Almost ASAP Semantics: from Timed Models to Timed Implementations*

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

Computer Science Department, 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. Several verification and control problems have been studied for hybrid automata or interesting subclasses (see for example [HKPV98]). Tools like HyTech [HHWT95] 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.

* 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)

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 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 [AFP$^+$03]. But even if we prove our controller model non-zeno, that does not mean that it can be implemented. In fact, we recently showed in [CHR02] that there are (very simple) timed automata that respect a syntactic criterion that ensures nonzenoness 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.

Structure of the paper. The paper is organized as follows. In section 2, we recall the notions of *timed transition systems*, *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 timed 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*. Proofs and other examples can be found in a longer version of this paper at the following web page: `http://www.ulb.ac.be/di/ssd/jfr`.

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.

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 $\mathcal{T}$ is noted $\mathsf{Reach}(\mathcal{T})$.

We need to compose TTS. For that purpose, we need TTS with 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. 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 2 [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.

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 3 [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 4 [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:

- $\sigma \in \Sigma^1_{\mathsf{out}} \cup \Sigma^2_{\mathsf{out}} \cup \mathbb{R}^{\geq 0}$ and $(s^1_1, \sigma, s^1_2) \in \rightarrow^1$ and $(s^2_1, \sigma, s^2_2) \in \rightarrow^2$
- $\sigma \in \Sigma^1_\tau$ and $(s^1_1, \sigma, s^1_2) \in \rightarrow^1$ and $s^2_1 = s^2_2$
- $\sigma \in \Sigma^2_\tau$ 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 STTS wants to issue an output that is not accepted by the other one.

Definition 5 [Freedom of receptiveness problems] 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 a notion of *simulation*. That notion of simulation makes explicit the notion of refusal in order to preserve the potential freedom of receptiveness problem property of the model.

Definition 6 [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:

- $(\iota^1, \iota^2) \in R$
- for any $(s^1_1, s^2_1) \in R$, we have that:
  - 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 $(s^1_2, s^2_2) \in R$;
  - $\mathsf{Ref}_{\mathcal{T}^1}(s^1_1) = \mathsf{Ref}_{\mathcal{T}^2}(s^2_1)$.

The notion of simulation we have defined can be used 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 an* 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.*

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 7 [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:

- $\mathcal{T}^1 \| \mathcal{T}^2$ is free of receptiveness problems;
- $\mathsf{Reach}(\mathcal{T}^1 \| \mathcal{T}^2) \cap \{(s^1, s^2) \mid s^1 \in S^1 \wedge s^2 \in B\}$ 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.*

3 Timed automata

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

Let X be a finite set of 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 domain 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 $v(x) \in I$ 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 the rectangular constraint $x \in I$ if "$x \in I$" is the constraint over x in g and true if there is 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 timed automata and their *classical* semantics.

Definition 8 [Timed automata - syntax] A timed automaton is a tuple $\langle \mathsf{Loc}, l_0, \mathsf{Var}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg} \rangle$ where **(i)**Loc is a finite set of locations representing the discrete states of the automaton. **(ii)**$l_0 \in \mathsf{Loc}$ is the initial location. **(iii)**$\mathsf{Var} = \{x_1, \ldots, x_n\}$ is a finite set of real-valued clocks which value continuously increase as time passes with first derivative equal to one. **(iv)**$\mathsf{Inv} : \mathsf{Loc} \to \mathsf{Rect}(\mathsf{Var})$ 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. **(v)**$\mathsf{Lab} = \mathsf{Lab_{in}} \cup \mathsf{Lab_{out}} \cup \mathsf{Lab}_\tau$ is a structured finite alphabet of labels, partitioned into input labels $\mathsf{Lab_{in}}$, output labels $\mathsf{Lab_{out}}$, and internal labels Lab_τ. **(vi)**$\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Loc} \times \mathsf{Rect}(\mathsf{Var}) \times \mathsf{Lab} \times 2^{\mathsf{Var}}$ is a set of edges. Every edge (l, l', g, σ, R)

represents a discrete transition 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 9 [Timed automata - semantics] Let $A = \langle \mathsf{Loc}, l_0, \mathsf{Var}, \mathsf{Inv}, \mathsf{Lab}, \mathsf{Edg} \rangle$ be a timed automaton, the semantics of A, noted $[\![A]\!]$, is the STTS $\mathcal{T} = (S, \iota, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_{\tau}, \rightarrow)$ where: **(i)**$S = \{(l, v) \mid l \in \mathsf{Loc} \wedge v \in [\![\mathsf{Inv}(l)]\!]\}$. **(ii)**$\iota = (l_0, v_0)$ such that for any $x \in \mathsf{Var} : v_0(x) = 0$. **(iii)**$\Sigma_{\mathsf{in}} = \mathsf{Lab}_{\mathsf{in}}$, $\Sigma_{\mathsf{out}} = \mathsf{Lab}_{\mathsf{out}}$, and $\Sigma_{\tau} = \mathsf{Lab}_{\tau}$. **(iv)**the transition relation $\rightarrow$ is defined as follows: (a) 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 $v \models g$, $v' = v[R := 0]$. (b) For the continuous transitions, $((l, v), t, (l', v')) \in\rightarrow$ iff $l = l'$ and for each variable $x \in \mathsf{Var}$ we have the two following conditions satisfied : $v'(x) = v(x) + t$ and $\forall t' \in [0, t] : v + t' \in [\![\mathsf{Inv}(l)]\!]$.

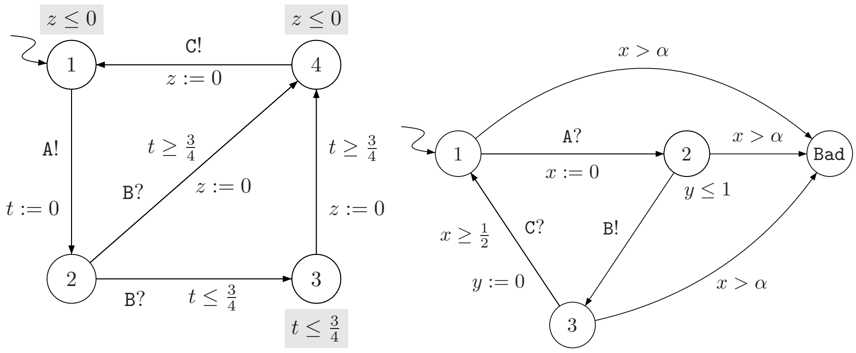
For simplicity, we restrict ourselves in this paper to environments modeled as timed automata. Nevertheless, all the results presented below hold if hte environment is modeled using any class of hybrid automata.

Running example. Consider Fig. 1. The timed automaton of Fig. 1(b) models a simple environment (a plant): when a request `A` is received, the response `B` is emitted before $y = 1$, and then the event `C` is accepted but it should occur at least one half time unit after `A` was received. Moreover, the event `A` *must* occur at least every α time units. If it was not the case, the environment would enter the location `Bad` modeling a fatal error. We will try to control the environment for $\alpha = 1$ and $\alpha = 2$.

The role of the controller is to produce an event `A` at least every α time units, to accept the subsequent event `B` and to output `C` respecting the timing constraint. An example of such a controller is given in Fig. 1(a). The designer has chosen here to react to the event `B` only after three quarter time unit. Given this controller for the system, we must verify that it gives orders in such a way that any resulting behavior of the environment avoids to enter the bad state. We must additionally verify that the controller is receptive to the event `B` from the environment (otherwise it could simply control the environment to avoid `Bad` by refusing to synchronize with `B`). We must also verify that the environment is ready to receive the orders (`A` and `C`) when emitted by the controller. The reader can check that, with the classical semantics of timed automata, the controller controls the environment such that the location `Bad` is not reachable for $\alpha = 1$ and $\alpha = 2$. Later, we will see that if $\alpha = 1$ then the controller is not implementable, on the other hand, if $\alpha = 2$ then the controller can be implemented and control the environment to avoid `Bad`.

As we already pointed out in the introduction, the classical semantics given in definition 9 is problematic for the controller part if our goal is to transfer the properties verified on the model to an implementation. Below, we illustrate the properties of the classical semantics that makes it 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



(a) The ASAP controller (b) The environment

Fig. 1. Running example.

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 A! proceeds exactly when $z = 0$, *i.e.* instantaneously. Clearly, no hardware can guarantee that the transition will always proceed without any delay. Second, synchronizations between the environment and the controller (*e.g.* transitions labeled B) cannot be implemented as instantaneous: some time is needed by the hardware to detect the incoming event B 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, as in [IKL$^+$00], 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 an 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[1] controllers.

Definition 10 [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 11 [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:

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

Definition 12 [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 $\{{}_\Delta g(x)_\Delta \mid g(x) \in g\}$.

[1] 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.

We are now ready to define the AASAP semantics. Intuitions are given right after the definition.

Definition 13 [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

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

where:

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

($A3$) $\Sigma_{\mathsf{in}} = \mathsf{Lab}_{\mathsf{in}}$, $\Sigma_{\mathsf{out}} = \mathsf{Lab}_{\mathsf{out}}$, 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 $I(\sigma) = \perp$ 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 $(l, l', g, \sigma, R) \in \mathsf{Edg}$, $v \models {}_{\Delta}g_{\Delta}$, $I(\sigma) \neq \perp$, $v' = v[R := 0]$ and $I' = I[\sigma := \perp]$;
  - ($A4.4$) let $\sigma \in \mathsf{Lab}_{\tau}$. We have $((l, v, I, d), \sigma, (l', v', I, 0)) \in\rightarrow$ iff there exists $(l, l', g, \sigma, R) \in \mathsf{Edg}$, $v \models {}_{\Delta}g_{\Delta}$, 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 $(l, l', g, \sigma, R) \in \mathsf{Edg}$ 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 $(l, l', g, \sigma, R) \in \mathsf{Edg}$ 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 $\perp$. The time elapsed since the last location change in the controller is recorded by d. 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 than 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 than Δ time units $(d+t' > \Delta)$, the guard of the transition has been true for more that Δ time units $(\mathsf{TS}(v+t', g) > \Delta)$, the last occurrence of σ event has not been treated yet and 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 $[\![A]\!]^{\mathsf{AAsap}}_{\Delta_2} \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_{\Delta_1}$.*

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

Corollary 1 *Let E be a timed 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 timed 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 AASAP 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 14 [Clock Rounding] Let $T \in \mathbb{R}, \Delta \in \mathbb{Q}^{>0}$, $\lfloor T \rfloor_\Delta = \lfloor \frac{T}{\Delta} \rfloor \Delta$.

We are now ready to define the program semantics. Intuitions are given right after the definition.

Definition 15 [Program Semantics] Let A be an ELASTIC controller and Δ_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 $\mathcal{T} = \langle S, \iota, \Sigma_{\mathsf{in}}, \Sigma_{\mathsf{out}}, \Sigma_\tau, \rightarrow\rangle$ 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 $(l, l', g, \sigma, R) \in \mathsf{Edg}$ 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 $\sigma \in \mathsf{Lab}_{\mathsf{in}}$. $((l,r,T,I,u,d,f),\sigma,(l,r,T,I',u,d,f)) \in\rightarrow$ iff $I(\sigma) = \bot$ and $I' = I[\sigma := 0]$;

($P4.3$) let $\bar{\sigma} \in \overline{\mathsf{Lab}_{\mathsf{in}}}$. $((l,r,T,I,u,d,\bot),\bar{\sigma},(l',r',T,I',u,0,\top)) \in\rightarrow$ iff there exists $(l,l',g,\sigma,R) \in \mathsf{Edg}$ 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 $(l,l',g,\sigma,R) \in \mathsf{Edg}$ 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 $\sigma \in \mathsf{Lab}_{\mathsf{in}}$, for any $(l,l',g,\sigma,R) \in \mathsf{Edg}$, we have that either $\lfloor T \rfloor_{\Delta_P} - r \not\models {}_{\Delta_S} g_{\Delta_S}$ or $I(\sigma) \leq u$
  - for any $\sigma \in \mathsf{Lab}_{\mathsf{out}} \cup \mathsf{Lab}_{\tau}$, for any $(l,l',g,\sigma,R) \in \mathsf{Edg}$, 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 $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P} \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.*

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 $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P} \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.*

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 timed 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: for any $\Delta \in \mathbb{Q}^{\geq 0}$, for any Elastic controller A, the AASAP semantics of A can be encoded using the classical semantics of a timed automaton A^{Δ} constructed from A and 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 $[\![A]\!]^{\mathsf{AAsap}}_{\Delta} \sqsubseteq^r [\![\mathcal{A}^{\Delta}]\!]$ and $[\![\mathcal{A}^{\Delta}]\!] \sqsubseteq^r [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.*

Corollary 3 *For any* Elastic *controller A, for any $\Delta \in \mathbb{Q}^{>0}$, for any timed automaton E with state space S^E, for any set of states $B \subseteq S^E$, we have that $[\![A]\!]^{\mathsf{AAsap}}_{\Delta}$ 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 a parameter);

– 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 the construction of theorem 6 to our running example (Fig. 1), we can ask HyTech to establish for which value of Δ, the tube of control strategies defined by the timed automaton obtained by the construction of theorem 6 is valid.

In the case $\alpha = 1$, the result is $\Delta = 0$. We can interpret this as follows: we have a correct model w.r.t to the classical ASAP semantics but it is nevertheless not guaranteed to be correct when implemented on a real hardware. In fact, the condition $\Delta = 0$ means that some transitions in the controller are required to be taken instantaneously, which is impossible in the real world. It must be admitted that the synchrony hypothesis was not realistic in that case. The second case ($\alpha = 2$) is more useful: the model is correct 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 $3\Delta_L + 4\Delta_P < 250ms$. For instance, we can implement the controller on the Lego Mindstorms™ platform, since it allows Δ_L to be as low as 6ms and offers a digital clock with $\Delta_P = 1$ms which is thus ample enough.

References

[AFP+03] Tobias Amnell, Elena Fersman, Paul Pettersson, Hongyan Sun, and Wang Yi. Code synthesis for timed automata. *Nordic Journal of Computing(NJC)*, 9, 2003.

[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.

[IKL+00] 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).*, 2000.

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