# Model Checking Real-Time Systems

Written Abstract for the Seminar “Recent Advances in Model Checking”

Vincent Trélat

## Organizational information

This abstract is based on Chapter 29 of the Handbook of Model Checking [CHVB18]. Section 1 first introduces and motivates model checking applied to real-time systems, building on [CHVB18, Chapters 29.1 and 29.2]. Section 2 gives some formal definitions from [CHVB18, Chapters 29.2 and 29.7] about timed automata and related notions such as region equivalence and zones. Finally, Section 3 briefly introduces time-extended formalisms such as *Timed Temporal Logic* and *Timed Games* related to timed automata for the verification of real-time systems, building on [CHVB18, Chapters 29.6 and 29.9]. An additional Section 4 brings some further material about language-theoretic properties that are not related to model checking and will not be discussed in the talk.

## 1 Introduction

**Motivation** In the early 1990s, Rajeev Alur and David Dill introduced a powerful and expressive formalism named *timed automata* which have proven very convenient for modeling and reasoning about real-time systems. It is nowadays a standard tool for the verification of real-time systems and has many applications in the industry, while also being a very active research area.

**Outlook** The most basic problem in model-checking is *reachability*, i.e. given a structure, one wants to determine whether a set of locations is reachable from an initial location. Since model-checking reasons on properties written in temporal logic, we need *timed temporal logic* that can express properties on timed automata. With such a formalism, one may eventually be interested in the controller synthesis problem, which can be expressed in the formalism of *timed games*. My presentation will therefore be articulated around these notions.

## 2 Timed Automata

**Preliminaries** In this chapter, time values are equated with non-negative real numbers of $\mathbb{R}_{\geq 0}$. A *time sequence* is a finite or infinite non-decreasing sequence of time values. A *timed word* over $\Sigma \times \mathbb{R}_{\geq 0}$ is a word over the alphabet $\Sigma$ sequentially paired with a time sequence.

Let $C$ be a finite set of variables called *clocks*. A *valuation* over $C$ is a mapping $v \colon C \to \mathbb{R}_{\geq 0}$. The set of valuations over $C$ is denoted $\mathbb{R}_{\geq 0}^{C}$ and $\mathbf{0}_C$ denoted the valuation assigning 0 to every clock of C.

For any valuation $v$ and any time value $t$, the valuation $v + t$ denotes the valuation obtained by shifting all values of $v$ by $t$. For any subset $r$ of $C$, $v[r]$ is the valuation obtained by resetting all clocks of $r$ in $v$.

A *constraint* $\varphi$ over $C$ is recursively defined by the following grammar:

$$\varphi ::= x \odot k \mid \varphi \wedge \varphi$$

where $x \in C$, $k \in \mathbb{Z}$ and $\odot \in \{<, \leq, =, \geq, >\}$. The set of constraints over $C$ is denoted $\Phi(C)$. We say that a valuation $v$ over $C$ satisfies $x \odot k$ when $v(x) \odot k$, and when $v$ satisfies a constraint $\varphi$, we write $v \models \varphi$. The set of valuations satisfying a constraint $\varphi$ is denoted $[\![\varphi]\!]_C$.

**Timed Automata** A timed automaton is basically a finite automaton with (real-time) constraints on the states. The following formal definition is a reformulation of [CHVB18, Chapter 29.2, Definition 1].

**Definition 1.** A *Timed Automaton* (TA) $\mathcal{A}$ is the data $(L, l_0, C, \Sigma, I, E)$ where $L$ is a finite set of *locations* with initial location $l_0 \in L$, $C$ is a finite set of *clocks*, $\Sigma$ is a finite set of *actions*, $I\colon L \to \Phi(C)$ is an *invariant mapping* and $E \subseteq L \times \Phi(C) \times \Sigma \times 2^C \times L$ is a set of edges.

Any edge $(\ell, \varphi, a, r, \ell') \in E$ is denoted $\ell \xrightarrow{\varphi,a,r} \ell'$ where $\varphi$ is a *guard*, and $r$ is a subset of clocks that are set to zero after taking the transition.

An example of TA is given in Fig. 1.

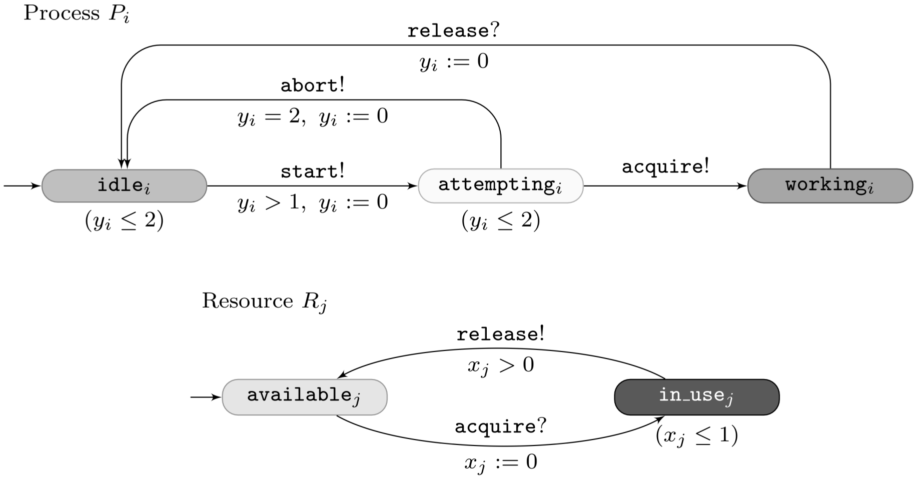


Figure 1: Two TA modeling processes which can use resources.
Figure taken from [CHVB18, Chapter 29.2]

**Semantics** The *operational semantics* of a TA $\mathcal{A} = (L, \ell_0, C, \Sigma, I, E)$ is the infinite-state timed transition system $[\![A]\!] = (S, s_0, \mathbb{R}_{\geq 0} \times \Sigma, T)$, where

$$S := \{(\ell, v) \in L \times \mathbb{R}_{\geq 0}, v \models I(\ell)\}, \quad s_0 := (\ell_0, \mathbf{0}_C),$$
$$T := \{(\ell, v) \xrightarrow{d,a} (\ell', (v+d)[r]) \mid d \in \mathbb{R}_{\geq 0}, \forall d' \in [0, d], v + d' \models I(\ell) \wedge \exists \ell \xrightarrow{\varphi,a,r} \ell' \in E, v + d \models \varphi\}$$

**Parallel Composition** It is possible to compose several TA in parallel. Informally, parallel composition roughly consists in pairing the automata and taking the conjunction of their invariants, distinguishing two types of transitions, namely the synchronous[1] and asynchronous transitions. The point is that systems can be defined in a compositional way.

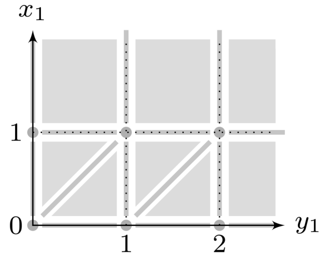
**Region Equivalence** Informally, two valuations are region equivalent if a TA cannot differentiate between them and we write[2] this relation $\cong_M$. More formally, two valuations are region equivalent

[1] Synchronous transitions are given by a *synchronization function*.

[2] $M := (M_x)_{x \in C}$ where $M_x$ is the maximal constant clock $x$ is compared to in the TA.

if their evaluations on any clock have the same integral parts and one has its fractional part null iff the other has its fractional part null and the order of the fractional parts is the same.

This notion is extended to states of the TA by defining $(\ell, v) \cong_M (\ell', v')$ iff $v \cong_M v'$ and $\ell = \ell'$. The equivalence class of $(\ell, v)$ is denoted $[\ell, v]_{\cong_M}$. As an example, we give a representation of clock regions for the parallel composition of $P_1$ and $R_1$ in Fig. 2.



| $P_1$ | $R_1$ | $f_1$ |
|---|---|---|
| start! | – | start |
| abort! | – | abort |
| acquire! | acquire? | acquire |
| release! | release? | release |

Figure 2: Clock regions for $(P_1||R_1)_{f_1}$ where $f_1$ is given by its table of values.

Figure taken from [CHVB18, Chapter 29.3]

From this notion, we can define the *region automaton* $\mathcal{R}_{\cong_M}(\mathcal{A}) = (S, s_0, \Sigma, T)$ which basically consists in quotienting $\mathcal{A}$ w.r.t. $\cong_M$:

$$S = (L \times \mathbb{R}_{\geq 0})_{/\cong_M} \quad s_0 = [\ell_0, \mathbf{0}_C]_{\cong_M} \quad T = \{[\ell, v]_{\cong_M} \xrightarrow{a} [\ell', v']_{\cong_M} \mid \exists d, (\ell, v) \xrightarrow{d,a} (\ell', v')\}$$

The region automaton $\mathcal{R}_{\cong_M}(\mathcal{A})$ is a finite automaton whose size is exponential compared with the size of $\mathcal{A}$. It can be used to check, reachability properties and "one of the most fundamental theorems" [CHVB18, Chapter 29.2, Theorem 1] states that the reachability problem (or equivalently emptiness) in TA is PSPACE-complete [AD94, Section 4.5].

**Zones** The number of states obtained from the region partitionning is unfortunately extremely large, so the notion of *zones* was introduced as a coarser representation of the state space. *Zones* are exactly the sets $[\![\varphi]\!]_C$ of valuations satisfying any guard $\varphi \in \Phi_d(C)$[3]. Similarly to the region automaton, we can define the *zone automaton*, except that the zone automaton is infinite, so zones need to be *normalized*, which basically consists in quotienting each zone w.r.t. the equivalence relation $\cong_M$. This operation does not produce zones in general, other operators are therefore used in practice, such as the *extrapolation* of a *shortest-path-closed* DBM (*difference-bound matrix*).

**Extensions** Many extensions of TA were proposed in the literature, such as allowing *diagonal constraints*, *updatable* TA[4] or adding *urgency* requirements. However, it has been shown that such extensions are no more expressive than the original class of TA. Furthermore, most attempted extensions generally lead to undecidability of the reachability problem, e.g. by simply adding constraints of the form $\lambda x + \mu y \leq k$ or $x := x + k$, or more complex classes such as *hybrid* automata.

## 3 Model checking timed systems

**Timed Temporal Logic** We extend the usual theory of LTL presented in [CHVB18, Chapter 2] to timed systems with MTL (*Metric Temporal Logic*). Given a set of atomic propositions $P$, an interval $I$ and a *time-constrained until* operator $\mathbf{U}_I$, the formulas of MTL are defined as follows:

$$\varphi ::= p \in P \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \mathbf{U}_I \varphi$$

[3] $\Phi_d(C)$ is the set of *diagonal constraints* over $C$, i.e. constraints of the form $x - y \odot k$.

[4] Clocks can be updated to any value instead of being reset to 0.

Similarly, the *constrained always* operator $\Box_I$ and *constrained eventually* operator $\Diamond_I$ can be defined with $\mathbf{U}_I$ in this time-constrained setting. Two different semantics are usually adopted for MTL, namely the *continuous* and *pointwise* semantics. Informally, the pointwise semantics allows one to assert formulas only at discrete set of points, while the continuous semantics allows one to assert formulas at arbitrary time points.

As one may imagine, bringing such expressiveness affects decidability. Although model checking and satisfiability for LTL are PSPACE-complete over both semantics, it is easy to see that the continuous semantics is strictly more expressive than the pointwise semantics, which results in losing decidability for model checking and satisfiability for MTL over the continuous semantics, as well as over the pointwise semantics for infinite words. By reducing expressiveness of MTL[5], one can recover decidability for model checking and satisfiability over both semantics.

The case of branching-time such as CTL and CTL* can be adapted to timed systems as well to TCTL and TCTL*. Unfortunately, only TCTL model checking is decidable (and PSPACE-complete).

**Timed Games** As explained in [CHVB18, Chapter 27], games provide a convenient framework for modeling and reasoning about environments that may be affected by several (uncontrollable) agents. An important result is that (time-optimal) reachability and safety games are decidable and EXPTIME-complete for timed games. The key ingredient of the proof of this result is that it is sufficient for those specific control objectives to consider *memoryless* strategies, i.e. strategies that do not depend on the history of the game. By authorizing time divergent[6] strategies only, deciding the existence of a winning strategy for reachability and safety games is EXPTIME-complete.

Timed games can also be used to decide bisimilarity of two states of an automaton (in exponential time). Informally, two equivalent states of a TA are time bisimilar (for some equivalence relation) if the TA can take the same transitions from those states.

Timed games have received a lot of attention recently, some work has conducted on extensions in which, for instance, players have different objectives.

## 4 Language-Theoretic Properties

As one might expect, all the above sections are related to language-theoretic properties of timed automata. In this section, we present some of the most important results related to *timed languages*. For instance, contrary to the untimed case, $\varepsilon$-TA are strictly more expressive than TA and one cannot decide whether a given $\varepsilon$-TA has a TA equivalent (without $\varepsilon$-transitions). Although in the general case the language-inclusion problem is undecidable for TA, it may recover its decidability in some special cases, such as when the containing TA is an *event-clock automaton* (over infinite words) or has at most one clock (over finite words), or under *closed-time* or *bounded-time* assumptions.

## References

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[CHVB18] Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of Model Checking.* Springer Publishing Company, Incorporated, 1st edition, 2018.

[5] See for instance MITL (EXPSPACE-complete), ECTL (PSPACE-complete) and MTL over *bounded time* (EXPSPACE-complete).

[6] Such an assumption stems from the condition of *non-Zenoness* which is another interesting notion. Basically, a run is *Zeno* if it can take infinitely many transitions in finite time.