

Robustness and Implementability of Timed Automata

Martin Dewulf, Laurent Doyen, Nicolas Markey, Jean-François Raskin

▶ **To cite this version:**

Martin Dewulf, Laurent Doyen, Nicolas Markey, Jean-François Raskin. Robustness and Implementability of Timed Automata. Proceedings of the Joint Conferences Formal Modelling and Analysis of Timed Systems (FORMATS'04) and Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFT'04), 2004, Grenoble, France. pp.118-133. hal-01194618

HAL Id: hal-01194618
https://hal.science/hal-01194618

Submitted on 7 Sep 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Robustness and Implementability of Timed Automata*

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

Computer Science Departement, Université Libre de Bruxelles, Belgium

Abstract. In a former paper, we defined a new semantics for timed automata, the Almost ASAP semantics, which is parameterized by Δ to cope with the reaction delay of the controller. We showed that this semantics is implementable provided there exists a strictly positive value for the parameter Δ for which the strategy is correct. In this paper, we define the implementability problem to be the question of existence of such a Δ. We show that this question is closely related to a notion of robustness for timed automata defined in [Pur98] and prove that the implementability problem is decidable.

1 Introduction

Timed automata are an important formal model for the specification and analysis of real-time systems. Formalisms like timed automata and hybrid automata are central in the so-called *model-based development methodology for embedded controllers*. The steps underlying that methodology can be summarized as follows: (*i*) construct a (timed/hybrid automaton) model Env of the environment in which the controller will be embedded; (*ii*) make clear what is the control objective: for example, prevent the environment to enter a set of Bad states; (*iii*) design a (timed automata) model Cont of the control strategy; (*iv*) verify that $\mathsf{Reach}(\llbracket \mathsf{Env} \parallel \mathsf{Cont} \rrbracket) \cap \mathsf{Bad} = \varnothing$ (where $\mathsf{Reach}(\llbracket \mathsf{Env} \parallel \mathsf{Cont} \rrbracket)$ denotes the set of states reachable in the transition system associated to the synchronized product of the automaton for the environment and the automaton for the controller). When Cont has been proven correct, it would be valuable to ensure that an implementation Impl of that model can be obtained in a systematic way in order to ensure the conservation of correctness, that is to ensure that $\mathsf{Reach}(\llbracket \mathsf{Env} \parallel \mathsf{Impl} \rrbracket) \cap \mathsf{Bad} = \varnothing$ is obtained by construction.

Unfortunately, this is often not possible for several *fundamental* and/or *technical* reasons. First, the notion of time used in the traditional semantics of timed automata is *continuous* and defines *perfect clocks* with *infinite precision* while implementations can only access time through *digital* and *finitely precise* clocks. Second, timed automata react *instantaneously* to events and time-outs while implementations can only react within a given, usually small but not zero, *reaction*

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

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

delay. Third, timed automata may describe control strategies that are *unrealistic*, like *zeno-strategies* or strategies that ask the controller *to act faster and faster* [CHR02]. For one of those three reasons, a model for a digital controller that has been proven correct may not be implementable (at all) or it may not be possible to turn it systematically into an implementation that is proven correct w.r.t. this model.

To overcome those problems, we recently proposed in [DDR04] an alternative semantics to timed automata. This semantics is called the Almost ASAP semantics, AASAP for short. The AASAP-semantics of a timed automaton A, noted $[\![A]\!]_{\Delta}^{\mathsf{AASAP}}$, is a parametric semantics that leaves as a parameter $\Delta \in \mathbb{Q}^{\geq 0}$ the *reaction delay* of the controller. This semantics relaxes the classical semantics of timed automata in that it does not impose on the controller to react instantaneously but imposes on the controller to react *within Δ time units*. We have proven that a timed controller is implementable with a *sufficiently fast* hardware if and only if there exists $\Delta \in \mathbb{Q}^{>0}$ such that $\mathsf{Reach}([\![\mathsf{Env} \parallel \mathsf{Cont}]\!]_{\Delta}) \cap \mathsf{Bad} = \varnothing$. Details on the notion of implementability can be found in [DDR04]. The *implementability problem* is to determine the existence of such a Δ. The decidability of that important problem is open. We will close this open question here.

The use of the AASAP-semantics in the verification phase can be understood intuitively as follows. When we verify a control strategy using the AASAP-semantics, we test if the proposed strategy is *robust* in the following sense[1]: "*Is the strategy still correct if it is perturbed a little bit when executed on a device that has a finite speed and uses finitely precise clocks ?*"

In this paper, we show that this intuition relating robustness and implementability allows us to draw an interesting and important link with a paper by Puri [Pur98] and allows us to answer *positively* the open question about the decidability of the implementability problem.

Related works. In this paper, we focus on timed controllers and environments that can be modeled using timed automata. There exist related works where the interested reader will find other ideas about implementability.

In [AFILS03], Rajeev Alur et al consider the more general problem of generating code from hybrid automata, but they only sketch a solution and state interesting research questions. The work in this paper should be useful in that context.

In [AFM+02,AFP+03], Wang Yi et al present a tool called TIMES that generates executable code from timed automata models. However, they make the synchrony hypothesis and so they *assume* that the hardware on which the code is executed is infinitely fast and precise.

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

[1] Our notion of robustness is different from another interseting one introduced in [GHJ97].

a mathematical model. So, GIOTTO can be seen an intermediary step between mathematical models like hybrid automata and real execution platforms.

Our paper is structured as follows. In Section 2, we recall some classical definitions related to timed automata and we introduce a general notion of enlarged semantics for those automata. In Section 3, we recall the essential notions and problems related to the AASAP-semantics, and we recall the notion of robustness as introduced by Puri in [Pur98]. In Section 4, we present a small example that illustrates the enlarged semantics and the problems that we want to solve on this semantics. In Section 5, we make formal the link between our notion of implementability and the notion of robustness introduced by Puri. In Section 6, we give a direct proof that the implementability problem is decidable. Finally, we close the paper by a conclusion.

Complete proofs can be found in a longer version of this paper at the following web page: `http://www.ulb.ac.be/di/ssd/cfv/publications.html`.

2 Preliminaries

Definition 1 [TTS] A *timed transition system* $\mathcal{T}$ is a tuple $\langle S, \iota, \Sigma, \rightarrow \rangle$ where S is a (possibly infinite) set of states, $\iota \in S$ is the initial state, Σ is a finite set of labels, and $\rightarrow \subseteq S \times \Sigma \cup \mathbb{R}^{\geq 0} \times S$ is the transition relation where $\mathbb{R}^{\geq 0}$ is the set of positive real numbers. If $(q, \sigma, q') \in \rightarrow$ we write $q \xrightarrow{\sigma} q'$. □

A *trajectory* of a TTS $\mathcal{T} = \langle S, \iota, \Sigma, \rightarrow \rangle$ is a sequence $\pi = (s_0, t_0) \dots (s_k, t_k)$ such that for $0 \leq i \leq k$, $(s_i, t_i) \in S \times \mathbb{R}$ and for $0 \leq i < k$, $s_i \xrightarrow{\sigma} s_{i+1}$ and either $\sigma \in \Sigma$ and $t_{i+1} = t_i$, or $\sigma \in \mathbb{R}^{>0}$ and $t_{i+1} = t_i + \sigma$. We sometimes refer to this trajectory as $\pi[t_0, t_k]$ and write $\pi(t_i)$ instead of q_i. A trajectory is *stutter-free* iff it is not the case for any i that $q_i \xrightarrow{\tau_i} q_{i+1}$ and $q_{i+1} \xrightarrow{\tau_{i+1}} q_{i+2}$ with $\tau_i, \tau_{i+1} \in \mathbb{R}^{>0}$. A state s of $\mathcal{T}$ is *reachable* if there exists a trajectory $\pi = (s_0, t_0) \dots (s_k, t_k)$ such that $s_0 = \iota$ and $s_n = s$. The set of reachable states of $\mathcal{T}$ is noted $\mathsf{Reach}(\mathcal{T})$.

Given a set $\mathsf{Var} = \{x_1, \dots, x_n\}$ of clocks, a *clock valuation* is a function $v \colon \mathsf{Var} \rightarrow \mathbb{R}^{\geq 0}$. In the sequel, we often say that a clock valuation is a point in $\mathbb{R}^n$. If $R \subseteq \mathsf{Var}$, then $v[R := 0]$ denotes the valuation v' such that

$$v'(x) = \begin{cases} 0 & \text{if } x \in R \\ v(x) & \text{if } x \notin R \end{cases}$$

A *closed rectangular guard* g over $\{x_1, \dots, x_n\}$ is a set of inequalities of the form $a_i \leq x_i \leq b_i$, one for each x_i where $a_i, b_i \in \mathbb{Q}^{\geq 0} \cup \{+\infty\}$ and $a_i \leq b_i$. We write $\mathsf{Rect_c}(\mathsf{Var})$ for the set of closed rectangular guards over Var. For $\Delta \geq 0$, we define $[\![g]\!]_\Delta = \{(x_1, \dots, x_n) \mid a_i - \Delta \leq x_i \leq b_i + \Delta\} \subseteq \mathbb{R}^n$. When $\Delta = 0$, we write $[\![g]\!]$ instead of $[\![g]\!]_0$.

We slightly modify the classical definitions related to timed automata [AD94]. In particular, guards on edges are rectangular and closed[2]. Also the value of

[2] In the sequel, the guards are enlarged by strictly positive parameter Δ, and so it is natural to consider them closed.

their clock is bounded by M, the largest constant appearing in guards. The last restriction does not reduce the expressive power of timed automata.

Definition 2 [Timed automaton] A *timed automaton* is a tuple $A = \langle \mathsf{Loc}, \mathsf{Var}, q_0, \mathsf{Lab}, \mathsf{Edg} \rangle$ where

- Loc is a finite set of locations representing the discrete states of the automaton.
- $\mathsf{Var} = \{x_1, \ldots, x_n\}$ is a finite set of real-valued variables.
- $q_0 = (l_0, v_0)$ where $l_0 \in \mathsf{Loc}$ is the initial location and v_0 is the initial clock valuation such that $\forall x \in \mathsf{Var} : v_0(x) \in \mathbb{N} \wedge v_0(x) \leq M$.
- Lab is a finite alphabet of labels.
- $\mathsf{Edg} \subseteq \mathsf{Loc} \times \mathsf{Loc} \times \mathsf{Rect_c}(\mathsf{Var}) \times \mathsf{Lab} \times 2^{\mathsf{Var}}$ is a set of edges. An edge (l, l', g, σ, R) represents a jump from location l to location l' with guard g, event σ and a subset $R \subseteq \mathsf{Var}$ of variables to be reset. □

We now define a family of semantics for timed automata that is parameterized by $\epsilon \in \mathbb{Q}^{\geq 0}$ (drift on clocks) and $\Delta \in \mathbb{Q}^{\geq 0}$ (imprecision on guards).

Definition 3 [Enlarged semantics of timed automata] The semantics of a timed automaton $A = \langle \mathsf{Loc}, \mathsf{Var}, q_0, \mathsf{Lab}, \mathsf{Edg} \rangle$, when the two parameters ϵ and Δ are fixed is given by the TTS $[\![A]\!]^{\epsilon}_{\Delta} = \langle S, \iota, \Sigma, \rightarrow \rangle$ where

1. $S = \{(l, v) \mid l \in \mathsf{Loc} \wedge v \colon \mathsf{Var} \rightarrow [0, M]\}$.
2. $\iota = q_0$.
3. $\Sigma = \mathsf{Lab}$.
4. The transition relation $\rightarrow$ is defined by
   (a) For the discrete transitions: $((l, v), \sigma, (l', v')) \in \rightarrow$ iff there exists an edge $(l, l', g, \sigma, R) \in \mathsf{Edg}$ such that $v \in [\![g]\!]_{\Delta}$ and $v' = v[R := 0]$.
   (b) For the continuous transitions: $((l, v), t, (l', v')) \in \rightarrow$ iff $l = l'$ and $v'(x_i) - v(x_i) \in [(1 - \epsilon)t, (1 + \epsilon)t]$ for $i = 1 \ldots n$. □

In the sequel, we write $[\![A]\!]$ for $[\![A]\!]^0_0$, which is the classical semantics of timed automata.

Remark Our definition of timed automata does not use strict inequalities; this simplifies the presentation and is not restrictive. Indeed, consider a timed automaton A with (possibly open) rectangular guards and the closure automaton $\widehat{A}$ resulting from A by replacing strict inequalities by non-strict ones. It appears obviously that $\mathsf{Reach}([\![\widehat{A}]\!]^{\epsilon}_{\frac{\Delta}{2}}) \subseteq \mathsf{Reach}([\![A]\!]^{\epsilon}_{\Delta})$ and $\mathsf{Reach}([\![A]\!]^{\epsilon}_{\Delta}) \subseteq \mathsf{Reach}([\![\widehat{A}]\!]^{\epsilon}_{\Delta})$, and hence the implementability problem on A *"Does there exist* $\Delta \in \mathbb{Q}^{>0}$ *such that* $\mathsf{Reach}([\![A]\!]^{\epsilon}_{\Delta}) \cap \mathsf{Bad} = \varnothing$ *?"* is equivalent to the the implementability problem on $\widehat{A}$.

We now recall some additional classical notions related to timed automata.

Let $\lfloor x \rfloor$ denote the integer part of x (the greatest integer $k \leq x$), and $\langle x \rangle$ denote its fractional part.

Definition 4 [Clock regions] A *clock region* is an equivalence class of the relation $\sim$ defined over the clock valuations in $\mathsf{Var} \to [0, M]$. We have $v \sim w$ iff all the following conditions hold:

- $\forall x \in \mathsf{Var} : \lfloor v(x) \rfloor = \lfloor w(x) \rfloor$.
- $\forall x, y \in \mathsf{Var} : \langle v(x) \rangle \leq \langle v(y) \rangle$ iff $\langle w(x) \rangle \leq \langle w(y) \rangle$.
- $\forall x \in \mathsf{Var} : \langle v(x) \rangle = 0$ iff $\langle w(x) \rangle = 0$. □

We write $]v[$ for the clock region containing v. It is easy to see that $]v[$ contains the valuations that agree with v on the integer part of the variables, and on the ordering of their fractional part and 0. The closure of the region containing v is noted $[v]$. Such a set is called a *closed region.*

Definition 5 [Region graph] Given the $\mathsf{TTS}\ [\![A]\!] = \langle S, s_0, \Sigma, \to_A \rangle$ of a timed automaton A, we define the corresponding *region graph* $G = \langle \mathcal{C}, \to_G \rangle$ of A :

- $\mathcal{C} = \{(l, [v]) \mid (l, v) \in S\}$ is the set of closed regions.
- $\to_G \subseteq \mathcal{C} \times \mathcal{C}$. $((l, [v]), (l', [v'])) \in \to_G$ if $(l, v) \to_A (l', v')$ and $(l, [v]) \neq (l', [v'])$. □

This definition is meaningful since $\mathcal{C}$ is finite and whenever $(l, [v]) \to_G (l', [v'])$, for any $s \in [v]$ there exists $s' \in [v']$ such that $(l, s) \to_A (l', s')$, and for any $s' \in [v']$ there exists $s \in [v]$ such that $(l, s) \to_A (l', s')$ [AD94].

Let $W = |\mathcal{C}|$ be the total number of regions.

Definition 6 [Zones] A *zone* $Z \subseteq \mathbb{R}^n$ is a *closed set* defined by inequalities of the form

$$x_i - x_j \leq m_{ij}, \quad \alpha_i \leq x_i \leq \beta_i$$

where $1 \leq i, j \leq n$ and $m_{ij}, \alpha_i, \beta_i \in \mathbb{Z}$. □

A set of states is called a *zone-set* if it is a finite union of sets of the form $\{l\} \times Z$ where l is a location and Z is a zone.

Definition 7 [Progress cycle] A *progress cycle* in the region graph of a timed automaton is a cycle in which each clock is reset at least once. □

Assumption 8 We make the assumption that every cycle in the region graph of the timed automata we consider is a progress cycle.

This assumption, made by Puri in his paper [Pur98], is not a very restrictive assumption, since it is weaker than classical non-Zeno assumptions in the literature (for example in [AMPS98], they impose that "in every cycle in the transition graph of the automaton, there is at least one transition which resets a clock variable x_i to zero, and at least one transition which can be taken only if $x_i \geq 1$").

3 AASAP semantics and enlarged semantics

In [DDR04], we introduced the Almost ASAP semantics. This semantics relaxes the usual semantics of timed automata, its main characteristics are summarized as follows:

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

In the same paper, in Theorem 6, we show that this semantics can be encoded using a syntactical transformation of the automaton controller and by enlarging the guards by the parameter Δ which takes its value in the positive rationals. So we can study the AASAP-semantics of $\mathsf{Env} \parallel \mathsf{Cont}$ by considering the semantics $[\![\mathsf{Env} \parallel \mathsf{Cont}']\!]^0_\Delta$ where Cont' is obtained syntactically from Cont. So in the rest of this paper, we will consider the Δ-enlarged semantics instead of the AASAP-semantics.

In this previous work, we have shown that the AASAP-semantics and so the Δ-enlarged semantics allow us to reason about the implementability of a control strategy defined by a timed automaton. The problems that we want to solve algorithmically on the Δ-enlarged semantics are the following ones:

- [Fixed] given a zone-set of Bad states, the timed automata Env and Cont, and a fixed value of $\Delta \in \mathbb{Q}^{>0}$ decide whether $\mathsf{Reach}([\![\mathsf{Env} \parallel \mathsf{Cont}']\!]^0_\Delta) \cap \mathsf{Bad} = \varnothing$;
- [Existence] given a zone-set of Bad states, Env and Cont, decide whether there exists $\Delta \in \mathbb{Q}^{>0}$ such that $\mathsf{Reach}([\![\mathsf{Env} \parallel \mathsf{Cont}']\!]^0_\Delta) \cap \mathsf{Bad} = \varnothing$. This is also called the implementability problem.
- [Maximization] given a zone-set of Bad states, Env and Cont, compute the least upper bound of the set of $\Delta \in \mathbb{Q}^{>0}$ such that $\mathsf{Reach}([\![\mathsf{Env} \parallel \mathsf{Cont}']\!]^0_\Delta) \cap \mathsf{Bad} = \varnothing$. Intuitively, this gives us the information about the slowest hardware that can implement correctly the strategy.

To solve the fixed version, we use the usual reachability algorithm for timed automaton defined in [AD94]. To solve the maximization version (in an approximative way), we observe that for any timed automaton A, any two positive rational numbers Δ_1, Δ_2, if $\Delta_1 \leq \Delta_2$ then $\mathsf{Reach}([\![A]\!]^0_{\Delta_1}) \subseteq \mathsf{Reach}([\![A]\!]^0_{\Delta_2})$. So, given a tolerance $\eta \in \mathbb{Q}^{>0}$, the maximal value of Δ can be approached by η as follows: assuming Bad is reachable in $[\![A]\!]^0_{\Delta=1}$, it suffices to solve the [Fixed] problems with values $\Delta_i = i\eta$ $(0 \leq i \leq \lceil \frac{1}{\eta} \rceil)$, and take as approximation of the maximal Δ the value Δ_i such that the answer of the [Fixed] problem is YES for Δ_i and NO for Δ_{i+1}, which can be found more efficiently with a dichotomy search.

The decidability of the implementability problem is established in the next sections. To achieve this, we draw a strong link with a robust semantics defined

by Puri in [Pur98]. In that paper, Puri shows that the traditional reachability analysis defined in [AD94] is not correct when the clocks drift even by a very small amount. He then reformulates the reachability problem as follows: given a timed automaton A, instead of computing $\mathsf{Reach}([\![A]\!]_0^0)$, he proposes an algorithm that computes $\cap_{\epsilon\in\mathbb{Q}^{>0}}\mathsf{Reach}([\![A]\!]_0^\epsilon)$. When A is clear from the context, this set is denoted by R_ϵ^*. This is the set of states that can be reached when the clocks drift by an infinitesimally small amount. He shows that this set has nice robustness properties with respect to modeling errors. In particular, he establishes that if the clocks are drifting, then guards can be checked with some small imprecisions (see [Pur98] for details).

In our paper, in order to make the link with the implementability problem, we study a variant of this robust semantics where only small imprecisions on guards checking are allowed: the set of reachable states in this semantics is the set $\cap_{\Delta\in\mathbb{Q}^{>0}}\mathsf{Reach}([\![A]\!]_\Delta^0)$. When A is clear from the context, this set is abbreviated by R_Δ^*. We first show that for any timed automaton A, any zone-set Bad, we have that: $\cap_{\Delta\in\mathbb{Q}^{>0}}\mathsf{Reach}([\![A]\!]_\Delta^0)\cap\mathsf{Bad}=\varnothing$ iff there exists $\Delta\in\mathbb{Q}^{>0}$ such that $\mathsf{Reach}([\![A]\!]_\Delta^0)\cap\mathsf{Bad}=\varnothing$. After, we establish that the algorithm proposed by Puri to compute the set $\cap_{\epsilon\in\mathbb{Q}^{>0}}\mathsf{Reach}([\![A]\!]_0^\epsilon)$ is also valid to compute the set of states $\cap_{\Delta\in\mathbb{Q}^{>0}}\mathsf{Reach}([\![A]\!]_\Delta^0)$. As corollaries, we obtain that $\cap_{\epsilon\in\mathbb{Q}^{>0}}\mathsf{Reach}([\![A]\!]_0^\epsilon)=\cap_{\Delta\in\mathbb{Q}^{>0}}\mathsf{Reach}([\![A]\!]_\Delta^0)$ and so we obtain an algorithm to decide the implementability problem.

The proofs of our results follow the general ideas of the proofs of Puri and are based on the structure of limit cycles of timed automata (a fundamental notion introduced by Puri) but we needed new techniques to treat the imprecisions on guards instead of the drifts of clocks as in the original paper. Also the proofs in the paper of Puri are not always convincing, so we reproved a large number of his lemmas that are needed to establish our proof and had to correct one of them.

4 Example

Consider automaton A of Fig. 1. We examine two cases : $\alpha=2$ and $\alpha=3$. For both cases, the reachable states of the classical semantics $[\![A]\!]$ are the same and are depicted in Fig. 2 (the points $v_0\dots v_7$ will be used later in the paper). The safety property we want to verify is that the location *err* is not reachable. Note that in the classical semantics this is true in both cases.

Consider now the enlarged semantics $[\![A]\!]_\Delta^0$ with $\Delta>0$. In this semantics, guards are enlarged by the amount Δ. The edge from l_1 to l_2 has the guard $a\le 2+\Delta$ and the edge from l_2 to l_1 has the guard $b\ge 2-\Delta$. Starting from the initial state $(l_1, a=1, b=0)$, the jump to l_2 can occur Δ time units later, so that the states $(l_2, a=0, b\le 1+\Delta)$ are reached. Similarly, the transition from l_2 back to l_1 is enabled Δ time units earlier and the states $(l_1, a\ge 1-2\Delta, b=0)$ can be reached. By iterating the cycle, the states $(l_1, a\ge 1-2k\Delta, b=0)$ and $(l_2, a=0, b\le 1+(2k-1)\Delta)$ are reachable. So, for any $\Delta>0$ some new states

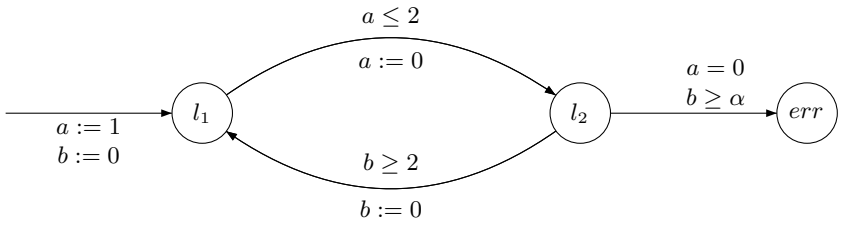


Fig. 1. A timed automaton A.

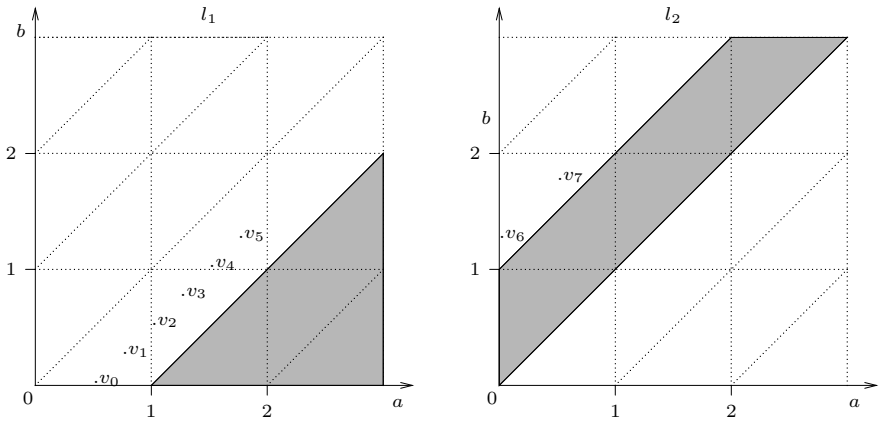


Fig. 2. $\mathsf{Reach}([\![A]\!])$ for the timed automaton A of Fig. 1.

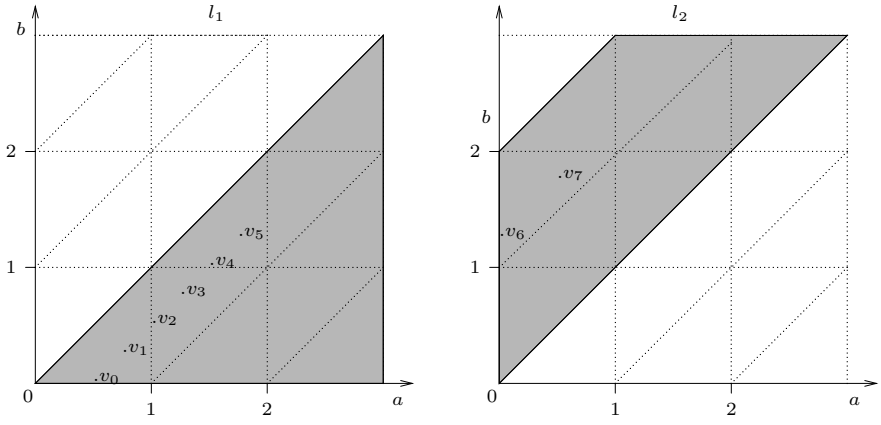


Fig. 3. The set R^*_Δ for the timed automaton A of Fig. 1.

are reachable in $[\![A]\!]^0_\Delta$ that were not reachable in the classical semantics. Those states are represented in Fig. 3.

From those new states that become reachable in location l_2, if $\alpha = 3$, the location *err* remains unreachable but if $\alpha = 2$ it becomes reachable.

Clearly, from this example, one sees that a correct timed system (in the sense of the classical semantics) could have a completely different (and potentially bad) behavior due to an infinitesimally small inaccuracy in testing of the clocks (which is unavoidable since the clocks are discrete in embedded systems). This is the case for the automaton of figure 1. When $\alpha = 2$, the classical semantics is not robust since even the slightest error in guard checking allows new discrete transition to be taken. In other words, there is no strictly positive value for the parameter Δ that still ensures the safety property for $[\![A]\!]^0_\Delta$. Systems with such features cannot have a correct implementation because their correctness relies on the idealization of the mathematical model.

But this is not always the case: for the same automaton when $\alpha = 3$, no more discrete transitions are possible in the enlarged semantics than in the classical one. In this case, we can answer positively to the question "Is there a strictly positive value for parameter Δ that allows the enlarged semantics to still satisfy the safety property?". And indeed, we can prove that any value $\Delta < \frac{1}{3}$ is satisfying.

5 Linking robustness and implementability

The classical semantics of timed automaton A is $[\![A]\!]^{\epsilon=0}_{\Delta=0}$, which is a mathematical idealization of how we expect an implementation would behave: it makes the hypothesis that the hardware is perfectly precise and infinitely fast. Unfortunately, the execution of a timed automaton on a real hardware cannot be considered as ideal in the mathematical sense. It is thus an interesting question to know whether a small drift or imprecision of the clocks could invalidate some properties satisfied by the classical semantics. Drifts in clocks have been studied in [Pur98]. We are interested in studying imprecisions in the evaluation of the guards since it is directly connected to the question of implementability, as explained above. The main result of this paper is the following.

Theorem 9 *There exists an algorithm that decide for any timed automaton A, any zone-set* Bad *if there exists $\Delta \in \mathbb{Q}^{>0}$ such that* $\mathsf{Reach}([\![A]\!]^0_\Delta) \cap \mathsf{Bad} = \varnothing$.

To prove Theorem 9, we show that the set Bad intersects R^*_Δ iff it intersects $\mathsf{Reach}([\![A]\!]^0_\Delta)$ for some $\Delta > 0$. As shown in the next section, Algorithm 1 computes R^*_Δ. Consequently, the implementability problem is decidable.

Theorem 10 *For any timed automaton A, any zone-set* Bad, $R^*_\Delta \cap \mathsf{Bad} = \varnothing$ *iff* $\exists \Delta > 0 : \mathsf{Reach}([\![A]\!]^0_\Delta) \cap \mathsf{Bad} = \varnothing$.

The proof of Theorem 10 relies on two intermediate lemmas, one of which corrects a wrong claim in [Pur98]: it gives a bound on the distance between

two zones with empty intersection. This bound is claimed to be $\frac{1}{2}$. We show that $\frac{1}{n}$ where n is the dimension of the space is the tightest bound. However the final results of [Pur98] are not deeply affected by this mistake. The distance considered is defined by $d_\infty(x,y) = \|x-y\|_\infty = \max_{1\leq i\leq n}(|x_i - y_i|)$. Let us reformulate that lemma.

Lemma 11 (Corrected from [Pur98, Lemma 6.4**])** *Let $Z_1 \subseteq \mathbb{R}^n$ and $Z_2 \subseteq \mathbb{R}^n$ be two zones such that $Z_1 \cap Z_2 = \varnothing$. Then, for any $x \in Z_1$ and $y \in Z_2$, $d_\infty(x,y) \geq \frac{1}{n}$. This bound is tight.*

In the sequel, when a distance d or a norm $\|\ \|$ is used, we always refer to d_∞ and $\|\ \|_\infty$. The following lemma relies on the theory of real numbers and the basics of topology.

Lemma 12 *Let $A_\Delta (\Delta \in \mathbb{R}^{>0})$ be a collection of sets such that $A_{\Delta_1} \subseteq A_{\Delta_2}$ if $\Delta_1 \leq \Delta_2$. Let $A = \bigcap_{\Delta>0} A_\Delta$ be nonempty. If $d(A,B) > 0$, then there exists $\Delta > 0$ such that $A_\Delta \cap B = \varnothing$.*

Proof of theorem 10. If $R^*_\Delta \cap \mathsf{Bad} = \varnothing$, since R^*_Δ and Bad are unions of sets of the form $\{l\} \times Z_l$ where Z_l is a zone, Lemma 11 applies and we have $d(R^*_\Delta, \mathsf{Bad}) > 0$. From Lemma 12, we obtain that there exists $\Delta > 0$ such that $\mathsf{Reach}([\![A]\!]^0_\Delta) \cap \mathrm{Bad} = \varnothing$.

If there exists $\Delta > 0$ such that $\mathsf{Reach}([\![A]\!]^0_\Delta) \cap \mathrm{Bad} = \varnothing$, then trivially $R^*_\Delta \cap \mathsf{Bad} = \varnothing$. ■

6 Algorithm for computing R^*_Δ

Algorithm 1: Algorithm from [Pur98] for computing $R^*_\epsilon(A)$ for a timed automaton A.

Data : A timed automaton $A = \langle \mathsf{Loc}, \mathsf{Var}, q_0, \mathsf{Lab}, \mathsf{Edg} \rangle$
Result : The set $J^* = R^*_\epsilon$
begin
1. Construct the region graph $G = (R_A, \longrightarrow_A)$ of A ;
2. Compute $SCC(G)$ = strongly connected components of G;
3. $J^* \leftarrow [(q_0)]$;
4. $J^* \leftarrow \mathsf{Reach}(G, J^*)$;
5. **if** for some $S \in SCC(G)$, $S \not\subseteq J^*$ and $J^* \cap S \neq \varnothing$ **then**
   $J^* := J^* \cup S$;
   Goto 4 ;

end

In this section, we prove that the algorithm proposed in [Pur98] computes R^*_Δ. This implies that $R^*_\epsilon = R^*_\Delta$. The algorithm is shown as Algorithm 1.

Let us first examine how Algorithm 1 performs on the example of section 4. In the region graph of the timed automaton of Fig. 1, there is a cycle that runs from valuation v_0 to itself through v_1 to v_7 (see Fig. 2). Thus there is a cycle through the regions containing valuations v_0 to v_7. Furthermore, these regions have an intersection with the set of reachable states in the classical semantics (in gray). Indeed since we consider closed regions, the intersection of two adjacent regions is not empty. Since those regions form a strongly connected component of the region graph and their intersection with the reachable states in the classical semantics is not empty, the algorithm adds all those regions to the set J^*.

One can check that all regions of R^*_Δ for the automaton A of figure 1 will be correctly added by Algorithm 1 (see figure 3).

In the rest of this section, we prove that this algorithm computes R^*_Δ, by proving $J^* \subseteq R^*_\Delta$ on the one hand, and $R^*_\Delta \subseteq J^*$ on the other hand.

6.1 Limit cycles

This section studies the behavior of limit cycles. A *limit cycle* of a timed automaton A is a trajectory $\pi = (q_0, t_0)(q_1, t_1) \dots (q_k, t_k)$ of $[\![A]\!]$ such that $t_k > t_0$ and $q_k = q_0$. As suggested in [Pur98], given a progress cycle in the region graph and a region on this cycle, we focus on the subset of points of this region having a limit cycle. We first define this subset:

Definition 13 [See [Pur98, Section 7.1]] Consider a cyclic path $p = p_0\, p_1\, \dots\, p_N$ with $p_N = p_0$ in the region graph of a timed automaton A. We define the *return map* $R_p \colon 2^{p_0} \to 2^{p_0}$ by $R_p(S) = \cup_{q \in S} R_p(\{q\})$ for $S \subseteq p_0$, and, for singletons,

$$R_p(\{q_0\}) = \left\{ q_N \;\middle|\; \begin{array}{l} \text{there exists a trajectory } \pi \text{ in } [\![A]\!] \text{ s.t.} \\ \pi = (q_0, t_0)(q_1, t_1) \dots (q_N, t_N) \text{ and } \forall i.\ q_i \in p_i \end{array} \right\}.$$

The set $L_{i,p}$ of points which can return back to themselves after i cycles through p, is defined as follows: $L_{i,p} = \{q \mid q \in R^i_p(q)\}$. The set of points with limit cycles through p is $L_p = \cup_{i \in \mathbb{N}} L_{i,p}$. □

In the sequel, we write R or L instead of R_p or L_p when the path p is clear from the context. The interesting property of L_p is that it is always reachable from any valuation in p:

Theorem 14 ([Pur98, Lemma 7.10**])** *Let $p = p_0\, p_1\, \dots\, p_N$ be a cycle in G. Then for any $z \in p_0$, there exists $z', z'' \in L$ s.t. there exist trajectories in $[\![A]\!]$ from z to z' and from z'' to z.*

6.2 Soundness of Algorithm 1: $J^* \subseteq R^*_\Delta$

Let $r \in \mathbb{R}^{\geq 0}$ and $x \in \mathbb{R}^n$. The *closed ball* of radius r centered in x is the set $B(x, r) = \{x' \mid d(x, x') \leq r\}$. The following Lemma shows how a trajectory may be modified when enlarging guards in timed automata:

Lemma 15 *Let A be a timed automaton with n clocks, $\Delta \in \mathbb{Q}^{>0}$, and $\delta = \frac{\Delta}{n}$. Let $p = p_0\, p_1\, p_2 \dots p_N$ be a cycle in the region graph of A. Let u be a valuation in p_0 having a limit cycle, i.e. for which there exists a trajectory $\pi[0,T]$ in $[\![A]\!]_0^0$ following p and with $\pi(0) = \pi(T) = u$. Let $v \in p_0 \cap B(u,\delta)$ be a neighbor valuation. Then there exists a trajectory from u to v in $[\![A]\!]_\Delta^0$.*

Intuitively, the result is proved by slightly anticipating or delaying the transition dates when a clock is reset.

Consider for instance the following path:

$$(x = a, y = b) \xrightarrow{\tau_1} \xrightarrow[x:=0]{} (x = 0, y = b + \tau_1) \xrightarrow{\tau_2} (x = a', y = b').$$

with $a' = \tau_2$ and $b' = b + \tau_1 + \tau_2$. By slightly modifying the continuous transition labels τ_i, we get

$$(x = a, y = b) \xrightarrow{\tau_1 - \delta} \xrightarrow[x:=0]{} (x = 0, y = b + \tau_1 - \delta) \xrightarrow{\tau_2 + \delta} (x = a' + \delta, y = b').$$

Thus the final value of clock y is identical in both cases, while the final value of x has been slightly modified. By carefully repeating this procedure, we can independently modify the final valuations of each clock. This corresponds to the idea behind Lemma 15.

Theorem 16 *Let A be a timed automaton. Let $p = p_0\, p_1 \dots p_N$ be a cyclic path in the region graph of A, and let x and y be two valuations in p_0. For any $\Delta \in \mathbb{Q}^{>0}$, there exists a trajectory from x to y in $[\![A]\!]_\Delta^0$.*

Proof. From Lemma 14, there exists $u, v \in L$ s.t. $x \to u$ and $v \to y$ in $[\![A]\!]$. Then u satisfies the conditions of Lemma 15, and thus any $w \in L$ "close to" u is reachable from u in $[\![A]\!]_\Delta^0$. Since L is convex, by recursively applying the previous argument, any $w \in L$ is reachable from u in $[\![A]\!]_\Delta^0$. In particular, v is reachable from u in $[\![A]\!]_\Delta^0$, and y is reachable from x in $[\![A]\!]_\Delta^0$. ■

As a consequence:

Theorem 17 *The set J^* computed by Algorithm 1 is a subset of R_Δ^*, i.e. $J^* \subseteq R_\Delta^*$.*

Proof. Let $\Delta > 0$. If a set of regions J is a subset of $\mathsf{Reach}([\![A]\!]_\Delta^0)$, then so is the set of reachable regions from J in the region graph G. Moreover, whenever a state of J appears in a cyclic region path p, then Theorem 16 ensures that any state in that region path is reachable in $[\![A]\!]_\Delta^0$. Thus $J \cup p \subseteq \mathsf{Reach}([\![A]\!]_\Delta^0)$. Since J^* is built by successively applying the above two operations, this ensures that $J^* \subseteq \mathsf{Reach}([\![A]\!]_\Delta^0)$. This holds for any $\Delta > 0$, thus $J^* \subseteq R_\Delta^*$. ■

6.3 Completeness of Algorithm 1: $R^*_\Delta \subseteq J^*$

To prove the completeness of Algorithm 1, we need to better understand the relationship between trajectories of $[\![A]\!]^0_\Delta$ and those of $[\![A]\!]$. In particular, Theorem 18 states that any trajectory π' of $[\![A]\!]^0_\Delta$ can be approached by a trajectory π of $[\![A]\!]$, making the same discrete transitions and passing by the same regions.

Theorem 18 *For any distance $\delta \in \mathbb{R}^{>0}$, for any number of steps $k \in \mathbb{N}$, there exists $\Delta \in \mathbb{Q}^{>0}$ such that for any k-step trajectory $\pi' = (q'_0, t'_0) \dots (q'_k, t'_k)$ of $[\![A]\!]^0_\Delta$, there exists a k-step trajectory $\pi = (q_0, t_0) \dots (q_k, t_k)$ of $[\![A]\!]$ such that for any position i $(0 \leq i \leq k)$*

- *$q_i \in [q'_i]$, that is the two trajectories cross the same regions,*
- *and the distance between corresponding states q_i and q'_i is bounded by δ, that is $q_i = (l_i, v_i)$, $q'_i = (l'_i, v'_i)$ where $l_i = l'_i$ and $\|v_i - v'_i\| \leq \delta$.*

Below, we give the general ideas underlying the proof. It uses the following lemma, stating that if v is reachable from u by letting time elapse, then any point $x \in [u]$ in the neighborhood of u can reach some point $y \in [v]$ in the neighborhood of v. More precisely:

Lemma 19 *Suppose $r \rightarrow_G r'$ in the region graph G of a timed automaton, and $u \xrightarrow{\tau} v$ with $u \in r$, $v \in r'$ and $\tau \in \mathbb{R}^{\geq 0}$. Then for any $x \in B(u, R) \cap r$, there exists $y \in B(v, 2R) \cap r'$ such that $x \xrightarrow{\tau'} y$ for some τ' with $|\tau - \tau'| \leq R$.*

Proof (Sketch) of theorem 18.

The outline of the proof is as follows. We define a "simulation" relation $\mathcal{R}$ between valuations, which is a subset of $\mathbb{R}^n \times \mathbb{N} \times \mathbb{R}^n$ and we write $\mathcal{R}_a(x', x)$ if $(x', a, x) \in \mathcal{R}$. We show that

$$\text{If } \begin{cases} a \leq k \\ \mathcal{R}_a(x', x) \\ x' \xrightarrow{t'} y' \xrightarrow{\sigma} z' \text{ in } [\![A]\!]^0_\Delta \end{cases} \quad \text{then } \exists y, z : \begin{cases} x \xrightarrow{t} y \xrightarrow{\sigma} z \text{ in } [\![A]\!] \\ \mathcal{R}_{a+1}(z', z) \\ \|z - z'\| \leq \|y - y'\| \leq \delta \end{cases}$$

The relation $\mathcal{R}_a$ takes into account the current position in the trajectory: if x' is the a^{th} state in the trajectory, then x is in a neighborhood of size parameterized by a. This size increases with a, because the longer is the trajectory, the greater is the possible deviation (Fig. 4). This is not a classical simulation relation because it only works for a fixed trajectory length k (which is sufficient for proving Theorem 18).

Intuitively, $\mathcal{R}$ links a valuation x with a valuation x' if whenever two clock have *close* fractional parts in x' (for a special distance $\mathcal{D}$), those clocks have *identical* fractional parts in x.

Fig. 5 illustrates the necessity of this choice: consider the trajectory $(x', 0)$ $(y', t')(z', t')$ of $[\![A]\!]^0_\Delta$. We must construct a trajectory $(x, 0)(y, t)(z, t)$ of $[\![A]\!]$

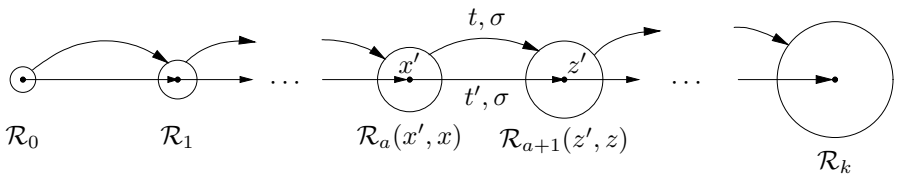


Fig. 4. Simulation of the trajectory π' of $[\![A]\!]^0_\Delta$.

taking the same discrete transition (labeled with σ). In the enlarged semantics, the guard on this transition is satisfied by y' (*i.e.* $y' \in [\![g]\!]_\Delta$).

The only possible choice for y is to both lie in the region $[y']$ and in $[\![g]\!]_0$ as shown in the figure. This imposes the choice of x on the diagonal (*i.e.* with identical fractional parts for x_i and x_j).

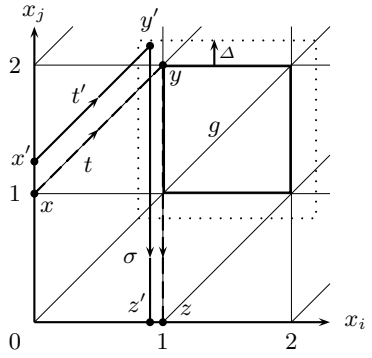


Fig. 5. Simulating $x' \xrightarrow{t'} y' \xrightarrow{\sigma} z'$ enforces to take x with $\langle x_i \rangle = \langle x_j \rangle$. This is because $\langle x'_i \rangle$ and $\langle x'_j \rangle$ are too close.

$\mathcal{R}$ is formally defined in the complete proof, and it is shown that the Δ can be chosen such that $\Delta < \frac{\delta}{2^k(2+4n)}$. ■

The following theorem is a direct consequence of Theorem 18. It says that points in J^* cannot reach points that are more than distance α away from J^* in $[\![A]\!]^0_\Delta$ for sufficiently small Δ.

Theorem 20 *Suppose $x \in J^*$ and there exists a trajectory from x to y in $[\![A]\!]^0_\Delta$ with $\Delta < \frac{\alpha}{2^{W+1}(2+4n)}$ where $\alpha < \frac{1}{2n}$, and W is the total number of regions. Then $d(y, J^*) < \alpha$.*

Finally, we can establish the completeness of Algorithm 1.

Theorem 21 *The set J^* computed by Algorithm 1 contains R^*_Δ, i.e. $R^*_\Delta \subseteq J^*$.*

Proof. Since J^* is a closed set containing the initial state, from Theorem 20, for any $y \in \mathsf{Reach}([\![A]\!]^0_\Delta)$, $d(y, J^*) \leq \alpha$ where α can be made arbitrarily small. It follows that $R^*_\Delta \subseteq J^*$. ■

This concludes the proof that Algorithm 1 computes R^*_Δ. According to [Pur98], we have the corollary that $R^*_\Delta = R^*_\epsilon = R^*$ where $R^* = \cap_{\Delta \in \mathbb{Q}^{>0}} \cap_{\epsilon \in \mathbb{Q}^{>0}} \mathsf{Reach}([\![A]\!]^\epsilon_\Delta)$ so that the enlarged semantics R^*_Δ is also robust against drifts in clocks.

6.4 Complexity

Complexity issues have been studied in [Pur98], so we recall the main result.

Theorem 22 ([Pur98]) *Given a timed automaton $A = \langle \mathsf{Loc}, \mathsf{Var}, q_0, \mathsf{Lab}, \mathsf{Edg} \rangle$ and a location $l_f \in \mathsf{Loc}$, determining whether a state $(l_f, v) \in R^*_\Delta$ for some valuation v is PSPACE-Complete.*

7 Conclusion

In this paper, we have shown that a notion of robustness defined by Puri [Pur98] is closely related to a notion of implementability that we recently introduced in [DDR04]. Making this link formal allowed us to show that our notion of implementability is decidable for the class of timed automata. To establish this link, we have proved that the algorithm proposed by Puri computes the set of reachable states of timed automata where guards are enlarged by an infinitesimally small rational value. The existence of such a value implies the implementability as shown in our previous paper. The proofs of the decidability result rely on non trivial adaptations of the main ideas underlying the study of drift in the rate of clocks made by Puri.

The algorithm that is used to check implementability manipulates strongly connected components of the region graph. It can be seen as defining exact accelerations of cycles of the timed automaton.

We will work in the future on making those accelerations practical and as a consequence, we will work on to turn the theoretical algorithm proposed in this paper into a practical one. If we succeed in this task, the results of this paper and of our previous paper [DDR04] will allow us to propose a practical procedure to produce provably correct code from proved correct controller modeled as timed automata.

References

[AD94] Rajeev Alur and David 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 *ACM Symposium on Languages, Compilers, and Tools for Embedded Systems*, 2003.

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

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

[AMPS98] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *Proc. System Structure and Control*. Elsevier, 1998.

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

[DDR04] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. In *HSCC 04: Hybrid Systems—Computation and Control*, Lecture Notes in Computer Science 2993, pages 296–310. Springer-Verlag, 2004.

[GHJ97] Vineet Gupta, Thomas A. Henzinger, and Radha Jagadeesan. Robust timed automata. In O. Maler, editor, *Hybrid and Real-Time Systems*, Lecture Notes in Computer Science, pages 331–345. Springer Verlag, March 1997.

[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.

[Pur98] Anuj Puri. Dynamical properties of timed automata. In *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems, 5th International Symposium, FTRTFT'98, Lyngby, Denmark, September 14-18, 1998*, volume 1486 of *Lecture Notes in Computer Science*, pages 210–227. Springer, 1998.