Robustness in Timed Automata*

Patricia Bouyer[1], Nicolas Markey[1], and Ocan Sankur[1,2]

[1] LSV, CNRS & ENS Cachan, France
[2] Université Libre de Bruxelles, Belgium
{bouyer,markey,sankur}@lsv.ens-cachan.fr

Abstract. In this paper we survey several approaches to the robustness of timed automata, that is, the ability of a system to resist to slight perturbations or errors. We will concentrate on robustness against timing errors which can be due to measuring errors, imprecise clocks, and unexpected runtime behaviors such as execution times that are longer or shorter than expected.

We consider the perturbation model of guard enlargement and formulate several robust verification problems that have been studied recently, including robustness analysis, robust implementation, and robust control.

1 Introduction

Timed automata are an extension of finite automata with analog clock variables, and a convenient formalism for modelling real-time systems [AD94]. The theory allows the model-checking against a rich set of properties, including temporal logics and their timed extensions. These algorithms have been implemented in several tools (*e.g.* Uppaal, IF2.0) and applied to many case studies.

The idea behind timed automata is to consider clocks with continuous values. The resulting abstraction is appealing in terms of modelling and allows for efficient symbolic algorithms. However, this formalism only allows validating designs under the assumptions that the clock variables are perfectly continuous, their values can be measured instantly and exactly, etc. Because concrete implementations cannot always be assumed to satisfy these assumptions, there is a need to study verification methodologies for timed automata where these idealistic assumptions are relaxed. A comparison of the semantics of timed automata and real-world systems is given in Table 1.

In this paper we survey several approaches to the robustness of timed automata. What we mean by robustness is the ability of a system to resist to slight perturbations or errors. We will concentrate on robustness against timing errors which can be due to measuring errors, imprecise clocks, and unexpected runtime behaviors such as execution times that are longer or shorter than expected. In particular, robustness in timed automata consists in relaxing the idealistic assumptions behind its semantics.

* Partly supported by ANR project ImpRo (ANR-10-BLAN-0317), by ERC Starting grants EQualIS (FP7-308087) and inVEST (FP7-279499), and by European project Cassting (FP7-601148).

Table 1. A comparison between the abstract semantics of timed automata and real-world systems

	Timed automata	**Real-world system**
Frequency	Infinite	Finite
Precision	Arbitrary	Bounded
Synchronization	Perfect	Delayed

The benefits of studying robustness for timed systems are twofold. First, robustness is a desired property of real-time systems since it requires the system to tolerate errors upto a given bound. Hence, the properties proven "robustly" will hold in the system even when the environment assumptions change slightly. This property is crucial for critical systems.

Second, a "robust theory" of timed automata will reconcile abstract models and real-world systems, in the sense that the behaviors of the design would more closely correspond to the behaviors of a real system. As a consequence, the analysis made on the abstract models can be transferred to the real-world system. Robustness is thus closely related to implementability.

Robustness in timed automata was first considered in [GHJ97] where a topological semantics was defined with the idea of excluding isolated behaviors, but the emphasis was rather on obtaining the decidability of some hard verification problems. Timed automata with clock drifts were considered in [Pur98, Pur00], where algorithms for safety analysis in presence of clock drifts were given. This work triggered a series of results on robust model-checking timed automata with guard enlargement and clock drifts, e.g. [DDMR08]. See Section 2.4 for more on related work.

In this paper, we formulate several robustness problems in timed automata by considering the perturbation models of guard enlargement, which models time measurement errors and jitter. These perturbations can both be considered as syntactic transformations, or as reactive semantics which we model as games. In all cases, we consider an unknown parameter which expresses the magnitude of the perturbations. We formulate several robust verification problems that have been studied recently, including robustness analysis, robust implementation, and robust control.

2 Definition

2.1 Timed Automata

Given a finite set of clocks $\mathcal{C}$, we call *valuations* the elements of $\mathbb{R}_{\geq 0}^{\mathcal{C}}$. For a subset $R \subseteq \mathcal{C}$ and a valuation ν, $\nu[R \leftarrow 0]$ is the valuation defined by $\nu[R \leftarrow 0](x) = \nu(x)$ for $x \in \mathcal{C} \setminus R$ and $\nu[R \leftarrow 0](x) = 0$ for $x \in R$. Given $d \in \mathbb{R}_{\geq 0}$ and a valuation ν, the valuation $\nu + d$ is defined by $(\nu + d)(x) = \nu(x) + d$ for all $x \in \mathcal{C}$. We extend these operations to sets of valuations in the obvious way. We write $\mathbf{0}$ for the valuation that assigns 0 to every clock.

An atomic clock constraint is a formula of the form $k \preceq x \preceq' l$ where $x \in \mathcal{C}$, $k, l \in \mathbb{Z} \cup \{-\infty, \infty\}$ and $\preceq, \preceq' \in \{<, \leq\}$. A *guard* is a conjunction of atomic clock constraints. A valuation ν satisfies a guard g, denoted $\nu \models g$, if all constraints are satisfied when each $x \in \mathcal{C}$ is replaced with $\nu(x)$. We write $\Phi_{\mathcal{C}}$ for the set of guards built on $\mathcal{C}$.

Definition 1. *A* timed automaton $\mathcal{A}$ *is a tuple* $(\mathcal{L}, \mathcal{C}, \ell_0, E)$*, where* $\mathcal{L}$ *is a finite set of locations,* $\mathcal{C}$ *is a finite set of clocks,* $E \subseteq \mathcal{L} \times \Phi_{\mathcal{C}} \times 2^{\mathcal{C}} \times \mathcal{L}$ *is a set of edges, and* $\ell_0 \in \mathcal{L}$ *is the initial location. An edge* $e = (\ell, g, R, \ell')$ *is also written as* $\ell \xrightarrow{g,R} \ell'$.

A closed timed automaton is a timed automaton whose all guards have closed atomic clock constraints.

A configuration of a timed automaton is a pair (ℓ, ν) where $\ell \in \mathcal{L}$ is the current location and ν is a clock valuation, assigning a value to each clock. At first sight it might look natural to assume that clocks take integer values, as timed automata will be used to model digital systems (using a cycle of the CPU as the unit of time). This would have several drawbacks: first, our results would only be valid for this given frequency of the CPU; more importantly, when the model consists of several components, a discrete semantics forces the components to have synchronoous transitions; finally, this would not be convenient for modelling the external environment, which should not be restricted to have finite frequency. We refer to [BS91, Alu91] for a longer discussion on this question.

In the rest of this paper, following [AD94], we thus assume that clocks take real values. The set of configurations (or states) of a timed automaton $\mathcal{A} = (\mathcal{L}, \mathcal{C}, \ell_0, E)$ is then $\mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}}$. A *run* of $\mathcal{A}$ is a sequence $s_1 e_1 s_2 e_2 \ldots$ where $s_i \in \mathcal{L} \times \mathbb{R}_{\geq 0}^{\mathcal{C}}$, and writing $s_i = (\ell_i, \nu_i)$, either $e_i \in \mathbb{R}_{>0}$, in which case $s_{i+1} = (\ell_i, \nu_i + e_i)$, or $e_i = (\ell_i, g, R, \ell') \in E$, in which case $s_{i+1} = (\ell', \nu[R \leftarrow 0])$. We denote by $\mathsf{state}_i(\rho)$ the i-th state of any run ρ, by $\mathsf{first}(\rho)$ its first state, and, if ρ is finite, $\mathsf{last}(\rho)$ denotes its last state of ρ.

Example 1. Fig. 1 displays an example of a timed automaton, representing the behavior of a computer mouse: it uses one clock to measure the delays between pushes on the buttons, and translates them into single- or double-clicks, depending on these delays.

2.2 Timed-Automata-Based Model Checking

Theorem 1 ([AD94]). *Reachability and Büchi properties can be decided in polynomial space in timed automata. Both problems are PSPACE-complete.*

The proof of this important theorem relies on the construction of the *region abstraction*: intuitively, if two clock valuations are *close enough*, in the sense that they give to each clock the same integral part and define the same order w.r.t. their fractional parts, then they will give rise to similar behaviours: the same transitions are available, and the valuations reached by letting time elapse can still be made *close enough*. After noticing that we do not need to care

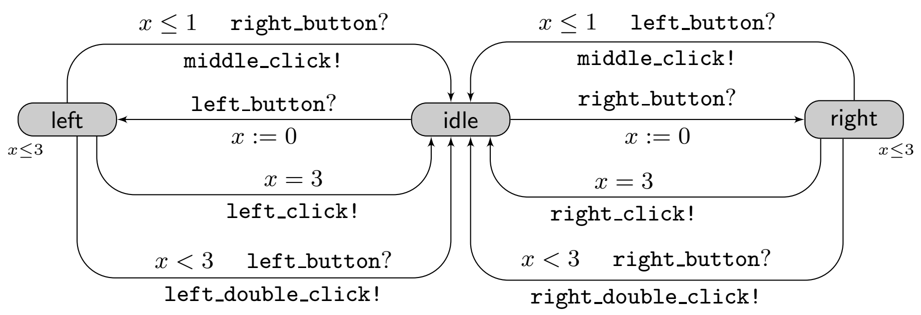


Fig. 1. *Clicks* and *double-clicks* of a mouse

about the value of a clock when it is larger than the maximal integer appearing in the automaton, this gives rise to an equivalence relation (called *region equivalence*) having finite index. Quotienting the automaton with the region equivalence provides us with the *region automaton*, which can be used to verify ω-regular properties. Though the region automaton has size exponential, the algorithm for checking ω-regular properties can be made to run on-the-fly, thus using only polynomial space.

Symbolic approaches based on coarser equivalence relations and adequate data structures have been developed, which make the verification of timed automata usable in practice. Several tools implement optimised versions of these algorithms (*e.g.* Uppaal [BDL$^+$06], Kronos [BDM$^+$98], ...), and have been successfully applied on industrial case studies.

Example 2. We illustrate the region equivalence on the two-clock timed automaton depicted on Fig. 2. The maximal constant appearing in this automaton is 2, so that the set of classes of the region-equivalence is as depicted on Fig. 3. Its region automaton is depicted on Fig. 4, in which we can observe that the rightmost location of the automaton is not reachable.

2.3 Discussion on the Semantics: Are We Doing the Right Job?

We opted for the dense-time semantics to overcome the drawbacks of discrete-time, but is dense-time *really* better?

Obviously, the continuous-time semantics can be used to mimick the discrete-time semantics (by forcing transitions to occur only at integer dates), which in some sense shows that it is not worse. But this semantics is mostly appropriate for abstract designs and high-level analysis of timed systems, and suffers from several inaccuracies when used for more concrete analyses. In particular:

– by assuming infinite frequency (*i.e.*, zero-delay transitions), infinite precision, and immediate communications between components, the continuous-time semantics is not adequate for implementation. Indeed, in practice, clocks do

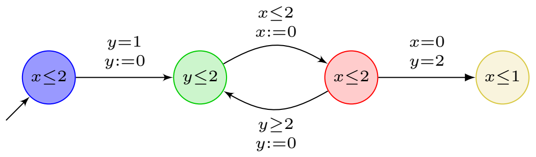


Fig. 2. A two-clock timed automaton $\mathcal{A}$

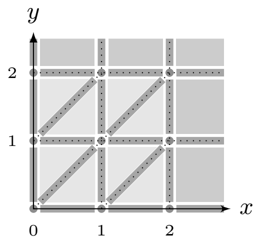


Fig. 3. Region equivalence for $\mathcal{A}$

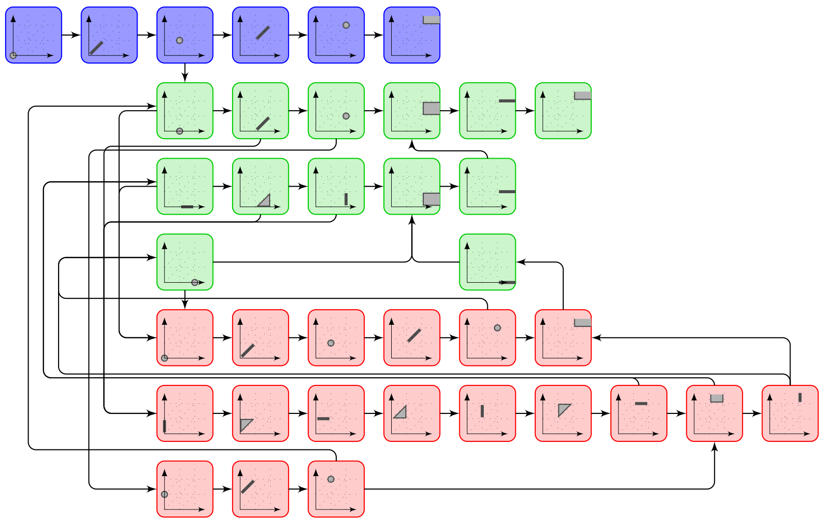


Fig. 4. The (reachable part of the) region automaton for $\mathcal{A}$

not all run at the exact same speed. Moreover, since computers are digital, the value of the clock is updated periodically. Hence a clock constraint might be evaluated to true at a time when it is not true anymore. Both phenomena give rise to emergent behaviors that are not taken into account during verification.

– because of the infinite frequency and precision, timed automata may exhibit non-realistic behaviors, especially convergence phenomena. The best-known example of convergence phenomena are *Zeno runs*, which are infinite runs (in terms of their number of transitions) having finite duration. More complex convergence phenomena can be hidden, which may be difficult to detect: such unrealistic behaviors can nevertheless be used by a standard verification process, witnessing in an inappropriate way the truth of some property.

Example 3. One realizes that any infinite behavior in the automaton of Fig. 2 is such that the value of clock x when entering the green location is non-decreasing and bounded by 2, hence converging (whereas globally time diverges).

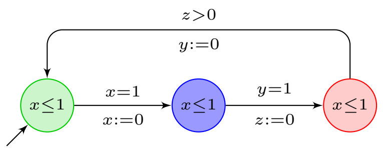


Fig. 5. An automaton with a (non-Zeno) convergence phenomenon

Another example (from [CHR02]) where such a phenomenon occurs is depicted at Fig. 5: because of the constraint on clock z, the time elapsed in the right-most state is always positive, and it can easily be proven that the value of x when entering the left-most state is the accumulated delay spent in the right-most state. Hence infinite runs exist in this automaton, but the delays in the right-most state will have to be arbitrarily small.

The remarks above raise several important questions on the real impact of standard verification processes applied to timed systems. First, given a model that has been proven to satisfy some properties, is it possible to implement it while preserving these properties? Also, to what extent does the result of the verification transfer to real-world systems?

This advocates for the development of a theory of robustness for timed systems. An important issue is to understand what is the real system behind the mathematical model, and also which implementation of a system we have in mind, if any. Also, it should be clear under which model of perturbations the theory applies (perturbations can arise from multiple sources). Finally, we should provide the user with methodologies to develop correct real-world (or implemented) systems.

The aim of this paper is to present a possible approach to the robustness of timed systems. It consists in developping refined techniques to check properties of timed automata in a robust setting: we do not want to model the execution platform explicitly, nor do we want to involve extra formalisms. Our algorithms will take timed automata as inputs, and they will verify properties under a semantics that takes timing imprecisions into account.

2.4 Related Work

An early attempt in defining a robustness notion for timed automata is [GHJ97], where a topological definition is introduced with the hope that the language inclusion problem could become decidable under this semantics; but the undecidability was later shown to hold even in the robust setting [HR00].

Robustness analysis algorithms in presence of parameterized guard enlargement have been studied extensively starting with safety properties in [DDMR08]. These results will be summarized in Section 3.

Following [Pur00, DDMR08], clock drifts have been studied for general timed automata (rather than closed timed automata) in [Dim07, SFK09]. These have also been considered in the context of distributed timed automata in [ABG$^+$08] where one distinguishes existential languages, that is, those behaviors that can be achieved under some evolution of the clocks, and universal languages, which is, the behaviors that are present whatever the drifts are. The emptiness of the former is decidable, while the latter is shown undecidable.

The semantics under an unknown regular sampling of time was considered in [CHR02, KP05, AKY10]. Reachability under unknown sampling rate was shown to be undecidable in [CHR02], while safety is decidable [KP05]. Here, the goal is to synthesize a sampling parameter under which a reachability or safety objective holds, or the untimed language is preserved. This problem is related to the implementability of timed automata with discrete clocks. Such discrete time approaches are interesting when the model is a low-level designs, that is, timed automata where one time unit is comparable to the clock period in the target hardware platform.

The work [AT05] considers encoding in timed automata models several perturbations including clock drifts, enlargement, sampling. The resulting approach allows analyzing systems with fixed perturbation parameters, and often with a fixed granularity (e.g. for clock drifts), but not applying parameter synthesis. Many cases from the literature do in fact use such encodings since robustness algorithms are not yet available in verification tools.

The robustness of timed models against decreases in execution times have been considered in [ACS10] using simulations, and an application in a multimedia system was discussed.

3 Robustness Analysis

Standard analysis of the timed-automaton model might not be satisfactory if we are interested in the implementation of this model. In this section, we describe a framework in which one implements directly the system which is designed. Unfortunately, as mentioned earlier, the implemented system might not behave precisely according to the model, since imprecisions might occur while executing. The idea is then to understand how one should adapt the analysis process in order to capture and analyze the real behavior of the implemented system.

Robustness analysis. We write $\mathcal{A}_\delta$ for the timed automaton obtained from $\mathcal{A}$ by enlarging its guards by a parameter δ: that is, every upper-bounded constraint $x \leq b$ or $x < b$ is replaced by $x \leq b+\delta$, and every lower-bounded constraint $x \geq a$ or $x > a$ is replaced by $x \geq a-\delta$. Obviously every behavior in $\mathcal{A}$ is also a behavior in $\mathcal{A}_\delta$. Also quite obviously, there are behaviors in $\mathcal{A}_\delta$ that are absent in $\mathcal{A}$ (since some delays cannot be exactly matched), but maybe more surprisingly, there are *qualitative behaviors* that can be found in any $\mathcal{A}_\delta$ (however small $\delta > 0$ may be) but that cannot be found in $\mathcal{A}$. In particular, a simple (untimed safety) property proven in $\mathcal{A}$ can be violated in any $\mathcal{A}_\delta$, however small $\delta > 0$ may be. We illustrate this fact in the example below.

Example 4. Consider the automaton of Fig. 2. When enlarged by δ, the first transition of the loop can be anticipated by δ, hence taken when $x = 2 - \delta$, while the second transition of the loop can be delayed until $y = 2 + \delta$. Starting from a configuration where $x = 1$ and $y = 0$, we end up with $x = 1 - 2\delta$ and $y = 0$ after applying these two transitions. This can be repeated, until we come back to the left-most state of the loop with x being close to zero, which will then give access to the right-most state of the automaton.

The idea of *robustness analysis* (or *robust model-checking*) is to verify the correctness of the enlarged timed automaton. We define it formally below.

Definition 2 (Robustness analysis). *Given a linear-time property φ[1] and a timed automaton $\mathcal{A}$, decide whether there is some $\delta_0 > 0$ such that for all $\delta \in [0, \delta_0]$, all executions in $\mathcal{A}_\delta$ satisfy property φ. If this is the case, we say that $\mathcal{A}$* robustly satisfies *φ, and that δ_0 witnesses this robust satisfaction.*

Why robustness analysis? The idea behind robustness analysis is that the enlargement by δ of all the guards should capture the imprecisions of the executed (or implemented) system. The accurateness of the approach then relies on the assumption that for some $\delta > 0$, $\mathcal{A}_\delta$ over-approximates the real behavior of the system (with δ depending on the characteristics of the processor). To support this assumption, program semantics are given in [DDR05] and in [SBM11]. They take into account various delay parameters of a processor, and are shown to satisfy this assumption.

This yields the following methodology for the development of correct implemented real-time systems.

1. Design $\mathcal{A}$;
2. Verify the robust satisfaction of φ by $\mathcal{A}$, that is verify $\mathcal{A}_\delta$, where δ is a parameter;
3. Implement $\mathcal{A}$: its correctness will be implied by that of $\mathcal{A}_\delta$.

Let us comment the robust satisfaction. It is easy to get convinced that if δ_0 witnesses the robust satisfaction of φ by $\mathcal{A}$, then so does any $0 < \delta \leq \delta_0$. This "faster-is-better" property is a desirable property when studying implementability: if a system is correct when implemented on a processor, then it will remain correct on a faster processor.

Decidability and complexity results. When $\delta_0 > 0$ is fixed, verifying that $\mathcal{A}_{\delta_0}$ satisfies φ can be done using standard model-checking techniques. However, robust model-checking is rather interested in a paramaterized analysis, and in the synthesis of a positive value for δ_0. In the theorem below, by timed automata with *progress cycles*, we refer to timed automata whose all cycles reset each clock at least once.

[1] That can be any ω-regular or LTL property, or even some timed property.

Theorem 2 ([DDMR08, BMR06, BMS11, BMR08]). *Robust model-checking of safety, Büchi, LTL properties for closed timed automata is PSPACE-complete. Robust model-checking of coFlatMTL (a fragment of MTL) for closed timed automata with progress cycles is EXPSPACE-complete.*

One can notice that the theoretical complexity of robust model-checking is the same as that of standard model-checking, which is quite surprising but very positive. Unfortunately no efficient symbolic algorithms have been developed so far, and a big effort is required in this direction, so that the practical impact of these results is consolidated.

4 Robust Implementation

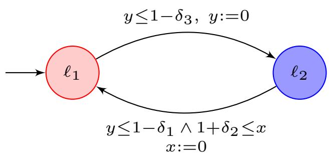
In this section, we will explore techniques that allow generating implementations that are robust to perturbations by construction. These techniques also ensure that some desired properties of the initial model are preserved in the implementation. Hence, this approach has the advantage of allowing the system designer to concentrate on the initial model, and use existing tools to prove correctnes, while leaving the implementation effort to a computer.

More precisely, we describe results that allow one to automatically transform a given timed automaton into a robust one, while preserving at least its time-abstract behaviors. Thus, our target implementation formalism is again timed automata subject to guard enlargement. We will present two ways of achieving robust implementation, each corresponding to a different point of view on the models.

4.1 Shrinking

Assume that we are interested in strictly respecting the timing constraints described by the guards of given timed automata, that is, we do not tolerate any enlargement of the guards. To motivate this assumption, consider the following scenario. We are designing a system that communicates with another component which only receives signals in a given time interval, say $x \in [l, u]$; any signal sent outside this interval is lost. While an abstract model of the system assuming instantaneous communication might simply require the signals to be sent inside the same interval, that is, $x \in [l, u]$, the system would not be correct under timing imprecisions, say, due to communication delays. A natural idea is to obtain a correct implementation by *shrinking* the guard into $x \in [l+\delta, u-\delta]$, where $\delta > 0$ is an appropriate parameter. In fact, assuming that timing imprecisions are modelled by an enlargement of Δ, we have $[l+\delta-\Delta, u-\delta+\Delta] \subseteq [l, u]$. In other terms, the behaviors induced by the implementation of the shrunk guard is included in the behaviors of the original abstract model.

While shrinking can offer an easy way to implement timed automata while preserving behaviors, it can also remove some significant behaviors from the model, such as liveness. In fact, some timed automata become blocking under the slightest shrinking of their guards. Such an example is given in Fig. 6.



$y \leq 1-\delta_3,\ y:=0$
ℓ_1
ℓ_2
$y \leq 1-\delta_1 \wedge 1+\delta_2 \leq x$
$x:=0$

Fig. 6. A shrunk timed automaton that is blocking whenever $\delta_2 > 0$ or $\delta_3 > 0$. To see this, consider any infinite execution and let $d_1, d_3, \ldots$ denote the delays at location l_1, and $d_2, d_4, \ldots$ those at l_2. One can show that $1 \leq d_{2i-1} + d_{2i}$ for all $i \geq 1$, which means that time diverges, but also $\delta_2 + \delta_3 \leq d_{2i+2} - d_{2i}$ and $d_{2i} \leq 1$. The latter means that the sequence $(d_{2i})_i$ increases at least by $\delta_2 + \delta_3$ at each step and is bounded above by 1, which is possible only when $\delta_2 = \delta_3 = 0$. Note that even if $\delta_2 = \delta_3 = 0$, non-blockingness requires consecutive delays to be equal, which is not realistic for digital systems.

We are interested in obtaining an implementation by shrinking the guards of a timed automaton such that some desired properties are preserved in the resulting model. We will consider a possibly different shrinking parameter for each atomic guard in order to increase the chance of success. Formally, given a timed automaton $\mathcal{A}$, let I denote the set of its atomic guards. Each atomic guard will be shrunk by $k_i\delta$ where $\delta > 0$ is a uniform parameter, and k_i's are possibly different natural numbers. Notice that any vector of rational numbers can be written as $\boldsymbol{k}\delta$ where $\boldsymbol{k}$ is the vector of k_i's. The resulting shrunk timed automaton is denoted $\mathcal{A}_{-\boldsymbol{k}\delta}$ (Note that this is simply enlargement by a negative amount). The *shrinkability* problem asks for property preservation under possible shrinking; its two variants are defined formally as follows.

A timed automaton is said to be *non-blocking* if from any state some discrete transition is enabled after some time delay.

Definition 3 (Shrinkability). *Given a timed automaton $\mathcal{A}$, decide whether for some $\delta_0 > 0$, and some positive integer vector $\boldsymbol{k}$,*

1. *$\mathcal{A}_{-\boldsymbol{k}\delta}$ is non-blocking,*
2. *and $\mathcal{A}_{-\boldsymbol{k}\delta}$ time-abstract-simulates $\mathcal{A}$,*

for all $\delta \in [0, \delta_0]$.

We also consider variants of the shrinkability problem. *Non-blocking-shrinkability* only asks for condition 1 to be satisfied in Definition 3, while *simulation-shrinkability* asks only for condition 2. The latter can be refined by only requiring the shrunk timed automaton to time-abstract simulate a given finite automaton $\mathcal{F} \sqsubseteq_{\text{t.a.}} \mathcal{A}$. In this case, only some part $\mathcal{F}$ of the time-abstract behavior of $\mathcal{A}$ is preserved in $\mathcal{A}_{-\boldsymbol{k}\delta}$.

Theorem 3 ([SBM11]). *For closed non-blocking timed automata, non-blocking-shrinkability is decidable in polynomial space, and in NP if the maximum degree of the locations is fixed.*

For closed timed automata, under some technical assumptions, simulation-shrinkability is decidable in pseudopolynomial time in the sizes of $\mathcal{A}$ and $\mathcal{F}$.

Algorithms of Theorem 3 allow computing the parameters $\boldsymbol{k}$ and the maximum $\delta > 0$. This enables the following design methodology which we sketched at the beginning of this section.

1. Design and verify timed automaton $\mathcal{A}$;
2. Apply shrinkability, which yields $\mathcal{A}_{-\boldsymbol{k}\delta}$;
3. Implement $\mathcal{A}_{-\boldsymbol{k}\delta}$ which is a safe refinement even under imprecisions: $\mathcal{A}_{-\boldsymbol{k}\delta+\Delta} \sqsubseteq \mathcal{A}$, and moreover it is non-blocking and it preserves all time-abstract behaviors of $\mathcal{A}$.

Tool support. The simulation-shrinkability algorithm described above was implemented in a software tool called Shrinktech [San13]. The tool is integrated with the Kronos model-checker: it takes as input (network of) timed automata and checks for shrinkability with respect to a given finite automaton $\mathcal{F}$. If shrinkability succeeds, then the tool outputs the computed parameters δ and $\boldsymbol{k}$, and the witnessing simulator sets. Otherwise, a counter-example is output, which is a finite automaton that cannot be simluated by any shrinking of the timed automaton. Some benchmarks are given in [San13], and the tool is available at `http://www.lsv.ens-cachan.fr/Software/shrinktech`.

4.2 Approximately Bisimilar Implementation

We now assume that the implementation of the timed automaton design can tolerate small timings errors, but we would like to avoid the accumulation of these over time as in Example 4. Under this "relaxation", we will show that we can achieve more than with the shrinking technique we described in the previous subsection: under some assumptions, all timed automata can be "approximately" implemented.

In order to compare the behaviors of two timed automata allowing bounded errors in timings, we use ε-bisimilarity, an extension of timed bisimilarity. Formally, an ε-bisimulation R is a relation between the states of a timed automaton defined as follows. For any s and t with $s\ R\ t$, whenever $s \xrightarrow{\sigma} s'$, there exists t' such that $t \xrightarrow{\sigma} t'$, and $t\ R\ t'$, and whenever $s \xrightarrow{d} s'$ for some $d \geq 0$, then there exists $d' \in [\max(0, d-\varepsilon), d+\varepsilon]$ such that $s'\ R\ t + d'$. We denote $s \sim_\varepsilon t$, if there is an ε-bisimulation between states s and t. Note that $\sim_0$ is the usual timed bisimulation, also written $\sim$.

Now, given timed automata $\mathcal{A}$ and $\mathcal{A}'$, we say that $\mathcal{A}'$ is an *ε-implementation*, if $\mathcal{A} \sim \mathcal{A}'$ and $\mathcal{A}' \sim_\varepsilon \mathcal{A}'_\delta$ for some $\delta, \varepsilon > 0$. Here, the former condition means that the alternative timed automaton $\mathcal{A}'$ is "equivalent" to $\mathcal{A}$ in the exact semantics, while the latter condition requires the robustness of $\mathcal{A}'$: under enlargement, all behaviors are approximately included in those of $\mathcal{A}$.

We also consider the simpler notion of *safe implementation* by requiring that $\mathcal{A} \sim \mathcal{A}'$ and that $\mathcal{A}'$ and $\mathcal{A}'_\delta$ have the same set of reachable locations for some $\delta > 0$.

Theorem 4. *For any timed automaton $\mathcal{A}$, and any $\varepsilon > 0$, one can construct an ε-implementation, and a safe implementation, both in exponential time.*

Although one can use ε-implementations to obtain safe implementations, the latter are smaller in size in practice.

Methodology. The above theorem allows one to completely separate design and implementation of timed automata models. In fact, since the implementation can be ensured for all models, one can only concentrate on designing the model and proving properties in the exact semantics; imprecisions do not need to be taken into account. One can then specify the precision ε depending on the required accuracy.

1. Design and verify timed automaton $\mathcal{A}$;
2. Automatically generate an implementation.

5 Robust Realisability and Control

In this section, we are interested in checking whether a given desired behavior is realisable in a timed automaton when the time delays are subject to perturbations. Our goal is to detect behaviors (such as defined by Büchi conditions) that can be realised even when the delays can only be chosen upto a bounded error. In fact, although the theory of timed automata allows characterizing the existence of infinite runs accepting for a Büchi condition, not all such runs are realisable with finite precision (hence, not in hardware neither); we already saw in Example 3 a timed automaton where any infinite run contains delay and clock value sequences that are convergent, that is, requires infinite precision. If the input model is timed games, then this problem is closely related to that of *robust controller synthesis*, where the goal is to find winning strategies that resist to systematic perturbations in the time delays.

We thus present the problem both as a realisability problem for timed automata, and as robust controller synthesis in timed games. We formalise this problem as a turn-based game between a controller that chooses delays and edges and an environment that perturbs the chosen delays. Two variants of this game semantics were studied in [CHP11, SBMR13] and [BMS12] respectively. We first present the two variants and related results, and compare them at the end of this section.

5.1 Conservative Semantics

We start with the *conservative game semantics*. Intuitively, the semantics is a turn-based two-player game parameterized by $\delta > 0$, where Player 1, also called *Controller* chooses a delay $d > \delta$ and an edge whose guard is satisfied after any delay in $d + [-\delta, \delta]$. Then, Player 2, also called *Perturbator* chooses an actual delay $d' \in d + [-\delta, \delta]$ after which the edge is taken. Hence, the delays suggested by Controller are perturbed by Perturbator by an amount from $[-\delta, \delta]$, but the guard is required to be satisfied whatever the perturbation is.

We also consider two-player timed games where, in addition to being able to suggest delays and actions, Perturbator can perturb by an amount in $[-\delta, \delta]$ the delays chosen by Controller.

While for $\delta = 0$, a strategy for Controller in this game semantics merely yields a run, for $\delta > 0$, it describes how Controller reacts to perturbations in the past. It may, for instance, choose the delays so as to correct the effect of the earlier perturbations.

We are interested in deciding whether for some $\delta > 0$, there exists a strategy of Controller whose all outcomes satisfy a given ω-regular condition. This is the *parameterized robust controller synthesis for Büchi objectives.* If the parameter is given as part of the input, then we will call the problem *fixed-parameter robust controller synthesis.* While the timed automaton of Fig. 6 is uncontrollable in this sense (which follows from the fact that it is not shrinkable), Fig. 7 shows example of a controllable timed automaton.

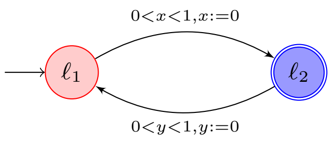


Fig. 7. A timed automaton (from [BA11]) that is robustly controllable for the Büchi objective $\{\ell_2\}$. In fact, perturbations at a given transition do not affect the rest of the run; they are *forgotten.*

For a fixed parameter $\delta > 0$, we have the following result for parity conditions.

Theorem 5 ([CHP11]). *The fixed-parameter robust controller synthesis for timed games and parity conditions in the conservative game semantics can be solved in exponential time.*

The above algorithm is obtained by reduction to timed games; in fact, when δ is known, it is possible to encode the game semantics as a timed game, and apply known algorithms.

For Büchi objectives, the parameterized version of the problem can be solved in polynomial space:

Theorem 6 ([SBMR13]). *Parameterized robust controller synthesis for timed automata and Büchi conditions in the conservative game semantics is PSPACE-complete.*

The above theorem is established by characterising those cycles that can be repeated infinitely by Controller against any strategy of Perturbator. The characterisation is expressed in terms of the reachability graph between the vertices of the visited regions (a.k.a. the orbit graph), and was introduced in [BA11] in a different context. [BA11] characterizes those cycles of a timed automaton along which the clock values are not convergent; these are called *forgetful cycles.*

The contribution of [SBMR13] consists in showing that any long enough run along non-forgetful cycles can be made blocking by an appropriate strategy of Perturbator.

5.2 Excess Semantics

We now present a variant of the previous semantics, similar in definition, but different in terms of usage. The *excess game semantics* is a turn-based two-player game between Controller and Perturbator, and parameterized by $\delta > 0$. The difference is that at each turn, Controller now only has to suggest a delay $d \geq \delta$ and an edge whose guard is satisfied only after the delay d. After such a move, Perturbator can modify the delay by any amount in $[-\delta, \delta]$, and the perturbed delay and the edge chosen by Controller are taken whether or not the guard is satisfied after the perturbation. We refer to Example 5 for an illustration of this semantics.

The following result shows the decidability of the parameterized robust controller synthesis problem for reachability objectives in timed automata and turn-based timed games.

Theorem 7 ([BMS12]). *Parameterized robust controller synthesis for turn-based timed games and reachability objectives in the excess game semantics is EXPTIME-complete.*

5.3 Comparison

We defined here two very similar game semantics. We believe both are meaningful and can appear naturally at different levels of abstraction. For instance, the excess-perturbation game semantics is a natural choice when modelling a real-time system with fixed task execution times, if we also know that these execution times will be subject to perturbation whose magnitude is unknown in advance, which may depend on the implementation platform to be chosen later. Incorporating these perturbations in the model, for instance, by replacing equality constraints by non-punctual intervals, requires a choice of a suitable interval length which may not be known. Moreover this will increase the state space in general. Hence the excess-perturbation game semantics allows one to keep the design abstract and still apply robustness analysis.

On the other hand, in some applications, specifying distinct lower and upper bounds in timings may be natural. For instance, if one is interested in modelling an embedded system that should send a signal to another component which accepts input signals in a time interval $[l, u]$, then it is natural to look for a controller that strictly respects this interval. Such a model and its semantics would be closer to the actual program. In this case, the conservative-perturbation game semantics is the natural choice.

Note also that while any outcome of the conservative game semantics is a run of the timed automaton (or game) in the usual semantics, this is not the case for the excess semantics. On the other hand, outcomes of the excess game semantics are runs of the enlarged automata.

5.4 Weighted Timed Automata and Games

Weighted timed automata [ALP01, BFH+01] are an extension of timed automata with one cost variable which grows with a possibly different constant derivative at each location. Cost-optimal reachability is decidable in this model, but undecidable in weighted timed games. Robustness problems can naturally be addressed for weighted timed automata and games, following similar motivations as timed automata. Moreover, one could hope that in weighted timed games, a suitable robust semantics could render the cost-optimal reachability problem decidable. We show in this section that undecidability still holds in both game semantics.

We consider conservative and excess perturbation game semantics. As Example 5 below shows, the optimal cost that can be achieved depends on δ. Because we assume that δ is a small parameter whose value can be adjusted as required, we will concentrate on the *limit-cost* of a strategy in a weighted timed automaton under the game semantics. The limit cost is the cost achieved by a strategy, when δ goes to 0.

Example 5. Figure 8 displays an example of a weighted timed game. Plain (resp. dashed) arrows are for Player 1 (resp. Player 2) edges. The slopes are indicated above each state. A strategy for Player 1 is to suggest a delay of 1 and choose

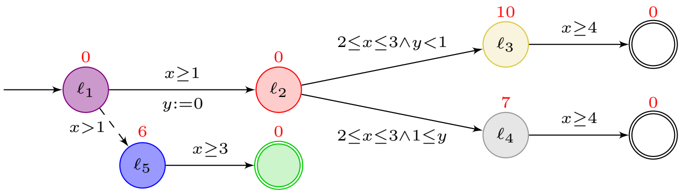


Fig. 8. Example of a WTG

the edge from ℓ_1 to ℓ_2. This prevents Player 2 from going down to location ℓ_5, where the cost of accepting is 12. From location ℓ_2, Player 1 can go to ℓ_4, from where a target location is reached with cost 7.

Under the excess-perturbation semantics, as in the exact case, Controller can suggest a delay of 1 and choose the edge from ℓ_1 to ℓ_2. The location ℓ_5 can thus be avoided. Now, one can see that the move of Perturbator determines the next location to be visited: if Perturbator adds a positive perturbation (*i.e.* if the delay is in $[1, 1+\delta]$), then only location ℓ_3 is reachable. Conversely, a negative perturbation enables only location ℓ_4. To maximize the cost, Perturbator will force the play to ℓ_3, so Controller can only ensure a cost of $10 + \Theta(\delta)$.

We now focus on the conservative semantics. The above strategy is no more valid since in this case, Controller can only suggest delays of at least $1 + \delta$. Then Perturbator can force the play to ℓ_5. Here, the cost of winning is $12 + \Theta(\delta)$.

We define the *optimal limit-cost (strong) decision problem* for the conservative or excess semantics as the problem of deciding whether some strategy achieves a limit cost of at least (resp. more than) a given threshold.

Theorem 8 ([BMS13])

- *The optimal limit-cost decision problem is undecidable for weighted timed automata under the excess game semantics.*
- *The optimal limit-cost strong decision problem is PSPACE-complete for weighted timed automata under the conservative game semantics, and undecidable for weighted timed games.*

Hence, the decision problems on optimal limit-cost in weighted timed games remain undecidable. Moreover, the problem even becomes undecidable for weighted timed *automata* in the excess game semantics. This is rather surprising since it is often believed that introducing imprecisions render problems easier to solve [AB01].

6 Conclusion

In this paper we have presented a recent approach to the theory of robustness for timed systems. The model of perturbation that is considered is that of guard enlargement, which captures time measurement errors and jitter. Other models of perturbation could be considered, which would more adequately take into account other sources of errors in the execution of timed systems.

Under this model of perturbation, we have described several robust verification problems which have been formulated and partly solved recently. Complexities of these problems are quite standard in the domain of timed systems, and symbolic technics need now to be developed for solving the various problems, that would support usability of the approach.

Robustness in timed systems is an important issue, the effort to develop a full theory of robustness needs therefore to be continued.

References

[AB01] Asarin, E., Bouajjani, A.: Perturbed turing machines and hybrid systems. In: Proc. 16th Annual Symposium on Logic in Computer Science (LICS 2001), pp. 269–278. IEEE Computer Society Press (2001)

[ABG$^+$08] Akshay, S., Bollig, B., Gastin, P., Mukund, M., Narayan Kumar, K.: Distributed Timed Automata with Independently Evolving Clocks. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 82–97. Springer, Heidelberg (2008)

[ACS10] Abdellatif, T., Combaz, J., Sifakis, J.: Model-based implementation of real-time applications. In: Proc. 10th International Workshop on Embedded Software (EMSOFT 2010), pp. 229–238. ACM (2010)

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

[AKY10] Abdulla, P.A., Krčál, P., Yi, W.: Sampled semantics of timed automata. Logical Methods in Computer Science 6(3:14) (2010)

[ALP01] Alur, R., La Torre, S., Pappas, G.J.: Optimal paths in weighted timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 49–62. Springer, Heidelberg (2001)

[Alu91] Alur, R.: Techniques for Automatic Verification of Real-Time Systems. PhD thesis, Stanford University, Stanford, CA, USA (1991)

[AT05] Altisen, K., Tripakis, S.: Implementation of timed automata: An issue of semantics or modeling? In: Pettersson, P., Yi, W. (eds.) FORMATS 2005. LNCS, vol. 3829, pp. 273–288. Springer, Heidelberg (2005)

[BA11] Basset, N., Asarin, E.: Thin and thick timed regular languages. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 113–128. Springer, Heidelberg (2011)

[BDL$^+$06] Behrmann, G., David, A., Larsen, K.G., Håkansson, J., Pettersson, P., Yi, W., Hendriks, M.: Uppaal 4.0. In: Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST 2006), pp. 125–126. IEEE Computer Society Press (2006)

[BDM$^+$98] Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: a model-checking tool for real-time systems. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 546–550. Springer, Heidelberg (1998)

[BFH$^+$01] Behrmann, G., Fehnker, A., Hune, T., Larsen, K.G., Pettersson, P., Romijn, J.M.T., Vaandrager, F.W.: Minimum-cost reachability for priced timed automata. In: Di Benedetto, M.D., Sangiovanni-Vincentelli, A.L. (eds.) HSCC 2001. LNCS, vol. 2034, pp. 147–161. Springer, Heidelberg (2001)

[BMR06] Bouyer, P., Markey, N., Reynier, P.-A.: Robust model-checking of linear-time properties in timed automata. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 238–249. Springer, Heidelberg (2006)

[BMR08] Bouyer, P., Markey, N., Reynier, P.-A.: Robust analysis of timed automata *via* channel machines. In: Amadio, R.M. (ed.) FOSSACS 2008. LNCS, vol. 4962, pp. 157–171. Springer, Heidelberg (2008)

[BMS11] Bouyer, P., Markey, N., Sankur, O.: Robust model-checking of timed automata via pumping in channel machines. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 97–112. Springer, Heidelberg (2011)

[BMS12] Bouyer, P., Markey, N., Sankur, O.: Robust reachability in timed automata: A game-based approach. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part II. LNCS, vol. 7392, pp. 128–140. Springer, Heidelberg (2012)

[BMS13] Bouyer, P., Markey, N., Sankur, O.: Robust weighted timed automata and games. In: Braberman, V., Fribourg, L. (eds.) FORMATS 2013. LNCS, vol. 8053, pp. 31–46. Springer, Heidelberg (2013)

[BS91] Brzozowski, J.A., Seger, C.-J.H.: Advances in asynchronous circuit theory. Bulletin of the European Association of Theoretical Computer Science, EATCS (1991)

[CHP11] Chatterjee, K., Henzinger, T.A., Prabhu, V.S.: Timed parity games: Complexity and robustness. Logical Methods in Computer Science 7(4) (2011)

[CHR02] Cassez, F., Henzinger, T.A., Raskin, J.-F.: A comparison of control problems for timed and hybrid systems. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC 2002. LNCS, vol. 2289, pp. 134–148. Springer, Heidelberg (2002)

[DDMR08] De Wulf, M., Doyen, L., Markey, N., Raskin, J.F.: Robust safety of timed automata. Formal Methods in System Design 33(1-3), 45–84 (2008)

[DDR05] De Wulf, M., Doyen, L., Raskin, J.-F.: Systematic Implementation of Real-Time Models. In: Fitzgerald, J.S., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 139–156. Springer, Heidelberg (2005)

[Dim07] Dima, C.: Dynamical properties of timed automata revisited. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 130–146. Springer, Heidelberg (2007)

[GHJ97] Gupta, V., Henzinger, T.A., Jagadeesan, R.: Robust timed automata. In: Maler, O. (ed.) HART 1997. LNCS, vol. 1201, pp. 331–345. Springer, Heidelberg (1997)

[HR00] Henzinger, T.A., Raskin, J.-F.: Robust undecidability of timed and hybrid systems. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 145–159. Springer, Heidelberg (2000)

[KP05] Krčál, P., Pelánek, R.: On sampled semantics of timed systems. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 310–321. Springer, Heidelberg (2005)

[Pur98] Puri, A.: Dynamical properties of timed automata. In: Ravn, A.P., Rischel, H. (eds.) FTRTFT 1998. LNCS, vol. 1486, pp. 210–227. Springer, Heidelberg (1998)

[Pur00] Puri, A.: Dynamical properties of timed automata. Discrete Event Dynamic Systems 10(1-2), 87–113 (2000)

[San13] Sankur, O.: Shrinktech: A tool for the robustness analysis of timed automata. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1006–1012. Springer, Heidelberg (2013)

[SBM11] Sankur, O., Bouyer, P., Markey, N.: Shrinking timed automata. In: Proc. 30th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2011). LIPIcs, vol. 13, pp. 375–386. Leibniz-Zentrum für Informatik (2011)

[SBMR13] Sankur, O., Bouyer, P., Markey, N., Reynier, P.-A.: Robust controller synthesis in timed automata. In: D'Argenio, P.R., Melgratti, H. (eds.) CONCUR 2013. LNCS, vol. 8052, pp. 546–560. Springer, Heidelberg (2013)

[SFK09] Swaminathan, M., Fränzle, M., Katoen, J.-P.: The surprising robustness of (closed) timed automata against clock-drift. In: Ausiello, G., Karhumäki, J., Mauri, G., Ong, L. (eds.) Proc. 5th IFIP International Conference on Theoretical Computer Science (TCS 2008). IFIP, vol. 273, pp. 537–553. Springer, Boston (2009)