Under consideration for publication in Formal Aspects of Computing

Almost ASAP Semantics: from Timed Models to Timed Implementations [1]

Martin De Wulf*, Laurent Doyen*[2] and Jean-François Raskin*

*Computer Science Department
Université Libre de Bruxelles
Boulevard du Triomphe CP 212
B - 1050 Brussels, 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[3] 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 and show how it can be analyzed using the tool HyTech.

Keywords: Timed and Hybrid Automata – Semantics – Implementability – Verification.

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* [Hen96], 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 and

Correspondence and offprint requests to: Jean-François Raskin, Université Libre de Bruxelles, Bld du Triomphe CP 212, B - 1050 Brussels, Belgium. e-mail: Jean-Francois.Raskin@ulb.ac.be, Phone:+3226505591, Fax:+3226505609.
[1] A preliminary and short version of this paper appeared in the *Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control (HSCC 2004)*, Lecture Notes in Computer Science, volume 2993, pp. 296–310. This work was supported by the FRFC project "Centre Fédéré en Vérification" funded by the Belgian National Science Fundation (FNRS) under grant nr 2.4530.02.
[2] Research fellow supported by the Belgian National Science Fundation (FNRS).
[3] ASAP stands for "as soon as possible"

interesting subclasses have been identified (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.

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 granularity* and take their values in a discrete domain. Furthermore, those clocks may also be subject to drifts and so may not be perfectly accurate. 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 enforce 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 (using the traditional semantics) 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 simplifies* 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 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 fine granular 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* and *safety control*. We also define a notion of *simulation* that will ensure the conservation of 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 enrich this notion of real-time program with a modelization of *clock drifts* and prove that even with this additional issue, the AASAP semantics remains implementable. In section 7, we explain how the AASAP semantics can be *analyzed* and *used in practice*.

2. Preliminaries

In this section, we introduce the definition of timed transition systems and how to compose them. We recall the notion of simulation which will be the formal basis for our notion of refinement. Finally, we introduce the problem of safety control and show how the 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 $\{x \in \mathbb{R} \mid x \geq 0\}$ of the nonnegative real numbers. □

A state $s \in S$ of a TTS $\mathcal{T} = \langle S, \iota, \Sigma, \rightarrow \rangle$ is *reachable* if there exists a finite sequence $s_0 s_1 \ldots s_n$ 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 $\overline{\Sigma'} \cap \Sigma = \varnothing$. This operator on alphabets will be used later: intuitively, for an event, σ represents its occurrence in the environment and $\bar{\sigma}$ its perception by a controller.

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 into 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 is embedded. We model the communication between the two STTS using the mechanism of synchronization on common labels. This is a blocking communication mechanism. Nevertheless, on one hand 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 avoid such problems we impose *input enabledness* of the STTS that we compose, which means that input labels have the property of being enabled in every state. *Input enabledness* is a concept introduced in [LT87]. Formally :

Definition 3. [Input enabled STTS] A STTS $\mathcal{T} = \langle S, \iota, \Sigma^{\mathsf{in}}, \Sigma^{\mathsf{out}}, \Sigma^{\tau}, \rightarrow \rangle$ is *input enabled* if for all $\sigma \in \Sigma^{\mathsf{in}}$, for all $s_1 \in S$ there exists $s_2 \in S$ such that $(s_1, \sigma, s_2) \in \rightarrow$. □

We chose this semantics for inputs because it clarifies the presentation, but other semantics are possible: for instance in a preliminary version of this work [DDR04], we imposed *receptiveness* of controllers. Under this assumption, a controller must be fast enough to treat each occurrence of an event before the next occurrence arrives. One could also imagine a semantics where inputs arriving at the wrong time are simply ignored. Those aspects are orthogonal to the implementability aspects of the AASAP semantics.

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

Definition 4. [Composition of STTS] Two input enabled 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 $((s_1, s_2), \sigma, (s_1', s_2)) \in \rightarrow$ 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 $(s_1, \sigma, s_2) \in \rightarrow_1$ and $(s_1', \sigma, s_2') \in \rightarrow_2$
- $\sigma \in \Sigma_1^{\tau}$ and $(s_1, \sigma, s_1') \in \rightarrow_1$ and $s_2 = s_2'$
- $\sigma \in \Sigma_2^{\tau}$ and $(s_2, \sigma, s_2') \in \rightarrow_2$ and $s_1 = s_1'$

□

For the sake of simplicity, we defined composition of two STTS only. The construction can be easily generalized if we assume that two STTS never share an output label as in [LT87].

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* [Mil80].

Definition 5. [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 $\mathcal{T}_2$ is *simulable* by $\mathcal{T}_1$, noted $\mathcal{T}_2 \sqsubseteq \mathcal{T}_1$, if there exists a relation $R \subseteq S_2 \times S_1$ (called a *simulation relation*) such that:

- $(\iota_2, \iota_1) \in R$;
- for any $(s_2, s_1) \in R$, for any $\sigma \in \Sigma \cup \mathbb{R}^{\geq 0}$, for any s_2' such that $(s_2, \sigma, s_2') \in \rightarrow_2$, there exists $s_1' \in S_1$ such that $(s_1, \sigma, s_1') \in \rightarrow_1$ and $(s_2', s_1') \in R$.

□

Simulation can be used to define a notion of refinement. We say that the STTS $\mathcal{T}_2$ *refines* the STTS $\mathcal{T}_1$, if $\mathcal{T}_2 \sqsubseteq \mathcal{T}_1$. In the following, we use simulation relations because they preserve safety properties [AL91], but they also preserve stronger properties such as the ones expressed in the logics LTL [Pnu77] or ACTL [CBG88].

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 6. [Safety Control] Let $\mathcal{T}_1 = \langle S_1, \iota_1, \Sigma_1^{\mathsf{in}}, \Sigma_1^{\mathsf{out}}, \Sigma_1^{\tau}, \rightarrow_1 \rangle$ (the controller) and $\mathcal{T}_2 = \langle S_2, \iota_2, \Sigma_2^{\mathsf{in}}, \Sigma_2^{\mathsf{out}}, \Sigma_2^{\tau}, \rightarrow_2 \rangle$ (the environment) be two composable STTS. Let $B \subseteq S_2$, we say that $\mathcal{T}_1$ controls $\mathcal{T}_2$ to avoid B if $\mathsf{Reach}(\mathcal{T}_1 \| \mathcal{T}_2) \cap \{(s_1, s_2) \mid s_1 \in S_1 \wedge s_2 \in B\}$ is empty. □

We can now state two theorems linking our notion of refinement with the notion of safety control.

Theorem 1. Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two composable STTS, let $\mathcal{T}_3$ be a STTS such that $\mathcal{T}_3 \sqsubseteq \mathcal{T}_1$, 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.

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 $\mathcal{T}_3 \sqsubseteq \mathcal{T}_2$, and let $B \subseteq S_3 \subseteq S_2$, if $\mathcal{T}_1$ controls $\mathcal{T}_2$ to avoid B then $\mathcal{T}_1$ controls $\mathcal{T}_3$ to avoid B.

Theorem 1 shows that safety control is preserved by refinements of the controller. This has practical interest since it may require less resources (time, memory) for automatic tools to check safety control of a coarser model of the controller. In the same way, an over-approximation of the environment can be sufficient for proving safety in practice. In this case, we can also infer safety control for the refined environment, as stated by Theorem 2. Again this can be useful in practice.

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 $v : X \rightarrow \mathbb{R}^{\geq 0}$. We write $[Y \rightarrow E]$ for the set of all valuations of set of variables Y to domain E. For a set $V \subseteq [X \rightarrow \mathbb{R}^{\geq 0}]$ 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 $a, b \in \mathbb{Q}^{\geq 0} \cup \{+\infty\}$. $\mathbb{Q}^{\geq 0}$ 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 over X is in normal form if it contains at most one rectangular constraint for each variable $x \in X$, with the convention that the empty predicate p (such that $[\![p]\!]= \varnothing$) is represented by $\{x \in [+\infty, +\infty] \mid x \in X\}$; 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 $[X \to \mathbb{R}^{\geq 0} \cup \{\bot\}]$ as follows: $(v + t)(x) = v(x) + t$, if $v(x) \in \mathbb{R}^{\geq 0}$, and $(v + t)(x) = \bot$ otherwise. We are now equipped to define timed automata and their *classical* semantics.

Definition 7. [Timed automata - syntax] A timed automaton is a tuple $\langle \mathsf{Loc}, l^0, \mathsf{Var}, \mathsf{Inv}, \mathsf{Lab^{in}}, \mathsf{Lab^{out}}, \mathsf{Lab}^\tau, \mathsf{Edg}\rangle$ 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 clocks which value continuously increases as time passes with first derivative equal to one.
- $\mathsf{Inv} : \mathsf{Loc} \to \mathsf{Rect}(\mathsf{Var})$ 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.
- $\mathsf{Lab} = \mathsf{Lab^{in}} \cup \mathsf{Lab^{out}} \cup \mathsf{Lab}^\tau$ is a structured finite alphabet of labels, partitioned into input labels $\mathsf{Lab^{in}}$, output labels $\mathsf{Lab^{out}}$, and internal labels Lab^τ.
- $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Loc} \times \mathsf{Rect}(\mathsf{Var}) \times \mathsf{Lab} \times 2^{\mathsf{Var}}$ is a set of edges. An edge (l, l', g, σ, R) represents a discrete transition from location l to location l' with guard g, label σ and a subset $R \subseteq \mathsf{Var}$ of the variables to be reset.

□

Definition 8. [Timed automata - semantics] Let $A = \langle \mathsf{Loc}, l_0, \mathsf{Var}, \mathsf{Inv}, \mathsf{Lab^{in}}, \mathsf{Lab^{out}}, \mathsf{Lab}^\tau, \mathsf{Edg}\rangle$ be a timed automaton, the semantics of A is the *input enabled* STTS $[\![A]\!]= (S, \iota, \Sigma^{\mathsf{in}}, \Sigma^{\mathsf{out}}, \Sigma^\tau, \to)$ where:

- $S = \{(l, v) \mid l \in \mathsf{Loc} \wedge v \in [\![\mathsf{Inv}(l)]\!]\}$.
- $\iota = (l_0, v_0)$ such that for any $x \in \mathsf{Var} : v_0(x) = 0$.
- $\Sigma^{\mathsf{in}} = \mathsf{Lab^{in}}$, $\Sigma^{\mathsf{out}} = \mathsf{Lab^{out}}$, and $\Sigma^\tau = \mathsf{Lab}^\tau$.
- the transition relation $\to$ is defined as follows:
  - For the discrete transitions, $((l, v), \sigma, (l', v')) \in \to$ iff
    - · either there exists an edge $(l, l', g, \sigma, R) \in \mathsf{Edg}$ such that $v \models g$, $v' = v[R := 0]$;
    - · or there does not exist such an edge, $\sigma \in \mathsf{Lab^{in}}$ and $(l, v) = (l', v')$.
  - For the continuous transitions, $((l, v), t, (l', v')) \in \to$ iff $l = l'$ and for each variable $x \in \mathsf{Var}$ we have the two following conditions satisfied : $v'(x) = v(x) + t$ and $\forall t' \in [0, t] : v + t' \in [\![\mathsf{Inv}(l)]\!]$.

□

This semantics for a timed automaton is an *input enabled* STTS. Indeed, if no transition has been foreseen in the syntax for a given state and a given input label, the semantics allow a self loop on that state for that label.

A *timed word* is a pair (σ, τ) where $\sigma = \sigma_0\sigma_1\sigma_2\dots$ is an infinite sequence of labels $\sigma_j \in \mathsf{Lab}$ and $\tau = \tau_0\tau_1\tau_2\dots$ is an infinite sequence of real numbers. A timed word (σ, τ) is *accepted* by A if there exists a sequence $s_0, s'_0, s_1, s'_1, s_2, s'_2, \dots$ of states $s_j, s'_j \in S$ such that $s_0 = \iota$ and for all $i \geq 0$ we have $(s_i, \tau_i - \tau_{i-1}, s'_i) \in \rightarrow$ (with $\tau_{-1} = 0$) and $(s'_i, \sigma_i, s_{i+1}) \in \rightarrow$.

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

Running example. Consider Fig. 1. A "!" corresponds to an output event and a "?" to an input event. The timed automaton of Fig. 1(b) models a simple environment (a plant): when a request `A` is received, the response `B` is emitted when $y = 1$, and then the event `C` is accepted which reset the clock y. Moreover, the event `A` *must* occur at least every 2 time units, and the reaction `C` should occur before the timeout condition $x \geq \alpha$ become true. If it is not the case, the environment enters the location `Bad` modeling a fatal error. We want to control the environment for $\alpha = 1$ and $\alpha = 2$.

The role of the controller is to produce an event `A` at least every 2 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 output an `A` every 1 time unit, and to react to the event `B` as quickly as possible by emitting a `C`. 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 set of bad states (all the states in which the environment is in control location `Bad`). Observe that since the semantics of timed automata is *input enabled*, we are sure that the controller could not simply control the environment to avoid `Bad` by refusing to synchronize with `B`.

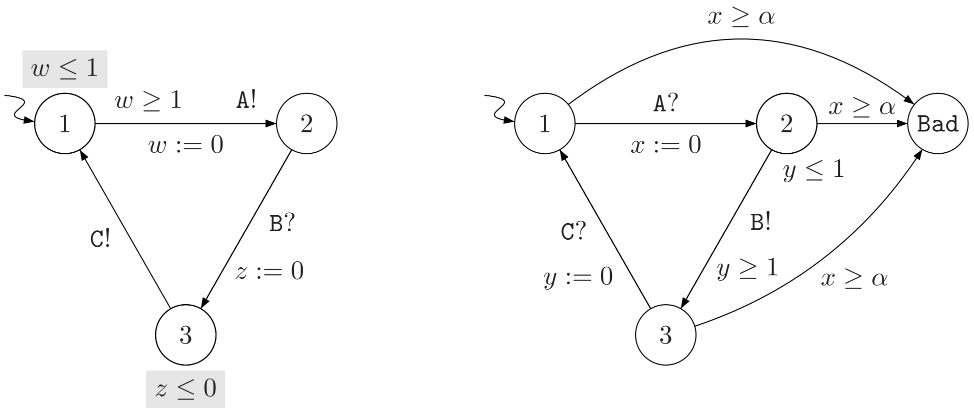
The reader can check that, with the classical semantics of timed automata, the controller controls the environment such that the location `Bad` is not reachable for $\alpha = 1$ and $\alpha = 2$. Later, we will see that if $\alpha = 1$ then the controller is not implementable, on the other hand, if $\alpha = 2$ then the controller can be implemented and controls the environment to avoid `Bad`.

4. Elastic Controllers and AASAP semantics

Main ingredients of our approach. As we already pointed out in the introduction, the classical semantics given in Definition 8 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 the example of Fig. 1(a)) are used to force the controller to take actions. Invariants can be removed if we assume an ASAP semantics for the controller: any action is taken as soon as possible, this is also called the *maximal progress assumption.* So, in the example, the transition labeled with `A`! fires exactly when $w = 1$, and the transition labeled with `C`! proceeds exactly when $z = 0$, *i.e.* instantaneously. Clearly, no hardware can guarantee that the transition will always be taken 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 granularity. It is then necessary to show that digital clocks can replace the real-valued clocks while preserving the verified safety properties.

These three problems illustrate 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 her/him 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.*



(a) The ASAP controller

(b) The environment

Fig. 1. Running example.

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 (sending σ), and in the controller (receiving $\bar{\sigma}$), however the time difference between the two events is bounded by Δ;
- guards are enlarged by some small amount depending on Δ.

We will now define formally this semantics and we will 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.

As stated previously, 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 and with closed guards. In the rest of the paper, we call the controller specified by this subclass ELASTIC[4] controllers.

Definition 9. [ELASTIC Controllers] An ELASTIC controller A is a tuple $\langle \mathsf{Loc}, l_0, \mathsf{Var}, \mathsf{Lab}^{\mathsf{in}}, \mathsf{Lab}^{\mathsf{out}}, \mathsf{Lab}^{\tau}, \mathsf{Edg}\rangle$ 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;
- $\mathsf{Lab} = \mathsf{Lab}^{\mathsf{in}} \cup \mathsf{Lab}^{\mathsf{out}} \cup \mathsf{Lab}^{\tau}$ 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', g, σ, 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.

□

[4] 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.

Before defining the AASAP semantics we need some more notations:

Definition 10. [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:

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

□

This definition is meaningful since $g \in \mathsf{Rect_c}(\mathsf{Var})$ defines a *closed* set.

Definition 11. [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. Intuitions are given right after the definition.

Definition 12. [AASAP semantics] Given an Elastic controller

$$A = \langle \mathsf{Loc}, l_0, \mathsf{Var}, \mathsf{Lab^{in}}, \mathsf{Lab^{out}}, \mathsf{Lab}^{\tau}, \mathsf{Edg} \rangle$$

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

$$\mathcal{T} = \langle S, \iota, \Sigma^{\mathsf{in}}, \Sigma^{\mathsf{out}}, \Sigma^{\tau}, \to \rangle$$

where:

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

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

($A4$) The transition relation is defined as follows:

- for the discrete transitions, we distinguish five cases:

  ($A4.1$) let $\sigma \in \mathsf{Lab^{out}}$. We have $((l,v,I,d), \sigma, (l',v',I,0)) \in \to$ 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 $\sigma \in \mathsf{Lab^{in}}$. We have $((l,v,I,d), \sigma, (l,v,I',d)) \in \to$ iff

  * either $I(\sigma) = \bot$ and $I' = I[\sigma := 0]$;
  * or $I(\sigma) \neq \bot$ and $I' = I$.

  ($A4.3$) let $\bar{\sigma} \in \overline{\mathsf{Lab^{in}}}$. We have $((l,v,I,d), \bar{\sigma}, (l',v',I',0)) \in \to$ iff there exists $(l,l',g,\sigma,R) \in \mathsf{Edg}$, $v \models {}_{\Delta}g_{\Delta}$, $I(\sigma) \neq \bot$, $v' = v[R := 0]$ and $I' = I[\sigma := \bot]$;

  ($A4.4$) let $\sigma \in \mathsf{Lab}^{\tau}$. We have $((l,v,I,d), \sigma, (l',v',I,0)) \in \to$ 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 $(l,v,I,d) \in S : ((l,v,I,d), \epsilon, (l,v,I,d)) \in \to$.

- for the continuous transitions:

  ($A4.6$) for any $t \in \mathbb{R}^{\geq 0}$, we have $((l,v,I,d), t, (l, v+t, I+t, d+t)) \in \to$ iff the two following conditions are satisfied:

  * for any edge $(l,l',g,\sigma,R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab^{out}} \cup \mathsf{Lab}^{\tau}$, 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 $\sigma \in \mathsf{Lab^{in}}$, 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 oldest "untreated" occurrence. The treatment of an event σ happens when a transition labelled with $\bar{\sigma}$ is fired. Once this oldest occurrence is treated, the function returns $\perp$ for σ until a new occurrence of σ, forgetting about the σ's that happened between the oldest occurrence and the treatment. 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 $(A4.2)$ defines how inputs from the environment are received by the controller. The controller maintains, through the function I, a list of events that have occurred and are not treated yet. An input event σ can be received at any time, but only the age of the oldest untreated σ is stored in the I function. Note that the rule ensures input enabledness of the controller. Rule $(A4.3)$ defines when inputs are treated by the controller. An input σ is treated when a transition with an enlarged guard and labelled with $\bar{\sigma}$ is fired. Once σ has been treated, the value of $I(\sigma)$ 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 $(\mathsf{TS}(v + t', g) > \Delta)$ and the last untreated occurrence of σ event has been emitted by the environment at least Δ time units ago $(I + t'(\sigma) > \Delta)$ (we define $\perp$ to be smaller than any rational value). This notion of urgency parameterized by Δ is the main difference between the AASAP semantics and the usual ASAP semantics.

Problems formulation We now define three problems that can be formulated about the AASAP semantics of an Elastic controller.

Definition 13. [Parametric safety control problem] Let E be a timed automaton, for which $[\![E]\!]$ has the set of states S_E, let $B \subseteq S_E$ be a set of bad states and let A be an Elastic controller, the *parametric safety control problem* asks

- **[Fixed]** whether $[\![A]\!]^{\mathsf{AAsap}}_{\Delta}$ controls $[\![E]\!]$ to avoid B for a given fixed value of Δ;
- **[Existence]** whether there exists $\Delta > 0$ such that $[\![A]\!]^{\mathsf{AAsap}}_{\Delta}$ controls $[\![E]\!]$ to avoid B;
- **[Maximization]** to maximize Δ such that $[\![A]\!]^{\mathsf{AAsap}}_{\Delta}$ controls $[\![E]\!]$ to avoid B.

□

As we will see later, the problem [Fixed] is useful when we know the characteristics of the hardware on which the control will be implemented, the problem [Existence] is useful to determine if the controller is implementable at all and the problem [maximization] is useful to determine what is the slowest hardware on which the controller can be implemented.

A property of the AASAP 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 $\Delta_2 \leq \Delta_1$ we have that $[\![A]\!]^{\mathsf{AAsap}}_{\Delta_2} \sqsubseteq [\![A]\!]^{\mathsf{AAsap}}_{\Delta_1}$.

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

Theorem 1 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.

Remark. Note that in general, faster is not always better, for example in scheduling theory. Furthermore, there is an extended research about the notion of "faster" in the field of process algebras (see [LV04] for example) which shows that it is better if you impose only upper bounds on the delays, exactly as in the AASAP semantics.

5. Implementability of the AASAP semantics

In this section, we show that any ELASTIC controller which controls (with $\Delta > 0$) 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 fine granular 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 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 we only want to prove that the AASAP semantics is implementable in a way or another, this implementation semantics is good enough. In section 7, we show how to use this semantics in the context of the LEGO MINDSTORMS™ platform.

This semantics is close to the one of PLC-automata introduced by Dierks [Die01]. The main difference is that we explicitly model the granularity of clocks.

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 $T \in \mathbb{R}^{\geq 0}, \Delta \in \mathbb{Q}^{>0}$, $\lfloor T \rfloor_\Delta = \lfloor \frac{T}{\Delta} \rfloor \Delta$ where $\lfloor x \rfloor$ is the greatest integer k such that $k \leq x$. Symmetrically, $\lceil T \rceil_\Delta = \lceil \frac{T}{\Delta} \rceil \Delta$ where $\lceil x \rceil$ is the smallest integer k such that $k \geq x$. □

Lemma 1 follows directly from the definition above.

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

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, $\Delta_P \in \mathbb{Q}^{>0}$. Let $\Delta_S = \lceil \Delta_L + \Delta_P \rceil_{\Delta_P}$. 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 $\mathsf{Lab}^{\mathsf{in}}$ into $\mathbb{R}^{\geq 0} \cup \{\perp\}$, $u \in \mathbb{R}^{\geq 0}$, $d \in \mathbb{R}^{\geq 0}$, and $f \in \{\top, \perp\}$;

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

$(P3)$ $\Sigma^{\mathsf{in}} = \mathsf{Lab}^{\mathsf{in}}$, $\Sigma^{\mathsf{out}} = \mathsf{Lab}^{\mathsf{out}}$, $\Sigma^{\tau} = \mathsf{Lab}^{\tau} \cup \overline{\mathsf{Lab}^{\mathsf{in}}} \cup \{\epsilon\}$;

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

– for the discrete transitions:

$(P4.1)$ let $\sigma \in \mathsf{Lab}^{\mathsf{out}}$. $((l, r, T, I, u, d, \bot), \sigma, (l', r', T, I, u, 0, \top)) \in\to$ 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 $\sigma \in \mathsf{Lab}^{\mathsf{in}}$. $((l, r, T, I, u, d, f), \sigma, (l, r, T, I', u, d, f)) \in\to$ iff

- either $I(\sigma) = \bot$ and $I' = I[\sigma := 0]$;
- or $I(\sigma) \neq \bot$ and $I' = I$.

$(P4.3)$ let $\bar{\sigma} \in \overline{\mathsf{Lab}^{\mathsf{in}}}$. $((l, r, T, I, u, d, \bot), \bar{\sigma}, (l', r', T, I', u, 0, \top)) \in\to$ 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 $\sigma \in \mathsf{Lab}^{\tau}$. $((l, r, T, I, u, d, \bot), \sigma, (l', r', T, I, u, 0, \top)) \in\to$ 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 $\sigma = \epsilon$. $((l, r, T, I, u, d, f), \sigma, (l, r, T + u, I, 0, d, \bot)) \in\to$ iff either $f = \top$ or the two following conditions hold:

- for any $\bar{\sigma} \in \overline{\mathsf{Lab}^{\mathsf{in}}}$, 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:

$(P4.6)$ $((l, r, T, I, u, d, f), t, (l, r, T, I + t, u + t, d + t, f)) \in\to$ 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 arrival of the last untreated input event, 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 $(P3)$ are straightforward. 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 $\Delta_S = \lceil \Delta_L + \Delta_P \rceil_{\Delta_P}$, this ensures that any event enabled at some point will be enabled sufficiently long so that the change can be detected by the procedure. The reason of the rounding to the least superior multiple of Δ_P is that Δ_S is a constant intended to be written in real code. Thus it has to be be expressed in the unit of time of the system. 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 $(P4.2)$ records the exact time at which the last untreated occurrence of an input event from the environment occurred. This rule simply ensures that the function I is updated when a new event, for which no occurrence is pending, is issued by the environment. Note that this rule ensures input enabledness. 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. This obviously ensures that the program semantics can not implement zeno strategies

The following simulation theorem expresses formally that if the hardware on which the program is implemented is fast enough (parameter Δ_L) and fine granular 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 rationals $\Delta, \Delta_L, \Delta_P \in \mathbb{Q}^{>0}$ such that $3\Delta_L + 4\Delta_P < \Delta$, we have $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P} \sqsubseteq [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.

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 all 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 $(l_2'', v_2'', I_2'', d_2'')$ such that: $((l_2, v_2, I_2, d_2), \Delta_L - u_1, (l_2'', v_2'', I_2'', d_2'')) \in \rightarrow_2$.

Let us show that R is a simulation relation.

1. $(\iota_1, \iota_2) \in R$. 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.
2. Let us assume that $(s_1, s_2) = ((l_1, r_1, T_1, I_1, u_1, d_1, f_1), (l_2, v_2, I_2, d_2)) \in R$ and that $(s_1, \sigma, s_1') \in \rightarrow_1$ (with $s_1' = (l_1', r_1', T_1', I_1', u_1', d_1', f_1')$). We must prove that for each value of σ, there exists a state $s_2' \in S_2$ such that $(s_2, \sigma, s_2') \in \rightarrow_2$ and $(s_1', s_2') \in R$.
   Since $(s_1, s_2) \in R$ we know that:

$(H1)$ $s_2 = (l_1, v_2, I_1, d_1)$
$(H2)$ $\forall x \in \mathsf{Var} : T_1 - r_1(x) + u_1 - \Delta_L - \Delta_P \leq v_2(x) \leq T_1 - r_1(x) + u_1 + \Delta_L + \Delta_P$
$(H3)$ there exists $s_2'' = (l_2'', v_2'', I_2'', d_2'') \in S_2$ such that: $((l_2, v_2, I_2, d_2), \Delta_L - u_1, (l_2'', v_2'', I_2'', d_2'')) \in \rightarrow_2$.

The rest of the proof works case by case on the different possible types of σ:

case (a) let $\sigma \in \Sigma^{\mathsf{in}}$
Since $(s_1, \sigma, s_1') \in \rightarrow_1$ we know that:

$$s_1' = \begin{cases} (l_1, r_1, T_1, I_1[\{\sigma\} := 0], u_1, d_1, f_1) & \text{if} \quad I_1(\sigma) = \bot \\ (l_1, r_1, T_1, I_1, u_1, d_1, f_1) & \text{if} \quad I_1(\sigma) \neq \bot \end{cases}$$

Let us first prove that $\exists s_2' \in S_2 : (s_2, \sigma, s_2') \in \rightarrow_2$. This is immediate since the AASAP semantics is *input enabled*. Now that we know s_2' exists we can say that:

$$s_2' = \begin{cases} (l_1, v_2, I_1[\{\sigma\} := 0], d_1) & \text{if} \quad I_1(\sigma) = \bot \\ (l_1, v_2, I_1, d_1) & \text{if} \quad I_1(\sigma) \neq \bot \end{cases}$$

It is now easy to prove that $(s_1', s_2') \in R$. Indeed, it is obvious that s_2' fulfills the five conditions of the simulation relation if $(s_1, s_2) \in R$.

case (b) let $\sigma \in \Sigma^{\mathsf{out}}$
Since $(s_1, \sigma, s_1') \in \rightarrow_1$ we know that:

$$(J1) \begin{cases} \exists (l_1, l_1', g, \sigma, R) \in \mathsf{Edg} : \lfloor T_1 \rfloor_{\Delta_P} - r_1 \models_{\Delta_S} g_{\Delta_S} \\ s_1' = (l_1', r_1[R := \lfloor T_1 \rfloor_{\Delta_P}], T_1, I_1, u_1, 0, \top) \end{cases}$$

Let us first prove that $\exists s_2' \in S_2 : (s_2, \sigma, s_2') \in \rightarrow_2$. We use the same edge as in the implementation semantics (see $(J1)$). This amounts to prove that: $\forall x \in \mathsf{Var} : v_2(x) \models_{\Delta} g_{\Delta}(x)$. Let $a_x = lb(g(x))$ and $b_x = rb(g(x))$. We know that $\forall x \in \mathsf{Var}$:

$$
\begin{array}{lll}
 & a_x - \Delta_S \leq \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) \leq b_x + \Delta_S & (J1) \\
\rightarrow & a_x - \Delta_S - \Delta_P \leq T_1 - r_1(x) \leq b_x + \Delta_S + \Delta_P & (\text{Lemma 1}) \\
\rightarrow & a_x - \lceil \Delta_L + \Delta_P \rceil_{\Delta_P} - \Delta_P \leq T_1 - r_1(x) \leq b_x + \lceil \Delta_L + \Delta_P \rceil_{\Delta_P} + \Delta_P & (\text{def. of } \Delta_S) \\
\rightarrow & a_x - \Delta_L - 3\Delta_P \leq T_1 - r_1(x) \leq b_x + \Delta_L + 3\Delta_P & (\text{Lemma 1}) \\
\rightarrow & a_x - 2\Delta_L - 4\Delta_P + u_1 \leq T_1 - r_1(x) + u_1 - \Delta_L - \Delta_P \wedge & \\
 & T_1 - r_1(x) + u_1 + \Delta_L + \Delta_P \leq b_x + 2\Delta_L + 4\Delta_P + u_1 & \\
\rightarrow & a_x - 2\Delta_L - 4\Delta_P + u_1 \leq v_2(x) \leq b_x + 2\Delta_L + 4\Delta_P + u_1 & (H2) \\
\rightarrow & a_x - 2\Delta_L - 4\Delta_P \leq v_2(x) \leq b_x + 3\Delta_L + 4\Delta_P & (0 \leq u_1 \leq \Delta_L) \\
\rightarrow & a_x - \Delta \leq v_2(x) \leq b_x + \Delta & (3\Delta_L + 4\Delta_P < \Delta) \\
\rightarrow & v_2(x) \models {}_\Delta g_\Delta &
\end{array}
$$

Now that it is established that $\exists s_2' \in S_2 : (s_1', \sigma, s_2') \in \rightarrow_2$ we know that $s_2' = (l_1', v_2[R := 0], I_1, 0)$. It remains to prove that $(s_1', s_2') \in R$ which means we must check the five rules of the simulation relation. $(R1)$,$(R3)$,$(R4)$ and $(R5)$ are clearly true.
To prove $(R2)$ we have to prove that:

$$\forall x \in \mathsf{Var} : \begin{cases} T_1 - r_1[R := \lfloor T_1 \rfloor_{\Delta_P}](x) + u_1 - \Delta_L - \Delta_P \leq v_2[R := 0](x) \\ v_2[R := 0](x) \leq T_1 - r_1[R := \lfloor T_1 \rfloor_{\Delta_P}](x) + u_1 + \Delta_L + \Delta_P \end{cases}$$

This proposition is the same as $(H2)$ for $x \notin R$. For $x \in R$, it amounts to prove:

$$T_1 - \lfloor T_1 \rfloor_{\Delta_P} + u_1 - \Delta_L - \Delta_P \leq 0 \leq T_1 - \lfloor T_1 \rfloor_{\Delta_P} + u_1 + \Delta_L + \Delta_P.$$

which is implied by $T_1 - \Delta_P - \lfloor T_1 \rfloor_{\Delta_P} \leq 0 \leq T_1 + \Delta_P - \lfloor T_1 \rfloor_{\Delta_P}$ since $u_1 - \Delta_L \leq 0$. This is a consequence of Lemma 1. This establishes $(R2)$.

case (c) 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 $(s_1, \epsilon, s_1') \in \rightarrow_1$ we know by $(P4.5)$ that

$(K1)$ $s_1' = (l_1, r_1, T_1 + u_1, I_1, 0, d_1, \bot)$

$(K2)$ $f_1 = \top$ or

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

By rule $(A4.5)$ of the AASAP-semantics, we know that there exists $s_2' \in S_2$ such that (s_2, ϵ, s_2') and

$(K3)$ $s_2' = s_2 = (l_1, v_2, I_1, d_1)$.

Now we have to prove that $(s_1', s_2') \in R$. $(R1), (R3)$ and $(R4)$ are clearly true. Proving $(R2)$ amounts to prove that

$$\forall x \in \mathsf{Var} : T_1 + u_1 - r_1(x) + 0 - \Delta_L - \Delta_P \leq v_2(x) \leq T_1 + u_1 - r_1(x) + 0 + \Delta_L + \Delta_P$$

which turns out to be equivalent to $(H2)$.
Let us now prove that there exists s_2'' s.t. $((l_1, v_2, I_1, d_1), \Delta_L, s_2'') \in \rightarrow_2$. According to rule $(A4.6)$, it amounts to prove that

$(L1)$ for any edge $(l_1, l', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}^{\mathsf{out}} \cup \mathsf{Lab}^\tau$, we have that:

$$\forall t' : 0 \leq t' \leq \Delta_L : (d_1 + t' \leq \Delta \vee \mathsf{TS}(v_2 + t', g) \leq \Delta)$$

$(L2)$ for any edge $(l_1, l', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}^{\mathsf{in}}$, we have that:

$$\forall t' : 0 \leq t' \leq \Delta_L : (d_1 + t' \leq \Delta \vee \mathsf{TS}(v_2 + t', g) \leq \Delta \vee (I_1 + t')(\sigma) \leq \Delta)$$

If $f_1 = \top$ it implies that the program has made a discrete transition during the last loop, which means that $d_1 \leq \Delta_L$ and thus that $d_1 + \Delta_L \leq 2\Delta_L \leq \Delta$ because we know that, by hypothesis, $2\Delta_L < \Delta$, which makes $(L1)$ and $(L2)$ true for any t'.
If $f_1 \neq \top$, the proof is less trivial. We first make a proof for labels of $(L1)$.

$\forall (l_1, l', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}^{\mathsf{out}}$ we have $\lfloor T_1 \rfloor_{\Delta_P} \not\models_{\Delta_S} g_{\Delta_S}$ by $(K2)$. Let $a_x = lb(g(x))$ and $b_x = rb(g(x))$. There are two possible cases:

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

$$
\begin{array}{lll}
 & \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) < a_x - \Delta_S & \\
\rightarrow & \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) < a_x - \lceil \Delta_L + \Delta_P \rceil_{\Delta_P} & (\Delta_S = \lceil \Delta_L + \Delta_P \rceil_{\Delta_P}) \\
\rightarrow & T_1 - r_1(x) < a_x - \Delta_L & (\text{Lemma 1}) \\
\rightarrow & T_1 - r_1(x) + u_1 + \Delta_L + \Delta_P < a_x + u_1 + \Delta_P & \\
\rightarrow & v_2(x) < a_x + u_1 + \Delta_P & (H2) \\
\rightarrow & v_2(x) < a_x + \Delta_L & (u_1 \leq \Delta_L + \Delta_P) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : v_2(x) + t' \leq a_x + 2\Delta_L + \Delta_P & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2(x) + t', g(x)) \leq 2\Delta_L + \Delta_P & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2(x) + t', g(x)) \leq \Delta & (2\Delta_L + \Delta_P < \Delta) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2 + t', g) \leq \Delta &
\end{array}
$$

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

$$
\begin{array}{lll}
 & \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) > b_x + \Delta_S & \\
\rightarrow & \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) > b_x + \lceil \Delta_L + \Delta_P \rceil_{\Delta_P} & (\Delta_S = \lceil \Delta_L + \Delta_P \rceil_{\Delta_P}) \\
\rightarrow & T_1 - r_1(x) > b_x + \Delta_L + \Delta_P & (\text{Lemma 1}) \\
\rightarrow & T_1 - r_1(x) + u_1 - \Delta_L - \Delta_P > b_x + u_1 & \\
\rightarrow & v_2(x) > b_x + u_1 & (H2) \\
\rightarrow & v_2(x) > b_x & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : v_2(x) + t' > b_x & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2(x) + t', g(x)) \leq \Delta & (v_2(x) + t' \not\models g(x)) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2 + t', g) \leq \Delta &
\end{array}
$$

Thus, both cases imply that $(L1)$ is true.
The proof for $(L2)$ is the same if we have, by $(K2)$, $\lfloor T_1 \rfloor_{\Delta_P} \not\models_{\Delta_S} g_{\Delta_S}$. If not, we have $I_1(\sigma) < u_1$ which also proves $(L2)$. Indeed

$$
\begin{array}{lll}
 & I_1(\sigma) < u_1 & \\
\rightarrow & I_1(\sigma) < \Delta_L & (u_1 \leq \Delta_L) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : I_1(\sigma) + t' < 2\Delta_L & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : I_1(\sigma) + t' < \Delta & (2\Delta_L \leq \Delta)
\end{array}
$$

case (d) let $\sigma \in \mathbb{R}^{\geq 0}$. For the sake of clarity let us consider that $\sigma = t$.
Since $(s_1, t, s_1') \in \rightarrow_1$ we know by $(P4.6)$ that

$(M1)$ $s_1' = (l_1, r_1, T_1, I_1 + t, u_1 + t, d_1 + t, f_1)$;

$(M2)$ $u_1 + t \leq \Delta_L$.

With those facts, we know that there exists $s_2' = (l_2', v_2', I_2', d_2') \in S_2$ such that $(s_2, t, s_2') \in \rightarrow_2$ because $(s_2, \Delta_L - u_1, s_2'') \in \rightarrow_2$ by $(H3)$ and $t \leq \Delta_L - u_1$ by $(M2)$.
Now we have $(s_2, t, s_2') \in \rightarrow_2$ and we know that:

$(M3)$ $s_2' = (l_1, v_2 + t, I_1 + t, d_1 + t)$

We can now prove that $(s_1', s_2') \in R$. We have to check the five points of the simulation relation: $(R1), (R2), (R3)$ and $(R4)$ are easy to prove using hypothesis $(H1)$ to $(H3)$ and $(M1)$ to $(M3)$.
For $(R5)$, since by $(H3)$, there exists $s_2'' = (l_2'', v_2'', I_2'', d_2'') \in S_2$ such that $((l_2, v_2, I_2, d_2), \Delta_L - u_1, (l_2'', v_2'', I_2'', d_2'')) \in \rightarrow_2$, we have $((l_1, v_2 + t, I_1 + t, d_1 + t), (\Delta_L - u_1 - t), (l_2'', v_2'', I_2'', d_2'')) \in \rightarrow_2$.

■

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

Proof. For any $\Delta > 0$, since parameters Δ_L and Δ_P are in the rational numbers, they can always be chosen such that $3\Delta_L + 6\Delta_P < \Delta$ which implies $\Delta_S + 2\Delta_L + 3\Delta_P < \Delta$ by Lemma 1. ■

And so, given a sufficiently fast hardware with a sufficiently small granularity for its 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. Implementability with clock drifts

In the previous section, we have shown that a control strategy that has been shown correct for the AASAP semantics can be implemented with a sufficiently fast hardware equipped with a sufficiently fine granular clock. To keep the proof and the exposition of the concepts simple, we have assumed that the hardware clock was delivering exactly spaced ticks. If we want to model the imprecision due to clock drifts in the hardware, we have to modify our program semantics. This is done in the Definition 16.

There are two modifications in the definition of the program semantics :

1. The main change is in rule $(Q4.5)$: to model a possible drift of ε every time unit, we change the way the variable T holding the current time of the system is updated. T is not incremented by the exact value u since its last update, but with a value $u' \in [(1-\varepsilon)u, (1+\varepsilon)u]$. This clearly models a drift bounded by ε every time unit.
2. As a drifting clock is less precise, we have to enlarge the guards more than in the previous semantics to ensure that no guard is missed.

Definition 16. [Program Semantics with clock drifts] Let A be an ELASTIC controller, Δ_L, $\Delta_P \in \mathbb{Q}^{>0}$, $\varepsilon \in \mathbb{Q}^{\geq 0}$ with $\varepsilon < 1$. Let $M \in \mathbb{N}$ be the largest constant a clock is compared with and finally, let $\Delta_S = \lceil (1+\varepsilon)(\Delta_L + \Delta_P) + \varepsilon M \rceil_{\Delta_P}$. The $(\Delta_L, \Delta_P, \varepsilon)$ program semantics of A, noted $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P,\varepsilon}$ is the *input enabled* structured timed transition system $\mathcal{T} = \langle S, \iota, \Sigma^{\mathsf{in}}, \Sigma^{\mathsf{out}}, \Sigma^{\tau}, \rightarrow\rangle$ where:

$(Q1)$ 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 $\mathsf{Lab}^{\mathsf{in}}$ into $\mathbb{R}^{\geq 0} \cup \{\bot\}$, $u \in \mathbb{R}^{\geq 0}$, $d \in \mathbb{R}^{\geq 0}$, and $f \in \{\top, \bot\}$;

$(Q2)$ $\iota = (l_0, r, 0, I, 0, 0, \bot)$ where r is such that for any $x \in \mathsf{Var}$, $r(x) = 0$, I is such that for any $\sigma \in \mathsf{Lab}^{\mathsf{in}}$, $I(\sigma) = \bot$;

$(Q3)$ $\Sigma^{\mathsf{in}} = \mathsf{Lab}^{\mathsf{in}}$, $\Sigma^{\mathsf{out}} = \mathsf{Lab}^{\mathsf{out}}$, $\Sigma^{\tau} = \mathsf{Lab}^{\tau} \cup \overline{\mathsf{Lab}^{\mathsf{in}}} \cup \{\epsilon\}$;

$(Q4)$ the transition relation $\rightarrow$ is defined as follows:

- for the discrete transitions:

  $(Q4.1)$ let $\sigma \in \mathsf{Lab}^{\mathsf{out}}$. $((l, r, T, I, u, d, \bot), \sigma, (l', r', T, I, u, 0, \top)) \in\rightarrow$ 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 $r' = r[R := \lfloor T \rfloor_{\Delta_P}]$.

  $(Q4.2)$ 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

  * either $I(\sigma) = \bot$ and $I' = I[\sigma := 0]$;
  * or $I(\sigma) \neq \bot$ and $I' = I$.

  $(Q4.3)$ 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 $I' = I[\sigma := \bot]$;

  $(Q4.4)$ 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 $r' = r[R := \lfloor T \rfloor_{\Delta_P}]$.

  $(Q4.5)$ let $\sigma = \epsilon$. $((l, r, T, I, u, d, f), \sigma, (l, r, T + u', I, 0, d, \bot)) \in\rightarrow$ where $u' \in [(1-\varepsilon)u, (1+\varepsilon)u]$ iff either $f = \top$ or the two following conditions hold:

  * for any $\bar{\sigma} \in \overline{\mathsf{Lab}^{\mathsf{in}}}$, 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:

(Q4.6) $((l,r,T,I,u,d,f),t,(l,r,T,I+t,u+t,d+t,f)) \in\rightarrow$ iff $u+t \leq \Delta_L$.

□

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

Theorem 6 (Simulation with drifting clocks). Let A be an ELASTIC controller. Let M be the largest constant a clock is compared with in A. For any $\Delta, \Delta_L, \Delta_P \in \mathbb{Q}^{>0}$, $\varepsilon \in \mathbb{Q}^{\geq 0}$ with $\varepsilon < 1$ such that $\frac{2\varepsilon M+(3+\varepsilon)\Delta_L+(4+2\varepsilon)\Delta_P}{1-\varepsilon} < \Delta$, we have $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P,\varepsilon} \sqsubseteq [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$. ■

Proof. The proof of this theorem follows closely the lines of the proof of Theorem 4 and is given in appendix. ■

One can observe that for $\varepsilon = 0$ we get the same constraint as in Theorem 4.

We can now immediately state the following theorem and corollary, which are the counterparts with clock drifts of Theorem 5 and Corollary 2 of the previous section.

Theorem 7 (Simulability with clock drifts). For any ELASTIC controller A, for any $\Delta \in \mathbb{Q}^{>0}$, there exists $\Delta_L, \Delta_P, \varepsilon \in \mathbb{Q}^{>0}$ with $\varepsilon < 1$, such that $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P,\varepsilon} \sqsubseteq [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.

Proof. Let M be the largest constant a clock is compared with in A. For any $\Delta > 0$, since parameters Δ_L, Δ_P and ε are in the rational numbers, they can always be chosen such that $\frac{2\varepsilon M+(3+\varepsilon)\Delta_L+(4+2\varepsilon)\Delta_P}{1-\varepsilon} < \Delta$. ■

And so, given a sufficiently fast hardware with a sufficient precision and a sufficiently small granularity for its clock, we can implement any controller that have been proved correct for the AASAP semantics. This is expressed by the following corollary:

Corollary 3 (Implementability with clock drifts). 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 $\Delta_L, \Delta_P, \varepsilon \in \mathbb{Q}^{>0}$ with $\varepsilon < 1$, such that $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P,\varepsilon}$ controls $[\![E]\!]$ to avoid B.

7. In practice

In this section, we show how the AASAP semantics can be analyzed automatically using the model-checker 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 $\mathcal{A}^{\Delta}$ constructed from A and Δ.

Theorem 8. 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 $[\![A]\!]^{\mathsf{AAsap}}_{\Delta} \sqsubseteq [\![\mathcal{A}^{\Delta}]\!]$ and $[\![\mathcal{A}^{\Delta}]\!] \sqsubseteq [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$.

Proof. We give the construction of $\mathcal{F}(A,\Delta)$. Let A be the tuple $\langle \mathsf{Loc}_1, l_1^0, \mathsf{Var}_1, \mathsf{Lab}_1^{\mathsf{in}}, \mathsf{Lab}_1^{\mathsf{out}}, \mathsf{Lab}_1^{\tau}, \mathsf{Edg}_1\rangle$ and let $\mathcal{F}(A,\Delta) = \langle \mathsf{Loc}_2, l_2^0, \mathsf{Var}_2, \mathsf{Inv}_2, \mathsf{Lab}_2^{\mathsf{in}}, \mathsf{Lab}_2^{\mathsf{out}}, \mathsf{Lab}_2^{\tau}, \mathsf{Edg}_2\rangle$ be the timed automaton such that:

- $\mathsf{Loc}_2 = \{(l,b) \mid l \in Loc_1 \wedge b \in [\Sigma^{\mathsf{in}} \rightarrow \{\top,\bot\}]\}$;
- $l_2^0 = (l_1^0, b_\bot)$ where $b_\bot$ is such that $b_\bot(\sigma) = \bot$ for any $\sigma \in \Sigma^{\mathsf{in}}$;
- $\mathsf{Var}_2 = \mathsf{Var}_1 \cup \{y_\sigma \mid \sigma \in \Sigma^{\mathsf{in}}\} \cup \{d\}$;
- $\mathsf{Lab}_2^{\mathsf{in}} = \mathsf{Lab}_1^{\mathsf{in}}$, $\mathsf{Lab}_2^{\mathsf{out}} = \mathsf{Lab}_1^{\mathsf{out}}$ and $\mathsf{Lab}_2^{\tau} = \mathsf{Lab}_1^{\tau} \cup \overline{\mathsf{Lab}_1^{\mathsf{in}}}$;

- Edg_2 is defined as follows.
$((l,b),(l',b'),{}_\Delta g_\Delta,\sigma,R') \in \mathsf{Edg}_2$ iff one of the following condition holds:
  - $\sigma \in \mathsf{Lab}_1^{\mathsf{in}}$ and
    1. $l' = l$
    2. $b(\sigma) = \bot$
    3. $b' = b[\sigma := \top]$
    4. $g = \mathsf{true}$
    5. $R' = \{y_\sigma\}$
  - $\sigma \in \mathsf{Lab}_1^{\mathsf{in}}$ and
    1. $l' = l$
    2. $b(\sigma) = \top$
    3. $b' = b$
    4. $g = \mathsf{true}$
    5. $R' = \varnothing$
  - $\sigma \in \mathsf{Lab}_1^{\mathsf{out}}$ and
    1. there exists $(l,l',g,\sigma,R) \in \mathsf{Edg}_1$
    2. $b' = b$
    3. $R' = R \cup \{d\}$
  - $\sigma \in \mathsf{Lab}_1^{\tau}$ and
    1. there exists $(l,l',g,\sigma,R) \in \mathsf{Edg}_1$
    2. $b' = b$
    3. $R' = R \cup \{d\}$
  - $\sigma = \bar{\alpha} \in \overline{\mathsf{Lab}_1^{\mathsf{in}}}$ and
    1. there exists $(l,l',g,\alpha,R) \in \mathsf{Edg}_1$
    2. $b(\alpha) = \top$
    3. $b' = b[\alpha := \bot]$
    4. $R' = R \cup \{d\}$
  - $\sigma = \epsilon$ and
    1. $l' = l$
    2. $b' = b$
    3. $g = \mathsf{true}$
    4. $R' = \varnothing$
- The function Inv_2 is defined as follows. Let $EVT((l,b)) = \{((l,l',g,\sigma,R) \in \mathsf{Edg}_1 \mid \sigma \in \overline{\mathsf{Lab}_1^{\mathsf{in}}} \wedge b(\sigma) = \top\}$. Let $ACT((l,b)) = \{(l,l',g,\sigma,R) \in \mathsf{Edg}_1 \mid \sigma \in \mathsf{Lab}_1^{\mathsf{out}} \cup \mathsf{Lab}_1^{\tau}\}$. $\mathsf{Inv}_2((l,b)) = \varphi_1(l,b) \wedge \varphi_2(l,b)$ where

$$\varphi_1(l,b) = \bigwedge_{(l,l',g,\sigma,R)\in EVT((l,b))} \left(d \le \Delta \vee \neg({}^\Delta g) \vee y_\sigma \le \Delta\right)$$
$$\varphi_2(l,b) = \bigwedge_{(l,l',g,\sigma,R)\in ACT((l,b))} \left(d \le \Delta \vee \neg({}^\Delta g)\right)$$

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

Let $[\![A]\!]_\Delta^{\mathsf{AAsap}} = (S_1, \iota_1, \Sigma_1^{\mathsf{in}}, \Sigma_1^{\mathsf{out}}, \Sigma_1^{\tau}, \rightarrow_1)$ and $[\![\mathcal{F}(A,\Delta)]\!] = (S_2, \iota_2, \Sigma_2^{\mathsf{in}}, \Sigma_2^{\mathsf{out}}, \Sigma_2^{\tau}, \rightarrow_2)$.
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.

- It appears obviously that $(\iota_1, \iota_2) \in R$.
- Let us assume that $(s_1,s_2) = ((l_1,v_1,I_1,d_1),((l_2,b_2),v_2)) \in R$ and that $(s_1,\sigma,s_1') \in \rightarrow_1$ (with $s_1' = (l_1', v_1', I_1', d_1')$).
For each value of σ, we must establish the existence of a state $s_2' \in S^2$ such that $(s_2,\sigma,s_2') \in \rightarrow_2$ and $(s_1', s_2') \in R$.
Since $(s_1,s_2) \in R$ we know that:

$(H0)$ $s_2 = ((l_1,b_2),v_2)$
$(H1)$ $v^2_{1|\mathsf{Var}^1} = v_1$, $v_2(d) = d_1$, and $v_2(y_\sigma) = I_1(\sigma)$ iff $I_1(\sigma) \neq \bot$.

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 $(s_1, t, s_1') \in \rightarrow_1$ we know by $(A4.6)$ that

$(H2)$ for any edge $(l_1, l', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}_1^{\mathsf{out}} \cup \mathsf{Lab}_1^{\tau}$, we have that:

$$\forall t' : 0 \leq t' \leq t : (d_1 + t' \leq \Delta \vee \mathsf{TS}(v_1 + t', g) \leq \Delta)$$

$(H3)$ for any edge $(l_1, l', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}_1^{\mathsf{in}}$, we have that:

$$\forall t' : 0 \leq t' \leq t : d_1 + t' \leq \Delta \vee \mathsf{TS}(v_1 + t', g) \leq \Delta \vee (I_1 + t')(\sigma) \leq \Delta$$

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

- $d_1 + t' \leq \Delta \leftrightarrow v_2(d) + t' \leq \Delta$
- $(I_1 + t')(\sigma) \leq \Delta \leftrightarrow v_2(y_\sigma) + t' \leq \Delta$
- $\mathsf{TS}(v_1 + t', g) \leq \Delta \leftrightarrow v_2 + t' \models \neg({}^{\Delta}g)$

The third proposition can be proved as follows (let $a_x = lb(g(x))$ and $b_x = rb(g(x))$):

$$\begin{array}{ll}
 & \mathsf{TS}(v_1 + t', g) \leq \Delta \\
\leftrightarrow & (v_1 + t' \models g) \rightarrow \exists x \in \mathsf{Var}_1 : v_1(x) + t' - \Delta \leq a_x \\
\leftrightarrow & (v_1 + t' \models g) \rightarrow (\exists x \in \mathsf{Var}_1 : a_x \leq v_1(x) + t' \leq b_x \ \wedge \ v_1(x) + t' - \Delta \leq a_x) \\
\leftrightarrow & (v_1 + t' \not\models g) \vee (\exists x \in \mathsf{Var}_1 : a_x \leq v_1(x) + t' \leq \min(b_x, a_x + \Delta)) \\
\leftrightarrow & (\exists x \in \mathsf{Var}_1 : v_1(x) + t' < a_x \vee v_1(x) + t' > b_x) \\
 & \vee(\exists x \in \mathsf{Var}_1 : a_x \leq v_1(x) + t' \leq \min(b_x, a_x + \Delta)) \\
\leftrightarrow & \exists x \in \mathsf{Var}_1 : v_1(x) + t' \leq a_x + \Delta \vee v_1(x) + t' > b_x \\
\leftrightarrow & \exists x \in \mathsf{Var}_1 : v_1(x) + t' \notin (a_x + \Delta, b_x] \\
\leftrightarrow & v_1 + t' \not\models {}^{\Delta}g \\
\leftrightarrow & v_1 + t' \models \neg({}^{\Delta}g) \\
\leftrightarrow & v_2 + t' \models \neg({}^{\Delta}g)
\end{array}$$

Consequently, using $(H2)$ and $(H3)$, we have $\forall 0 \leq t' \leq t : v_2 + t' \models \mathsf{Inv}_2(l_2)$. Thus, the state $s_2' = ((l_2, b_2), v_2')$ where $v_2' = v_2 + t$ is such that $(s_2, t, s_2') \in \rightarrow_2$.
The reader can easily check that $(s_1', s_2') \in R$.

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.

■

Corollary 4. 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 $[\![A]\!]_{\Delta}^{\mathsf{AAsap}}$ controls $[\![E]\!]$ to avoid B iff $[\![\mathcal{F}(A, \Delta)]\!]$ controls $[\![E]\!]$ to avoid B.

In practice, we use Theorem 8 to reduce the controllability problem to a reachability problem. We first construct $\mathcal{F}(A, \Delta)$ (where we can leave Δ as a parameter) and generate a HyTech file with a description of $\mathcal{F}(A, \Delta)$ and E. We then use HyTech to synthesize the weakest constraint on Δ that ensures the correctness of the control strategy for the AASAP semantics. In other words, we ask HyTech for which parameter value $\mathsf{Reach}([\![\mathcal{F}(A, \Delta)]\!] \parallel [\![E]\!]) \cap B = \varnothing$. If HyTech terminates, it returns a linear constraint $\psi(\Delta)$ over Δ which allows to solve the three problems of Definition 13:

- **[Fixed]** For a given value $D \in \mathbb{Q}^{\geq 0}$ for the parameter Δ, answering the [Fixed] question amounts to ask if $\psi(D)$ is true.
- **[Existence]** To solve the [Existence] question, it suffices to ask if there exists $D \in \mathbb{Q}^{\geq 0}$ such that $\psi(D)$.
- **[Maximization]** To solve the [Maximization] question, we ask if
  - either there exists $D_{max} \in \mathbb{Q}^{\geq 0}$ such that $\psi(D_{max})$ and $\forall D > D_{max} : \neg\psi(D)$;
  - or there exists $D_{sup} \in \mathbb{Q}^{\geq 0}$ such that $\neg\psi(D_{sup})$ and $\forall 0 \leq D < D_{sup} : \psi(D)$.

All those question are expressed as formulas of the additive theory of the reals and are thus solvable [Wei99] .

In case the HyTech analysis does not terminate, we still have a practical solution for two of the problems of Definition 13 if the environment is specified as a timed automaton:

- **[Fixed]** In this case we can obviously respond to the [Fixed] version of the correctness problem as it amounts to a reachability question on timed automata [AD94],
- **[Maximization]** In this case, we can approximate as close as needed the solution of the [Maximization] question thanks to the "faster is better" property of the AASAP semantics: by doing a binary search on the value space of the parameter Δ, using the [Fixed] question, we can approximate the maximal value of Δ for which the controller is correct up to any precision.

To answer those two previous questions, we are not restricted to the use of HyTech or other parametric tools and we for example can use Uppaal [PL00]. Finally, we prove in [DDMR04] the decidability of the [Existence] question when the environment is a timed automata.

Running example If we apply the construction of Theorem 8 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 8 is valid.

First, consider the case ($\alpha = 1$). The condition of correctness is $\Delta = 0$. We can interpret this as follows: we have a correct model w.r.t to the classical ASAP semantics (the controller imposes the repetition of events `ABC`, at integer points in time, that is the divergent timed word $((\mathtt{ABC})^\omega, \tau)$ with $\tau_{3i} = \tau_{3i+1} = \tau_{3i+2} = i+1$, $i \in \mathbb{N}$). But if we accept a positive *fixed* delay (no matter how small it is) between the event `B` and the reaction `C`, we cannot anymore guarantee that the environment will not eventually reach `Bad`. The insightful reader has maybe noticed that if the delay between `B` and `C` can vary (and here decrease), then it is possible to avoid `Bad`. However, the greatest lower bound of the delays will be zero so that implementation is not possible. For example, a (non-zeno) controller could issue orders such that $\tau = (1, 1, 1\frac{1}{2})(2, 2\frac{1}{2}, 2\frac{3}{4})(3, 3\frac{3}{4}, 3\frac{7}{8})(4, 4\frac{7}{8}, 4\frac{15}{16})\ldots$ Clearly this time sequence, however divergent, is not acceptable as the output of an implementable controller because there is no lower bound on the difference between two consecutive time instants. This shows that not only zeno controllers require infinitely fast hardware to be implemented. Hence, it must be admitted that the synchrony hypothesis is not only a matter of non-zenoness; in this example, the condition $\Delta = 0$ shows that it is impossible for a finite-speed hardware to implement the controller.

In the second case ($\alpha = 2$), the model is correct for any $\Delta < \frac{1}{3}$. If we assume that the unit of time is the second, Theorem 6 then tells us that, to preserve the desired property with a systematic implementation of the Elastic controller (as described in Section 5), we should have a platform with loop time Δ_L clock granularity Δ_P and clock precision ε such that $\frac{2\varepsilon M+(3+\varepsilon)\Delta_L+(4+2\varepsilon)\Delta_P}{1-\varepsilon} < 333$ms, where $M = 2$ is the greatest constant appearing in the controller. For instance, we can implement the controller on the Lego Mindstorms™ platform, since it allows Δ_L to be as low as 6ms, offers a digital clock with $\Delta_P = 1$ms and a drift certainly not greater than 1%.

7.1. Relaxing the AASAP semantics

The construction of Theorem 8 gives a model that can be used in practice to verify the implementability of a controller (Corollary 4). This model is a timed automaton that contains *exactly* the same reachability information than the AASAP semantics. This is why the construction is not as simple as one might expect. Other constructions can be used (for example if it facilitates the verification), at the condition that it can *simulate* the AASAP semantics. Also, if a controller controls an environment E', it ensures that it can control any refinement E of E'. This results from Theorems 1, 2 and 8.

Corollary 5. For any Elastic controller A, for any $\Delta \in \mathbb{Q}^{>0}$, for any timed automata E' with state space $S_{E'}$, and E with state space $S_E \subseteq S_{E'}$ such that $[\![E]\!] \sqsubseteq [\![E']\!]$; for any timed automaton C such that $[\![\mathcal{F}(A,\Delta)]\!] \sqsubseteq [\![C]\!]$, for any set of states $B \subseteq S_E$, we have that if $[\![C]\!]$ controls $[\![E']\!]$ to avoid B then $[\![A]\!]^{\mathsf{AAsap}}_{\Delta}$ controls $[\![E]\!]$ to avoid B.

In Corollary 5, the automaton C is an *over-approximation* of the AASAP semantics. For verification concerns, a promising such over-approximation is, given a timed automaton T, to close and enlarge (left and right) every guard by Δ to obtain the timed automaton we call $\mathcal{X}(T,\Delta)$. The model $\mathcal{X}(\mathcal{F}(A,\Delta),\Delta)$ has interesting properties. We show in [DDMR04] that the problem to decide whether there exists a rational value Δ such that it controls $\mathcal{X}(E, \Delta)$ to avoid B is decidable .

8. Conclusion and related works

In this paper, we have introduced the notion of Almost ASAP semantics for timed automata. This semantics is a relaxation of the usual ASAP semantics: the controller does not have to react instantaneously to events and time-outs, it only has to do so within a given time bound (that can be leaved as a parameter). We have shown that this semantics is useful to formally reason about the implementability of mathematical models for timed controllers: any controller that has been shown correct for the Almost ASAP semantics can be systematically implemented. The properties that have been proven on the model are transfered to the implementation (without making the synchrony hypothesis) provided that the implementation is executed on a hardware which is sufficiently fast and which uses a sufficiently fine granular digital clock subject to a sufficiently small drift.

We now compare our work with some recent related works and also point out several future research directions.

Related works. In [AFM+02, AFP+03], 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 [KMTY04], the authors agree that an event must remain observable during some (usually small but not singular) period. They propose a digitalized semantics for timed automata in order to model the fact that the environment cannot be observed continuously, but only at discrete instants. However, they do not make a formal link with automatic code generation. On the contrary, we propose such a link, and our semantics is more high-level in that it is continuous-time and thus closer to the designer's point of view.

In [AFILS03], 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], 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+00], 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 fix point 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. Partial answers are given in [DDMR04].

- 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.

Acknowledgements We thank the anonymous referees for their valuable criticism and useful comments. In particular, one reviewer made useful suggestions about the notion of *input enabledness* that lead to nice simplifications.

References

[AD94] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[AFILS03] R. Alur, J. Kim F. Ivancic, I. Lee, and O. Sokolsky. Generating embedded software from hierarchical hybrid models. In *Proc. of LCTES 2003*, volume 38 of *ACM SIGPLAN Notices*, pages 171 – 182, 2003.

[AFM+02] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: A tool for modelling and implementation of embedded systems. In *Proc. of TACAS' 2002*, volume 2280 of *LNCS*, pages 460–464. Springer–Verlag, 2002.

[AFP+03] T. Amnell, E. Fersman, P. Pettersson, H. Sun, and W. Yi. Code synthesis for timed automata. *Nordic Journal of Computing*, 9(4), 2003.

[AL91] M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

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

[CBG88] E.M Clarke, M.C. Browne, and O. Grumberg. Characterizing finite kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59:115–131, 1988.

[CHR02] F. Cassez, T.A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In *Proc. of HSCC' 02*, volume 2289 of *LNCS*, pages 134–148. Springer-Verlag, 2002.

[DDMR04] M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robustness and implementability of timed automata. In *Proc. of FORMATS-FTRTFT 2004*, volume 3253 of *LNCS*, pages 118–133. Springer-Verlag, 2004.

[DDR04] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. In *Proc. of HSCC' 04*, volume 2993 of *LNCS*, pages 296–310. Springer-Verlag, 2004.

[Die01] H. Dierks. PLC-automata: a new class of implementable real-time automata. *Theoretical Computer Science*, 253(1):61–93, 2001.

[Hen96] T.A. Henzinger. The theory of hybrid automata. In *Proc. of LICS '96*, pages 278–292. IEEE, 1996.

[HHWT95] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. A user guide to HyTech. In *Proc. of TACAS 95*, volume 1019 of *LNCS*, 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] M. Hendriks and K. G. Larsen. Exact acceleration of real-time model checking. In *Proc. of TPTS'02*, volume 65 of *ENTCS*. Elsevier Science, 2002.

[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. of ECRTS '00*, pages 147–155. IEEE, 2000.

[KMTY04] P. Krčál, L. Mokrushin, P.S. Thiagarajan, and W. Yi. Timed vs time triggered automata. In *Proc. of CONCUR'04*, volume 3170 of *LNCS*, pages 340–354. Springer–Verlag, 2004.

[LT87] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proc. of the 6 th PODC*, pages 137–151. ACM, 1987.

[LV04] G. Lüttgen and W. Vogler. Bisimulation on speed: worst-case efficiency. *Information and Computation*, 191(2):105–144, 2004.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, 1980.

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

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of FOCS 1977*, pages 46–57. IEEE, 1977.

[Wei99] V. Weispfenning. Mixed real-integer linear quantifier elimination. In *Proc. of ISSAC 99*, pages 129–136. ACM, 1999.

A. Proof of theorem 6

Theorem 6 (Simulation with drifting clocks) Let A be an ELASTIC controller. Let M be the largest constant a clock is compared with in A. For any $\Delta, \Delta_L, \Delta_P \in \mathbb{Q}^{>0}$, $\varepsilon \in \mathbb{Q}^{\geq 0}$ with $\varepsilon < 1$ such that $\frac{2\varepsilon M+(3+\varepsilon)\Delta_L+(4+2\varepsilon)\Delta_P}{1-\varepsilon} < \Delta$, we have $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P,\varepsilon} \sqsubseteq [\![A]\!]^{\mathsf{AAsap}}_{\Delta}$. ■

Proof. Let $[\![A]\!]^{\mathsf{Prg}}_{\Delta_L,\Delta_P,\varepsilon} = (S_1, \iota_1, \Sigma_1^{\mathsf{in}}, \Sigma_1^{\mathsf{out}}, \Sigma_1^{\tau}, \rightarrow_1)$ 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 $x \in \mathsf{Var}$, $\frac{T_1 - r_1(x) - (1+\varepsilon)(\Delta_L+\Delta_P)}{1+\varepsilon} + u_1 \leq v_2(x) \leq \frac{T_1 - r_1(x) + (1+\varepsilon)(\Delta_L+\Delta_P)}{1-\varepsilon} + u_1$
$(R3)$ for any $\sigma \in \mathsf{Lab}^{\mathsf{in}}$, $I_1(\sigma) = I_2(\sigma)$;
$(R4)$ $d_1 = d_2$;
$(R5)$ there exists $(l_2'', v_2'', I_2'', d_2'')$ such that: $((l_2, v_2, I_2, d_2), \Delta_L - u_1, (l_2'', v_2'', I_2'', d_2'')) \in \rightarrow_2$.

Let us show that R is a simulation relation.

1. $(\iota_1, \iota_2) \in R$. 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.
2. Let us assume that $(s_1, s_2) = ((l_1, r_1, T_1, I_1, u_1, d_1, f_1), (l_2, v_2, I_2, d_2)) \in R$ and that $(s_1, \sigma, s_1') \in \rightarrow_1$ (with $s_1' = (l_1', r_1', T_1', I_1', u_1', d_1', f_1')$). We must prove that for each value of σ, there exists a state $s_2' \in S_2$ such that $(s_2, \sigma, s_2') \in \rightarrow_2$ and $(s_1', s_2') \in R$.
   Since $(s_1, s_2) \in R$ we know that:

$(H1)$ $s_2 = (l_1, v_2, I_1, d_1)$
$(H2)$ $\forall x \in \mathsf{Var} : \frac{T_1 - r_1(x) - (1+\varepsilon)(\Delta_L+\Delta_P)}{1+\varepsilon} + u_1 \leq v_2(x) \leq \frac{T_1 - r_1(x) + (1+\varepsilon)(\Delta_L+\Delta_P)}{1-\varepsilon} + u_1$
$(H3)$ there exists $s_2'' = (l_2'', v_2'', I_2'', d_2'') \in S_2$ such that: $((l_2, v_2, I_2, d_2), \Delta_L - u_1, (l_2'', v_2'', I_2'', d_2'')) \in \rightarrow_2$.

The rest of the proof works case by case on the different possible types of σ:

case (a) let $\sigma \in \Sigma^{\mathsf{in}}$
Since $(s_1, \sigma, s_1') \in \rightarrow_1$ we know that:

$$s_1' = \begin{cases} (l_1, r_1, T_1, I_1[\{\sigma\} := 0], u_1, d_1, f_1) & \text{if } I_1(\sigma) = \bot \\ (l_1, r_1, T_1, I_1, u_1, d_1, f_1) & \text{if } I_1(\sigma) \neq \bot \end{cases}$$

Let us first prove that $\exists s_2' \in S_2 : (s_2, \sigma, s_2') \in \rightarrow_2$. This is immediate since the AASAP semantics is *input enabled*. Now that we know s_2' exists we can say that:

$$s_2' = \begin{cases} (l_1, v_2, I_1[\{\sigma\} := 0], d_1) & \text{if } I_1(\sigma) = \bot \\ (l_1, v_2, I_1, d_1) & \text{if } I_1(\sigma) \neq \bot \end{cases}$$

It is now easy to prove that $(s_1', s_2') \in R$. Indeed, it is obvious that s_2' fulfills the five conditions of the simulation relation if $(s_1, s_2) \in R$.

case (b) let $\sigma \in \Sigma^{\mathsf{out}}$
Since $(s_1, \sigma, s_1') \in \rightarrow_1$ we know that:

$$(J1) \begin{cases} \exists (l_1, l_1', g, \sigma, R) \in \mathsf{Edg} : \lfloor T_1 \rfloor_{\Delta_P} - r_1 \models_{\Delta_S} g_{\Delta_S} \\ s_1' = (l_1', r_1[R := \lfloor T_1 \rfloor_{\Delta_P}], T_1, I_1, u_1, 0, \top) \end{cases}$$

Let us first prove that $\exists s_2' \in S_2 : (s_2, \sigma, s_2') \in \rightarrow_2$. We use the same edge as in the implementation semantics (see $(J1)$). This amounts to prove that: $\forall x \in \mathsf{Var} : v_2(x) \models_{\Delta} g_{\Delta}(x)$. Let $a_x = lb(g(x))$ and $b_x = rb(g(x))$. We know that $\forall x \in \mathsf{Var}$:

$$\begin{array}{lll}
 & a_x - \Delta_S \leq \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) \leq b_x + \Delta_S & (J1) \\
\rightarrow & a_x - \lceil (1+\varepsilon)(\Delta_L + \Delta_P) + \varepsilon M \rceil_{\Delta_P} \leq \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) \wedge & \\
 & \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) \leq b_x + \lceil (1+\varepsilon)(\Delta_L + \Delta_P) + \varepsilon M \rceil_{\Delta_P} & (\text{def. of } \Delta_S) \\
\rightarrow & a_x - (1+\varepsilon)(\Delta_L + \Delta_P) - \varepsilon M - \Delta_P \leq T_1 - r_1(x) \wedge & \\
 & T_1 - r_1(x) \leq b_x + (1+\varepsilon)(\Delta_L + \Delta_P) + \varepsilon M + 2\Delta_P & (\text{Lemma 1}) \\
\rightarrow & \frac{a_x - 2(1+\varepsilon)(\Delta_L + \Delta_P) - \varepsilon M - \Delta_P}{1+\varepsilon} + u_1 \leq \frac{T_1 - r_1(x) - (1+\varepsilon)(\Delta_L + \Delta_P)}{1+\varepsilon} + u_1 \wedge & \\
 & \frac{T_1 - r_1(x) + (1+\varepsilon)(\Delta_L + \Delta_P)}{1-\varepsilon} + u_1 \leq \frac{b_x + 2(1+\varepsilon)(\Delta_L + \Delta_P) + \varepsilon M + 2\Delta_P}{1-\varepsilon} + u_1 & \\
\rightarrow & \frac{a_x - 2(1+\varepsilon)(\Delta_L + \Delta_P) - \varepsilon M - \Delta_P}{1+\varepsilon} + u_1 \leq v_2(x) \wedge & \\
 & v_2(x) \leq \frac{b_x + 2(1+\varepsilon)(\Delta_L + \Delta_P) + \varepsilon M + 2\Delta_P}{1-\varepsilon} + u_1 & (H2) \\
\rightarrow & \frac{a_x - (2+2\varepsilon)\Delta_L - (3+2\varepsilon)\Delta_P - \varepsilon M}{1+\varepsilon} \leq v_2(x) \wedge & \\
 & v_2(x) \leq \frac{b_x + (3+\varepsilon)\Delta_L + (4+2\varepsilon)\Delta_P + \varepsilon M}{1-\varepsilon} & (0 \leq u_1 \leq \Delta_L) \\
\rightarrow & a_x - \frac{2\varepsilon M + (2+2\varepsilon)\Delta_L + (3+2\varepsilon)\Delta_P}{1+\varepsilon} \leq v_2(x) \wedge & \\
 & v_2(x) \leq b_x + \frac{2\varepsilon M + (3+\varepsilon)\Delta_L + (4+2\varepsilon)\Delta_P}{1-\varepsilon} & (M \geq a_x \wedge M \geq b_x) \\
\rightarrow & a_x - \Delta \leq v_2(x) \leq b_x + \Delta & (\frac{2\varepsilon M + (3+\varepsilon)\Delta_L + (4+2\varepsilon)\Delta_P}{1-\varepsilon} < \Delta) \\
\rightarrow & v_2(x) \models_{\Delta} g_{\Delta} &
\end{array}$$

Now that it is established that $\exists s_2' \in S_2 : (s_1', \sigma, s_2') \in \rightarrow_2$ we know that: $s_2' = (l_1', v_2[R := 0], I_1, 0)$
It remains to prove that $(s_1', s_2') \in R$ which means we must check the five rules of the simulation relation. $(R1)$,$(R3)$,$(R4)$ and $(R5)$ are clearly true.
To prove $(R2)$ we have to prove that

$$\forall x \in \mathsf{Var} : \begin{cases} \frac{T_1 - r_1[R := \lfloor T_1 \rfloor_{\Delta_P}](x) - (1+\varepsilon)(\Delta_L + \Delta_P)}{1+\varepsilon} + u_1 \leq v_2[R := 0](x) \\ v_2[R := 0](x) \leq \frac{T_1 - r_1[R := \lfloor T_1 \rfloor_{\Delta_P}](x) + (1+\varepsilon)(\Delta_L + \Delta_P)}{1-\varepsilon} + u_1 \end{cases}$$

This proposition is the same as $(H2)$ for $x \notin R$. For $x \in R$, it amounts to prove:

$$\frac{T_1 - \lfloor T_1 \rfloor_{\Delta_P} - (1+\varepsilon)(\Delta_L + \Delta_P)}{1+\varepsilon} + u_1 \leq 0 \leq \frac{T_1 - \lfloor T_1 \rfloor_{\Delta_P} + (1+\varepsilon)(\Delta_L + \Delta_P)}{1-\varepsilon} + u_1.$$

which is implied by $T_1 - \Delta_P - \lfloor T_1 \rfloor_{\Delta_P} \leq 0 \leq T_1 + \Delta_P - \lfloor T_1 \rfloor_{\Delta_P}$, which is a consequence of Lemma 1, and $u_1 - \Delta_L \leq 0$. This establishes $(R2)$.

case (c) 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 $(s_1, \epsilon, s_1') \in \rightarrow_1$ we know by $(Q4.5)$ that

$(K1)$ $s_1' = (l_1, r_1, T_1 + u_1', I_1, 0, d_1, \perp)$ where $u_1' \in [(1-\varepsilon)u_1, (1+\varepsilon)u_1]$;

$(K2)$ $f_1 = \top$ or

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

By rule $(A4.5)$ of the AASAP-semantics, we know that there exists $s_2' \in S_2$ such that (s_2, ϵ, s_2') and

$(K3)$ $s_2' = s_2 = (l_1, v_2, I_1, d_1)$.

Now we have to prove that $(s_1', s_2') \in R$. $(R1)$, $(R3)$ and $(R4)$ are clearly true. Proving $(R2)$ amounts to prove that

$$\forall x \in \mathsf{Var} : \frac{T_1 + u_1' - r_1(x) - (1+\varepsilon)(\Delta_L + \Delta_P)}{1+\varepsilon} \leq v_2(x) \leq \frac{T_1 + u_1' - r_1(x) + (1+\varepsilon)(\Delta_L + \Delta_P)}{1-\varepsilon}$$

which is implied by $(H2)$ and $(1-\varepsilon)u_1 \leq u_1' \leq (1+\varepsilon)u_1$.

Let us now prove that there exists s_2'' s.t. $((l_1, v_2, I_1, d_1), \Delta_L, s_2'') \in \rightarrow_2$. According to rule $(A4.6)$, it amounts to prove that

$(L1)$ for any edge $(l_1, l', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}^{\mathsf{out}} \cup \mathsf{Lab}^{\tau}$, we have that:

$$\forall t' : 0 \leq t' \leq \Delta_L : (d_1 + t' \leq \Delta \vee \mathsf{TS}(v_2 + t', g) \leq \Delta)$$

$(L2)$ for any edge $(l_1, l', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}^{\mathsf{in}}$, we have that:

$$\forall t' : 0 \leq t' \leq \Delta_L : (d_1 + t' \leq \Delta \vee \mathsf{TS}(v_2 + t', g) \leq \Delta \vee (I_1 + t')(\sigma) \leq \Delta)$$

If $f_1 = \top$ it implies that the program has made a discrete transition during the last loop, which means that $d_1 \leq \Delta_L$ and thus that $d_1 + \Delta_L \leq 2\Delta_L \leq \Delta$ because we know that, by hypothesis, $\Delta > 2\Delta_L$, which makes $(L1)$ and $(L2)$ true for any t'.
If $f_1 \neq \top$, the proof is less trivial. We first make a proof for labels of $(L1)$.
$\forall (l_1, l', g, \sigma, R) \in \mathsf{Edg}$ with $\sigma \in \mathsf{Lab}^{\mathsf{out}}$ we have $\lfloor T_1 \rfloor_{\Delta_P} \not\models_{\Delta_S} g_{\Delta_S}$ by $(K2)$. Let $a_x = lb(g(x))$ and $b_x = rb(g(x))$. There are two possible cases:

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

$$\begin{array}{lll}
 & \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) < a_x - \Delta_S & \\
 & \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) < a_x - \lceil (1+\varepsilon)(\Delta_L + \Delta_P) + \varepsilon M \rceil_{\Delta_P} & \text{(def. of } \Delta_S) \\
\rightarrow & T_1 - r_1(x) - \Delta_P < a_x - (1+\varepsilon)(\Delta_L + \Delta_P) - \varepsilon M & \text{(Lemma 1)} \\
\rightarrow & \frac{T_1 - r_1(x) + (1+\varepsilon)(\Delta_L + \Delta_P)}{1-\varepsilon} + u_1 < \frac{a_x + \Delta_P - \varepsilon M}{1-\varepsilon} + u_1 & \\
\rightarrow & v_2(x) < \frac{a_x + \Delta_P - \varepsilon M}{1-\varepsilon} + u_1 & (H2) \\
\rightarrow & v_2(x) < \frac{a_x + \Delta_P - \varepsilon M}{1-\varepsilon} + \Delta_L & (u_1 \leq \Delta_L) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : v_2(x) + t' \leq \frac{a_x + \Delta_P - \varepsilon M}{1-\varepsilon} + 2\Delta_L & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : v_2(x) + t' \leq a_x + \frac{\varepsilon a_x + 2(1-\varepsilon)\Delta_L + \Delta_P - \varepsilon M}{1-\varepsilon} & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : v_2(x) + t' \leq a_x + \frac{2(1-\varepsilon)\Delta_L + \Delta_P}{1-\varepsilon} & (M \geq a_x) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2(x) + t', g(x)) \leq \frac{2(1-\varepsilon)\Delta_L + \Delta_P}{1-\varepsilon} & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2(x) + t', g(x)) \leq \Delta & (\frac{2(1-\varepsilon)\Delta_L + \Delta_P}{1-\varepsilon} < \Delta) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2 + t', g) \leq \Delta &
\end{array}$$

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

$$\begin{array}{lll}
 & \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) > b_x + \Delta_S & \\
 & \lfloor T_1 \rfloor_{\Delta_P} - r_1(x) > b_x + \lceil (1+\varepsilon)(\Delta_L + \Delta_P) + \varepsilon M \rceil_{\Delta_P} & \text{(def. of } \Delta_S) \\
\rightarrow & T_1 - r_1(x) > b_x + (1+\varepsilon)(\Delta_L + \Delta_P) + \varepsilon M & \text{(Lemma 1)} \\
\rightarrow & \frac{T_1 - r_1(x) - (1+\varepsilon)(\Delta_L + \Delta_P)}{1+\varepsilon} + u_1 > \frac{b_x + \varepsilon M}{1+\varepsilon} + u_1 & \\
\rightarrow & v_2(x) > \frac{b_x + \varepsilon M}{1+\varepsilon} + u_1 & (H2) \\
\rightarrow & v_2(x) > \frac{b_x + \varepsilon M}{1+\varepsilon} & (u1 \geq 0) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : v_2(x) + t' > \frac{b_x + \varepsilon M}{1+\varepsilon} & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : v_2(x) + t' > b_x + \frac{-\varepsilon b_x + \varepsilon M}{1+\varepsilon} & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : v_2(x) + t' > b_x + \frac{-\varepsilon M + \varepsilon M}{1+\varepsilon} & (b_x \leq M) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : v_2(x) + t' > b_x & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2(x) + t', g(x)) \leq \Delta & (v_2(x) + t' \not\models g(x)) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : \mathsf{TS}(v_2 + t', g) \leq \Delta &
\end{array}$$

Thus, both cases imply that $(L1)$ is true.
The proof for $(L2)$ is the same if we have, by $(K2)$, $\lfloor T_1 \rfloor_{\Delta_P} \not\models_{\Delta_S} g_{\Delta_S}$. If not, we have $I_1(\sigma) < u_1$ which also proves $(L2)$. Indeed

$$\begin{array}{lll}
 & I_1(\sigma) < u_1 & \\
\rightarrow & I_1(\sigma) < \Delta_L & (u_1 \leq \Delta_L) \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : I_1(\sigma) + t' < 2\Delta_L & \\
\rightarrow & \forall t' : 0 \leq t' \leq \Delta_L : I_1(\sigma) + t' < \Delta & (2\Delta_L \leq \Delta)
\end{array}$$

case (d) let $\sigma \in \mathbb{R}^{\geq 0}$. For the sake of clarity let us consider that $\sigma = t$.
Since $(s_1, t, s_1') \in \rightarrow_1$ we know by $(Q4.6)$ that

$(M1)$ $s'_1 = (l_1, r_1, T_1, I_1 + t, u_1 + t, d_1 + t, f_1)$;

$(M2)$ $u_1 + t \leq \Delta_L$.

With those facts, we know that there exists $s'_2 = (l'_2, v'_2, I'_2, d'_2) \in S_2$ such that $(s_2, t, s'_2) \in \rightarrow_2$ because $(s_2, \Delta_L - u_1, s''_2) \in \rightarrow_2$ by $(H3)$ and $t \leq \Delta_L - u_1$ by $(M2)$.
Now we have $(s_2, t, s'_2) \in \rightarrow_2$ and we know that:

$(M3)$ $s'_2 = (l_1, v_2 + t, I_1 + t, d_1 + t)$

We can now prove that $(s'_1, s'_2) \in R$. We have to check the five points of the simulation relation: $(R1), (R2), (R3)$ and $(R4)$ are easy to prove using hypothesis $(H1)$ to $(H3)$ and $(M1)$ to $(M3)$.
For $(R5)$, since by $(H3)$, there exists $s''_2 = (l''_2, v''_2, I''_2, d''_2) \in S_2$ such that $((l_2, v_2, I_2, d_2), \Delta_L - u_1, (l''_2, v''_2, I''_2, d''_2)) \in \rightarrow_2$, we have $((l_1, v_2 + t, I_1 + t, d_1 + t), (\Delta_L - u_1 - t), (l''_2, v''_2, I''_2, d''_2)) \in \rightarrow_2$.

■