \Sigma \cup \mathbb{R}^{\geq 0}$ such that $(s_i, \sigma, s_{i+1}) \in \rightarrow$. The set of reachable states of $\mathcal{T}$ are noted $\mathsf{Reach}(\mathcal{T})$.

We need to compose TTS. For that purpose, we need TTS with structured set of labels.

Definition 3 [Structured set of labels] We say that a finite set of labels Σ is *structured* if it is partitioned into three subsets: Σ_{in} the set of input labels, Σ_{out} the set of output labels, and Σ_τ the set of internal labels.

In the sequel, we need some more notations. Let Σ be a structured alphabet and $\Sigma' \subseteq \Sigma$ be a subset of labels, then we note $\overline{\Sigma'}$ for the set $\{\bar{\sigma} \mid \sigma \in \Sigma'\}$, and assume this set is such that $\overline{\Sigma'} \cap \Sigma = \emptyset$.

Definition 4 [STTS] A *structured timed transition system* $\mathcal{T}$ is a tuple $\langle S, \iota, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau, \rightarrow\rangle$, where S is a (possibly infinite) set of states, $\iota \in S$ is the initial state, the set of labels is partitioned in three subsets: Σ_{in} is the finite set of incoming labels, Σ_{out} is the finite set of outgoing labels, Σ_τ is the finite set of internal labels, and $\rightarrow \subseteq S \times \Sigma_{\mathsf{in}} \cup \Sigma_{\mathsf{out}} \cup \Sigma_\tau \cup \mathbb{R}^{\geq 0} \times S$ is the transition relation.

The notion of reachability is extended to STTS as expected.

In the sequel, we use one STTS to model a timed controller and one to model the environment in which the controller has to be embedded. We model the communication between the two STTS using the mechanism of synchronization on common labels. This is a blocking communication mechanism. But we want to verify that the controller does not control the environment by refusing to synchronize on its output, and on the other hand, we do not want our controller to issue outputs that can not be accepted by the environment. To verify the absence of those synchronization problems, we make their potential presence explicit by introducing the notion of refusal function.

Definition 5 [Refusal function of a STTS] Given a STTS $\mathcal{T} = \langle S, \iota, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau, \rightarrow\rangle$, we define its *refusal function* $\mathsf{Ref}_{\mathcal{T}} : S \rightarrow 2^{\Sigma_{\mathsf{in}}}$ as follows :

$$\mathsf{Ref}_{\mathcal{T}}(s) = \{\sigma \in \Sigma_{\mathsf{in}} \mid \neg\exists s' \in S : (s, \sigma, s') \in \rightarrow\}$$

We now define when and how two STTS can be composed to define a timed transition system.

Definition 6 [Composition of STTS] Two STTS $\mathcal{T}^1 = \langle S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau, \rightarrow^1\rangle$ and $\mathcal{T}^2 = \langle S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau, \rightarrow^2\rangle$ are *composable* if $\Sigma^1_{\mathsf{in}} = \Sigma^2_{\mathsf{out}}$ and $\Sigma^2_{\mathsf{in}} = \Sigma^1_{\mathsf{out}}$. Their composition, noted $\mathcal{T}^1 \| \mathcal{T}^2$ is the TTS $\mathcal{T} = \langle S, \iota, \Sigma, \rightarrow\rangle$ such that $S = \{(s^1, s^2) \mid s^1 \in S^1 \text{ and } s^2 \in S^2\}$, $\iota = (\iota^1, \iota^2)$, $\Sigma = \Sigma^1_{\mathsf{out}} \cup \Sigma^2_{\mathsf{out}} \cup \Sigma^1_\tau \cup \Sigma^2_\tau$, and $\rightarrow$ is such that for any $\sigma \in \Sigma \cup \mathbb{R}^{\geq 0}$, we have that $((s^1_1, s^2_1), \sigma, (s^1_2, s^2_2)) \in \rightarrow$ iff one of the following three assertions holds:

$(C1)$ $\sigma \in \Sigma^1_{\mathsf{out}} \cup \Sigma^2_{\mathsf{out}} \cup \mathbb{R}^{\geq 0}$ and $(s^1_1, \sigma, s^1_2) \in \rightarrow^1$ and $(s^2_1, \sigma, s^2_2) \in \rightarrow^2$
$(C2)$ $\sigma \in \Sigma^1_\tau$ and $(s^1_1, \sigma, s^1_2) \in \rightarrow^1$ and $s^2_1 = s^2_2$

$(C3)$ $\sigma \in \Sigma^2_\tau$ and $(s^2_1, \sigma, s^2_2) \in \rightarrow^2$ and $s^1_1 = s^1_2$

When composing two STTS, we say that the result is free of receptiveness problem if there is no reachable state in the product where one STTSwant to issue an output that is not accepted by the other one.

Definition 7 [Freedom of receptiveness problems] We say that the composition of two composable STTS $\mathcal{T}^1 = \langle S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau, \rightarrow^1\rangle$ and $\mathcal{T}^2 = \langle S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau, \rightarrow^2\rangle$ is *free of receptiveness problems* if their composition $\mathcal{T}^1 \| \mathcal{T}^2 = \langle S, \iota, \Sigma, \rightarrow\rangle$ is such that there does not exist $(s^1_1, s^2_1) \in \mathsf{Reach}(\mathcal{T}^1 \| \mathcal{T}^2)$, such that either:

- there exist $\sigma \in \Sigma^1_{\mathsf{out}}$, $s^1_2 \in S^1$ such that $(s^1_1, \sigma, s^1_2) \in \rightarrow^1$ and $\sigma \in \mathsf{Ref}_{\mathcal{T}^2}(s^2_1)$
- there exist $\sigma \in \Sigma^2_{\mathsf{out}}$, $s^2_2 \in S^2$ such that $(s^2_1, \sigma, s^2_2) \in \rightarrow^2$ and $\sigma \in \mathsf{Ref}_{\mathcal{T}^1}(s^1_1)$

Implementations of controllers are also formalized using STTS. To reason about the correctness of implementations w.r.t. higher level models, we use the notion of *simulation*. That notion of simulation is classical but makes explicit the notion of refusal in order to preserve the potential freedom of receptiveness problem property of the model.

Definition 8 [Simulation relation for STTS] Given two STTS $\mathcal{T}^1 = \langle S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau, \rightarrow^1\rangle$ and $\mathcal{T}^2 = \langle S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau, \rightarrow^2\rangle$, let $\Sigma = \Sigma^1_{\mathsf{out}} \cup \Sigma^1_{\mathsf{in}} \cup \Sigma^1_\tau$, we say that $\mathcal{T}^1$ is *simulable* by $\mathcal{T}^2$ and *as receptive* as $\mathcal{T}^2$, noted $\mathcal{T}^1 \sqsubseteq^r \mathcal{T}^2$, if there exists a relation $R \subseteq S^1 \times S^2$ (called a *simulation relation*) such that:

$(S1)$ $(\iota^1, \iota^2) \in R$

$(S2)$ for any $(s^1_1, s^2_1) \in R$, we have that:

- $(S21)$ for any $\sigma \in \Sigma \cup \mathbb{R}^{\geq 0}$, for any s^1_2 such that $(s^1_1, \sigma, s^1_2) \in \rightarrow^1$, there exists $s^2_2 \in S^2$ such that $(s^2_1, \sigma, s^2_2) \in \rightarrow^2$ and $(s^1_2, s^2_2) \in R$;
- $(S22)$ $\mathsf{Ref}_{\mathcal{T}^1}(s^1_1) = \mathsf{Ref}_{\mathcal{T}^2}(s^2_1)$.

The notion of simulation we have defined can be used as usual to define a notion of refinement. We say that the STTS $\mathcal{T}^2$ *refines* the STTS $\mathcal{T}^1$, if $\mathcal{T}^1 \sqsubseteq^r \mathcal{T}^2$. The following theorem shows that our notion of refinement ensures that if a STTS $\mathcal{T}^1$ is free of receptiveness problems when composed with a STTS $\mathcal{T}^2$, then we can conclude the same for any STTS $\mathcal{T}^3$ that refines $\mathcal{T}^1$.

Theorem 1 *Let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two composable* STTS*, let $\mathcal{T}^3$ be a* STTS *such that $\mathcal{T}^3 \sqsubseteq^r \mathcal{T}^1$, if $\mathcal{T}^1 \| \mathcal{T}^2$ is free of receptiveness problems then $\mathcal{T}^3 \| \mathcal{T}^2$ is free of receptiveness problems.*

Proof. This proof is given in appendix.

We are now equipped to define the notion of safety control. This notion together with the notion of refinement we have introduced above allow us to formalize in section 4 and 5, the notion of correct implementation of an embedded timed controller.

Definition 9 [Safety Control] Let $\mathcal{T}^1 = \langle S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau, \rightarrow^1\rangle$ and $\mathcal{T}^2 = \langle S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau, \rightarrow^2\rangle$ be two composable STTS. Let $B \subseteq S^2$, we say that $\mathcal{T}^1$ controls $\mathcal{T}^2$ to avoid B if the following two conditions hold:

$(C1)$ $\mathcal{T}^1 \| \mathcal{T}^2$ is free of receptiveness problems;

$(C2)$ $\mathsf{Reach}(\mathcal{T}^1 \| \mathcal{T}^2) \cap \{(s^1, s^2) \mid s^1 \in S^1 \wedge s^2 \in B\}$ is empty.

We can now state a theorem linking our notion of refinement with the notion of safety control.

Theorem 2 *Let $\mathcal{T}^1 = \langle S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau, \rightarrow^1\rangle$ and $\mathcal{T}^2 = \langle S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau, \rightarrow^2\rangle$ be two composable* STTS*, let $\mathcal{T}^3$ be a* STTS *such that $\mathcal{T}^3 \sqsubseteq^r \mathcal{T}^1$, and let $B \subseteq S^2$, if $\mathcal{T}^1$ controls $\mathcal{T}^2$ to avoid B then $\mathcal{T}^3$ controls $\mathcal{T}^2$ to avoid B.*

Proof. This proof is given in appendix.

3 Timed and rectangular automata

The STTS of previous section are specified using the formalism of rectangular automata. We recall their definition in this section.

Let X be a finite set of positively real valued variables. A valuation for X is a function $v : X \to \mathbb{R}^{\geq 0}$. We write $[Y \to E]$ for the set of all valuations of set of variables Y to E. For a set $V \subseteq [X \to \mathbb{R}^{\geq 0}]$ of valuations, and $x \in X$, define $V(x) = \{v(x) \mid v \in V\}$. A *rectangular constraint* over X is a formula of the form $x \in I$ where x belongs to X, and I is one of the intervals $(a, b), [a, b), (a, b]$ or $[a, b]$ where $a, b \in \mathbb{Q}^{\geq 0} \cup \{+\infty\}$, and $a \leq b$. $\mathbb{Q}^{\geq 0}$ denotes the positive rational numbers and, in the sequel, we also use $\mathbb{Q}^{>0}$ to denote the strictly positive rational numbers. A *rectangular predicate* is a finite set of rectangular constraints. For a rectangular predicate p and a valuation v, we write $v \models p$ if $v(x) \in I$ is true for all $x \in I$ appearing in p. For a rectangular predicate p, $[\![p]\!]$ denotes the set $\{v \mid v \models p\}$. We say that a rectangular predicate is in normal form if it contains at most one rectangular constraint for any variable $x \in X$; any rectangular predicate can be put in that normal form. Let g be a rectangular predicate in normal form, then $g(x)$ denotes $x \in I$ if $x \in I$ is the constraint over x in g and $\mathtt{true}$ if there are no constraint over x in g. We note $\mathsf{Rect}(X)$ the set of rectangular predicates built using variables in X. $\mathsf{Rect_c}(X)$ is the subset of rectangular predicates containing only closed rectangular constraints. Let $g(x)$ denote the closed rectangular constraints $x \in [a, b]$, $lb(g(x))$ denotes the value a and $rb(g(x))$ denotes the value b. Let $v : E_1 \to E_2$ be a valuation, let $E_3 \subseteq E_1$, and $c \in E_2$, then $v[E_3 := c]$ denotes the valuation v' such that

$$v'(e) = \begin{cases} c & \text{if } e \in E_3 \\ v(e) & \text{if } e \notin E_3 \end{cases}$$

In the sequel, we sometimes write $v[e := c]$ instead of $v[\{e\} := c]$. Let $v : X \to \mathbb{R}^{\geq 0}$ be a valuation, for any $t \in \mathbb{R}^{\geq 0}$, $v - t$ is a valuation in $[X \to \mathbb{R}]$ such that for any $x \in X$ $v - t(x) = v(x) - t$. We define $v + t$ in a similar way. We extend this definition to valuation v in $[X \to \mathbb{R}^{\geq 0} \cup \{\perp\}]$ as follows: $(v + t)(x) = v(x) + t$if $v(x) \in \mathbb{R}^{\geq 0}$ and $(v + t)(x) = \perp$ otherwise.

We are now equipped to define rectangular automata and their *classical* semantics.

Definition 10 [Rectangular automata - syntax] A rectangular automaton is a tuple $\langle \mathsf{Loc}, l_0, \mathsf{Var}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Lab}, \mathsf{Edg} \rangle$ where

- Loc is a finite set of locations representing the discrete states of the automaton.
- $l_0 \in \mathsf{Loc}$ is the initial location.
- $\mathsf{Var} = \{x_1, \ldots, x_n\}$ is a finite set of real-valued variables, we write $\dot{\mathsf{Var}} = \{\dot{x} \mid x \in \mathsf{Var}\}$ for the set of corresponding dotted variables, which represent first derivatives.
- $\mathsf{Inv} : \mathsf{Loc} \to \mathsf{Rect}(\mathsf{Var})$ is the invariant condition. The automaton can stay in location l as long as each variable x has a value in the interval $[\![\mathsf{Inv}(l)]\!]\ (x)$. We require that for any $x \in \mathsf{Var}$, $0 \in [\![\mathsf{Inv}(l_0)]\!]\ (x)$, to ensure the existence of an initial state.
- $\mathsf{Flow} : \mathsf{Loc} \to \mathsf{Rect}(\dot{\mathsf{Var}})$ is the flow condition. If the automaton is in location l, then each variable $x \in X$ can evolve nondeterministically with a derivative in the interval $[\![\mathsf{Flow}(l)]\!]\ (\dot{x})$.
- $\mathsf{Lab} = \mathsf{Lab_{in}} \cup \mathsf{Lab_{out}} \cup \mathsf{Lab_\tau}$ is a structured finite alphabet of labels, partitioned into input labels $\mathsf{Lab_{in}}$, output labels $\mathsf{Lab_{out}}$, and internal labels $\mathsf{Lab_\tau}$.
- $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Loc} \times \mathsf{Rect}(\mathsf{Var}) \times \mathsf{Lab} \times 2^{\mathsf{Var}}$ is a set of edges. Every edge (l, l', g, σ, R) represents a move from location l to location l' with guard g, event σ and a subset $R \subseteq \mathsf{Var}$ of the variables to be reset.

Definition 11 [Rectangular automata - semantics] Let $A = \langle \mathsf{Loc}, l_0, \mathsf{Var}, \mathsf{Inv}, \mathsf{Flow}, \mathsf{Lab}, \mathsf{Edg} \rangle$ be a rectangular automaton, the semantics of A, noted $[\![A]\!]$, is the STTS $\mathcal{T} = (S, \iota, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau, \to)$ where:

- $S = \{(l, v) \mid l \in \mathsf{Loc} \wedge v \models \mathsf{Inv}(l)\}$.
- $\iota = (l_0, v_0)$ such that for any $x \in \mathsf{Var} : v_0(x) = 0$.
- $\Sigma_{\mathsf{in}} = \mathsf{Lab}_{\mathsf{in}}$, $\Sigma_{\mathsf{out}} = \mathsf{Lab}_{\mathsf{out}}$, and $\Sigma_\tau = \mathsf{Lab}_\tau$.
- the transition relation $\rightarrow$ is defined as follows:
  - for the discrete transitions, $((l, v), \sigma, (l', v')) \in\rightarrow$ iff there exists an edge $(l, l', g, \sigma, R) \in \mathsf{Edg}$ such that $v \models g$, $v' = v[R := 0]$.
  - for the continuous transitions, $((l, v), t, (l', v')) \in\rightarrow$ iff $l = l'$ and for each variable $x \in \mathsf{Var}$ there exists a differentiable function $f_x : [0, t] \rightarrow [\![Inv(l)]\!]\ (x)$ such that (i) $f_x(0) = v(x)$, (ii) $f_x(t) = v'(x)$ and (iii) $\forall 0 < t' < t : \dot{f}(t') \in [\![flow(l)]\!]\ (x)$.

In the sequel, we need to refer to the subclass of timed automata.

Definition 12 [Timed automata] A *timed automaton* is a rectangular automaton where continuous variables are clocks, that is, for any $x \in \mathsf{Var}$ and any $l \in \mathsf{Loc}$, $[\![flow(l)]\!]\ (x) = 1$.

Running example. Consider Fig. 1. The product[1] of rectangular automata of Fig.1(b) to 1(d) models a moving belt on which luggages are conveyed (say to a plane). The moving belt is equipped with a scanner and a motorized arm. The scanner is used to detect suspicious luggages and the arm is used to remove those suspicious luggages from the moving belt. If a suspicious luggage is detected by the scanner, the signal `Lug` is emitted.

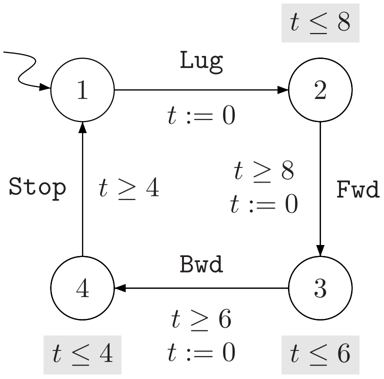
In that case, the motorized arm has to remove the luggage when it passes in front of it (`AtP2`) by pushing the luggage out of the moving belt. The arm can be controlled by issuing the order `Fwd`, to move the arm forward, `Bwd` to move the arm backward, and `Stop` to bring it to an halt.

The role of the controller is to capture the events emitted by the scanner and to give orders to the arm respecting a strict timing in order to properly remove suspicious luggages. An example of such a controller is given in Fig. 1(a). An observer modeled by the timed automaton of Fig. 1(e) ensures that whenever the event `Lug` is emitted then the events `AtP2` and `Hit` are at most 2 time units apart. As the luggage has some thickness, this ensures that they are properly removed from the moving belt. If this is not the case, the observer enter the location `Bad`. The following additional physical constraints are modeled by the product of automata: (i) any luggage takes between 9 and 10 time units to go from the scanner to the center of the arm, (ii) the arm takes between 1 and 2 time units to reach the center of the moving belt when it is launched forward from its idle position, (iii) it takes at most 4 times units to come back to its idle position when launched backward. Given a controller for this system, we must verify that the controller gives orders to the arm such that any resulting behavior of the environment avoid to enter the bad state of the observer. We must additionally verify that the controller is receptive to the event `Lug` from the environment (otherwise it could simply control the environment to avoid bad by refusing to synchronize on `Lug`). We must also verify that the arm is ready to receive the order the controller sent to him.

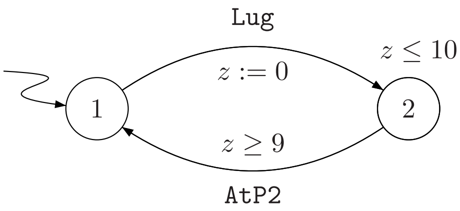
As we already pointed out in the introduction, the classical semantics given in definition 11 is problematic for the controller part if our goal is to transfer the properties verified on the model to an implementation. Consider the controller for the arm in Fig. 1(a). Below, we illustrate the properties of the classical semantics that makes impossible to both implement the controller and ensure formally that the properties of the model are preserved:

- First, note that invariants (grayed constraints in Fig. 1(a)) are used to force the controller to take actions. Invariants can be removed if we assume a ASAP semantics for the controller: any action is taken as soon as possible, this is also called the *maximal progress assumption.* So the transition labeled with `Fwd` proceeds exactly when $t = 8$. Clearly, no hardware can guarantee that the transition will always proceed when $t = 8$.

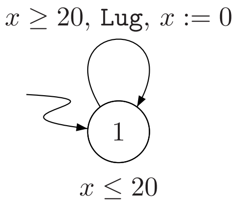
[1] defined as usual [ACH+95]



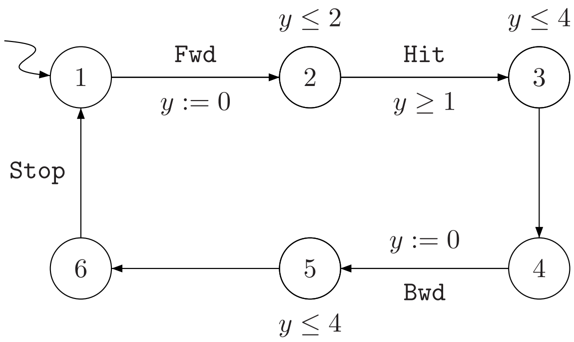
(a) Elastic controller for an arm.



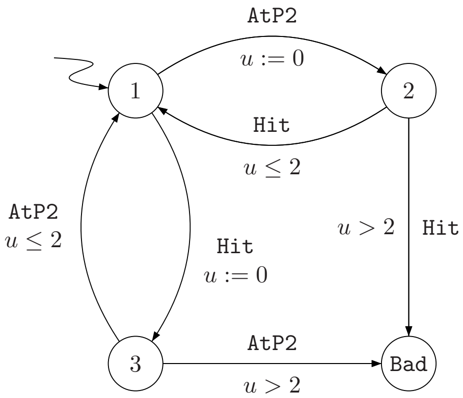
(b) Timed automaton modeling the moving of the luggage.



(c) Timed automaton generating Lug.



(d) Timed automaton modeling the arm.



(e) Observer timed automaton.

Fig. 1. Running example.

– Second, synchronizations between the environment and the controller (*e.g.* transition labeled `Lug`) cannot be implemented as instantaneous: some time is needed by the hardware to detect the incoming event `Lug` and for the software that implements the control strategy to take this event into account.
– Third, the use of real-valued clocks is only possible in the model: implementations use digital clocks with finite precision. It is then necessary to show that a digital clock can replace the real-valued clocks while preserving the verified safety properties.

These three problems illustrates that even if we have formally verified our control strategy, we can not conclude that an implementation will conserve any of the properties that we have proven on the model. This is unfortunate. If we simplify, there are two options to get out of this situation: (i) we ask the designer to give up the synchrony hypothesis and ask the designer to model the platform on which the control strategy will be implemented, or (ii) we let the designer go on with the synchrony hypothesis at the modeling level but relax the ASAP semantics during the verification phase in order to *formally validate the synchrony hypothesis*.

We think that the second option is *much more appealing* and we propose in the next section a framework that makes the second option possible *theoretically* but also feasible *practically*. The framework we propose is centered on a relaxation of the ASAP semantics that we call the AASAP semantics. The main characteristics of this semantics are summarized below:

– any transition that can be taken by the controller becomes urgent only after a small delay Δ (which may be left as a parameter);
– a distinction is made between the occurrence of an event in the environment (sent) and in the controller (received), however the time difference between the two events is bounded by Δ;
– guards are enlarged by some small amount depending on Δ.

We define formally this semantics in the next section and show in section 5 that it is *robust* in the sense that it defines a *tube of strategies* (instead of a unique strategy as in the ASAP semantics) which can be refined in a formal way into an implementation while preserving the safety properties imposed by this tube of strategies.

4 ELASTIC Controllers and AASAP semantics

As explained in the previous section, invariants are useful when modeling controllers with the classical semantics in order to force the controller to take actions but they are useless with a ASAP semantics. This is also true with the semantics we define in this section. So, we restrict our attention to the subclass of timed automata without invariants. In the rest of the paper, we call the controller specified by this subclass ELASTIC[2] controllers.

Definition 13 [ELASTIC Controllers] An ELASTIC controller A is a tuple $\langle \mathsf{Loc}, l_0, \mathsf{Var}, \mathsf{Lab}, \mathsf{Edg} \rangle$ where Loc is a finite set of locations, $l_0 \in \mathsf{Loc}$ is the initial location, $\mathsf{Var} = \{x_1, \ldots, x_n\}$ is a finite set of clocks, Lab is a finite structured alphabet of labels, partitioned into input labels $\mathsf{Lab}_{\mathsf{in}}$, output labels $\mathsf{Lab}_{\mathsf{out}}$, and internal labels Lab_{τ}, Edg is a set of edges of the form (l, l', σ, g, R) where $l, l' \in \mathsf{Loc}$ are locations, $\sigma \in \mathsf{Lab}$ is a label, $g \in \mathsf{Rect_c}(\mathsf{Var})$ is a guard and $R \subseteq \mathsf{Var}$ is a set of clocks to be reset.

Before defining the AASAP semantics we need some more notations:

Definition 14 [True Since] We define the function "True Since", noted $\mathsf{TS} : [\mathsf{Var} \to \mathbb{R}^{\geq 0}] \times \mathsf{Rect_c}(\mathsf{Var}) \to \mathbb{R}^{\geq 0} \cup \{-\infty\}$, as follows:

[2] ELASTIC stands for Event-based LAnguage for Simple TImed Controllers; we also give to those timed controllers a semantics which is elastic in a sense that will be clear to the reader soon.

$$\mathsf{TS}(v,g) = \begin{cases} t & \text{if } v \models g \wedge v-t \models g \wedge \forall t' > t : v-t' \not\models g \\ -\infty & \text{otherwise} \end{cases}.$$

Definition 15 [Guard Enlargement] Let $g(x)$ be the rectangular constraint $x \in [a,b]$, the rectangular constraint ${}_{\Delta}g(x)_{\Delta}$ with $\Delta \in \mathbb{Q}^{\geq 0}$ is the formula $x \in [a-\Delta, b+\Delta]$ if $a-\Delta \geq 0$ and $x \in [0, b+\Delta]$ otherwise. If g is a closed rectangular predicate then ${}_{\Delta}g_{\Delta}$ is the set of closed rectangular constraints $\{{}_{\Delta}g(x)_{\Delta} \mid g(x) \in g\}$.

We are now ready to define the AASAP semantics.

Definition 16 [AASAP semantics] Given an ELASTIC controller

$$A = \langle \mathsf{Loc}, l_0, \mathsf{Var}, \mathsf{Lab}, \mathsf{Edg} \rangle$$

and $\Delta \in \mathbb{Q}^{\geq 0}$, the AASAP semantics of A, noted $[\![A]\!]_{\Delta}^{\mathsf{AAsap}}$ is the STTS

$$\mathcal{T} = (S, \iota, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_{\tau}, \rightarrow)$$

where:

($A1$) S is the set of tuples (l,v,I,d) where $l \in \mathsf{Loc}$, $v \in [\mathsf{Var} \rightarrow \mathbb{R}^{\geq 0}]$, $I \in [\Sigma_{\mathsf{in}} \rightarrow \mathbb{R}^{\geq 0} \cup \{\bot\}]$ and $d \in \mathbb{R}^{\geq 0}$;

($A2$) $\iota = (l_0, v, I, 0)$ where v is such that for any $x \in \mathsf{Var} : v(x) = 0$, and I is such that for any $\sigma \in \Sigma_{\mathsf{in}}$, $I(\sigma) = \bot$;

($A3$) $\Sigma_{\mathsf{in}} = \mathsf{Lab}_{\mathsf{in}}$, $\Sigma_{\mathsf{out}} = \mathsf{Lab}_{\mathsf{out}}$, and $\Sigma_{\tau} = \mathsf{Lab}_{\tau} \cup \overline{\mathsf{Lab}_{\mathsf{in}}} \cup \{\epsilon\}$;

($A4$) The transition relation is defined as follows:
- for the discrete transitions, we distinguish five cases:
  - ($A4.1$) let $\sigma \in \mathsf{Lab}_{\mathsf{out}}$. We have $((l,v,I,d),\sigma,(l',v',I,0)) \in \rightarrow$ iff there exists $(l,l',g,\sigma,R) \in \mathsf{Edg}$ such that $v \models {}_{\Delta}g_{\Delta}$ and $v' = v[R := 0]$;
  - ($A4.2$) let $\sigma \in \mathsf{Lab}_{\mathsf{in}}$. We have $((l,v,I,d),\sigma,(l,v,I',d)) \in \rightarrow$ iff $I(\sigma) = \bot$ and $I' = I[\sigma := 0]$;
  - ($A4.3$) let $\bar{\sigma} \in \overline{\mathsf{Lab}_{\mathsf{in}}}$. We have $((l,v,I,d),\bar{\sigma},(l',v',I',0)) \in \rightarrow$ iff there exists $(l,l',\sigma,g,R) \in \mathsf{Edg}$, $v \models {}_{\Delta}g_{\Delta}$, $I(\sigma) \neq \bot$, $v' = v[R := 0]$ and $I' = I[\sigma := \bot]$;
  - ($A4.4$) let $\sigma \in \mathsf{Lab}_{\tau}$. We have $((l,v,I,d),\sigma,(l',v',I,0)) \in \rightarrow$ iff there exists $(l,l',\sigma,g,R) \in \mathsf{Edg}$, $v \models {}_{\Delta}g_{\Delta}$, and $v' = v[R := 0]$;
  - ($A4.5$) let $\sigma = \epsilon$. We have for any $(l,v,I,d) \in S : ((l,v,I,d),\epsilon,(l,v,I,d)) \in \rightarrow$.
- for the continuous transitions:
  - ($A4.6$) for any $t \in \mathbb{R}^{\geq 0}$, we have $((l,v,I,d),t,(l,v+t,I+t,d+t)) \in \rightarrow$ iff the two following conditions are satisfied:
    - for any edge $(l,l',\sigma,g,R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}_{\mathsf{out}} \cup \mathsf{Lab}_{\tau}$, we have that:
    $$\forall t' : 0 \leq t' \leq t : (d+t' \leq \Delta \vee \mathsf{TS}(v+t',g) \leq \Delta)$$
    - for any edge $(l,l',\sigma,g,R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}_{\mathsf{in}}$, we have that:
    $$\forall t' : 0 \leq t' \leq t : (d+t' \leq \Delta \vee \mathsf{TS}(v+t',g) \leq \Delta \vee (I+t')(\sigma) \leq \Delta)$$

Comments on the AASAP *semantics.* Rule ($A1$) defines the states that are tuples of the form $\langle l,v,I,d \rangle$. The first two components, location l and valuation v, are the same as in the classical semantics; I and d are new. The function I records, for each input event σ, the time elapsed since its last occurrence if this occurrence has not been "treated" yet otherwise the function returns the special value $\bot$. d records the time elapsed since the last location change in the controller. Rule ($A2$) and ($A3$) are straightforward. Rules ($A4.1-6$) require more explanations. Rule ($A4.1$) defines when it is allowed for the controller to emit an output event. The only difference with the classical semantics is that we enlarge the guard by the parameter Δ. Rules ($A4.2-3$) defines how inputs from the environment are received ($A4.2$) and treated ($A4.3$) by the controller. First, the controller maintains, through the function I, a list of events that have occurred and are not yet treated. An input event σ can be received by the controller if no occurrence of σ is already present. An input event σ can be treated if $I(\sigma)$ is different of $\perp$. Once treated, the value of I for that event goes back to $\perp$. Rule $(A4.4)$ is similar to $(A4.1)$. Rule $(A4.5)$ expresses that the ϵ event can always be emitted. Rule $(A4.6)$ specifies how much time can elapse. Intuitively, time can pass as long as no transition starting from the current location is *urgent*. A transition labeled with an output or an internal event is urgent in a location l when the control has been in l for more than Δ time units $(d + t' > \Delta)$ and the guard of the transition has been true for more that Δ time units $(\mathsf{TS}(v + t', g) > \Delta)$. A transition labeled with an input event σ is urgent in a location l when the control has been in l for more that Δ time units $(d + t' > \Delta)$, the guard of the transition has been true for more that Δ time units$(\mathsf{TS}(v + t', g) > \Delta)$, and the last occurrence of σ event has not been treated yet but has been emitted by the environment at least Δ time units ago$(I + t'(\sigma) > \Delta)$. This notion of urgency parameterized by Δ is the main difference between the AASAP semantics and the usual ASAP semantics.

We now state a first property of the AASAP semantics. The following theorem and corollary state formally the informal statement "faster is better", that is if an environment is controllable with an Elastic controller reacting within the bound Δ_1 then this environment is controllable by the same controller for any reaction time $\Delta_2 \leq \Delta_1$. This is clearly a desirable property.

Theorem 3 *Let A be an* Elastic *controller, for any $\Delta_1, \Delta_2 \in \mathbb{Q}^{\geq 0}$ such that $\Delta_1 \geq \Delta_2$ we have that $[\![A]\!]^{\mathsf{AAsap}}_{\Delta_2} \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_{\Delta_1}$.*

Proof It is clear that the identity between the set of states of the two STTS $[\![A]\!]^{\mathsf{AAsap}}_{\Delta_2}$ and $[\![A]\!]^{\mathsf{AAsap}}_{\Delta_1}$ is a simulation relation between them.

Theorem 2 and theorem 3 allow us to state the following corollary:

Corollary 1 *Let E be a rectangular automaton, $[\![E]\!]$ be an* STTS *with set of states S^E, $B \subseteq S^E$ be a set of bad states, and A be an* Elastic *controller. For any $\Delta_1, \Delta_2 \in \mathbb{Q}^{\geq 0}$, such that $\Delta_1 \geq \Delta_2$, if $[\![A]\!]^{\mathsf{AAsap}}_{\Delta_1}$ controls $[\![E]\!]$ to avoid B then $[\![A]\!]^{\mathsf{AAsap}}_{\Delta_2}$ controls $[\![E]\!]$ to avoid B.*

We say that an Elastic controller is able to control an environment modeled as a rectangular automaton E for a safety property modeled by a set of bad states B if there exists $\Delta > 0$ such that $[\![A]\!]^{\mathsf{AAsap}}_{\Delta}$ controls $[\![E]\!]$ to avoid B.

5 Implementability of the AASAP semantics

In this section, we show that any Elastic controller which controls an environment E for a safety property modeled by a set of bad states B can be implemented provided there exists a hardware sufficiently fast and providing sufficiently precise digital clocks.

To establish this result, we proceed as follows. First, we define what we call the program semantics of an Elastic controller. The so-called program semantics can be seen as a formal semantics for the following procedure interpreting Elastic controllers. This procedure repeatedly executes what we call *execution rounds*. An execution round is defined as follows:

- first, the current time is read in the clock register of the CPU and stored in a variable, say `T`;
- the list of input events to treat is updated: the input sensors are checked for new events issued by the environment;
- guards of the edges of the current locations are evaluated with the value stored in `T`. If at least one guard evaluates to true then take nondeterministically one of the enabled transitions;
- the next round is started.

All we require from the hardware is to respect the following two requirements: (i) the clock register of the CPU is incremented every Δ_P time units and (ii) the time spent in one loop is bounded by a certain fixed value Δ_L. We choose this semantics for its simplicity and also because it is obviously implementable. There are more efficient ways to interpret ELASTIC controllers but as the ASAP semantics is such that "faster is better", this semantics is good enough for our purpose. In section 6, we show how to use this semantics in the context of the LEGO MINDSTORMS™ platform.

We proceed now with the definition of the program semantics. This semantics manipulates digital clocks, so we need the following definition:

Definition 17 [Clock Rounding] Let $T \in \mathbb{R}, \Delta \in \mathbb{Q}^{>0}$, $\lfloor T \rfloor_\Delta = \lfloor \frac{T}{\Delta} \rfloor \times \Delta$.

Lemma 1 follows directly from the definition above.

Lemma 1 *For any $T \in \mathbb{R}^{\geq 0}$, any $\Delta \in \mathbb{Q}^{>0}$, $T - \Delta \leq \lfloor T \rfloor_\Delta \leq T + \Delta$.*

We are now ready to define the program semantics:

Definition 18 [Program Semantics] Let A be an ELASTIC controller and $\Delta_L, \Delta_P \in \mathbb{Q}^{>0}$. We define $\Delta_S = \Delta_L + 2\Delta_P$. The (Δ_L, Δ_P) program semantics of A, noted $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P}$ is the structured timed transition system $\mathcal{T} = (S, \iota, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau, \rightarrow)$ where:

$(P1)$ S is the set of tuples (l, r, T, I, u, d, f) such that $l \in \mathsf{Loc}$, r is a function from Var into $\mathbb{R}^{\geq 0}$, $T \in \mathbb{R}^{\geq 0}$, I is a function from $\mathsf{Lab_{in}}$ into $\mathbb{R}^{\geq 0} \cup \{\bot\}$, $u \in \mathbb{R}^{\geq 0}$, $d \in \mathbb{R}^{\geq 0}$, and $f \in \{\top, \bot\}$;

$(P2)$ $\iota = (l_0, r, 0, I, 0, 0, \bot)$ where r is such that for any $x \in \mathsf{Var}$, $r(x) = 0$, I is such that for any $\sigma \in \mathsf{Lab_{in}}$, $I(\sigma) = \bot$;

$(P3)$ $\Sigma_{\mathsf{in}} = \mathsf{Lab_{in}}$, $\Sigma_{\mathsf{out}} = \mathsf{Lab_{out}}$, $\Sigma_\tau = \mathsf{Lab}_\tau \cup \overline{\mathsf{Lab_{in}}} \cup \{\epsilon\}$;

$(P4)$ the transition relation $\rightarrow$ is defined as follows:

- for the discrete transitions:
  - $(P4.1)$ let $\sigma \in \mathsf{Lab_{out}}$. $((l, r, T, I, u, d, \bot), \sigma, (l', r', T, I, u, 0, \top)) \in\rightarrow$ iff there exists $(l, l', \sigma, g, R) \in \mathsf{Edg}$ such that $\lfloor T \rfloor_{\Delta_P} - r \models_{\Delta_S} g_{\Delta_S}$ and $r' = r[R := \lfloor T \rfloor_{\Delta_P}]$.
  - $(P4.2)$ let $\sigma \in \mathsf{Lab_{in}}$. $((l, r, T, I, u, d, f), \sigma, (l, r, T, I', u, d, f)) \in\rightarrow$ iff $I(\sigma) = \bot$ and $I' = I[\sigma := 0]$;
  - $(P4.3)$ let $\bar{\sigma} \in \overline{\mathsf{Lab_{in}}}$. $((l, r, T, I, u, d, \bot), \bar{\sigma}, (l', r', T, I', u, 0, \top)) \in\rightarrow$ iff there exists $(l, l', \sigma, g, R) \in \mathsf{Edg}$ such that $\lfloor T \rfloor_{\Delta_P} - r \models_{\Delta_S} g_{\Delta_S}$, $I(\sigma) > u$, $r' = r[R := \lfloor T \rfloor_{\Delta_P}]$ and $I' = I[\sigma := \bot]$;
  - $(P4.4)$ let $\sigma \in \mathsf{Lab}_\tau$. $((l, r, T, I, u, d, \bot), \sigma, (l', r', T, I, u, 0, \top)) \in\rightarrow$ iff there exists $(l, l', \sigma, g, R) \in \mathsf{Edg}$ such that $\lfloor T \rfloor_{\Delta_P} - r \models_{\Delta_S} g_{\Delta_S}$ and $r' = r[R := \lfloor T \rfloor_{\Delta_P}]$.
  - $(P4.5)$ let $\sigma = \epsilon$. $((l, r, T, I, u, d, f), \sigma, (l, r, T + u, I, 0, d, \bot)) \in\rightarrow$ iff either $f = \top$ or the two following conditions hold:
    - · for any $\bar{\sigma}$ such that $\sigma \in \mathsf{Lab_{in}}$, for any $(l, l', \sigma, g, R) \in \mathsf{Edg}$, we have that either $\lfloor T \rfloor_{\Delta_P} - r \not\models_{\Delta_S} g_{\Delta_S}$ or $I(\sigma) \leq u$
    - · for any $\sigma \in \mathsf{Lab_{out}} \cup \mathsf{Lab}_\tau$, for any $(l, l', \sigma, g, R) \in \mathsf{Edg}$, we have that $\lfloor T \rfloor_{\Delta_P} - r \not\models_{\Delta_S} g_{\Delta_S}$
- for the continuous transitions:
  - $(P4.6)$ $((l, r, T, I, u, d, f), t, (l, r, T, I + t, u + t, d + t, f)) \in\rightarrow$ iff $u + t \leq \Delta_L$.

Comments on the program semantics. Rule $(P1)$ defines the states which are tuples (l, r, T, I, u, d, f), where l is the current location, r maps each clock to the digital time when it has last been reset, T records the (exact) time at which the last round has started; I, as in the AASAP semantics, records the time elapsed since the last arrival of each input event not yet treated, u records the time elapsed since the last round was started (so that $T + u$ is the exact current time), d records the time elapsed since the last location change, and f is a flag which is set to $\top$ if a location change has occurred in the current round. Rules $(P2)$ and $(P3)$ should be clear. We comment rules $(P4.1 - 6)$. First, we make some general comments on digital clocks and guards of discrete transitions of the controller. Note that in those rules, we evaluate the guards with the valuation $\lfloor T \rfloor_{\Delta_P} - r$ for the clocks, that is, for variable x, the difference between the digital value of the variable T at the beginning of the current round and the digital value of x at the beginning of the round when x was last reset. This value approximates the real time difference between the exact time at which the guard is evaluated and the exact time at which the clock x has been reset. Let t be this exact time difference, then we know that: $\lfloor T \rfloor_{\Delta_P} - r(x) - \Delta_L - \Delta_P \leq t \leq \lfloor T \rfloor_{\Delta_P} - r(x) + \Delta_L + \Delta_P$. Also note that the guard g has been enlarged by the value $\Delta_S = \Delta_L + 2\Delta_P$, this ensures that any event enabled at some point will be enabled sufficiently long so that the change can be detected by the procedure. Rule $(P4.1)$ expresses when transitions labeled with output events can be taken. Note that variables are reset to the digital time of the current round. Rule $(P4.2)$ simply records the exact time at which input event from the environment occurred. This rule simply ensures that the function I is updated when a new event is issued by the environment. Rule $(P4.3)$ says when an input of the environment can be treated by the controller: it has to be present at the beginning of the current round and the enlargement of the guard labelling the transition has to be true for digital values of the clocks at the beginning of the round, and no other discrete transitions should have been taken in the current round. Rule $(P4.4)$ is similar to rule $(P4.1)$ but applies to internal events. Rule $(P4.5)$ expresses that the event ϵ is issued when the current round is finished and the system starts a new round. Note that this is only possible if the program has taken a discrete transition or there were no discrete transition to take. This ensures that the program always takes discrete transitions when possible. Rule $(P4.6)$ expresses that the program can always let time elapse unless it violates the maximal time spent in one round.

The following simulation theorem expresses formally that if the hardware on which the program is implemented is fast enough (parameter Δ_L) and precise enough (parameter Δ_P) then the program semantics can be simulated by the AASAP semantics.

Theorem 4 (Simulation) *Let A be an* ELASTIC *controller, for any $\Delta, \Delta_L, \Delta_P \in \mathbb{Q}^{\geq 0}$ be such that $\Delta > 3\Delta_L + 4\Delta_P$, we have $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P} \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.*

Proof. Let $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P} = (S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_{\tau}, \rightarrow^1)$ and $[\![A]\!]^{\mathsf{AAsap}}_{\Delta} = (S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_{\tau}, \rightarrow^2)$. Consider the relation $R \subseteq S^1 \times S^2$ that contains the pairs:

$$(s^1, s^2) = ((l^1, r^1, T^1, I^1, u^1, d^1, f^1), (l^2, v^2, I^2, d^2))$$

such that the following conditions hold:

$(R1)$ $l^1 = l^2$;
$(R2)$ for any $x \in \mathsf{Var}$, $|v^2(x) - (T^1 - r^1(x) + u^1)| \leq \Delta_L + \Delta_P$
$(R3)$ for any $\sigma \in \mathsf{Lab}_{\mathsf{in}}$, $I^1(\sigma) = I^2(\sigma)$;
$(R4)$ $d^1 = d^2$;
$(R5)$ there exists (l^3, v^3, I^3, d^3) such that: $((l^2, v^2, I^2, d^2), \Delta_L - u^1, (l^3, v^3, I^3, d^3)) \in \rightarrow^2$.

We have now to prove that R is a simulation relation in the precise sense of definition 8. This is done in the appendix. ∎

Theorem 5 (Simulability) *For any* ELASTIC *controller A, for any $\Delta \in \mathbb{Q}^{>0}$, there exists $\Delta_L, \Delta_P \in \mathbb{Q}^{>0}$ such that $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P} \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.*

Proof For any Δ, as parameters Δ_L and Δ_P are in the rational numbers,they can always be chosen such that $\Delta > 3\Delta_L + 4\Delta_P$. ∎

And so, given a sufficiently fast hardware with a sufficiently precise digital clock, we can implement any controller that have been proved correct. This is expressed by the following corollary:

Corollary 2 (Implementability) *Let E be a rectangular automaton , let $[\![E]\!]$ be a* STTS*with set of states S^E, $B \subseteq S^E$ be a set of bad states. For any* ELASTIC *controller A, for any $\Delta \in \mathbb{Q}^{>0}$, such that $[\![A]\!]^{\mathsf{AAsap}}_{\Delta}$ controls $[\![E]\!]$ to avoid B, there exist $\Delta_L, \Delta_P \in \mathbb{Q}^{>0}$ such that $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P}$ controls $[\![E]\!]$ to avoid B.*

6 In practice

In this section, we show that the AASAP semantics can be analyzed automatically using the tool HyTech [HHWT95]. This is a direct corollary of the next theorem: the AASAP semantics of any Elastic controller can be encoded using the classical semantics of a timed automaton. The proof of this result is constructive for any Δ.

Theorem 6 *For any* Elastic *controller A, for any $\Delta \in \mathbb{Q}^{>0}$, we can construct effectively a timed automaton $\mathcal{A}^{\Delta} = \mathcal{F}(A, \Delta)$ such that $[\![A]\!]_{\Delta}^{\mathsf{AAsap}} \sqsubseteq^r [\![\mathcal{A}^{\Delta}]\!]$ and $[\![\mathcal{A}^{\Delta}]\!] \sqsubseteq^r [\![A]\!]_{\Delta}^{\mathsf{AAsap}}$.*

Proof. We give the construction of $\mathcal{F}(A, \Delta)$. Let A be the tuple $\langle \mathsf{Loc}^1, l_0^1, \mathsf{Var}^1, \mathsf{Lab}^1, \mathsf{Edg}^1 \rangle$ and let $\mathcal{F}(A, \Delta) = \langle \mathsf{Loc}^2, l_0^2, \mathsf{Var}^2, \mathsf{Inv}^2, \mathsf{Flow}^2, \mathsf{Lab}^2, \mathsf{Edg}^2 \rangle$ be the timed automaton such that:

- $\mathsf{Loc}^2 = \{(l, b) \mid l \in Loc^1 \wedge b \in [\Sigma_{\mathsf{in}} \to \{\top, \bot\}]\}$;
- $l_0^2 = (l_0^1, b_\bot)$ where $b_\bot$ is such that $b_\bot(\sigma) = \bot$ for any $\sigma \in \Sigma_{\mathsf{in}}$;
- $\mathsf{Var}^2 = \mathsf{Var}^1 \cup \{y_\sigma \mid \sigma \in \Sigma_{\mathsf{in}}\} \cup \{d\}$;
- Flow^2 is such that for any $x \in \mathsf{Var}^2$, for any $l \in Loc^2$:$[\![\mathsf{Flow}^2(l)]\!]\ (x) = 1$;
- $\mathsf{Lab}^2 = \mathsf{Lab}^1_{\mathsf{out}} \cup \mathsf{Lab}^1_{\mathsf{int}} \cup \mathsf{Lab}^1_{\mathsf{in}} \cup \overline{\mathsf{Lab}^1_{\mathsf{in}}}$;
- Edg^2 is defined as follows.
  $((l, b), (l', b'), \sigma, {}_{\Delta}g_{\Delta}, R') \in \mathsf{Edg}^2$ iff one of the following condition holds:
  - $\sigma \in \mathsf{Lab}^1_{\mathsf{out}}$ and
    1. there exists $(l, l', \sigma, g, R) \in \mathsf{Edg}^1$
    2. $b' = b$
    3. $R' = R \cup \{d\}$
  - $\sigma = \bar{\alpha} \in \overline{\mathsf{Lab}^1_{\mathsf{in}}}$ and
    1. there exists $(l, l', \alpha, g, R) \in \mathsf{Edg}^1$
    2. $b(\alpha) = \top$
    3. $b' = b[\alpha := \bot]$
    4. $R' = R \cup \{d\}$
  - $\sigma \in \mathsf{Lab}^1_{\mathsf{int}}$ and
    1. there exists $(l, l', \sigma, g, R) \in \mathsf{Edg}^1$
    2. $b' = b$
    3. $R' = R \cup \{d\}$
  - $\sigma \in \mathsf{Lab}^1_{\mathsf{in}}$ and
    1. $l' = l$
    2. $b(\sigma) = \bot$
    3. $b' = b[\sigma := \top]$
    4. $g = \mathsf{true}$
    5. $R' = \{y_\sigma\}$
  - $\sigma = \epsilon$ and
    1. $l' = l$
    2. $b = b'$
    3. $g = \mathsf{true}$
    4. $R' = \emptyset$
- The function Inv^2 is defined as follows. Let $EVT(l) = \{(l, l', \sigma, g, R) \in \mathsf{Edg}^2 \mid \sigma \in \overline{\mathsf{Lab}^1_{\mathsf{in}}}\}$. Let $ACT(l) = \{(l, l', \sigma, g, R) \in \mathsf{Edg}^2 \mid \sigma \in \mathsf{Lab}^1_{\mathsf{out}} \cup \mathsf{Lab}^1_{\mathsf{int}}\}$.

$$\mathsf{Inv}^2(l) = \Big(\bigwedge_{(l,l',\sigma,g,R) \in EVT(l)} \big(d \leq \Delta \vee \neg({}^{\Delta}g) \vee y_\sigma \leq \Delta\big)\Big) \wedge \Big(\bigwedge_{(l,l',\sigma,g,R) \in ACT(l)} \big(d \leq \Delta \vee \neg({}^{\Delta}g)\big)\Big)$$

  where ${}^{\Delta}g(x)$ is the expression $x \in [a + \Delta, b]$ if $g(x)$ is the expression $x \in [a, b]$[3].

The proof that this construction is correct is given in appendix. ∎

Corollary 3 *For any* Elastic *controller A, for any $\Delta \in \mathbb{Q}^{>0}$, for any rectangular automaton E with state space S^E, for any set of states $B \subseteq S^E$, we have that $[\![A]\!]_{\Delta}^{\mathsf{AAsap}}$ controls $[\![E]\!]$ to avoid B iff $[\![\mathcal{F}(A, \Delta)]\!]$ controls $[\![E]\!]$ to avoid B.*

In practice, we use theorem 6 to reduce the controllability problem to a reachability problem:

- we construct $\mathcal{F}(A, \Delta)$ (where we can leave Δ as parameter);

[3] However disjunctions are not allowed in invariants of timed automata, they can be easily modeled by splitting locations.

– we construct a HyTech file with a description of $\mathcal{F}(A, \Delta)$ and E;
– we ask for which parameter value $reach([\![\mathcal{F}(A, \Delta)]\!] \parallel [\![E]\!]) \cap Bad = \emptyset$ (where Bad is a set of bad states) and the system is free of receptiveness problems.

If we apply that construction to our running example, HyTech establishes that the tube of control strategies defined by the timed automaton of obtained by the construction of theorem 6 (see Fig. 2) is valid for any $\Delta \leq \frac{1}{4}$. If we assume that the unit of time is the second, theorem 4 then tells us that, to preserve the desired property, with a systematic implementation of the Elastic controller, we should have a platform with loop time Δ_L and clock precision Δ_P such that $3\Delta_L + 4\Delta_P < 250ms$. We implemented the procedure of section 5 on the Lego Mindstorms™ platform, using a slightly modified version of the open-source operating system brickOS™. We then ran this implementation using the Elastic controller of figure 1(a). The platform allows Δ_L to be as low as 6ms and offers a digital clock with Δ_P = 1ms which is thus ample enough.

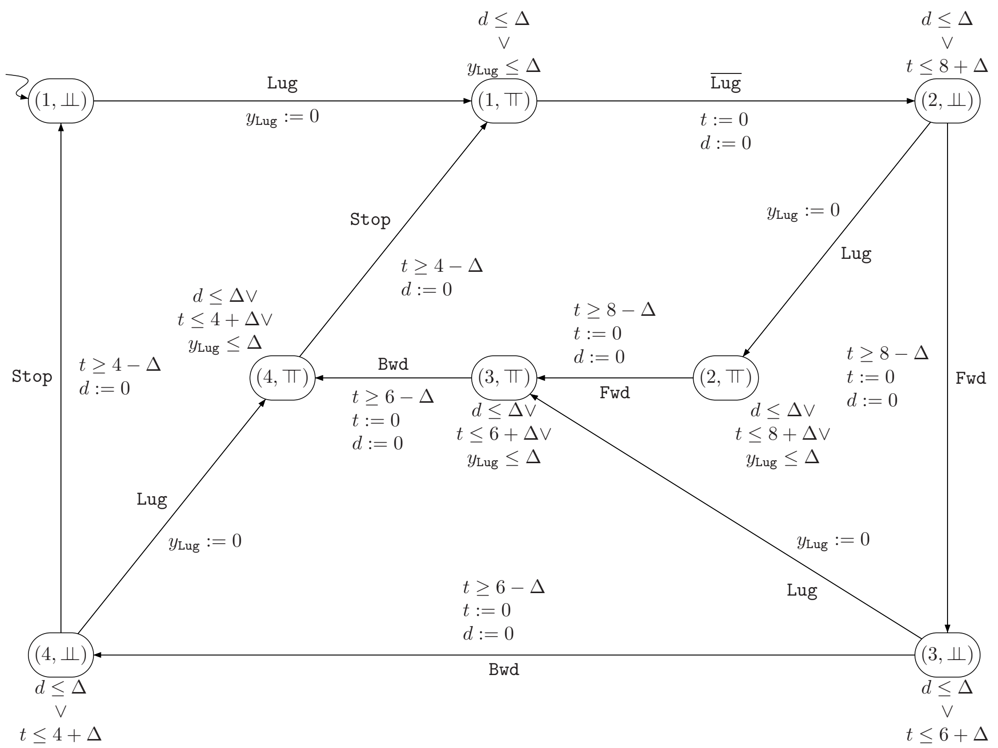


Fig. 2. Semantic timed automaton for the Elastic of figure 1(a).

7 Related and future works

In this section, we compare our work with some recent related works. We also point out several future research directions.

Related works In [AFM+02,Yi03], Wang Yi et al present a tool called Times that generates executable code (C code for the Lego Mindstorms™ platform) from timed automata models. The code is generated with the synchrony hypothesis. This work does not tackle the problem on which we concentrate in this paper. The properties proved on the models are not guaranteed to be preserved by their code generation. On the other hand, this work also integrate interesting schedulability analysis. In our paper, we have only concentrated on simple control centered programs. In our approach, tasks that are computing expensive, should be modeled explicitly (with their worst-case execution time for example). This is coherent with the approach they propose.

In [AFILS03], Rajeev Alur et al introduce a methodology to generate code from hybrid automata. The class of models they consider is larger than the class we consider here, i.e. the ELASTIC controllers. As, in the work of Yi et al, they adopt the synchrony hypothesis. Nevertheless, they plan to explore further this translation in order to see how to achieve the translation without the synchrony hypothesis. The work in this paper should be useful in that context.

In [HKSP03], Tom Henzinger et al introduce a programming model for real-time embedded controllers called GIOTTO. GIOTTO is an embedded software model that can be used to specify a solution to a given control problem independently of an execution platform but which is closer to executable code than a mathematical model. So, GIOTTO can be seen an intermediary step between mathematical models like hybrid automata and real execution platform.

In [IKL$^+$], Kim Larsen et al show how to model code for real-time controllers using UPPAAL models in order to formally verify the code behavior. Usually, they encounter the problem that the obtained description is difficult to analyze because the time unit at the controller level (time slice of the real-time OS for example) is much smaller than the natural time unit of the environment. This leads to what they call *symbolic state space fragmentation*. They proposed in [HL02] a partial solution to that problem. In our framework, we do not encounter that problem. In fact, the larger reaction delay computed during the analysis phase of the AASAP semantics is usually close to the time unit of the environment to control, and usually, at least, much larger than the time unit of the hardware on which the control program is executed. The program is generated automatically from the ELASTIC model and is guaranteed to be correct by construction (no need to verify it).

Future works As future works, we plan to:

- study in details the parameter synthesis problem defined by the AASAP semantics. Currently, we analyze the semantics with HYTECH using the standard fixpoint algorithm with no guarantee of termination. We do not know if the synthesis problem is decidable or not. If the problem turns out to be undecidable, we will look for heuristics.
- study GIOTTO as a possible intermediary step for code generation in the setting of our method. This intermediary step may allow us to simplify parts of our construction.
- compare more extensively the practicability of our approach compared to the approach consisting in verifying code.

References

[ABD$^+$00] E. Asarin, O. Bournier, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. In *Proc. IEEE*, 2000.

[ACH$^+$95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[AFILS03] R. Alur, J. Kim F. Ivancic, I. Lee, and O. Sokolsky. Generating embedded software from hierarchical hybrid models. In *ACM Symposium on Languages, Compilers, and Tools for Embedded Systems*, 2003.

[AFM$^+$02] Tobias Amnell, Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Times: A tool for modelling and implementation of embedded systems. In J.-P. Katoen and P. Stevens, editors, *Proc. Of The 8*Th *International Conference On Tools And Algorithms For The Construction And Analysis Of Systems*, number 2280 in Lecture Notes In Computer Science, pages 460–464. Springer–Verlag, 2002.

[Ber00] G. Berry. *The Foundations of Esterel*. MIT Press, 2000.

[CHR02] F. Cassez, T.A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In *HSCC 02: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2289, pages 134–148. Springer-Verlag, 2002.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science 1019, pages 41–71. Springer-Verlag, 1995.

[HKPV98] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[HKSP03] T.A. Henzinger, C.M. Kirsch, M.A. Sanvido, and W. Pree. From control models to real-time code using Giotto. *IEEE Control Systems Magazine*, 23(1):50–64, 2003.

[HL02] Martijn Hendriks and Kim G. Larsen. Exact acceleration of real-time model checking. In *Workshop on Theory and Practice of Timed Systems*, 2002.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science*, pages 394–406. IEEE Computer Society Press, 1992.

[IKL$^+$] T. Iversen, K. Kristoffersen, K. Larsen, M. Laursen, R. Madsen, S. Mortensen, P. Petterson, and C. Thomasen. Model-checking real-time control programs – verifying LEGO mindstorms systems using UPPAAL. In *Proc. 12th Euromicro Conf. on Real-Time Systems (ECRTS'00).*

[Mil80] R. Milner. *A Calculus of Communicating Systems.* Lecture Notes in Computer Science 92. Springer-Verlag, 1980.

[PL00] Paul Pettersson and Kim G. Larsen. Uppaal2k. *Bulletin of the European Association for Theoretical Computer Science*, 70:40–44, February 2000.

[Yi03] Tobias Amnelland Elena Fersmanand Paul Petterssonand Hongyan Sunand Wang Yi. Code synthesis for timed automata. *Accepted for publication in Nordic Journal of Computing*, 2003.

A Proof of theorems from section 2

To structure the proof, we need some more notations and lemmas. We refine the definition of reachable states of a TTS.

Definition 19 [n-Reachable States of TTS] A state s of a TTS $\mathcal{T} = \langle S, \iota, \Sigma, \rightarrow\rangle$ is *n-reachable* if there exists a finite sequence $s_0 s_1 \dots s_n$ of states such that $s_0 = \iota$, $s_n = s$ and for any i, $0 \leq i < n$, there exists $\sigma \in \Sigma \cup \mathbb{R}^{\geq 0}$ such that $(s_i, \sigma, s_{i+1}) \in \rightarrow$.

This definition is extended to STTS as expected. Let us introduce an intermediate lemma.

Lemma 2 *Let $\mathcal{T}^1 = \langle S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau, \rightarrow^1\rangle$ and $\mathcal{T}^2 = \langle S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau, \rightarrow^2\rangle$ be two composable STTS, let $\mathcal{T}^3 = \langle S^3, \iota^3, \Sigma^3_{\mathsf{in}}, \Sigma^3_{\mathsf{out}}, \Sigma^3_\tau, \rightarrow^3\rangle$ be a STTS such that $\mathcal{T}^3 \sqsubseteq^r \mathcal{T}^1$ (with simulation relation R). If (s^3_n, s^2_n) is n-reachable (in $\mathcal{T}^3 \| \mathcal{T}^2$) then there exists s^1_n such that (s^1_n, s^2_n) is n-reachable (in $\mathcal{T}^1 \| \mathcal{T}^2$) and $(s^3_n, s^1_n) \in R$.*

Proof of lemma 2
We show this by induction on n.

- Case $n = 0$
  It is clearly true since $(s^3_0, s^2_0) = (\iota^3, \iota^2)$, (ι^1, ι^2) is 0-reachable and $(\iota^3, \iota^1) \in R$.
- Case $n = k + 1$
  $(H0)$ Assume it is true for $n = k$, and let show it remains true for $n = k + 1$.
  Let $\mathcal{T}^3 \| \mathcal{T}^2 = \langle S^{3,2}, \iota^{3,2}, \Sigma^{3,2}, \rightarrow^{3,2}\rangle$ and $\mathcal{T}^1 \| \mathcal{T}^2 = \langle S^{1,2}, \iota^{1,2}, \Sigma^{1,2}, \rightarrow^{1,2}\rangle$.
  Let (s^3_{k+1}, s^2_{k+1}) be $(k + 1)$-reachable (in $\mathcal{T}^3 \| \mathcal{T}^2$). Then there exists (s^3_k, s^2_k) k-reachable (in $\mathcal{T}^3 \| \mathcal{T}^2$) such that $((s^3_k, s^2_k), \sigma, (s^3_{k+1}, s^2_{k+1})) \in\rightarrow^{3,2}$ for some $\sigma \in \Sigma^{3,2} \cup \mathbb{R}^{\geq 0}$. $(H1)$
  By $(H0)$, there exists s^1_k such that
  $(H2)$ (s^1_k, s^2_k) is k-reachable (in $\mathcal{T}^1 \| \mathcal{T}^2$) and
  $(H3)$ $(s^3_k, s^1_k) \in R$
  We consider the 3 possible cases for $(H1)$:
  1. $\sigma \in \Sigma^3_{\mathsf{out}} \cup \Sigma^2_{\mathsf{out}} \cup \mathbb{R}^{\geq 0}$ and then $\begin{cases} (s^3_k, \sigma, s^3_{k+1}) \in\rightarrow^3 \\ (s^2_k, \sigma, s^2_{k+1}) \in\rightarrow^2 \end{cases}$
     As $(s^3_k, s^1_k) \in R$ $(H3)$, we know that there exists s^1_{k+1} such that $(s^1_k, \sigma, s^1_{k+1}) \in\rightarrow^1$ and $(s^3_{k+1}, s^1_{k+1}) \in R$ (see $(S21)$ in definition 8?). So that $((s^1_k, s^2_k), \sigma, (s^1_{k+1}, s^2_{k+1})) \in\rightarrow^{1,2}$ since $\sigma \in \Sigma^3_{\mathsf{out}} \cup \Sigma^2_{\mathsf{out}} \cup \mathbb{R}^{\geq 0}$ and $\Sigma^3_{\mathsf{out}} = \Sigma^2_{\mathsf{in}} = \Sigma^1_{\mathsf{out}}$. Finally, (s^1_{k+1}, s^2_{k+1}) is $(k + 1)$-reachable (in $\mathcal{T}^1 \| \mathcal{T}^2$) (by $(H2)$).
  2. $\sigma \in \Sigma^3_\tau$ and then $\begin{cases} (s^3_k, \sigma, s^3_{k+1}) \in\rightarrow^3 \\ s^2_k = s^2_{k+1} \end{cases}$
     As $(s^3_k, s^1_k) \in R$ $(H3)$, we know that there exists s^1_{k+1} such that $(s^1_k, \sigma, s^1_{k+1}) \in\rightarrow^1$ and $(s^3_{k+1}, s^1_{k+1}) \in R$ (see1 $(S21)$ in definition 8?) So that $((s^1_k, s^2_k), \sigma, (s^1_{k+1}, s^2_{k+1})) \in\rightarrow^{1,2}$ since $\sigma \in \Sigma^3_\tau$ and $\Sigma^3_\tau \subseteq \Sigma^1_\tau$. Finally, (s^1_{k+1}, s^2_{k+1}) is $(k + 1)$-reachable (in $\mathcal{T}^1 \| \mathcal{T}^2$) (by $(H2)$).
  3. $\sigma \in \Sigma^2_\tau$ and then $\begin{cases} s^3_k = s^3_{k+1} \\ (s^2_k, \sigma, s^2_{k+1}) \in\rightarrow^2 \end{cases}$
     We have directly $((s^1_k, s^2_k), \sigma, (s^1_{k+1}, s^2_{k+1})) \in\rightarrow^{1,2}$ and $(s^3_{k+1}, s^1_{k+1}) \in R$ (take $s^1_{k+1} = s^1_k$). Finally, (s^1_{k+1}, s^2_{k+1}) is $(k + 1)$-reachable (in $\mathcal{T}^1 \| \mathcal{T}^2$) (by $(H2)$).

∎

Theorem 1 Let $\mathcal{T}^1$ and $\mathcal{T}^2$ be two composable STTS, let $\mathcal{T}^3$ be a STTS such that $\mathcal{T}^3 \sqsubseteq^r \mathcal{T}^1$, if $\mathcal{T}^1 \| \mathcal{T}^2$ is free of receptiveness problems then $\mathcal{T}^3 \| \mathcal{T}^2$ is free of receptiveness problems.

Proof. (*ad absurdum*). We have $\mathcal{T}^3 \sqsubseteq^r \mathcal{T}^1$ so that there exists $R \subseteq S^3 \times S^2$ such that $(S1)$ and $(S2)$ are satisfied.

$(H1)$ Assume there exists $(s_1^3, s_1^2) \in \mathsf{Reach}(\mathcal{T}^3 \| \mathcal{T}^2)$ such that:

$$\begin{array}{ll} \exists \sigma \in \Sigma^3_{\mathsf{out}},\ s_2^3 \in S^3 : (s_1^3, \sigma, s_2^3) \in \to^3 & (J1) \\ \qquad\qquad\qquad\quad \wedge\ \sigma \in \mathsf{Ref}_{\mathcal{T}^2}(s_1^2) & (J2) \end{array}$$

Then, by lemma 2, there exists s_1^1 such that : $(s_1^1, s_1^2) \in \mathsf{Reach}(\mathcal{T}^1 \| \mathcal{T}^2)$ $(J3)$
$\wedge\ (s_1^3, s_1^1) \in R$ $(J4)$

(since $s \in \mathsf{Reach}(\mathcal{T})$ iff s is n-reachable in $\mathcal{T}$ for some n)

By $(J1)$ and $(J4)$, there exists s_2^1 : $(s_1^1, \sigma, s_2^1) \in \to^1$. Moreover $\sigma \in \mathsf{Ref}_{\mathcal{T}^2}(s_1^2)$ by $(J2)$ and $\Sigma^3_{\mathsf{out}} = \Sigma^2_{\mathsf{in}} = \Sigma^1_{\mathsf{out}}$. In summary, we have $(s_1^1, s_1^2) \in \mathsf{Reach}(\mathcal{T}^1 \| \mathcal{T}^2)$, $\sigma \in \Sigma^1_{\mathsf{out}}$, and $s_2^1 \in S^1$ such that $(s_1^1, \sigma, s_2^1) \in \to^1$ and $\sigma \in \mathsf{Ref}_{\mathcal{T}^2}(s_1^2)$.

This is in contradiction with the fact that $\mathcal{T}^1 \| \mathcal{T}^2$ is free of receptiveness problems. This means that the assumption $(H1)$ above is false.

$(H2)$ Assume there exists $(s_1^3, s_1^2) \in \mathsf{Reach}(\mathcal{T}^3 \| \mathcal{T}^2)$ such that:

$$\begin{array}{ll} \exists \sigma \in \Sigma^2_{\mathsf{out}},\ s_2^2 \in S^3 : (s_1^2, \sigma, s_2^2) \in \to^2 & (J1) \\ \qquad\qquad\qquad\quad \wedge\ \sigma \in \mathsf{Ref}_{\mathcal{T}^3}(s_1^3) & (J2) \end{array}$$

Then, by lemma 2, there exists s_1^1 such that : $(s_1^1, s_1^2) \in \mathsf{Reach}(\mathcal{T}^1 \| \mathcal{T}^2)$ $(J3)$
$\wedge\ (s_1^3, s_1^1) \in R$ $(J4)$

(since $s \in \mathsf{Reach}(\mathcal{T})$ iff s is n-reachable in $\mathcal{T}$ for some n)

By $(J2)$ and $(J4)$, $\mathsf{Ref}_{\mathcal{T}^3}(s_1^3) = \mathsf{Ref}_{\mathcal{T}^1}(s_1^1)$ and $\sigma \in \mathsf{Ref}_{\mathcal{T}^1}(s_1^1)$. In summary, we have $(s_1^1, s_1^2) \in \mathsf{Reach}(\mathcal{T}^1 \| \mathcal{T}^2)$, $\sigma \in \Sigma^2_{\mathsf{out}}$, and $s_2^2 \in S^2$ such that $(s_1^2, \sigma, s_2^2) \in \to^2$ and $\sigma \in \mathsf{Ref}_{\mathcal{T}^2}(s_1^2)$.

This is in contradiction with the fact that $\mathcal{T}^1 \| \mathcal{T}^2$ is free of receptiveness problems. This means that the assumption $(H2)$ above is false.

Since $(H1)$ and $(H2)$ are false, it must be that $\mathcal{T}^3 \| \mathcal{T}^2$ is free of receptiveness problems.

∎

Theorem 2 Let $\mathcal{T}^1 = \langle S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_{\tau}, \to^1 \rangle$ and $\mathcal{T}^2 = \langle S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_{\tau}, \to^2 \rangle$ be two composable STTS, let $\mathcal{T}^3$ be a STTS such that $\mathcal{T}^3 \sqsubseteq^r \mathcal{T}^1$, and let $B \subseteq S^2$, if $\mathcal{T}^1$ controls $\mathcal{T}^2$ to avoid B then $\mathcal{T}^3$ controls $\mathcal{T}^2$ to avoid B.

Proof. We must show that

1. $\mathcal{T}^3 \| \mathcal{T}^1$ is free of receptiveness problems. This is a corollary of theorem 1 since $\mathcal{T}^1$ and $\mathcal{T}^2$ are composable and $\mathcal{T}^3 \sqsubseteq^r \mathcal{T}^1$.
2. $\mathsf{Reach}(\mathcal{T}^3 \| \mathcal{T}^2) \cap \{(s^3, s^2) \mid s^3 \in S^3 \wedge s^2 \in B\}$ is empty.
   $(H1)$ Assume there exists $(s_1^3, s_1^2) \in \mathsf{Reach}(\mathcal{T}^3 \| \mathcal{T}^2) \cap \{(s^3, s^2) \mid s^3 \in S^3 \wedge s^2 \in B\}$.

   Then $s_1^2 \in B$. By lemma 2, there exists s_1^1 such that $(s_1^1, s_1^2) \in \mathsf{Reach}(\mathcal{T}^1 \| \mathcal{T}^2)$ ((since $s \in \mathsf{Reach}(\mathcal{T})$ iff s is n-reachable in $\mathcal{T}$ for some n)). So that $(s_1^1, s_1^2) \in \mathsf{Reach}(\mathcal{T}^1 \| \mathcal{T}^2) \cap \{(s^1, s^2) \mid s^1 \in S^1 \wedge s^2 \in B\}$.
   This is in contradiction with the fact that $\mathcal{T}^1$ controls $\mathcal{T}^2$ to avoid B. This means that the assumption $(H1)$ above is false, that is $\mathsf{Reach}(\mathcal{T}^3 \| \mathcal{T}^2) \cap \{(s^3, s^2) \mid s^3 \in S^3 \wedge s^2 \in B\} = \emptyset$.

∎

B Proof of theorem of section 5

Theorem 4 [Simulation]

Let A be an ELASTIC controller, for any $\Delta, \Delta_L, \Delta_P \in \mathbb{Q}^{\geq 0}$ be such that $\Delta > 3\Delta_L + 4\Delta_P$, we have $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L, \Delta_P} \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.

Proof. Let $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L, \Delta_P} = (S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_{\tau}, \to^1)$ and $[\![A]\!]^{\mathsf{AAsap}}_{\Delta} = (S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_{\tau}, \to^2)$. Consider the relation $R \subseteq S^1 \times S^2$ that contains the pairs:

$$(s^1, s^2) = ((l^1, r^1, T^1, I^1, u^1, d^1, f^1), (l^2, v^2, I^2, d^2))$$

such that the following conditions hold:

$(R1)$ $l^1 = l^2$;
$(R2)$ for any $x \in \mathsf{Var}$, $|v^2(x) - (T^1 - r^1(x) + u^1)| \leq \Delta_L + \Delta_P$
$(R3)$ for any $\sigma \in \mathsf{Lab_{in}}$, $I^1(\sigma) = I^2(\sigma)$;
$(R4)$ $d^1 = d^2$;
$(R5)$ there exists (l^3, v^3, I^3, d^3) such that: $((l^2, v^2, I^2, d^2), \Delta_L - u^1, (l^3, v^3, I^3, d^3)) \in \rightarrow^2$.

Let us prove that R is a simulation relation:

$(S1)$ $(\iota^1, \iota^2) \in R$. We have to check the 5 rules of the simulation relation.
$(R1), (R2), (R3)$ and $(R4)$ are clearly true.
$(R5)$ To establish this property, we first note that $d^2 = 0$ and so $d^2 + \Delta_L < \Delta$ which implies $\forall t' \leq \Delta_L : d^2 + t' < \Delta$. Hence the two conditions of rule $(A4.6)$ are verified.
$(S2)$ Let us assume that $(s_1^1, s_1^2) = ((l_1^1, r_1^1, T_1^1, I_1^1, u_1^1, d_1^1, f_1^1), (l_1^2, v_1^2, I_1^2, d_1^2)) \in R$ and that $(s_1^1, \sigma, s_2^1) \in\rightarrow^1$ (with $s_2^1 = (l_2^1, r_2^1, T_2^1, I_2^1, u_2^1, d_2^1, f_2^1)$). We must prove the two following conditions:
$(S21)$ For each possible value for σ, we must establish the existence of a state $s_2^2 \in S^2$ such that $(s_1^2, \sigma, s_2^2) \in\rightarrow^2$ and $(s_2^1, s_2^2) \in R$.
$(S22)$ $\mathsf{Ref}_{\mathcal{T}^1}(s_1^1) = \mathsf{Ref}_{\mathcal{T}^2}(s_1^2)$.
We start with $(S21)$: Since $(s_1^1, s_1^2) \in R$ we know that:
$(H0)$ $s_1^2 = (l_1^1, v_1^2, I_1^1, d_1^1)$
$(H1)$ $\forall x \in \mathsf{Var} : T_1^1 - r_1^1(x) + u_1^1 - \Delta_L - \Delta_P \leq v_1^2(x) \leq T_1^1 - r_1^1(x) + u_1^1 + \Delta_L + \Delta_P$
$(H2)$ there exists $s^3 = (l^3, v^3, I^3, d^3) \in S^2$ such that: $((l_1^2, v_1^2, I_1^2, d_1^2), \Delta_L - u_1^1, (l^3, v^3, I^3, d^3)) \in\rightarrow^2$.
The rest of the proof works case by case on the different possible values for σ:

1. let $\sigma \in \Sigma_{\mathsf{in}}$
   Since $(s_1^1, \sigma, s_2^1) \in\rightarrow^1$ we know that:
   $(H3)$ $I_1^1(\sigma) = \perp$
   $(H4)$ $s_2^1 = (l_1^1, r_1^1, T_1^1, I_1^1[\{\sigma\} := 0], u_1^1, d_1^1, f_1^1)$
   - Let us first prove that $\exists s_2^2 \in S^2 : (s_1^2, \sigma, s_2^2) \in\rightarrow^2$. It amounts to prove that $I_1^1(\sigma) = \perp$ which is true by $(H3)$. Now that we know s_2^2 exists we can say that:
     $(H5)$ $s_2^2 = (l_1^1, v_1^2, I_1^1[\{\sigma\} := 0], d_1^1)$
   - It is now easy to prove that $(s_2^1, s_2^2) \in R$. Indeed, it is obvious that s_2^2 fulfills the five conditions of the simulation relation if $(s_1^1, s_1^2) \in R$.
2. let $\sigma \in \Sigma_{\mathsf{out}}$
   Since $(s_1^1, \sigma, s_2^1) \in\rightarrow^1$ we know that:
   $(H3)$ $\exists (l_1^1, l_2^1, g, \sigma, R) \in Edg : \lfloor T_1^1 \rfloor_{\Delta_P} - r_1^1 \models {}_{\Delta_S} g_{\Delta_S}$
   $(H4)$ $s_2^1 = (l_2^1, r_1^1[R := \lfloor T_1^1 \rfloor_{\Delta_P}], T_1^1, I_1^1, u_1^1, 0, \top)$
   - Let us first prove that $\exists s_2^2 \in S^2 : (s_1^2, \sigma, s_2^2) \in\rightarrow^2$. We use the same edge as in the implementation semantics (see $(H3)$). This amounts to prove that: $\forall x \in \mathsf{Var} : v_1^2(x) \models {}_{\Delta} g_{\Delta}(x)$. Now we know that $\forall x \in \mathsf{Var}$:

$$\begin{array}{ll}
\quad lb(g(x)) - \Delta_S \leq \lfloor T_1^1 \rfloor_{\Delta_P} - r_1^1(x) \leq rb(g(x)) + \Delta_S & (H3) \\
\rightarrow lb(g(x)) - \Delta_S - \Delta_P \leq T_1^1 - r_1^1(x) \leq rb(g(x)) + \Delta_S + \Delta_P & (\text{ lemma 1}) \\
\rightarrow lb(g(x)) - \Delta_L - 3\Delta_P \leq T_1^1 - r_1^1(x) \leq rb(g(x)) + \Delta_L + 3\Delta_P & (\Delta_S = \Delta_L + 2\Delta_P) \\
\rightarrow lb(g(x)) - 2\Delta_L - 4\Delta_P + u_1^1 \leq T_1^1 - r_1^1(x) + u_1^1 - \Delta_L - \Delta_P \wedge & \\
\quad T_1^1 - r_1^1(x) + u_1^1 + \Delta_L + \Delta_P \leq rb(g(x)) + 2\Delta_L + 4\Delta_P + u_1^1 & \\
\rightarrow lb(g(x)) - 2\Delta_L - 4\Delta_P + u_1^1 \leq v_1^2(x) \leq rb(g(x)) + 2\Delta_L + 4\Delta_P + u_1^1 & (H1) \\
\rightarrow lb(g(x)) - 2\Delta_L - 4\Delta_P \leq v_1^2(x) \leq rb(g(x)) + 3\Delta_L + 4\Delta_P & (0 \leq u_1^1 \leq \Delta_L) \\
\rightarrow lb(g(x)) - \Delta \leq v_1^2(x) \leq rb(g(x)) + \Delta & (3\Delta_L + 4\Delta_P < \Delta) \\
\rightarrow v_1^2(x) \models {}_{\Delta} g_{\Delta} &
\end{array}$$

     Now that it is established that $\exists s_2^2 \in S^2 : (s_2^1, \sigma, s_2^2) \in\rightarrow^2$ we know that:

$(H5)$ $s_2^2 = (l_2^1, v_1^2[R := 0], I_1^1, 0)$

- It remains to prove that $(s_2^1, s_2^2) \in R$ which means we must check the five rules of the simulation relation. $(R1), (R3)$ and $(R4)$ are clearly true.
  To prove $(R2)$ we have to prove that $\forall x \in \mathsf{Var}$:
  $\begin{cases} T_1^1 - r_1^1[R := \lfloor T_1^1 \rfloor_{\Delta_P}](x) + u_1^1 - \Delta_L - \Delta_P \leq v_1^2[R := 0](x) \\ v_1^2[R := 0](x) \leq T_1^1 - r_1^1[R := \lfloor T_1^1 \rfloor_{\Delta_P}](x) + u_1^1 + \Delta_L + \Delta_P \end{cases}$
  This proposition is the same as H_1 for any $x \notin R$. For $x \in R$, this amounts to prove:
  $$T_1^1 - \lfloor T_1^1 \rfloor_{\Delta_P} + u_1^1 - \Delta_L - \Delta_P \leq 0 \leq T_1^1 - \lfloor T_1^1 \rfloor_{\Delta_P} + u_1^1 + \Delta_L + \Delta_P.$$
  We first instantiate lemma 1:
  $T_1^1 - \Delta_P \leq \lfloor T_1^1 \rfloor_{\Delta_P} \leq T_1^1 + \Delta_P$
  $\rightarrow T_1^1 - \Delta_P - \lfloor T_1^1 \rfloor_{\Delta_P} \leq 0 \leq T_1^1 + \Delta_P - \lfloor T_1^1 \rfloor_{\Delta_P}$
  $\rightarrow T_1^1 - \lfloor T_1^1 \rfloor_{\Delta_P} + u_1^1 - \Delta_L - \Delta_P \leq 0 \leq T_1^1 - \lfloor T_1^1 \rfloor_{\Delta_P} + u_1^1 + \Delta_L + \Delta_P$ $(u_1^1 - \Delta_L \leq 0)$
  This establishes $(R2)$.
  To prove $(R5)$ we can intuitively state that at a discrete transition, the deadline of the next possible discrete transition can only stay still or move away but certainly not be earlier than before. As before, a time step of length $\Delta_L - u_1^1$ was possible, so must it still be the case now.

3. let $\sigma \in \Sigma_\tau . \Sigma_\tau = \mathsf{Lab}_\tau \cup \overline{\mathsf{Lab}_{\mathsf{in}}} \cup \{\epsilon\}$. The proof for the first two sets is similar to the previous case. Let $\sigma = \epsilon$
   Since $(s_1^1, \epsilon, s_2^1) \in \rightarrow^1$ we know by $(P4.5)$ that

$(H3)$ $s_2^1 = (l_1^1, r_1^1, T_1^1 + u_1^1, I_1^1, 0, d_1^1, \bot)$

$(H4)$ $f_1^1 = \bot$ or

  - for any $\bar{\sigma}$ such that $\sigma \in \mathsf{Lab}_{\mathsf{in}}$, for any $(l_1^1, l', \sigma, g, R) \in \mathsf{Edg}$, we have that either $\lfloor T_1^1 \rfloor_{\Delta_P} - r_1^1 \not\models {}_{\Delta_S} g_{\Delta_S}$ or $I_1^1(\sigma) \leq u_1^1$
  - for any $\sigma \in \mathsf{Lab}_{\mathsf{out}}$, for any $(l_1^1, l', \sigma, g, R) \in \mathsf{Edg}$, we have that $\lfloor T_1^1 \rfloor_{\Delta_P} - r_1^1 \not\models {}_{\Delta_S} g_{\Delta_S}$

By rule $(A4.5)$ of the AAsap-semantics, we know that there exists $s_2^2 \in S^2$ such that (s_1^2, ϵ, s_2^2) and

$(H5)$ $s_2^2 = s_1^2 = (l_1^1, v_1^2, I_1^1, d_1^1)$.

Now we have to prove that $(s_2^1, s_2^2) \in R$. $(R1), (R3)$ and $(R4)$ are clearly true. Proving $(R2)$ amounts to prove that

$$\forall x \in \mathsf{Var} : T_1^1 + u_1^1 - r_1^1(x) + 0 - \Delta_L - \Delta_P \leq v_1^2(x) \leq T_1^1 + u_1^1 - r_1^1(x) + 0 + \Delta_L + \Delta_P$$

which turns out to be equivalent to $(H1)$.

Let us now prover that there exists s^3 s.t. $((l_1^1, v_1^2, I_1^1, d_1^1), \Delta_L, s^3) \in \rightarrow^2$. According to rule $(A4.6)$, it amounts to prove that

$(T1)$ for any edge $(l_1^1, l', \sigma, g, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}_{\mathsf{out}} \cup \mathsf{Lab}_\tau$, we have that:
$$\forall t' : 0 \leq t' \leq \Delta_L : (d_1^1 + t' \leq \Delta \vee \mathsf{TS}(v_1^2 + t', g) \leq \Delta)$$

$(T2)$ for any edge $(l_1^1, l', \sigma, g, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}_{\mathsf{in}}$, we have that:
$$\forall t' : 0 \leq t' \leq \Delta_L : (d_1^1 + t' \leq \Delta \vee \mathsf{TS}(v_1^2 + t', g) \leq \Delta \vee (I_1^1 + t')(\sigma) \leq \Delta)$$

If $f_1^1 = \top$ it implies that the program has made a discrete transition during the last loop, which means that $d_1^1 \leq \Delta_L$ and thus that $d_1^1 + \Delta_L \leq 2\Delta_L \leq \Delta$ because we know that, by hypothesis, $\Delta > 2\Delta_L + 4\Delta_P$, which makes $(T1)$ and $(T2)$ true for any t'.

If $f_1^1 \neq \top$, the proof is less trivial. We first make a proof for labels of $(T1)$.

$\forall (l_1^1, l', \sigma, g, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}_{\mathsf{out}}$ we have $\lfloor T_1^1 \rfloor_{\Delta_P} \not\models {}_{\Delta_S} g_{\Delta_S}$ by $(H4)$. There are two possible cases:

(a) $\exists x \in \mathsf{Var}$ such that

$$\begin{array}{ll}
\lfloor T_1^1 \rfloor_{\Delta_P} - r_1^1(x) < lb(g(x)) - \Delta_S & \\
\to \lfloor T_1^1 \rfloor_{\Delta_P} - r_1^1(x) < lb(g(x)) - \Delta_L - 2\Delta_P & (\Delta_S = \Delta_L + 2\Delta_P) \\
\to T_1^1 - r_1^1(x) < lb(g(x)) - \Delta_L - \Delta_P & (\text{lemma 1}) \\
\to T_1^1 - r_1^1(x) + u_1^1 + \Delta_L + \Delta_P < lb(g(x)) + u_1^1 & \\
\to v_1^2(x) < lb(g(x)) + u_1^1 & (H1) \\
\to v_1^2(x) < lb(g(x)) + \Delta_L & (u_1^1 \leq \Delta_L) \\
\to \forall t' : 0 \leq t' \leq \Delta_L : v_1^2(x) + t' \leq lb(g(x)) + 2\Delta_L & \\
\to \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_1^2(x) + t', g(x)) \leq 2\Delta_L & \\
\to \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_1^2(x) + t', g(x)) \leq \Delta & (2\Delta_L < \Delta) \\
\to \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_1^2 + t', g) \leq \Delta &
\end{array}$$

(b) $\exists x \in Var$ such that

$$\begin{array}{ll}
\lfloor T_1^1 \rfloor_{\Delta_P} - r_1^1(x) > rb(g(x)) + \Delta_S & \\
\to \lfloor T_1^1 \rfloor_{\Delta_P} - r_1^1(x) > rb(g(x)) + \Delta_L + 2\Delta_P & (\Delta_S = \Delta_L + 2\Delta_P) \\
\to T_1^1 - r_1^1(x) > rb(g(x)) + \Delta_L + \Delta_P & (\text{lemma 1}) \\
\to T_1^1 - r_1^1(x) + u_1^1 - \Delta_L - \Delta_P > rb(g(x)) + u_1^1 & \\
\to v_1^2(x) > rb(g(x)) + u_1^1 & (H1) \\
\to v_1^2(x) > rb(g(x)) & \\
\to \forall t' : 0 \leq t' \leq \Delta_L : v_1^2(x) + t' > rb(g(x)) & \\
\to \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_1^2(x) + t', g(x)) \leq \Delta & (v_1^2(x) + t' \not\models g(x)) \\
\to \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_1^2 + t', g) \leq \Delta &
\end{array}$$

Thus, both case implies that $(T1)$ is true.

The proof for $(T2)$ is the same if we have, by $(H4)$, $\lfloor T_1^1 \rfloor_{\Delta_P} \not\models {}_{\Delta_S} g_{\Delta_S}$. If not, we have $I_1^1(\sigma) < u_1^1$ which also allows to prove $(T2)$. Indeed

$$\begin{array}{ll}
I_1^1(\sigma) < u_1^1 & \\
\to \forall t' : 0 \leq t' \leq \Delta_L : I_1^1(\sigma) + t' < \Delta_L & \\
\to \forall t' : 0 \leq t' \leq \Delta_L : I_1^1(\sigma) + t' < \Delta & (\Delta_L \leq \Delta)
\end{array}$$

4. let $\sigma \in \mathbb{R}^{\geq 0}$. For the sake of clarity let us consider that $\sigma = t$.

   Since $(s_1^1, t, s_2^1) \in \to^1$ we know by $(P4.6)$ that

   $(H3)$ $s_2^1 = (l_1^1, r_1^1, t_1^1, I_1^1 + t, u_1^1 + t, d_1^1 + t, f_1^1)$;

   $(H4)$ $u_1^1 + t \leq \Delta_L$.

   With those facts, we know that there exists $s_2^2 = (l_2^2, v_2^2, I_2^2, d_2^2, d_2^2) \in S^2$ such that $(s_1^2, t, s_2^2) \in \to^2$ because $(s_1^2, \Delta_L, s^3) \in \to^2$ $(H2)$ and $t \leq \Delta_L - u_1^1$ $(H4)$.

   Now that we have $(s_1^2, t, s_2^2) \in \to^2$, we know that:

   $(H5)$ $s_2^2 = (l_1^1, v_1^2 + t, I_1^1 + t, d_1^1 + t)$

   We can now prove that $(s_2^1, s_2^2) \in R$. We have to check the five points of the simulation relation: $(R1), (R2), (R3)$ and $(R4)$ are easy to prove using hypotheses $(H1)$ to $(H5)$.

   For $(R5)$, since by $(H2)$, there exists $s^3 = (l^3, v^3, I^3, d^3) \in S^2$ such that: $((l_1^2, v_1^2, I_1^2, d_1^2), \Delta_L - u_1^1, (l^3, v^3, I^3, d^3)) \in \to^2$. We have $((l_1^1, v_1^2 + t, I_1^1 + t, d_1^1 + t), (\Delta_L - u_1^1 - t), (l^3, v^3, I^3, d^3) \in S^2$.

The proof for $(S22)$ is trivial since the refusal function for a state s can be calculated in both of our semantics from the component I of the state s:

- $\forall s = (l, r, T, I, u, d, f) \in S^1 : \mathsf{Ref}_{[\![A]\!]^{\mathsf{Prg}}_{\Delta_L, \Delta_P}}(s) = \{\sigma \in \mathsf{Lab_{in}} \mid I(\sigma) \neq \perp\}$
- $\forall s = (l, v, I, d) \in S^2 : \mathsf{Ref}_{[\![A]\!]^{\mathsf{AAsap}}_{\Delta}}(s) = \{\sigma \in \mathsf{Lab_{in}} \mid I(\sigma) \neq \perp\}$

This fact and the condition $(R3)$ of the simulation R relation implies $(S22)$.

∎

C Proof of theorem of the section 6

Theorem 6 For any Elastic controller A, for any $\Delta \in \mathbb{Q}^{>0}$, we can construct effectively a timed automaton $\mathcal{A}^{\Delta} = \mathcal{F}(A, \Delta)$ such that $[\![A]\!]^{\mathsf{AAsap}}_{\Delta} \sqsubseteq^r [\![\mathcal{A}^{\Delta}]\!]$ and $[\![\mathcal{A}^{\Delta}]\!] \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.

Proof. We give the construction of $\mathcal{F}(A,\Delta)$. Let A be the tuple $\langle \mathsf{Loc}^1, l_0^1, \mathsf{Var}^1, \mathsf{Lab}^1, \mathsf{Edg}^1\rangle$ and let $\mathcal{F}(A,\Delta) = \langle \mathsf{Loc}^2, l_0^2, \mathsf{Var}^2, \mathsf{Inv}^2, \mathsf{Flow}^2, \mathsf{Lab}^2, \mathsf{Edg}^2\rangle$ be the timed automaton such that:

- $\mathsf{Loc}^2 = \{(l,b) \mid l \in Loc^1 \wedge b \in [\Sigma_{\mathsf{in}} \to \{\top,\bot\}]\}$;
- $l_0^2 = (l_0^1, b_\bot)$ where $b_\bot$ is such that $b_\bot(\sigma) = \bot$ for any $\sigma \in \Sigma_{\mathsf{in}}$;
- $\mathsf{Var}^2 = \mathsf{Var}^1 \cup \{y_\sigma \mid \sigma \in \Sigma_{\mathsf{in}}\} \cup \{d\}$;
- Flow^2 is such that for any $x \in \mathsf{Var}^2$, for any $l \in Loc^2$:$[\![\mathsf{Flow}^2(l)]\!]\ (x) = 1$;
- $\mathsf{Lab}^2 = \mathsf{Lab}^1_{\mathsf{out}} \cup \mathsf{Lab}^1_{\mathsf{int}} \cup \mathsf{Lab}^1_{\mathsf{in}} \cup \overline{\mathsf{Lab}^1_{\mathsf{in}}}$;
- Edg^2 is defined as follows.
  $((l,b),(l',b'),\sigma,{}_\Delta g_\Delta, R') \in \mathsf{Edg}^2$ iff one of the following condition holds:
  - $\sigma \in \mathsf{Lab}^1_{\mathsf{out}}$ and
    1. there exists $(l,l',\sigma,g,R) \in \mathsf{Edg}^1$
    2. $b' = b$
    3. $R' = R \cup \{d\}$
  - $\sigma = \bar{\alpha} \in \overline{\mathsf{Lab}^1_{\mathsf{in}}}$ and
    1. there exists $(l,l',\alpha,g,R) \in \mathsf{Edg}^1$
    2. $b(\alpha) = \top$
    3. $b' = b[\alpha := \bot]$
    4. $R' = R \cup \{d\}$
  - $\sigma \in \mathsf{Lab}^1_{\mathsf{int}}$ and
    1. there exists $(l,l',\sigma,g,R) \in \mathsf{Edg}^1$
    2. $b' = b$
    3. $R' = R \cup \{d\}$
  - $\sigma \in \mathsf{Lab}^1_{\mathsf{in}}$ and
    1. $l' = l$
    2. $b(\sigma) = \bot$
    3. $b' = b[\sigma := \top]$
    4. $g = \mathsf{true}$
    5. $R' = \{y_\sigma\}$
  - $\sigma = \epsilon$ and
    1. $l' = l$
    2. $b = b'$
    3. $g = \mathsf{true}$
    4. $R' = \emptyset$
- The function Inv^2 is defined as follows. Let $EVT(l) = \{(l,l',\sigma,g,R) \in \mathsf{Edg}^2 \mid \sigma \in \overline{\mathsf{Lab}^1_{\mathsf{in}}}\}$. Let $ACT(l) = \{(l,l',\sigma,g,R) \in \mathsf{Edg}^2 \mid \sigma \in \mathsf{Lab}^1_{\mathsf{out}} \cup \mathsf{Lab}^1_{\mathsf{int}}\}$.

  $$\mathsf{Inv}^2(l) = \Big(\bigwedge_{(l,l',\sigma,g,R)\in EVT(l)} \big(d \le \Delta \vee \neg({}^\Delta g) \vee y_\sigma \le \Delta\big)\Big) \wedge \Big(\bigwedge_{(l,l',\sigma,g,R)\in ACT(l)} \big(d \le \Delta \vee \neg({}^\Delta g)\big)\Big)$$

  where ${}^\Delta g(x)$ is the expression $x \in [a+\Delta, b]$ if $g(x)$ is the expression $x \in [a,b]$

To establish that the construction above is correct, we proceed as follows. We show that:

(1) $[\![A]\!]^{\mathsf{AAsap}}_\Delta \sqsubseteq^r [\![\mathcal{F}(A,\Delta)]\!]$
(2) $[\![\mathcal{F}(A,\Delta)]\!] \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_\Delta$

Let $[\![A]\!]^{\mathsf{AAsap}}_\Delta = (S^1, \iota^1, \Sigma^1_{\mathsf{in}}, \Sigma^1_{\mathsf{out}}, \Sigma^1_\tau, \to^1)$ and $[\![\mathcal{F}(A,\Delta)]\!] = (S^2, \iota^2, \Sigma^2_{\mathsf{in}}, \Sigma^2_{\mathsf{out}}, \Sigma^2_\tau, \to^2)$.

To prove (1) we can use the simulation relation $R \subseteq S^1 \times S^2$ such that $((l^1, v^1, I^1, d^1), ((l^2, b^2), v^2)) \in R$ iff:

1. $l^1 = l^2$
2. for any $\sigma \in \mathsf{Lab}_{\mathsf{in}}$, $\begin{cases} b^2(\sigma) = \bot & \text{iff } I^1(\sigma) = \bot \\ b^2(\sigma) = \top \wedge v^2(y_\sigma) = I^1(\sigma) & \text{iff } I^1(\sigma) \neq \bot \end{cases}$
3. $v^2_{|\mathsf{Var}^1} = v^1$ ($v_{|X}$ is the restriction of v to X)
4. $v^2(d) = d^1$

Let us prove that R is a simulation relation:

$(S1)$ $(\iota^1, \iota^2) \in R$. We have to check the 4 rules $(R1-4)$ of the simulation relation, which are clearly true.

$(S2)$ Let us assume that $(s_1^1, s_1^2) = ((l_1^1, v_1^1, I_1^1, d_1^1), ((l_1^2, b_1^2), v_1^2)) \in R$ and that $(s_1^1, \sigma, s_2^1) \in \to^1$ (with $s_2^1 = (l_2^1, v_2^1, I_2^1, d_2^1)$).
We must prove the two following conditions:

($S21$) For each possible value of σ, we must establish the existence of a state $s_2^2 \in S^2$ such that $(s_1^2, \sigma, s_2^2) \in \rightarrow^2$ and $(s_2^1, s_2^2) \in R$.

($S22$) $\mathsf{Ref}_{\mathcal{T}^1}(s_1^1) = \mathsf{Ref}_{\mathcal{T}^2}(s_1^2)$.

We start with ($S21$): Since $(s_1^1, s_1^2) \in R$ we know that:

($H0$) $s_1^2 = ((l_1^2, b_1^2), v_1^2)$

($H1$) $v^2_{1|\mathsf{Var}^1} = v_1^1$, $v_1^2(d) = d_1^1$, and $v_1^2(y_\sigma) = I_1^1(\sigma)$ iff $I_1^1(\sigma) \neq \bot$.

Let $\sigma \in \mathbb{R}^{\geq 0}$ (other cases are straightforward and left to the reader). For the sake of clarity let us consider that $\sigma = t$.

Since $(s_1^1, t, s_2^1) \in \rightarrow^1$ we know by ($A4.6$) that

($H2$) for any edge $(l_1^1, l', \sigma, g, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}^1_{\mathsf{out}} \cup \mathsf{Lab}^1_{\mathsf{int}}$, we have that:
$$\forall t' : 0 \leq t' \leq t : (d_1^1 + t' \leq \Delta \vee \mathsf{TS}(v_1^1 + t', g) \leq \Delta)$$

($H3$) for any edge $(l_1^1, l', \sigma, g, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}^1_{\mathsf{in}}$, we have that:
$$\forall t' : 0 \leq t' \leq t : (d_1^1 + t' \leq \Delta \vee (\mathsf{TS}(v_1^1 + t', g) \leq \Delta \vee (I_1^1 + t')(\sigma) \leq \Delta)$$

It is easy to prove by ($H1$) that:

* $d_1^1 + t' \leq \Delta \leftrightarrow v_1^2(d) + t' \leq \Delta$
* $(I_1^1 + t')(\sigma) \leq \Delta \leftrightarrow v_1^2(y_\sigma) + t' \leq \Delta$
* $\mathsf{TS}(v_1^1 + t', g) \leq \Delta \leftrightarrow v_1^2 + t' \models \neg({}^{\Delta}g)$

The third proposition can be proved as follows:

$$\begin{aligned}
&\mathsf{TS}(v_1^1 + t', g) \leq \Delta \\
\leftrightarrow\ & (v_1^1 + t' \models g) \rightarrow (\exists x \in \mathsf{Var}^1 : v_1^1(x) + t' - \Delta \leq lb(g(x)) \vee v_1^1(x) + t' - \Delta > rb(g(x))) \\
\leftrightarrow\ & (v_1^1 + t' \models g) \rightarrow (\exists x \in \mathsf{Var}^1 : lb(g(x)) \leq v_1^1(x) + t' \leq rb(g(x)) \wedge \\
& \qquad (v_1^1(x) + t' - \Delta \leq lb(g(x)) \vee v_1^1(x) + t' - \Delta > rb(g(x)))) \\
\leftrightarrow\ & (v_1^1 + t' \models g) \rightarrow (\exists x \in \mathsf{Var}^1 : lb(g(x)) \leq v_1^1(x) + t' \leq rb(g(x)) \ \wedge\ v_1^1(x) + t' - \Delta \leq lb(g(x)) \\
\leftrightarrow\ & (v_1^1 + t' \not\models g) \vee (\exists x \in \mathsf{Var}^1 : lb(g(x)) \leq v_1^1(x) + t' \leq min(rb(g(x)), lb(g(x)) + \Delta \\
\leftrightarrow\ & (\exists x \in \mathsf{Var}^1 : v_1^1(x) + t' < lb(g(x)) \vee v_1^1(x) + t' > rb(g(x))) \\
& \vee (\exists x \in \mathsf{Var}^1 : lb(g(x)) \leq v_1^1(x) + t' \leq min(rb(g(x)), lb(g(x)) + \Delta)) \\
\leftrightarrow\ & \exists x \in \mathsf{Var}^1 : v_1^1(x) + t' < lb(g(x)) + \Delta \vee v_1^1(x) + t' > rb(g(x)) \\
\leftrightarrow\ & \exists x \in \mathsf{Var}^1 : v_1^1(x) + t' \notin [lb(g(x)) + \Delta, rb(g(x))] \\
\leftrightarrow\ & v_1^1 + t' \not\models {}^{\Delta}g \\
\leftrightarrow\ & v_1^1 + t' \models \neg({}^{\Delta}g) \\
\leftrightarrow\ & v_1^2 + t' \models \neg({}^{\Delta}g)
\end{aligned}$$

In summary, we showed that ($H2$) and ($H3$) implies $\forall 0 \leq t' \leq t : v_1^2 + t' \models \mathsf{Inv}^2(l_1^2)$. Thus, the state $s_2^2 = ((l_1^2, b_1^2), v_2^2)$ where $v_2^2 = v_1^2 + t$ is such that $(s_1^2, t, s_2^2) \in \rightarrow^2$ (for each $x \in \mathsf{Var}^2$, we define the function f_x by $f_x(t) = v_1^2(x) + t$. It is easy to check that $(i) f_x(0) = v_1^2(x)$, $(ii) f_x(t) = v_2^2(x)$ and $(iii) \forall 0 < t' < t : \dot{f}(t') = 1$).

The reader can easily check that $(s_2^1, s_2^2) \in R$.

The proof for ($S22$) is obvious since the refusal function for s_1^1 and s_1^2 can be computed as:

- $\mathsf{Ref}_{[\![A]\!]^{\mathsf{AAsap}}_{\Delta}}(s_1^1) = \{\sigma \in \mathsf{Lab_{in}} \mid I_1^1(\sigma) \neq \bot\}$
- $\mathsf{Ref}_{[\![\mathcal{F}(A,\Delta)]\!]}(s_1^2) = \{\sigma \in \mathsf{Lab_{in}} \mid b_1^2(\sigma) \neq \bot\}$

This fact and the condition ($R2$) of the definition of the simulation relation R implies ($S22$).

To prove (2) we can use the simulation relation R' such that $R'(s^2, s^1)$ iff $R(s^1, s^2)$. The proof is similar since we only used equivalence in our reasonings. ∎

D Example: The two tanks controller

This example is inspired by the water-level monitor described in [ACH+95]. The version we present here is sensibly complexified.

The system to control is a set of two tanks (Fig. 3) with a valve: a pump which can be turned on and off, and can change its position between tank 1 and tank 2 (only when it is turned off). We

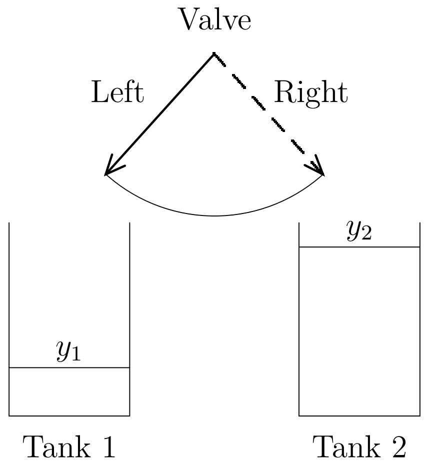


Fig. 3. Two tanks and a valve.

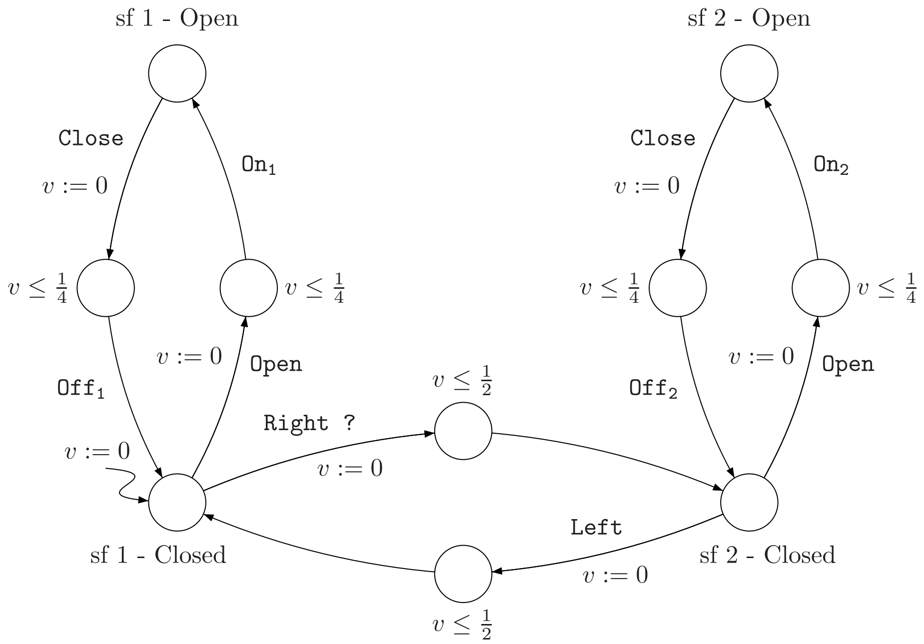


Fig. 4. Rectangular automaton specifying the behavior of the valve.

want to maintain the water level of both tanks between 0 and 15 units of length. When the valve is turned on and it is in left (or right) position, the water level of the corresponding tank 1 (or 2 respectively) rises by 2 units of length per time unit. In other cases (*e.g.* when the valve is turned off) the water level falls by 0.5 units of length per time unit.

The Fig. 4 shows the rectangular automaton for the valve; essentially, a received order (`Open`, `Close`, `Right` or `Left`) is followed by its execution which takes time: 15 units to turn the valve on/off, and 30 to change its position.

The rectangular automaton for the tank 1 is depicted in Fig. 5 (tank 2 is similar). Its water level (denoted by the variable y_1) has a derivative of 2 when the valve is on and a derivative of -0.5 when it is off. A signal ($\mathtt{Low}_1$ or $\mathtt{High}_1$) is emitted whenever the water level falls to 5 or rises to 10. At such moment it can be considered as urgent to turn the valve on (or off) and maybe to change its position before. Sometimes, the valve is turned off before the level reached 5 (or, conversely, it is turned on while the level is still above 10): in those cases, the corresponding signal ($\mathtt{Low}_1$ or $\mathtt{High}_1$) is emitted. If the level reaches 15 or goes below 0, then an error state (Err) is reached.

The ELASTIC controller is depicted in Fig. 6. When it receives a low-level or high-level signal from one of the tanks, it tries as quickly as possible to turn the valve on or off (with position change if required). In order to avoid oscillations between turning the valve on and off, and changing its position (which can damage the motor of the valve), we specify a delay of at least 1 time unit betweens two switching on/off and at least 2 time units between two position changes. The clock x_{valve} is reset whenever such an operation is done, so that a simple condition $x \geq 1$ or $x \geq 2$ has to be checked.

The analysis of this model by HYTECH is possible with the construction of theorem 6. Even though this example is quite big, we found that the error states can be avoided with $\Delta = \frac{1}{10}$, and subsequently for any lower value (corollary 2). We can use a LEGO MINDSTORMS™ platform to implement this model, if the hardware satisfy the condition $3\Delta_L + 4\Delta_P < 100$ms given by theorem 4. The safety property (the level in both tanks remains between 0 and 15 units of length) will automatically be true. In practice, it is thus implementable since the platform allows Δ_L to be as low as 6ms and offers a digital clock with $\Delta_P = 1$ms.

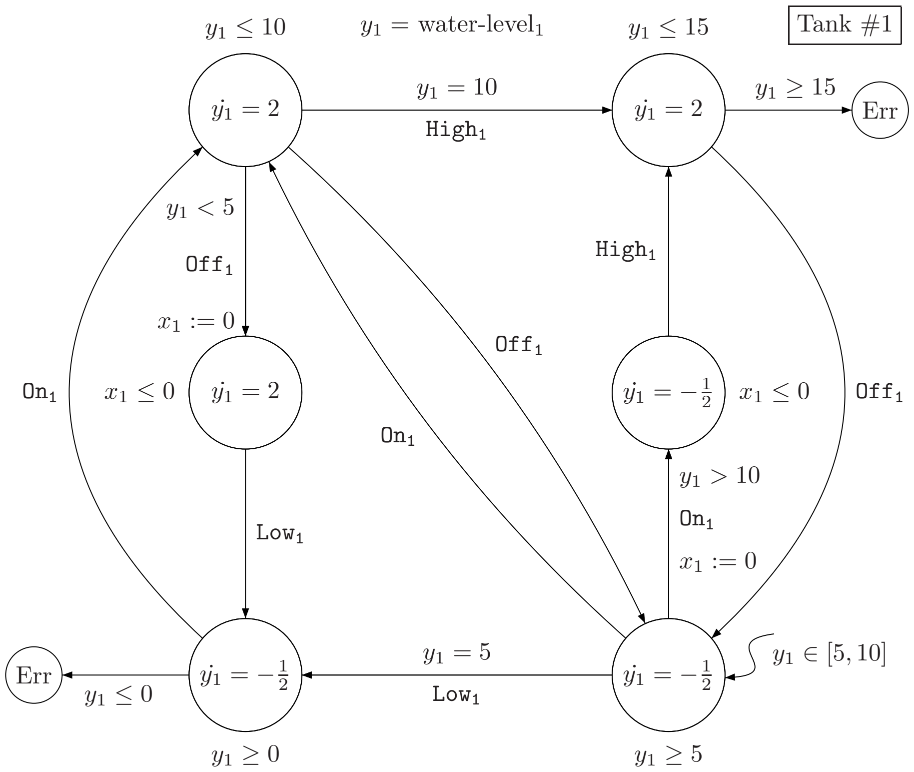


Fig. 5. Rectangular automaton specifying the behavior of one of the two tanks.

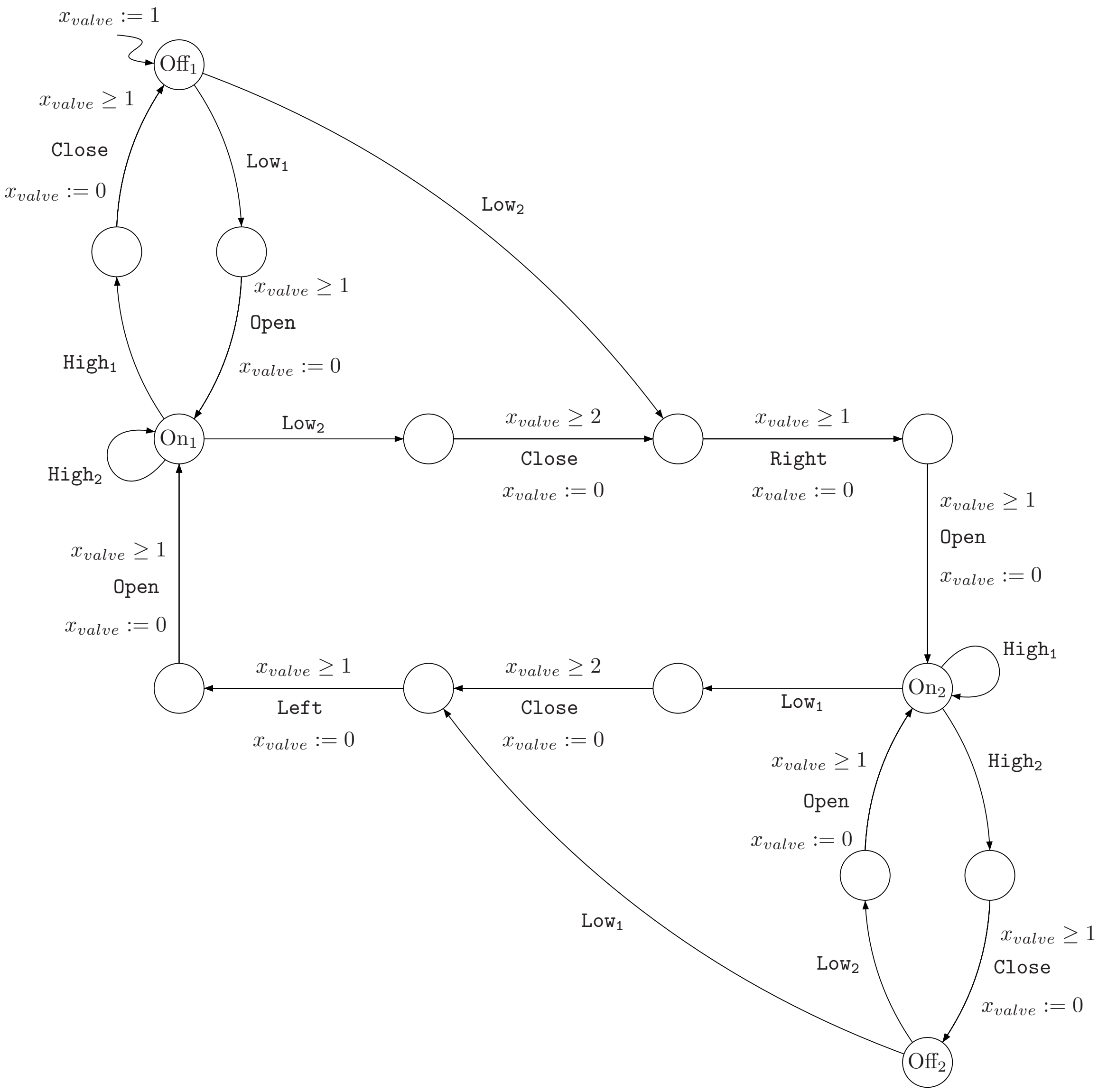


Fig. 6. ELASTIC controller for two tanks.