Robustness in Real-time Systems

Nicolas Markey*
LSV – CNRS & ENS Cachan, France
nicolas.markey@lsv.ens-cachan.fr

Abstract

We review several aspects of robustness of real-time systems, and present recent results on the robust verification of timed automata.

1 Introduction

Most embedded systems have strong real-time constraints, which it is compulsory to take into account in their modelling and verification. Timed automata [6] have been introduced twenty years ago for that purpose: they extend finite-state automata with real-valued clocks, which can be used to specify how much time may elapse between different events in the system. These timing constraints are usually just conjunctions of atomic constraints, comparing the value of a clock against an integer. Formally, given a finite set of proposition AP and a finite alphabet of actions Σ:

Definition 1. *A* timed automaton *is a tuple* $\mathcal{A} = \langle Q, \ell, \mathcal{C}, T, I\rangle$ *where* Q *is a finite set of states,* $\ell\colon Q \to 2^{\mathsf{AP}}$ *labels states with the atomic propositions they satisfy,* $\mathcal{C}$ *is a finite set of clock variables,* $T \subseteq Q \times C(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times Q$ *(where* $C(\mathcal{C})$ *is the set of timing constraints over* $\mathcal{C}$*) is the set of transitions, and* $I\colon Q \to C(\mathcal{C})$ *assigns an invariant to each state.*

A transition (q, ϕ, σ, r, q') in T indicates that it is allowed to go from q to q' only when the values of the clocks satisfy constraint ϕ. If the transition is taken, all the clocks in r are reset to zero. Invariants in states are clock constraints that must be satisfied for being allowed to let time elapse.

Fig. 1 depicts a small example of a timed automaton, modelling (part of) the driver for a one-button computer mouse. When a `click` is received, the driver waits for a small period of time, until it can decide whether it is a single- or a double-click. This delay is spent by the rightmost state: when entering this state, clock c is set to zero;

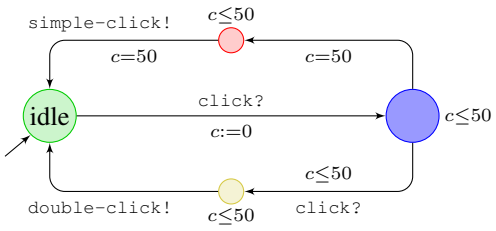

Fig. 1. A timed automaton modelling a computer mouse

as long as c is less than 50 (milliseconds, say), the driver will wait for a second click. After that delay, it has to leave the rightmost state, and will then send a `single-click` or `double-click` event.

We will not define the semantics of timed automata more formally, as the above intuition will be sufficient for understanding this paper. We refer the interested reader to the rich literature on the subject [6, 5, 12] for more details. An important point to notice is that timed automata are a finite representation of infinite-state systems: a *configuration* of a timed automaton contains both the state of the automaton and the (real-valued) valuation of the clocks. Still, the underlying infinite-state transition system has nice properties, which make reachability decidable over these automata. The main ingredient for deciding reachability is *region equivalence*: roughly, two clock valuations that satisfy the same timing constraints will lead to the same *untimed* behaviours (*i.e.*, the delays may have to be changed). Similarly, when a clock value is larger than the maximal constant it is compared to in the automaton, the exact value of the clock can be forgotten.

Formally, given a timed automaton $\mathcal{A} = \langle Q, \ell, \mathcal{C}, T, I\rangle$, write $M_{\mathcal{A}}$ for the maximal integer constant appearing in the timing constraints of $\mathcal{A}$. Two valuations of the clocks of $\mathcal{A}$ are said *region-equivalent*[1] if they satisfy exactly the same

*This work has been partly supported by the EU FP7 under grant number ICT-214755 (Quasimodo), and by the French projects ANR-06-SETI-003 (DOTS) and ANR-2010-BLAN-0317 (ImpRo).

[1] This definition is not the original one [6], but is simpler and will be sufficient for our purpose.

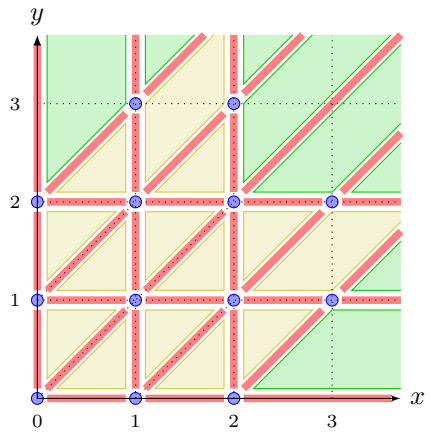

Fig. 2. Region equivalence for two clocks

set of timing constraints among

$$\left\{ \begin{array}{r} x \sim c \\ x-y \sim c \end{array} \;\middle|\; \begin{array}{l} x \in \mathcal{C},\ y \in \mathcal{C},\ c \in \{0,...,M_{\mathcal{A}}\}, \\ \sim \in \{<,\leq,=,\geq,>\} \end{array} \right\}$$

The number of equivalence classes is bounded by $[2M_{\mathcal{A}}+2]^{(\mathcal{C}+1)^2}$. This equivalence can be used to build the *region automaton* $\mathcal{R}_{\mathcal{A}}$ of $\mathcal{A}$, which is a finite-state automaton that can be proved (time-abstract) bisimilar to $\mathcal{A}$, and can thus satisfy the same untimed properties as $\mathcal{A}$. In particular:

Theorem 2 ([6])**.** *Reachability can be decided in exponential time (and is* PSPACE*-complete) on timed automata.*

This provides a way to check simple safety properties of timed automata, and has been extended to temporal-logic model checking. Efficient algorithms (based on a variation of regions called *zones*, which allow for more efficient algorithms in practice) have also been developed and implemented in tools like Uppaal and Kronos, leading to a wide use of timed automata in many industrial case studies [11, 33, 25].

2 Robustness issues

One problem remains: while timed automata have very nice algorithmic properties, their semantics is not always adequate: timed automata are used to model *real* systems, in which time cannot be measured as precisely as in this mathematical model. We now illustrate two of these problems on concrete examples.

First, the semantics of timed automata assumes that transitions are immediate. It was already noticed long ago that this induces undesirable behaviours, the most famous of which are *Zeno* executions, where infinitely many transitions can be taken during a single time unit. But these are by far not the only example: consider for instance the timed automaton depicted on Fig. 3 (originating from [19]). While this automaton does not contain Zeno executions, it can be proved that, along any run, the accumulated time elapsed in the rightmost state is bounded by 1.

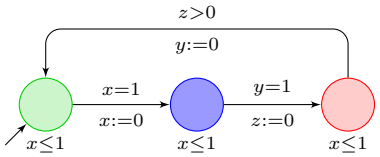

Fig. 3. Non-Zeno timed automaton with a convergence phenomenon

Second, timed automata also assume arbitrary precision in the value of the clocks. Consider the automaton depicted on Fig. 4, taken from [28]: it represents one of several similar components[2] (each having a different non-zero `id`) that want to access a critical section (represented as their square state). It can be proved that mutual exclusion is enforced by these automata: any two of them will never be in their critical section simultaneously. However, if the timing constraints on the transition leading to the critical state is enlarged to $x_{\texttt{id}} \geq 2$ (instead of $x_{\texttt{id}} > 2$), mutual exclusion is lost.

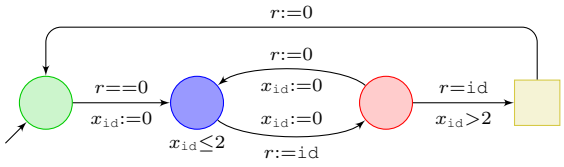

Fig. 4. Fischer's mutual exclusion protocol

These problems witness the fact that the formal semantics of timed automata is not realistic: it is very convenient for mathematical reasoning, but is also too precise: some timed automata may enjoy properties that may be lost if (arbitrary) small perturbation come into play. On the other hand, some timed automata may fail to satisfy some properties because of unreaslistic behaviours (such as Zeno executions), which will never occur on a physical device. In the rest of this paper, we review several approaches to these problems.

[2] In this example, the components communicate through a shared variable r. This could easily be encoded with standard synchronization with a central automaton storing the value of the variable.

3 Different approaches to robustness

3.1 Sampled semantics

One natural solution to these problems is to use a discrete-time semantics [7]: in that semantics, clocks take integer values (or integer multiples of a fixed granularity), and are all updated at the same time. This semantics is closer to the finite-frequency behaviour of microprocessors, and rules out all kinds of convergent behaviours; however, it strengthens the synchrony hypothesis, by preventing the systems to perform any action between two integer dates. This is not realistic: while the computerized system will indeed not perform any action between two ticks, the physical system may evolve continuously. This has been proved to make a difference, for instance, when modelling digital circuits [18]: it takes some time in each gate to propagate changes of the input signal, and the changes cannot be assumed to occur at integer dates (some behaviours might be lost).

The natural question of the existence of a *sufficiently small* sampling rate preserving properties of the dense-time semantics has been studied in different settings, and turns out to be quite difficult [27, 2].

On the algorithmic side, the discrete-time semantics is easier to work with than the real-time one: only punctual regions are visited, and several problems that are undecidable in dense-time become decidable. Still, reachability remains PSPACE-hard, and the gain on the algorithmic side is not so appealing. The corresponding algorithms have been implemented for instance in the tool Rabbit [14].

3.2 Robust timed automata and tube semantics

A more refined approach has been proposed in [24]: intuitively, it considers only the executions that would still be accepted (or generated) by the automaton if the delays were slightly perturbed along that execution. This is a topological approach: the semantics discards the executions whose set of neighbour executions that are accepted is not dense. This is called the *tube semantics*, reflecting the fact that there must be a dense "tube" of accepting executions around a given execution for this execution to be considered valid. In the example of Fig. 5, the execution in which the delay between the first two a is 2 (visiting the bottom state) would be discarded. The same for the execution where this delay is 1. On the other hand, the executions where this delay is in $(0, 1)$ are accepted.

It was proved in [24] that safety of robust timed automata (*i.e.*, deciding whether a bad state is reachable under that semantics) is in PSPACE. The idea of the algorithm is to consider the *interior automaton*: this is a (slightly generalized) timed automaton, only involving strict inequalities in its timing constraints, and having the same *tube language* as the initial automaton. Having only strict inequalities, its *tube language* equals its language in the classical semantics, so that emptiness checking is in PSPACE.

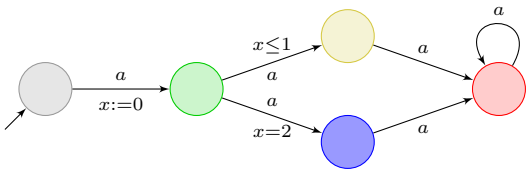

Fig. 5. Tube acceptance in timed automata

3.3 Probabilistic semantics of timed automata

A quite similar approach has been proposed recently in [9]: there, a *measure* on the set of executions of timed automata is defined, which provides a way of computing *how (un)likely* an execution is. Again, executions involving equality constraints will generally have probability 0, as would be the case of the execution visiting the bottom state of the automaton of Fig. 5. However, if no other transition were available, then this transition, despite its equality constraint, would have probability 1. This is the case of the first transition in the example of Fig. 6. Then, assuming uniform distributions on the delays, the probability of ending up in the top-most state when starting with $x = 0$ in the left-most state equals $3/4$: during the first two time units in the middle state (when t is between 1 and 3), both transitions have the same probability to occur. Over the last two time units (when t is between 3 and 5), only the top-most transition is available. In the end, the probability of taking this transition is

$$\frac{1}{4}\left(\int_1^3 \frac{1}{2}\mathrm{d}t + \int_3^5 \mathrm{d}t\right) = \frac{3}{4}.$$

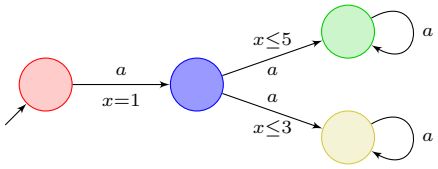

Fig. 6. Probabilistic semantics

Over finite paths, verifying whether an LTL property[3] holds with probability 1 been proved decidable [9]: it can be checked on the region automaton, by discarding the transitions that have probability zero. The problem is harder

[3] LTL is a logical formalism for expressing properties on sequence of events, based on modalities such as "until", "eventually" and "always".

when considering infinite paths, and has only been proved decidable on one-clock timed automata [10, 13].

4 Drifting and enlarged semantics

One last approach, which we develop in the rest of this paper, was introduced in [30]. As opposed to the above approaches, it *extends* the set of executions instead of shrinking it: the idea is to assume that there might be some imprecision on the values of the clock, hence allowing a transition at time $2+\epsilon$ where it was only allowed befire time 2. We develop this approach below, and present two results on this semantics.

4.1 Enlarging the semantics

The approach here is to add imprecision on the values of the clocks during the runs of the automaton. This is two-fold: on the one hand, clocks may not have the exact same speed, resulting in slight drifts, which can accumulate if they are not synchronized regularly; on the other hand, because clocks have finite frequency, their value is piecewise-constant: hence a transition that is to be taken at time t may also be available at time $t+\epsilon$. In his seminal paper, Puri defined the *enlarged semantics* $[\![\mathcal{A}]\!]_{\delta,\epsilon}$ as the semantics of the hybrid automaton obtained from $\mathcal{A}$ by replacing each guard of the form $a \leq x \leq b$ with $a-\delta \leq x \leq b+\delta$, and making the clocks evolve at a rate in $[1-\epsilon, 1+\epsilon]$.

The interest and usefulness of these semantics has recently been reinforced by [22], in the setting of *implementability*. There, yet another semantics is proposed, in which the timed automaton is run over a (simple) model of a CPU equipped with a digital clock: the value of the clock is updated only sporadically, at least once every δ_P time units, and may drift at a rate of ϵ; the CPU also has finite frequency: it sporadically (at least once every δ_L time units) performs the following sequence of actions: first, it reads the value the global clock (from which the values of the clocks of the automaton are derived); then it evaluates the guards of the transitions; and finally, it takes one of the possible transitions of the timed automaton. A timed automaton is said *implementable* w.r.t. a given condition if the condition is fulfilled under this semantics, for some (positive) values of ϵ, δ_P and δ_L.

This semantics is quite irregular: of course, following the ideas of [4], it can be modelled as a timed automaton (as far as only guard enlargements are involved). However, δ_L and δ_P will generally not have the same granularity as the other constants, which may dramatically increase the verification time; also, the representation of clock drifts involves rectangular hybrid automata. Moreover, we are more interested in the *parametrized problem*: does there exist positive values for ϵ, δ_P and δ_L for which the model behaves correctly? Indeed, the aim here is to decide the existence of a sufficiently precise hardware on which the timed automaton under study could be safely implemented.

An (over-)approximate solution has been proposed in [22, 21], using the enlarged semantics: the set of runs of the automaton under the program semantics is included in the set of runs under the enlarged semantics, provided that the enlargement is large enough compared to ϵ, δ_P and δ_L.

The enlarged semantics has received much attention from computer-scientists over the last five years [3, 8, 16, 17, 20, 21, 23, 31, ...]. In the rest of this paper, we develop two of these results.

4.2 Robust safety checking

Clearly enough, the enlarged semantics[4] does add new behaviours since it may allow a transition at time $1+\delta$ when this transition is only allowed at time 1 in the classical semantics. But what we really mean is whether it allows *really new* behaviours, and not just behaviours with slightly different clock values. Hence the first natural question is whether new (and harmful) behaviours are added. This is our first definition of robustness:

Definition 3. *A timed automaton $\mathcal{A}$* robustly satisfies a property ϕ *if there exist positive values for ϵ and δ for which all the runs of $[\![\mathcal{A}]\!]_{\delta,\epsilon}$ satisfy ϕ. Automaton $\mathcal{A}$ is said to be* robust w.r.t. a set of properties $\mathcal{S}$ *if $\mathcal{A}$ robustly satisfies any formula of $\mathcal{S}$ it satisfies in the classical sense.*

The example of Fig. 7 is a case where enlargement does add extra behaviours [30, 21]. It can be seen as a simplified version of the MPEG video encoder developed in [1]: the system receives one new frame every two time units, and encodes the frame within the next two time units. In an idealized world, the system would run perfectly. Unfortunately, it can be checked that the rightmost state (which, in our example, would correspond to losing a frame) can be reached from the initial state for any positive enlargement. Actually, the set of reachable clock valuations in each state is depicted on Fig. 8: the left-hand side of the figure displays the reachable configurations in the classical semantics. This can easily be computed on the region automaton. The right-hand side of the figure shows the configurations that are reachable *for any positive enlargement*, however small it may be: this is because the small imprecision on the values of the clocks can be accumulated by cycling in states ℓ_1 and ℓ_2. Of course, the smaller the enlargement, the longer the execution will have to loop in order to accumulate sufficiently many imprecision and reach state ℓ_3.

[4] This term encompasses both enlargement of guards and of the rate of the clocks.

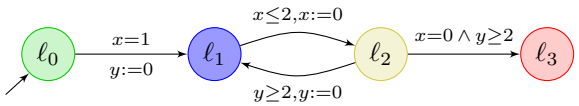

Fig. 7. A non-robust timed automaton

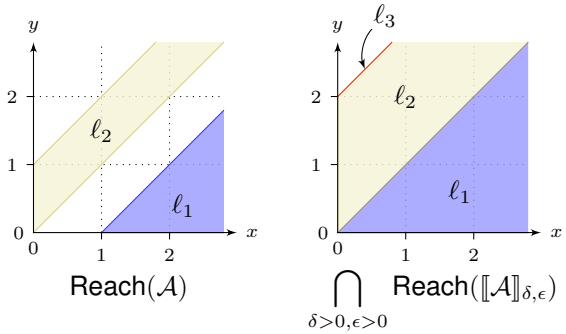

Fig. 8. Reachable configurations

While this example shows that any positive enlargement may come with extra reachable states, it also indicates how cycles in the automaton play an important role, by providing a way of growing the small imprecision into a one-time-unit gap, and thereby reaching new states. This was formalized in [30, 21], where an algorithm is given to compute the set of configurations that can be reached under any positive enlargement. More precisely, the algorithm computes (under some restrictions that we discuss later) the set $\bigcap_{\delta>0,\epsilon>0} \mathsf{Reach}([\![\mathcal{A}]\!]_{\delta,\epsilon})$, The idea of the algorithm is to extend the region automaton with extra transitions from region (ℓ, r) to region (ℓ, r') whenever the following conditions are met:

- (ℓ, r') belongs to a cycle in the region automaton of $\mathcal{A}$;
- r is a neighbouring region of r' (in other terms, the intersection of the closures of r and r' is non-empty).

Intuitively, the second condition implies that if region (ℓ, r') is reachable, then, because of the enlargement, a "border" of region (ℓ, r) can also be reached. Then the first condition provides a way of really getting into the region. In the end, the following result holds:

Theorem 4 ([30, 21])**.** *Robust safety is decidable, and* PSPACE*-complete (for a restricted class of timed automata).*

As an additional result, it was proved that the algorithm is also correct when only one kind of imprecision is involved; more precisely, the following equalities hold:

$$\bigcap_{\delta>0,\epsilon>0} \mathsf{Reach}([\![\mathcal{A}]\!]_{\delta,\epsilon}) = \bigcap_{\epsilon>0} \mathsf{Reach}([\![\mathcal{A}]\!]_{0,\epsilon}) \quad (1)$$
$$= \bigcap_{\delta>0} \mathsf{Reach}([\![\mathcal{A}]\!]_{\delta,0}).$$

The above approach can easily be extended to LTL *robust* model checking: roughly, it suffices to keep track of the sequence of states visited along cycles which are involved in the addition of extra transitions in the region automaton.

Theorem 5 ([16])**.** *LTL robust model-checking is decidable, and* PSPACE*-complete (under some restrictions).*

Theorems 4 and 5 only hold for a subclass of timed automata: besides boundedness (all clocks must be bounded) and closure (strict inequalities are forbidden), the theorem only applies to timed automata whose region automaton have no *weak cycles*, *i.e.*, cycles along which some clock is not reset. Fig. 9 displays an example of a timed automaton which does not satisfy this assumption. It can be checked on this automaton that Equation (1) does not hold: the rightmost state is reachable when enlarging the clock constraints by a positive δ, but clock drifts alone will have no effect.

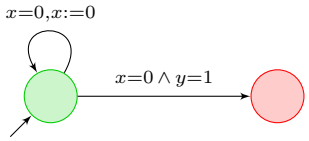

Fig. 9. A timed automaton with a weak cycle

4.3 Measuring robustness

A more refined approach consists in *measuring* how much the behaviour of a timed automaton may change when introducing imprecision. Following ideas from [26, 32], we say that two states are τ-bisimilar, for $\tau \in \mathbb{R}_{\geq 0}$, whenever the behaviours from one of them can be mimicked from the other one (and conversely), possibly with slightly different delays; the difference between delays can be at most τ: a delay of 1.5 time units from one state can be matched by a delay in $[1.5-\tau, 1.5+\tau]$ from the other state, while action transitions must be matched exactly. τ-bisimulations provide a way of quantifying *how close* two timed automata are to one another: we let $d(\mathcal{A}, \mathcal{B}) = \tau_0$ when τ_0 is the lower bound of the τ's for which the initial states of both automata are τ-bisimilar (and it is $+\infty$ is no such τ exists). Formally, this defines a pseudo-metric (because the distance can be zero between two different—even non-bisimilar—timed automata). Now, robustness can be defined as follows [15]:

Definition 6. *A timed automaton $\mathcal{A}$ is* robust *if there exist positive values for ϵ and δ under which $[\![\mathcal{A}]\!]_{\delta,\epsilon}$ and $[\![\mathcal{A}]\!]$ are τ-bisimilar for a finite τ.*

It must be noticed that this definition of robustness is not parametrized by a property. Moreover, if a timed automaton is robust (in the sense of Definition 6), then it is robust

w.r.t. any ω-regular formula (in the sense of Definition 3). More interestingly, this second definition opens the way to a quantitative notion of robustness: a timed automaton has robustness α when the distance between $[\![\mathcal{A}]\!]$ and $[\![\mathcal{A}]\!]_{\delta,\epsilon}$ is at most $\alpha \cdot \max(\delta, \epsilon)$. While there is currently no known procedure for deciding (or measuring) this notion of robustness, the following interesting result has been established (for guard enlargement only) [15]:

Theorem 7. *Given a timed automaton $\mathcal{A}$, we can effectively compute another timed automaton $\mathcal{A}'$ which is robust (against guard enlargement) and at pseudo-distance 0 from $\mathcal{A}$.*

In other terms, any timed automaton can be made robust. Basically, the idea behind this result is to consider the region automaton (seen as a timed automaton), and to strengthen the timing constraints in order to prevent errors to accumulate. This theoretically comes with an exponential blow-up, which we currently don't know if it can be avoided. On our example of Fig. 7, we can easily come up with a robust bisimilar timed automaton of the same size, which is depicted on Fig. 10.

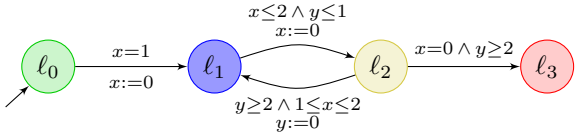

Fig. 10. A robust timed automaton

5 Conclusions and perspectives

We have reviewed several approaches to make the semantics automata more realistic: while being very convenient for modelling and reasoning about real-time systems, timed automata are governed by a mathematical semantics that may not reflect the real behaviour of a physical system: it may contain unrealistic behaviours on the one hand, which involve the convergence of time or isolated executions that are very unlikely; for a non-safety-critical system, you will not want to take these weird behaviours into account. On the other hand, the semantics of timed automata also assumes perfect measurement of time and does not consider the fact that clock values have finite precision. These small imprecisions may accumulate along the executions, and lead to unpredicted behaviours.

While the problem has been identified long ago and the computer-science community has come with various solutions, robustness issues in timed automata remain a very active research topic: timed automata are now rather well-understood under their classical semantics, but the problem of their robustness remains quite open: first, the very definition of what *robustness* encompasses is still not precisely settled; of particular interest is the link between robustness and implementability. Second, in the different approaches that have been defined, several important questions remain open, in particular in the development of efficient algorithms and tools.

A very interesting avenue for further research is to extend robust verification to robust controller synthesis, where the aim is to automatically design a controller for a timed automaton to satisfy a given property. There, robustness has many more facets, including imperfect information or the evaluation of the precision needed for the controller to really achieve its role. Robustness in weighted timed automata (where extra quantities can be measured besides time, such as energy consumption) is also a very promising extension [29], as it may in some cases have better decidability properties than the classical semantics.

Acknowledgements. I thank Patricia Bouyer-Decitre, Ocan Sakur, and the anonymous referees for helpful comments on earlier versions of this paper.

References

[1] T. Abdellatif, J. Combaz, and J. Sifakis. Model-based implementation of real-time applications. In EMSOFT'10, p. 229–238. ACM Press, October 2010.

[2] P. A. Abdulla, P. Krčál, and W. Yi. Sampled semantics of timed automata. *Logicical Methods in Computer Science*, 6(3), September 2010.

[3] S. Akshay, B. Bollig, P. Gastin, M. Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. In CONCUR'08, LNCS 5201, p. 82–97. Springer, August 2008.

[4] K. Altisen and S. Tripakis. Implementation of timed automata: An issue of semantics or modeling? In FORMATS'05, LNCS 3829, p. 273–288. Springer, September 2005.

[5] R. Alur. Timed automata. In CAV'99, LNCS 1633, p. 8–22. Springer, July 1999.

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

[7] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Information and Computation*, 104(1):35–77, May 1993.

[8] R. Alur, S. La Torre, and P. Madhusudan. Perturbed timed automata. In HSCC'05, LNCS 3414, p. 70–85. Springer, March 2005.

[9] Ch. Baier, N. Bertrand, P. Bouyer, Th. Brihaye, and M. Größer. Probabilistic and topological semantics for timed automata. In FSTTCS'07, LNCS 4855, p. 179–191. Springer, December 2007.

[10] Ch. Baier, N. Bertrand, P. Bouyer, Th. Brihaye, and M. Größer. Almost-sure model checking of infinite paths in one-clock timed automata. In LICS'08. IEEE Comp. Soc. Press, June 2008.

[11] G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal, 2005.

[12] J. Bengtsson and W. Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets*, LNCS 2098, p. 87–124. Springer, 2004.

[13] N. Bertrand, P. Bouyer, Th. Brihaye, and N. Markey. Quantitative model-checking of one-clock timed automata under probabilistic semantics. In QEST'08, p. 55–64. IEEE Comp. Soc. Press, September 2008.

[14] D. Beyer, C. Lewerentz, and A. Noack. Rabbit: A tool for BDD-based verification of real-time systems. In CAV'03, LNCS 2725, p. 122–125. Springer, July 2003.

[15] P. Bouyer, K. G. Larsen, N. Markey, O. Sankur, and C. Thrane. Timed automata can always be made implementable, 2011. Submitted.

[16] P. Bouyer, N. Markey, and P.-A. Reynier. Robust model-checking of linear-time properties in timed automata. In LATIN'06, LNCS 3887, p. 238–249. Springer, March 2006.

[17] P. Bouyer, N. Markey, and P.-A. Reynier. Robust analysis of timed automata via channel machines. In FoSSaCS'08, LNCS 4962, p. 157–171. Springer, March-April 2008.

[18] J. A. Brzozowski and C.-J. H. Seger. Advances in asynchronous circuit theory part II: Bounded inertial delay models, MOS circuits, design techniques. *EATCS Bulletin*, 43:199–263, February 1991.

[19] F. Cassez, T. A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In HSCC'02, LNCS 2289, p. 134–148. Springer, March 2002.

[20] C. Daws and P. Kordy. Symbolic robustness analysis of timed automata. In FORMATS'06, LNCS 4202, p. 143–155. Springer, September 2006.

[21] M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1-3):45–84, December 2008.

[22] M. De Wulf, L. Doyen, and J.-F. Raskin. Almost ASAP semantics: From timed models to timed implementations. *Formal Aspects of Computing*, 17(3):319–341, 2005.

[23] C. Dima. Dynamical properties of timed automata revisited. In FORMATS'07, LNCS 4763, p. 130–146. Springer, October 2007.

[24] V. Gupta, T. A. Henzinger, and R. Jagadeesan. Robust timed automata. In HART'97, LNCS 1201, p. 331–345. Springer, March 1997.

[25] K. Havelund, A. Skou, K. G. Larsen, and K. Lund. Formal modelling and analysis of an audio/video protocol: An industrial case study using Uppaal. In RTSS'97, p. 2–13. IEEE Comp. Soc. Press, December 1997.

[26] T. A. Henzinger, R. Majumdar, and V. S. Prabhu. Quantifying similarities between timed systems. In FORMATS'05, LNCS 3829, p. 226–241. Springer, September 2005.

[27] T. A. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In REX'91, LNCS 600, p. 226–251. Springer, 1992.

[28] K. J. Kristoffersen, F. Laroussinie, K. G. Larsen, P. Pettersson, and W. Yi. A compositional proof of a real-time mutual exclusion protocol. In TAPSOFT'97, LNCS 1214, p. 565–579. Springer, April 1997.

[29] N. Markey and P.-A. Reynier. 1-clock priced timed automata with energy constraints and imperfect information. In GASICS'10, September 2010.

[30] A. Puri. Dynamical properties of timed automata. In FTRTFT'98, LNCS 1486, p. 210–227. Springer, September 1998.

[31] M. Swaminathan, M. Fränzle, and J.-P. Katoen. The surprising robustness of (closed) timed automata against clock-drift. In IFIPTCS'08, IFIP Conference Proceedings 273, p. 537–553. Springer, September 2008.

[32] C. Thrane, U. Fahrenberg, and K. G. Larsen. Quantitative analysis of weighted transition systems. *Journal of Logic and Algebraic Programming*, 79(7):689–703, October 2010.

[33] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1(1-2):123–133, October 1997.