An Introduction to Timed Automata

Patricia Bouyer
LSV – CNRS & ENS de Cachan
61, avenue du Président Wilson
94230 Cachan – France
email: bouyer@lsv.ens-cachan.fr

1 Introduction

Explicit timing constraints are naturally present in real-life systems (transmission delays, response time, etc...). Classical models (finite automata, Petri nets, etc...) can not express such real-time constraints. Since their introduction by Rajeev Alur and David Dill in [6, 7], timed automata are one of the most studied models for real-time systems: in those systems, quantitative properties of delays between events can easily be expressed. Numerous works have been devoted to the "theoretical" comprehension of timed automata: determinization [9], minimization [3], power of clocks [5, 33], power of ε-transitions [15], extensions of the model [27, 35, 23, 13], logical characterizations [35], etc... have in particular been investigated. Practical aspects of the model have also been considered and several model-checkers are now available (HyTech [31], Kronos [25], Uppaal [38]). These model-checkers have been used to verify many industrial case studies (see the web pages of the tools, given page 13).

One of the major properties of timed automata is probably that reachability properties are decidable [7], though timed automata have an infinite number of configurations. The core of this result is the construction of the so-called region automaton, which finitely abstract behaviours of timed automata in such a way that checking reachability in a timed automaton reduces to checking reachability in a (somewhat larger) finite automaton. This construction has many other applications, as for example the decidability of the TCTL model-checking [2] (TCTL is the timed extension of the logic CTL). However, many problems remain undecidable, as not everything can be reduced to the untimed framework. For example, timed automata are neither determinizable, nor complementable [7]. Checking if a timed automaton is determinizable (or complementable) is even an undecidable problem [42]. An other important example is the undecidability of the universality problem for timed automata [7].

The aim of this tutorial is to give some understanding of the timed automata model. We will present the basic tools which are used in the domain of verification of timed systems. In particular, after having presented the model, we will present in details the region automata construction. For modeling reasons, it is important to have expressive models, but it is also important that the models remain decidable. We will then present several variants or extensions of timed automata, focusing on the decidability of reachability properties, and on the expressiveness of the models. We will terminate this tutorial with some implementation and algorithmics issues.

We would like to point out several recent surveys on timed automata which present current works and results on timed automata with a point of view somewhat different from the one adopted in this tutorial. A recent survey by Rajeev Alur and Madhusudan P. gives many hints about decidability issues for timed automata [10]. In [11], Eugene Asarin presents the current challenges in timed languages theory.

2 Timed Automata

If Z is a set, let Z^* be the set of *finite* sequences of elements in Z. We consider as time domain $\mathbb{T}$ the set $\mathbb{Q}_+$ of non-negative rationals or the set $\mathbb{R}_+$ of non-negative reals, and Σ as a finite set of *actions*. A *time sequence* over $\mathbb{T}$ is a finite non decreasing sequence $\tau = (t_i)_{1 \le i \le p} \in \mathbb{T}^*$. A *timed word* $\omega = (a_i, t_i)_{1 \le i \le p}$ is an element of $(\Sigma \times \mathbb{T})^*$, also written as a pair $\omega = (\sigma, \tau)$, where $\sigma = (a_i)_{1 \le i \le p}$ is a word in Σ^* and $\tau = (t_i)_{1 \le i \le p}$ a time sequence in $\mathbb{T}^*$ of same length.

Clock Valuations, Operations on Clocks. We consider a finite set X of variables, called *clocks*. A *clock valuation* over X is a mapping $v : X \to \mathbb{T}$ which assigns to each clock a time value. The set of all clock valuations over X is denoted $\mathbb{T}^X$. Let $t \in \mathbb{T}$, the valuation $v + t$ is defined by $(v + t)(x) = v(x) + t$, $\forall x \in X$. We also use the notation $(\alpha_i)_{1 \le i \le n}$ for the valuation v such that $v(x_i) = \alpha_i$. For a subset Y of X, we denote by $[Y \leftarrow 0]v$ the valuation such that for each $x \in Y$, $([Y \leftarrow 0]v)(x) = 0$ and for each $x \in X \setminus Y$, $([Y \leftarrow 0]v)(x) = v(x)$.

Clock Constraints. Given a finite set of clocks X, we introduce two sets of *clock constraints over* X. The most general one, denoted $\mathcal{C}(X)$, is defined by the grammar:

$$g \ ::= \ x \bowtie c \ \mid \ x - y \bowtie c \ \mid \ g \wedge g \ \mid \ \mathit{true}$$
where $x,\ y \in X,\ c \in \mathbb{Z}$ and $\bowtie \in \{<, \le, =, \ge, >\}$.

We also use the proper subset of *diagonal-free* constraints where the comparison between two clocks is not allowed. This set, denoted $\mathcal{C}_{df}(X)$, is defined by the grammar:

$$g \ ::= \ x \bowtie c \ \mid \ g \wedge g \ \mid \ \mathit{true},$$
where $x \in X$, $c \in \mathbb{Z}$ and $\bowtie \in \{<, \le, =, \ge, >\}$.

A *k-bounded clock constraint* is a clock constraint which involves only constants c between $-k$ and $+k$. The set of k-bounded (resp. k-bounded diagonal-free) clock constraints is denoted $\mathcal{C}^k(X)$ (resp. $\mathcal{C}^k_{df}(X)$). A constraint of the form $x - y \bowtie c$ is a *diagonal constraint*.

If v is a clock valuation we write $v \models g$ when v satisfies the clock constraint g and we say that v satisfies $x \bowtie c$ (resp. $x - y \bowtie c$) whenever $v(x) \bowtie c$ (resp. $v(x) - v(y) \bowtie c$). If g is a clock constraint, we note $[\![g]\!]$ the set of clock valuations $\{v \in \mathbb{T}^X \mid v \models g\}$.

Timed Automata. A *timed automaton* over $\mathbb{T}$ is a tuple $\mathcal{A} = (\Sigma, Q, T, I, F, X)$, where Σ is a finite alphabet of actions, Q is a finite set of states, X is a finite set of clocks, $T \subseteq Q \times [\mathcal{C}(X) \times \Sigma \times 2^X] \times Q$ is a finite set of transitions[1], $I \subseteq Q$ is the subset of initial states and $F \subseteq Q$ is the subset of final states. If all constraints appearing in $\mathcal{A}$ are diagonal-free, we say that $\mathcal{A}$ is a *diagonal-free timed automaton*.

A *path* in $\mathcal{A}$ is a finite sequence of consecutive transitions:

$$P = q_0 \xrightarrow{g_1, a_1, Y_1} q_1 \ \ldots \ q_{p-1} \xrightarrow{g_p, a_p, Y_p} q_p$$

where $q_{i-1} \xrightarrow{g_i, a_i, Y_i} q_i \in T$ for every $1 \le i \le p$.

The path is said to be *accepting* if it starts in an initial state ($q_0 \in I$) and ends in a final state ($q_p \in F$). A *run* of the automaton along the path P is a sequence of the form:

$$(q_0, v_0) \xrightarrow[t_1]{g_1, a_1, Y_1} (q_1, v_1) \ \ldots \ \xrightarrow[t_p]{g_p, a_p, Y_p} (q_p, v_p)$$

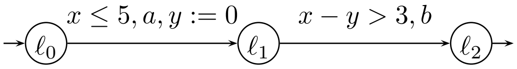
where $\tau = (t_i)_{1 \le i \le p}$ is a time sequence and $(v_i)_{1 \le i \le p}$ are clock valuations such that:

$$\begin{cases} v_0(x) = 0, \ \forall x \in X \\ v_{i-1} + (t_i - t_{i-1}) \models g_i \\ v_i = [C_i \leftarrow 0]\,(v_{i-1} + (t_i - t_{i-1})) \end{cases}$$

The label of the run is the timed word $w = (a_1, t_1) \ldots (a_p, t_p)$. If the path P is accepting then the timed word w is said to be accepted by $\mathcal{A}$. The set of all timed words accepted by $\mathcal{A}$ is denoted by $L_t(\mathcal{A})$.

Remark 1 *In these notes, we only consider finite paths and words with finitely many actions, but we could consider more general acceptance conditions (Büchi, Muller, etc...) as well, see [7].*

Example 1 *An example of timed automaton is given below.*



This timed automaton accepts the timed word $(a, 4.1)(b, 5.5)$. *An accepting run for this word is*

$$(\ell_0, (0, 0)) \xrightarrow[4.1]{a} (\ell_1, (4.1, 0)) \xrightarrow[5.5]{b} (\ell_2, (5.5, 1.4))$$

where $(4.1, 0)$ *represents the valuation* v *such that* $v(x) = 4.1$ *and* $v(y) = 0$.

3 Reachability Analysis

For verification purposes, the most fundamental properties that one should be able to verify are reachability properties: safety properties can for

[1] For more readability, a transition will often be written as $q \xrightarrow{g, a, Y} q'$ or even as $q \xrightarrow{g, a, Y := 0} q'$ instead of simply the tuple (q, g, a, Y, q').

example be expressed as reachability properties. Usually a class of models is said *decidable* whenever checking reachability properties in this class is decidable. Otherwise this class is said *undecidable*. For timed automata reachability properties we want to check are: "Is state q of timed automaton $\mathcal{A}$ reachable? *i.e.* is there a run starting in an initial state leading to q?" There is no requirement as what are the values of the clocks when reaching state q. This problem is equivalent to the *emptiness problem* (from a language-theoretical point of view), where the question is whether the language accepted by a timed automaton is empty or not.

The class of finite automata is obviously decidable, the reachability problem is even NLOGSPACE-complete [36], and efficient methods, symbolic techniques, data structures, etc... have been developed and implemented [24]. The problem with timed automata is that the number of configurations of a timed automaton is infinite (a configuration is a pair (q, v) where q is a state and v a clock valuation). Techniques used for verifying finite automata can thus not be used for timed automata. Specific symbolic techniques and abstractions have to be developed, which take into account the specific properties of timed automata, in particular the fact that clocks evolve synchronously with global time.

In the following, we will concentrate on the verification of reachability properties in timed automata, and present the basic technics for solving this problem. Of course, in the literature, more general properties have been considered. For example, the model-checking of TCTL [2], a timed extension of CTL, is decidable in PSPACE, and symbolic technics have been developed to efficiently model-check TCTL [34]. Note however that not everything can be reduced to the finite untimed case using the region automaton construction: for example, universality of timed automata is undecidable [7], and model-checking of most linear-time timed temporal logics are undecidable, when equality can be used in the constraints [8].

4 The Region Abstraction

The construction we will describe below is due to Alur and Dill first in [6]. The aim of this construction is to finitely abstract behaviours of timed automata, so that checking a reachability property in a timed automaton reduces to checking a reachability property in a finite automaton.

4.1 The Region Automaton Construction

Region Partitioning. Let us fix a finite set of clocks X. Let $\mathcal{R}$ be a finite partitioning of $\mathbb{T}^X$. Let $\mathcal{C}$ be a finite set of constraints over X. We define three compatibility conditions as follows:

① We say that $\mathcal{R}$ is *compatible with constraints* $\mathcal{C}$ if for every constraint g in $\mathcal{C}$, for every R in $\mathcal{R}$, either $[\![g]\!] \subseteq R$ or $[\![g]\!] \cap R = \emptyset$.

② We say that $\mathcal{R}$ is *compatible with elapsing of time* if for all R and R' in $\mathcal{R}$, if there exists some $v \in R$ and $t \in \mathbb{T}$ such that $v + t \in R'$, then for every $v' \in R$, there exists some $t' \in \mathbb{T}$ such that $v' + t' \in R'$.

③ We say that $\mathcal{R}$ is *compatible with resets* whenever for all R and R' in $\mathcal{R}$, for every subset $Y \subseteq X$, if $[Y \leftarrow 0]R \cap R' \neq \emptyset$, then $[Y \leftarrow 0]R \subseteq R'$.

If $\mathcal{R}$ satisfies these three conditions, we will say that $\mathcal{R}$ is a *set of regions* for the set of constraints $\mathcal{C}$ or simply a set of regions (if $\mathcal{C}$ is clear from the context). $\mathcal{R}$ defines in a natural way an equivalence relation $\equiv_{\mathcal{R}}$ over valuations ($v \equiv_{\mathcal{R}} v'$ iff for each region R of $\mathcal{R}$, $v \in R \iff v' \in R$). An equivalence class of $\equiv_{\mathcal{R}}$ (or equivalently an element of $\mathcal{R}$) is called a *region*. If v is a valuation we note $[v]_{\mathcal{R}}$ the region to which v belongs.

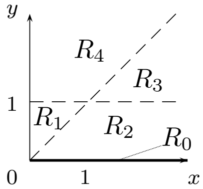
The intuition behind these conditions is the following: we want to finitely abstract behaviours of timed automata. To this aim, we finitely abstract the (infinite) set of valuations: a valuation v will be abstracted by the region $[v]_{\mathcal{R}}$. In order for the abstraction to preserve (at least) reachability properties, it must be the case that if two valuations are equivalent, then their future behaviours are also equivalent. The three conditions above precisely express this property: condition ① says that two equivalent valuations satisfy the same clock constraints, condition ② says that elapsing of time does not distinguish two equivalent valuations whereas condition ③ says that resetting clocks does not distinguish two equivalent valuations.

Region Graph. From a set of regions $\mathcal{R}$ one can define the so-called *region graph*, which represents the possible timing evolutions of the system: the region graph is a finite automaton whose set of states is $\mathcal{R}$ and whose transitions are:

$$\begin{cases} R \xrightarrow{\varepsilon} R' \text{ if } R' \text{ is a time successor of } R \\ R \xrightarrow{Y} R' \text{ if } [Y \leftarrow 0]R \subseteq R' \end{cases}$$

Intuitively, the region graph records possible timed evolutions of the system: there is a transition $R \xrightarrow{\varepsilon} R'$ if, from every valuation of R, it is possible to let some time elapse and reach R'. There is a transition $R \xrightarrow{Y} R'$ if, from R, R' can be reached by resetting clocks in Y.

Example 2 *Let us consider the following partitioning of* $\mathbb{R}_+^{\{x,y\}}$.



It is easy to verify that $\mathcal{R}$ is a set of regions for the constraints $\{y = 1, x = y\}$. The region graph associated with $\mathcal{R}$ is represented on **Fig.** *1.*

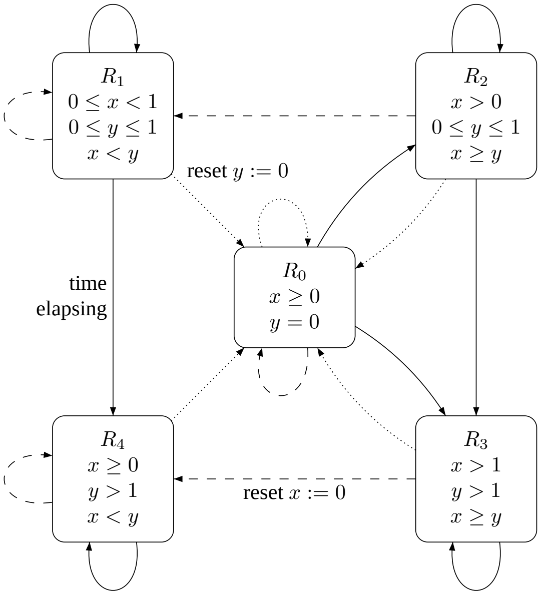


Fig. 1: *A simple example of region graph*

Region Automaton. Consider a timed automaton $\mathcal{A} = (\Sigma, Q, T, I, F, X)$ with set of constraints $\mathcal{C}$. Let $\mathcal{R}$ be a finite set of regions for $\mathcal{C}$ (*i.e.* a partitioning of $\mathbb{T}^X$ satisfying conditions ①, ② and ③). The *region automaton* $\Gamma_{\mathcal{R}}(\mathcal{A})$ is the finite automaton whose set of states is $Q \times \mathcal{R}$, whose initial states are $I \times \{R_0\}$ (where R_0 is the region containing the valuation assigning 0 to each clock), whose final states are $F \times \mathcal{R}$ and whose transitions are defined as follows:

- there is a transition $(\ell, R) \xrightarrow{a} (\ell', R')$ whenever there exists a transition $\ell \xrightarrow{g,a,Y} \ell'$ in $\mathcal{A}$ with $R \subseteq [\![g]\!]$ and $R \xrightarrow{Y} R'$ transition of the region graph
- there is a transition $(\ell, R) \xrightarrow{\varepsilon} (\ell, R')$ whenever $R \xrightarrow{\varepsilon} R'$ transition of the region graph

This automaton somehow simulates the original timed automaton: the first type of transitions simulates discrete actions (or transitions) whereas the second type of transitions simulates elapsing of time.

The fundamental property of this construction is the following:

Proposition 1 *Let $\mathcal{A}$ be a timed automaton with set of constraints $\mathcal{C}$. We assume we can construct a set of regions $\mathcal{R}$ for $\mathcal{C}$. Then,*

$$\mathsf{Untime}(L_t(\mathcal{A})) = L(\Gamma_{\mathcal{R}}(\mathcal{A}))$$

where $L(\Gamma_{\mathcal{R}}(\mathcal{A}))$ is the (untimed) language accepted by $\Gamma_{\mathcal{R}}(\mathcal{A})$, and

$$\mathsf{Untime}((a_1, t_1) \dots (a_p, t_p)) = a_1 \dots a_p.$$

More precisely, whenever in $\mathcal{A}$ we can wait some delay and do an a, then in $\Gamma_{\mathcal{R}}(\mathcal{A})$, we can take several ε-transitions and then do an a, and *vice-versa*. We will see in section 4.3 that this property naturally expresses in terms of time-abstract bisimulation. Checking reachability properties in $\mathcal{A}$ thus reduces to checking reachability properties in $\Gamma_{\mathcal{R}}(\mathcal{A})$. As $\Gamma_{\mathcal{R}}(\mathcal{A})$ is a finite automaton, we get that for every timed automaton $\mathcal{A}$ for which we can construct a set of regions (satisfying conditions ①, ② and ③), we can decide reachability properties using the region automaton construction

4.2 Region Automaton for Classical Timed Automata

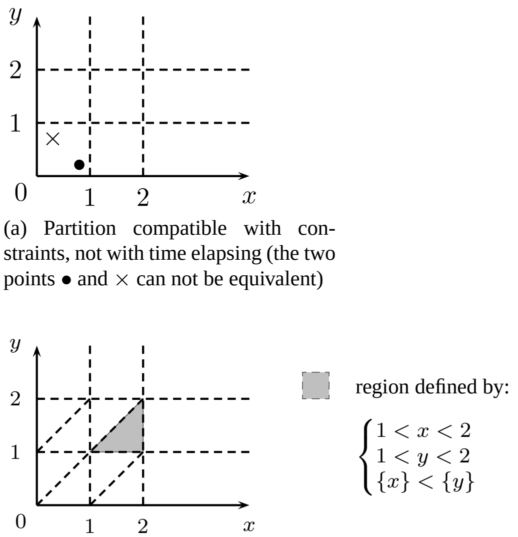
We fix for this subsection a finite set of clocks X.

Sets of regions for diagonal-free constraints. Let M be an integer. We define the following partitioning of $\mathbb{T}^X$. Let v and v' be two valuations of $\mathbb{T}^X$, we say that $v \equiv_{df}^M v'$ if all three following conditions hold:

- $v(x) > M$ iff $v'(x) > M$ for each $x \in X$,
- if $v(x) \le M$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ and $\Big(\{v(x)\} = 0$ iff $\{v'(x)\} = 0\Big)$ for each $x \in X$, and

- if $v(x) \leq M$ and $v(y) \leq M$, then $\{v(x)\} \leq \{v(y)\}$ iff $\{v'(x)\} \leq \{v'(y)\}$ for all $x,\ y \in X$.

The relation $\equiv_{df}^{M}$ is an equivalence relation of finite index. The partitioning $\mathcal{R}_{df}^{M}(X)$ is then defined as the set of equivalence classes of $\mathbb{T}^X{}_{/\equiv_{df}^{M}}$. **Fig.** 2 explains the region construction for two clocks.


y
2
1
0
1
2
x
region defined by:
$\begin{cases} 1 < x < 2 \\ 1 < y < 2 \\ \{x\} < \{y\} \end{cases}$

(a) Partition compatible with constraints, not with time elapsing (the two points ● and × can not be equivalent)

(b) Partition compatible with constraints, time elapsing (and resets)

Fig. 2: *Diagonal-free region partitioning for two clocks and maximal constant* 2

It is easy to prove (and left as an exercise) the following lemma:

Lemma 1 *The partitioning $\mathcal{R}_{df}^{M}(X)$ is a set of regions for the constraints $\mathcal{C}_{df}^{M}(X)$.*

Roughly counting all possible combinations above, we can bound the number of regions in $\mathcal{R}_{df}^{M}(X)$ by $2^{|X|}.|X|!.(2M+2)^{|X|}$ where $|X|$ is the cardinal of X.

Sets of regions for general constraints. Recall that the difference between diagonal-free clock constraints and general clock constraints stands in the fact that *diagonal constraints* (*i.e.* constraints of the form $x - y \bowtie c$) can be used. An easy extension of the previous construction can be done. We do not define it formally here, but only give a simple example with two clocks, see **Fig.** 3.

This set of regions is denoted $\mathcal{R}^M(X)$, and its cardinal can roughly be bounded by $(2M + 2)^{(|X|+1)^2}$. Note that this set of regions is also correct for M-bounded diagonal-free constraints.

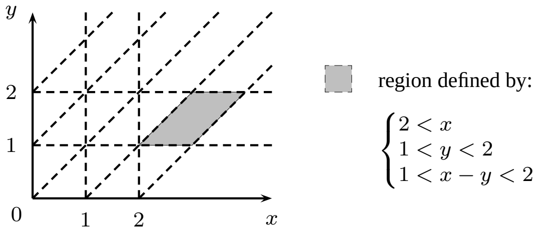

y
2
1
0
1
2
x
region defined by:
$\begin{cases} 2 < x \\ 1 < y < 2 \\ 1 < x - y < 2 \end{cases}$

Fig. 3: *Set of regions for* 2*-bounded general constraints with two clocks*

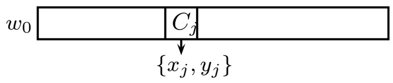
Region automata for classical timed automata. Let $\mathcal{A}$ be a timed automaton with set of clocks X. Let M be the maximal constant involved in one of the constraints of $\mathcal{A}$, the set $\mathcal{R}^M(X)$ is a set of regions for $\mathcal{A}$. From the results of the previous subsections, we get the following theorem, due to Alur and Dill [6, 7], which is the core of the verification of timed systems.

Theorem 1 (Alur & Dill 90's) *Reachability (or equivalently emptiness) is decidable for timed automata. It is a* PSPACE*-complete problem (for both diagonal-free as well as general timed automata).*

Although this theorem has been first proved in [7], the proof we choose to sketch is taken from [1], where it is written in details.

Proof. [Sketch] PSPACE membership is easy: the size of th region automaton is exponential in the size of the original automaton. Using the NLOGSPACE complexity of the reachability problem in classical untimed graphs, we get that reachability in timed automata can be done in PSPACE.

PSPACE-hardness can be proved by reducing the termination of a linearly bounded Turing machine (LBTM for short) on some input to reachability in timed automata. The encoding is done as follows: assuming the alphabet is $\{a, b\}$, the content of cell C_j of the track of the LBTM is encoded by two clocks x_j and y_j. Cell C_j contains an "a" when the constraint $x_j = y_j$ holds, and cell C_j contains a "b" when the constraint $x_j < y_j$ holds. Note that these two conditions are invariant by time elapsing.



If $q \xrightarrow{\alpha,\alpha',\delta} q'$ is a transition of the LBTM, then for each position i of the tape, there will be a transition $(q,i) \xrightarrow{g,Y:=0} (q',i')$ where:

- g is $x_i = y_i$ (resp. $x_i < y_i$) if $\alpha = a$ (resp. $\alpha = b$)
- $Y = \{x_i, y_i\}$ (resp. $Y = \{x_i\}$) if $\alpha = a$ (resp. $\alpha = b$)
- $i' = i+1$ (resp. $i' = i-1$) if δ is right and $i < n$ (resp. left)

We need to enforce time elapsing; this can be done by adding a clock t which is checked to 1 and reset to 0 on all transitions. Initially the track contains the encoding of the word w_0. This can be done by a transition from a state "init" to $(q_0, 1)$ where q_0 is the initial state of the LBTM, which checks whether $t = 1$, and resets clocks in Y_0 where $Y_0 = \{t\} \cup \{x_i \mid w_0[i] = b\}$. The computation over w_0 of the LBTM terminates iff there is a run from state "init" to some state (q_f, i) where q_f is the final state of the LBTM. □

Note that the above encoding uses diagonal constraints, but as will be seen later (see section 5.1), there is no need of these diagonals. A direct but more involved construction without diagonals can be found in the appendix of [1].

Remark 2 *Note that sets of regions we have described could be refined: there is no need to have the same maximal constant for all clocks, one maximal constant for each clock could be used. However, for our purpose here, there is no need for such a refinement.*

4.3 Interpretation in Terms of Finite Bisimulation

With what has been presented before, conditions ①, ② and ③ (compatibility of the set of regions with constraints, time elapsing and resets) have a natural interpretation in terms of ***time-abstract bisimulation***.

Timed transition system associated with a timed automaton. We have defined the semantics of a timed automaton as runs or timed words. We could have defined its semantics as a timed transition system as well. Transition systems (thus in particular timed transition systems) are more suitable for behavioural comparisons of systems.

Let $\mathcal{A} = (\Sigma, Q, T, I, F, X)$ be a timed automaton. The timed transition system associated with $\mathcal{A}$ has $Q \times \mathbb{T}^X$ for set of states and its transition relation is defined by the two following rules:

$$\begin{cases} (\ell, v) \xrightarrow{d} (\ell, v+d) & \text{for every } d \in \mathbb{T} \\ (\ell, v) \xrightarrow{a} (\ell', v') & \text{if there is } \ell \xrightarrow{g,a,Y} \ell' \text{ s.t.} \\ & v \models g,\ v' = [Y \leftarrow 0]v \end{cases}$$

Time-abstract bisimulation. Time-abstract bisimulation could be defined for two timed automata, but for our purpose, we follow the lines of [22] and define time-abstract bisimulation on a single timed automaton. Let $\mathcal{A} = (\Sigma, Q, T, I, F, X)$ be a timed automaton (over alphabet Σ). We say that a relation $\equiv \subseteq (Q \times \mathbb{T}^X) \times (Q \times \mathbb{T}^X)$ is a *time-abstract bisimulation* whenever it is an equivalence relation satisfying the following conditions:

- if $(\ell_1, v_1) \equiv (\ell_2, v_2)$ and $(\ell_1, v_1) \xrightarrow{d_1} (\ell_1, v_1 + d_1)$ for some $d_1 \in \mathbb{T}$, then there exists $d_2 \in \mathbb{T}$ such that $(\ell_2, v_2) \xrightarrow{d_2} (\ell_2, v_2 + d_2)$ and $(\ell_1, v_1 + d_1) \equiv (\ell_2, v_2 + d_2)$
- if $(\ell_1, v_1) \equiv (\ell_2, v_2)$ and $(\ell_1, v_1) \xrightarrow{a} (\ell'_1, v'_1)$, then there exists (ℓ'_2, v'_2) such that $(\ell_2, v_2) \xrightarrow{a} (\ell'_2, v'_2)$ and $(\ell'_1, v'_1) \equiv (\ell'_2, v'_2)$
- and *vice-versa*.

By definition, such a relation is an equivalence relation, and as such, $\equiv$ is said to have a *finite index* whenever there are finitely many equivalence classes. Informally, from two equivalent configurations, it is possible to do the same discrete actions and/or to wait some amount of time (possibly different in the two configurations) and stay equivalent.

Relation with the region automaton construction.

Proposition 2 *Let $\mathcal{A}$ be a timed automaton and $\mathcal{R}$ a set of regions for the constraints in $\mathcal{A}$. The relation $\{((\ell, v), (\ell, v')) \mid [v]_{\mathcal{R}} = [v']_{\mathcal{R}}\}$ is a time-abstract bisimulation with a finite index.*

Time-abstract bisimulation appears indeed as the right notion corresponding to the region automaton construction and formally justifies everything which has been explained previously. It proves more precisely that the region automaton construction can be used to verify all properties that are invariant by time-abstract bisimulation, *e.g.* reachability properties, safety properties, many untimed properties. However, notice that we can not use directly this construction to verify properties expressed in a timed logic like TCTL because a property like “reaching a state in exactly 5 units of time” is not invariant by time-abstract bisimulation. For these properties a more involved construction is needed which adds a clock for the formula, and then construct a region automaton taking into account this additional clock. We do not develop this construction here but better refer to original articles on the subject [2].

The converse of Proposition 2 also holds and it can be used to prove decidability of timed systems: if for a timed system we can compute a time-abstract bisimulation relation with a finite index, then reachability (and other time-abstract invariant properties) can be decided using a region automaton-like construction. Examples of such constructions can for example be found in [29, 22].

4.4 Partial Conclusion

Timed automata are an interesting model for representing systems with real-time constraints. Despite the infinite number of possible configurations of a timed automaton, model-checking of reachability properties has been proved decidable. This is probably the most fundamental property of timed automata, which has been proved at the beginning of the 90’s by Alur and Dill, and which is the starting point of numerous works on timed models. We have presented in this section the basics of the decidability of timed automata, which relies on a reduction to finite automata: this is fundamental for most of the works on timed systems. It is however worth to notice that not everything can be reduced to the finite automata case. For example (see [7] and also [42]),

- universality (the dual of reachability) is an undecidable problem;
- the class of timed languages accepted by timed automata is not closed under complementation;
- not all timed automata can be determinized, and, in addition, the problem of deciding whether a timed automaton can be determinized is an undecidable problem;

These problems will not be tackled in this tutorial, but we refer to [10] for a survey of (un)decidability results about timed automata.

In the rest of this tutorial, we will mostly consider extensions (or variants) of timed automata and study decidability of these models, and we will also concentrate on algorithmics and implementation aspects. We hope this should help better understanding timed behaviours and timed models.

5 Extensions of Timed Automata

For representing real-life systems, it is much convenient to have expressive and easy-to-use models. We will present in this section several extensions (or variants) of timed automata, and will focus on the decidability of their reachability problem. We will also give some expressiveness results.

A class of systems $\mathcal{S}$ is said *strictly more expressive* than a class of systems $\mathcal{S}'$ whenever there exists S in $\mathcal{S}$ such that no S' in $\mathcal{S}'$ accepts the same language as S, and for every system S' in $\mathcal{S}'$, there exists S in $\mathcal{S}$ which recognizes the same language as S'. A class of systems $\mathcal{S}$ is *as expressive as* $\mathcal{S}'$ whenever for every S in $\mathcal{S}$, there exists S' in $\mathcal{S}'$ which accepts the same language as S.

5.1 Role of Diagonal Clock Constraints

Diagonal constraints (*i.e.* clock constraints of the form $x - y \bowtie c$ where $x,\ y \in X$, $c \in \mathbb{Z}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$) have been first mentioned in the seminal paper of Alur & Dill [7], and are often considered as part of the model of timed automata. We have seen in previous section that diagonal constraints do not add any decidability and complexity problems to the model.

It was known as a folklore result that diagonal constraints can be eliminated from timed automata, and thus that they do not add expressive power to timed automata. A formal proof of this result has been done in [15].

Proposition 3 *For every timed automaton $\mathcal{A}$, possibly with diagonal constraints, there exists a timed automaton $\mathcal{B}$, with only diagonal-free constraints, which recognizes the same language. Note that $\mathcal{B}$ is* ***strongly bisimilar***[2] *to $\mathcal{A}$.*

This construction leads to an exponential (in the number of diagonal constraints) blowup of the number of states of the automaton, and this blowup is unavoidable as timed automata with diagonal constraints are exponentially more succinct than diagonal-free timed automata [19].

[2] Which means they are bisimilar (in a classical way) for actions taken in $\Sigma \cup \mathbb{T}$: if a system can do action, then so can also the other system, and if a system can wait d units of time, then so can also the other system.

5.2 Adding Silent Actions

For finite automata, it is well-known that *silent actions* (also known as ε*-transitions* or *internal actions*) do not add expressive power to finite automata and that they can be eliminated with no blowup in the number of states of the automaton. Silent actions in timed automata have been studied in details in [15], and the situation is far from the one in the untimed framework.

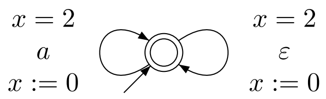
A first (easy) fact is that the region automaton construction can be done in a similar way when there are silent actions, we thus get:

Proposition 4 *The reachability problem is decidable for timed automata with silent actions. The complexity is also* PSPACE*-complete.*

However, and this is at first surprising, silent actions can not be removed, as it is the case for classical finite automata.

Theorem 2 *Timed automata with silent actions are strictly more expressive than classical timed automata.*

Several examples are given in [15]. Among them, there is the language $L = \{(a, t_1) \dots (a, t_i) \cdots \mid \forall i,\ i \mod 2 = 0\}$. This timed language is recognized by the following automaton but is recognized by no timed automaton without silent actions.



Proofs of non-expressivity by a classical timed automaton are always *ad-hoc* as there is no real criterion for a timed language to be recognized by a classical timed automaton. However a sufficient criterium is given in [15]: let $\mathcal{A}$ be a timed automaton possibly with silent actions; if, in $\mathcal{A}$, there is no loop in which a clock is reset on an ε-transition, then ε-transitions can be removed from $\mathcal{A}$, and we can construct a timed automaton $\mathcal{B}$ without ε-transitions which recognizes the same language.

5.3 Adding Additive Clock Constraints

We have seen that diagonal constraints can be used safely in timed automata. A natural idea is then to consider clock constraints of the form $x + y \bowtie c$. Such a constraint will be called an *additive clock constraint*. The model of timed automata which uses classical constraints and additive clock constraints has been studied in [16].

Two clocks. For timed automata with **two** clocks, a region construction can be done. We will not define it precisely here but the region partitioning when the maximal constant is 2 is illustrated on **Fig.** 4. The general case can be easily deduced from this representation.

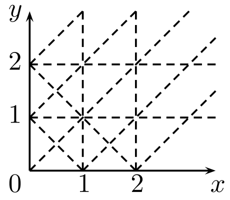
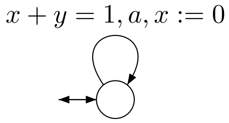


Fig. 4: *Region partitioning for additive clock constraints (two clocks)*

Proposition 5 *The reachability problem for timed automata with at most two clocks and possibly additive clock constraints is decidable.*

The language L^+ represented on **Fig.** 5 is accepted by a timed automaton with two clocks and additive clock constraints but is accepted by no timed automaton with classical clock constraints.



$L^+ = \{(a^n, t_1 \dots t_n) \mid n \geq 1$ and $t_i = 1 - \frac{1}{2^i}\}$

Fig. 5: *A language which needs additive clock constraints*

Four clocks or more. The following result holds for timed automata with four clocks or more, and additive clock constraints:

Theorem 3 *The reachability problem is undecidable for timed automata with four clocks or more, and additive clock constraints.*

This undecidability result is rather involved and is by reduction from the halting problem of a two counter machine [39]. The proof can be found in [16].

What about three clocks? The region graph construction done for two clocks does not extend to three clocks. Using the characterization of regions using time-abstract bisimulation, it has been proven in [41] that there is no finite partitioning satisfying the conditions ①, ② and ③ as soon as there are three clocks (x, y and z) and constraints $\{x+y=1, x=0, z=1\}$ are used. However the reduction presented above (for proving undecidability of reachability checking in timed automata with four clocks and additive clock constraints) can not be adapted if we allow only three clocks. It is still an open problem to know if the reachability problem for timed automata with three clocks and additive clock constraints is decidable or not.

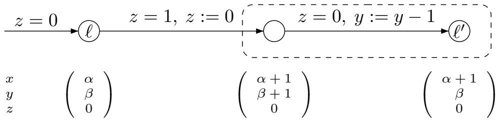
5.4 Adding New Operations on Clocks

Up to now, we can only reset clocks to zero. In [20], models using more general *updates* have been studied. In the model of *updatable timed automata*, a transition is of the form $\ell \xrightarrow{g,a,\mathsf{up}} \ell'$ where g is a clock constraint, a is an action and up is an *update*, *i.e.* for each clock x, an operation up_x of the form $x :\bowtie c$ or $x :\bowtie y+c$ where $c \in \mathbb{Z}$, y is a clock, and $\bowtie \in \{<,\leq,=,\geq,>\}$. Let us take two valuations v and v'. We have that $v' \in \mathsf{up}(v)$ whenever for each clock x, $v'(x) \in \mathsf{up}_x(v)$, where $\mathsf{up}_x(v) =$
$$\begin{cases} \{\alpha \mid \alpha \bowtie c\} & \text{if } \mathsf{up}_x(v) \text{ is } x :\bowtie c \\ \{\alpha \mid \alpha \bowtie v(y)+c\} & \text{if } \mathsf{up}_x(v) \text{ is } x :\bowtie y+c \end{cases}$$
For example, it is possible to decrement the value of a clock by 1, or to set a clock nondeterminiscally at a value less than 2.

This model is very general and it is easy to prove that the reachability problem is not decidable for the whole class of updatable timed automata, by reducing the computation of a two counter machine to the computation of an updatable timed automaton (decrementation (resp. incrementation) of counters are simulated by decrementation (resp. incrementation) of clocks). In [20], tighter undecidable classes and several decidable classes are described. We will not enter into details here, but will present two undecidability proofs and describe one decidable class.

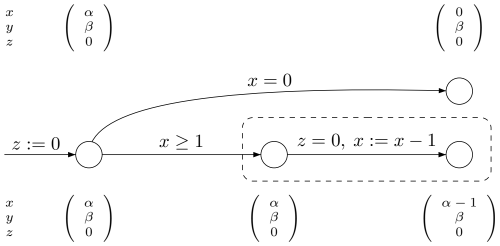
Decrementing clocks leads to undecidability. We now sketch the reduction from a two counter machine to updatable timed automata with resets to zero and decrementation. Let us consider a two counter machine $\mathcal{M}$ with the two counters c and d. We will construct a timed automaton $\mathcal{A}$ (with decrementations and resets to zero) such that the computation of $\mathcal{M}$ terminates if and only if a given state of $\mathcal{A}$ is reachable. The value of counter c (resp. counter d) is encoded by the value of clock x (resp. clock y). An additional clock z is used to rhythm the computation of automaton $\mathcal{A}$. Incrementation (and decrementation) of counters are simulated as follows.

- **Incrementation of counter c.**



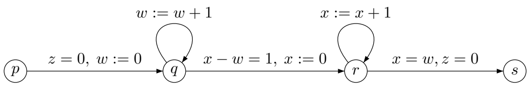
  For incrementing counter c, we let time elapse during one unit of time. The two clocks x and y thus increase by 1. It is then sufficient to decrease clock y by 1: the value of x in ℓ' is equal to the value of x in ℓ plus 1 whereas the value of y in ℓ' is equal to the value of y in ℓ. This correctly encodes an incrementation of c by 1.

- **Decrementation of counter c.**



  An explanation similar to the one for decrementation can be done.

Incrementing clocks also leads to undecidability as soon as diagonal constraints are used... From the previous reduction, it is sufficient to be able to simulate the part of the automaton which is framed with dashed lines, thus to decrease the value of a clock (say x) by 1.



It is easy to see that this module simulates an incrementation.

... but remains decidable when no diagonal constraints are used. We will see that the usual (diagonal-free) region partitioning is correct when also using incrementation of clocks. However this requires a more involved explanation. Indeed, the three conditions ①, ② and ③ are no more sufficient because more general operations on clocks are used. More precisely, we need to replace condition ③ by the following condition (where $\mathcal{R}$ is a finite partitioning of the set of valuations, and $\mathcal{U}$ is a finite set of updates):

③' We say that $\mathcal{R}$ is *compatible with updates in* $\mathcal{U}$ whenever for all $R, R' \in \mathcal{R}$, for each $\mathsf{up} \in \mathcal{U}$,if for some valuation $v \in R$, $\mathsf{up}(v) \cap R' \neq \emptyset$, then for every valuation $v' \in R$, $\mathsf{up}(v') \cap R' \neq \emptyset$.

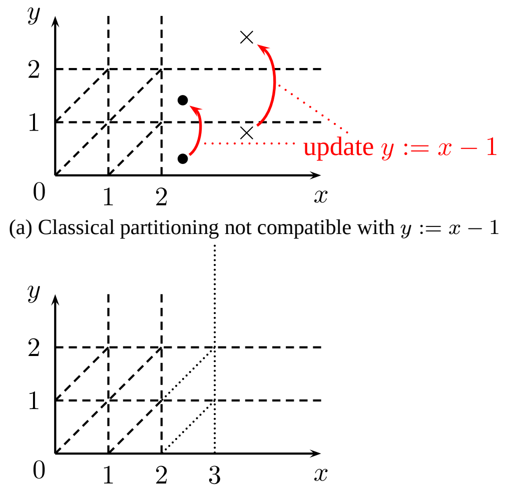
It is just an extension of Proposition 1 to prove that if, for a finite set of constraints $\mathcal{C}$ and a finite set of updates $\mathcal{U}$, we can construct a set of regions satisfying conditions ①, ② and ③', then the region automaton construction can be used to verify reachability (or more generally time-abstract invariant) properties.

Let us fix a finite set $\mathcal{C}$ of diagonal-free constraints, and a finite set of updates $\mathcal{U}$ of the form $x := y + c$ and possibly some resets of clocks. If the system of inequations

$$\begin{aligned} &\{\alpha_x \geq c \mid (x \bowtie c) \text{ is in } \mathcal{C}\} \\ \cup\, &\{\alpha_x \leq \alpha_y + c \mid (x := y + c) \text{ is in } \mathcal{U}\} \end{aligned}$$

has a solution $(m_x)_{x \in X}$, then the diagonal-free set of regions where the maximal constant for x is m_x satisfies the three above-mentioned conditions. Note that if only updates of the form $x := x + 1$ are authorized then, as claimed before, the usual region partitioning is correct (because constraints $\alpha_x \leq \alpha_x + 1$ are trivially true).

However the usual region partitioning needs sometimes to be refined a little bit. Consider the following example: the maximal constant to which the two clocks x and y are compared is 2, both resets of x and y are allowed, and the more elaborated update $y := x - 1$. The system of inequations is $\{\alpha_x \geq 2, \alpha_y \geq 2, \alpha_y \leq \alpha_x - 1\}$. It has a solution, *eg* $\alpha_x = 2$ and $\alpha_y = 3$. We explain the intuition behind these conditions on **Fig.** 6.



(a) Classical partitioning not compatible with $y := x - 1$

(b) Set of regions satisfying conditions ①, ② and ③'.

Fig. 6: *Partitioning for updates* $y := x - 1$

Updatable timed automata have been studied in details in [20], where the precise frontier between decidable and undecidable subclasses has been depicted: among other results, when only diagonal-free constraints are used, decrementation of clocks leads to undecidability whereas incrementation leads to decidability, which may appear as a surprising result. It has also been proved that for every updatable timed automaton belonging to some decidable subclass, we can construct a timed automaton with silent actions (but with an exponential complexity blowup) which recognizes the same timed language.

5.5 Partial Conclusion

We have shortly presented in this section several extensions and variants of timed automata, having in mind the decidability of reachability checking. Many other extensions or subclasses could have been presented as well, for example timed automata with modulo constraints [23], or timed automata with event-predicting or event-recording timed automata [9, 35].

Historically, (linear) hybrid automata [30, 32] have not been defined and studied as an extension of timed automata, but they can be viewed as such. A hybrid automaton is roughly a timed automaton where variables (instead of clocks) grow in every state following some differential equation. Linear hybrid automata are particular hybrid automata where variables evolve following linear differential equations. As soon as a variable has two different slopes, the hybrid automata model is undecidable [32]. In particular, *stopwatch automata*, *i.e.* timed automata in which clocks can be stopped, are undecidable. However, a decidable subclass has been exhibited, the so-called initialized rectangular automata. Hybrid automata are a very interesting model which would require a whole tutorial in itself. We better refer to [40] for an introduction to this model.

6 Algorithmics & Implementation

In practice the region automaton construction is not used in tools. Algorithms for "minimizing" the region automaton have been proposed for example in [3, 4, 43]. However in practice *on-the-fly* technics are preferred.

6.1 Reachability Analysis: Two Methods

There are two main families of (semi-)algorithms for analyzing reachability properties of systems (not only timed systems, but all kinds of systems).

Forward analysis. The general idea of forward analysis is to compute configurations which are reachable from initial configurations within 1 steps, 2 steps, etc... until final states are reached or until the computation terminates.

Backward analysis. The general idea of backward analysis is to compute configurations from which we can reach final configurations within 1 step, 2 steps, etc... until initial configurations are reached or until the computation terminates.

These two generic approaches are used for many models, for example counter machines, hybrid systems, etc... Of course, given a class of systems, specific technics (*e.g.* abstractions, widening operations, etc...) can be used for improving the computation. We will study how these approaches can be used for verifying timed automata.

6.2 Reachability Analysis in Timed Automata: Zones

We need now to look carefully at how the above-mentioned general methods can be used for verifying timed automata. In particular, as timed automata have an infinite number of configurations, we need to use symbolic representations for doing the computation. Given a transition e of a timed automaton $\ell \xrightarrow{g,a,Y} \ell'$, we need to be able to compute, given a set W of valuations, both sets

$$\{v' \mid \exists v \in W \ \exists t \in \mathbb{T} \text{ s.t. } v' = [Y \leftarrow 0](v+t)\}$$

$$\{v \mid \exists v' \in W \ \exists t \in \mathbb{T} \text{ s.t. } [Y \leftarrow 0](v+t) = v'\}$$

It is worth to notice that if the forward computation starts in an initial state with all clocks initialized to 0 or if the backward computation starts from the final states with clocks set to any value (which is sufficient as we are only interested in reachability of discrete states), sets of valuations which are computed are *zones*, *i.e.* sets of valuations defined by a general clock constraint. Recall that general clock constraints are defined by the grammar:

$$g \ ::= \ x \bowtie c \ \mid \ x - y \bowtie c \ \mid \ g \wedge g$$

where $c \in \mathbb{Z}$, $\bowtie \in \{\leq, <, =, >, \geq\}$ and x, y are clocks. A clock constraint g defines a zone $[\![g]\!] = \{v \in \mathbb{T}^X \mid v \models \varphi\}$. For analyzing timed automata, zones are the *symbolic representation* which is commonly used. For implementing forward and backward analysis, we need to be able to perform several operations on zones. From what has been said before, these operations are the following (Z and Z' are supposed to be zones):

- *Future of Z:* $\overrightarrow{Z} = \{v+t \mid v \in Z \text{ and } t \in \mathbb{T}\}$
- *Past of Z:* $\overleftarrow{Z} = \{v-t \mid v \in Z \text{ and } t \in \mathbb{T}\}$
- *Intersection of Z and Z':* $Z \cap Z' = \{v \mid v \in Z \text{ and } v \in Z'\}$
- *Reset to zero of Z w.r.t. set of clocks Y:* $[Y \leftarrow 0]Z = \{[Y \leftarrow 0]v \mid v \in Z\}$
- *Inverse reset to zero of Z w.r.t. set of clocks Y:* $[Y \leftarrow 0]^{-1}Z = \{v \mid [Y \leftarrow 0]v \in Z\}$
- *Test emptiness of Z:* decide whether $Z = \emptyset$

Using these operations, the basic steps of the forward and the backward computations can be rewritten as:

$$\begin{cases} \mathsf{Post}_e(Z) = [Y \leftarrow 0](\overrightarrow{Z} \cap [\![g]\!]) \\ \mathsf{Pre}_e(Z) = \overleftarrow{[Y \leftarrow 0]^{-1}(Z \cap [\![Y=0]\!]) \cap [\![g]\!]} \end{cases}$$

6.3 The DBM Data Structure

For representing zones, the most common data structure which is used is the so-called DBM data structure (where DBM stands for "Difference Bounded Matrice"). This data structure has been first introduced in [17] and then proposed in the framework of timed automata in [28]. Several presentations of this data structure can be found in the literature, for example in [24, 14, 18].

A *difference bounded matrice* (say *DBM* for short) for a set $X = \{x_1, \ldots, x_n\}$ of n clocks is an $(n+1)$-square matrice of pairs

$$(m; \prec) \in \mathbb{V} = (\mathbb{Z} \times \{<, \leq\}) \cup \{(\infty; <)\}.$$

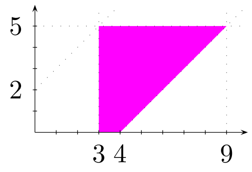
A DBM $M = (m_{i,j}, \prec_{i,j})_{i,j=1\dots n}$ defines the following subset of $\mathbb{T}^n$ (the clock x_0 is supposed to be always equal to zero, *i.e.* for each valuation v, $v(x_0) = 0$):

$$\{v : X \to \mathbb{T} \mid \forall\, i, j,\ v(x_i) - v(x_j) \prec_{i,j} m_{i,j}\}$$

where $\gamma < \infty$ simply means that γ is some real without bound. This subset of $\mathbb{T}^n$ is a zone and will be denoted, in what follows, by $[\![M]\!]$. In what follows, to simplify notations, we will assume that all constraints are non-strict, so that coefficient of DBMs will be elements of $\mathbb{Z} \cup \{\infty\}$.

Example 3 *Consider the zone defined by the constraints* $(x_1 \geq 3) \wedge (x_2 \leq 5) \wedge (x_1 - x_2 \leq 4)$. *This zone, depicted below on the right, can be represented by the DBM below (on the left).*

$$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ x_0 & \infty & -3 & \infty \\ x_1 & \infty & \infty & 4 \\ x_2 & 5 & \infty & \infty \end{array}$$



A zone can have several representations using DBMs. For example, the zone of the previous example can equivalently be represented by the DBM

$$\begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix}$$

A normal form can be defined on DBMs, which tightens all possible constraints. This can be done using a Floyd algorithm on the matrice (viewed as a weighted graph). A zone has a unique representation as a DBM in normal form. Tests like emptiness checking, or comparison of zones can then be done syntactically on the DBMs in normal form. For example, a zone Z is included in a zone Z' if the DBM in normal form representing Z is smaller than the DBM in normal form representing Z'. Finally all operations on zones described in section 6.2 can easily be done on the DBMs, details can be found in all mentioned papers on DBMs.

Let us just mention that the DBM data structure is the most basic data structure which is used for analyzing timed systems, some more involved BDD-like data structures can also be used, for example CDDs (which stands for "Clock Difference Diagrams") [37].

6.4 Backward Analysis

Let $\mathcal{A} = (\Sigma, Q, T, I, F, X)$ be a timed automaton. Backward analysis then consists in computing the following sets of symbolic configurations: $\mathcal{S}_0 = \{(f, \mathbb{T}^X) \mid f \in F\}$, and iteratively $\mathcal{S}_{p+1} = \{(\ell, Z) \mid \exists e = (\ell \xrightarrow{g,a,Y} \ell')\exists(\ell', Z') \in \mathcal{S}_p$ s.t. $Z = \mathsf{Pre}_e(Z')\}, \dots$

Theorem 4 *The backward computation terminates and is correct w.r.t. reachability, i.e. if a state is found reachable by the computation, then it is really reachable.*

Correctness is immediate as the computation is *exact* (as opposed to over-(or under-)approximate). Termination needs some additional argument, related to properties of the region partitioning associated with timed automata. The termination proof then relies on the following lemma, which can be proved as an exercise.

Lemma 2 *Let $\mathcal{A}$ be a timed automaton and let $\mathcal{R}$ be a set of regions satisfying conditions ①, ② and ③ (for $\mathcal{A}$). Consider a finite union of regions $\bigcup_{i=1}^{p} R_i$ (with $R_i \in \mathcal{R}$ for $1 \leq i \leq p$). Then the following holds:*

- $\overleftarrow{\bigcup_{i=1}^{p} R_i}$ *is a finite union of regions*
- $[Y \leftarrow 0]^{-1}(\bigcup_{i=1}^{p} R_i)$ *is a finite union of regions (for any set of clocks Y)*
- $g \cap (\bigcup_{i=1}^{p} R_i)$ *is a finite union of regions if g is a constraint of $\mathcal{A}$ (thus compatible with $\mathcal{R}$)*

Backward analysis thus appears as a very interesting method for analyzing timed systems. However, in practice, most commonly used tools (for example UPPAAL) prefer using a forward analysis procedure. A natural question then arises: what's the problem with backward analysis? It comes from the fact that the use of bounded integer variables really improves and eases the modeling of real systems. Backward analysis is then not suitable for arithmetical operations: for example if we know in which interval lies the variable i and if we know that i is assigned the value $j.k + \ell.m$, it is not easy to compute the possible values of variables j, k, ℓ, m (apart from listing all possible tuples of values). For this kind of operations, forward analysis is much more suitable.

6.5 Forward Analysis

Let $\mathcal{A} = (\Sigma, Q, T, I, F, X)$ be a timed automaton. Forward analysis then consists in computing the following sets of symbolic configurations:

$\mathcal{S}_0 = \{(i,\mathbf{0}) \mid i \in I\}$, and then iteratively $\mathcal{S}_{p+1} = \{(\ell',Z') \mid \exists e = (\ell \xrightarrow{g,a,Y} \ell')\ \exists(\ell,Z) \in \mathcal{S}_p$ s.t. $Z' = \mathsf{Post}_e(Z)\}$, ...The forward analysis gives a correct answer (if it gives an answer), but may not terminate. An example of automaton where the forward computation does not terminate is given on **Fig.** 7. The zones which are computed are represented on the right part of the figure, and it is easy to check that the computation will never terminate.

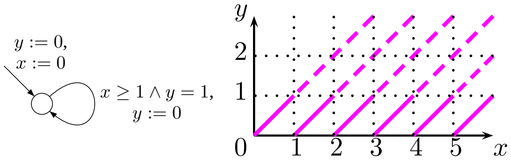


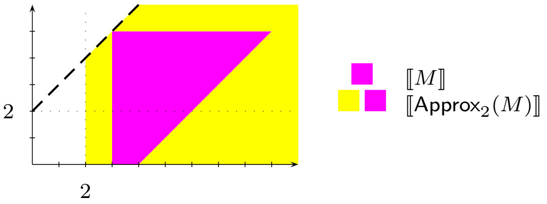
Fig. 7: *Forward computation does not always terminate...*

To overcome this problem, it is necessary to use some abstractions, several are proposed in [26]. For example, if Z and Z' are computed for the location ℓ, zones are replaced by the smallest zone containing both Z and Z': this approximation is called the "*convex-hull*"[3], it does not ensure termination and is only semi-correct w.r.t. reachability in the sense that a state which is announced as reachable may not be reachable. The most interesting abstraction studied in this paper is the *extrapolation* operator.

The extrapolation operator. The abstraction operator which is commonly used is called *extrapolation*, and sometimes *normalization* [14]. We will note it here Approx_k, it is defined up to a constant k as follows: if Z is a zone, $\mathsf{Approx}_k(Z)$ is the smallest k-bounded zone[4] which contains Z. This operation is well-defined on DBMs: if M is a DBM in normal form representing Z, a DBM representing $\mathsf{Approx}_k(Z)$ is M' where each coefficient less than $-k$ is replaced by $-k$ and all coefficients greater than k is replaced by $+\infty$, all other coefficients remain unchanged.

Example 4 *Consider the zone M of Example 3. Its extrapolation w.r.t. 2 is the following DBM:*

$$\mathsf{Approx}_2(M) = \begin{pmatrix} 0 & -2 & 0 \\ 9 & 0 & +\infty \\ +\infty & 2 & 0 \end{pmatrix}$$



Obviously,

- Approx_k is a finite abstraction operator because there are finitely many DBMs whose coefficients are either $+\infty$ or some integer between $-k$ and $+k$
- the computation of Approx_k is effective and can be done easily on DBMs
- Approx_k is a complete abstraction w.r.t. reachability because for every zone Z, $Z \subseteq \mathsf{Approx}_k(Z)$

The only problem stands in the correctness of Approx_k w.r.t. reachability: we have to find a constant k such that this abstraction operator will be correct w.r.t. reachability.

Theorem 5 *Let $\mathcal{A}$ be a **diagonal-free** timed automaton. Take k the maximal constant appearing in the constraints of $\mathcal{A}$. Then Approx_k is correct w.r.t. reachability in $\mathcal{A}$.*

Two different proofs of this theorem can be found in [18] and [12]. Note that this theorem does not extend to timed automata with general clock constraints. See [18] for a counter-example, and [21] for a solution to the problem.

6.6 Tools for Timed Systems

Several tools implement timed (and hybrid) automata.

- HyTech [31] is a model-checker for linear hybrid automata. Exact backward and forward computations can be done, reachability properties can thus be checked (but there is of course no guarantee the computation will terminate). Many other operations on polyhedra can be performed, for example hiding of variables (corresponding to projections), "`while`" loops, emptiness checks, etc... HyTech, which has been developed in Berkeley (USA), can be downloaded on

  `http://www-cad.eecs.berkeley.edu:80/~tah/HyTech/`

[3] It is a language abuse, because it is not reaaly the convex hull of the two zones, but it is the smallest zone containing the convex-hull of the two zones.

[4] A k-bounded zone is a zone defined by a k-bounded clock constraint.

- KRONOS [25] is a model-checker for timed automata. Exact as well as abstract backward and forward computations can be done. A backward procedure for the logic TCTL [2] is also implemented [34]. The tool KRONOS, which has been developed in Grenoble (France), can be downloaded on

  ```
  http://www-verimag.imag.fr/
       TEMPORISE/kronos/
  ```

- UPPAAL [38] is a model-checker for timed automata which performs forward analysis with extrapolation. It can verify reachability properties of timed systems with some extra features as bounded integer variables and broadcast channels. The tool UPPAAL, which is jointly developed in Aalborg University (Denmark) and Uppsala University (Sweden), can be downloaded on

  ```
  http://www.uppaal.com/
  ```

7 Conclusion

In this tutorial we have presented the basic model of timed automata, introduced at the beginning of the 90's by Rajeev Alur and David Dill [7]. One of the most important and most fundamental construction which is used in this domain is the region automaton construction: it finitely abstracts behaviours of timed automata into behaviours of finite automata, which allows to model-check many properties: although we only presented how reachability properties could be checked, properties in TCTL can also be verified using a region-like construction [2]. We have also presented several extensions of timed automata, concentrating on the decidability of the model-checking of reachability properties.

There are so many works which have been devoted to timed systems in general, and timed automata in particular, that it is hopeless to present the whole theory of timed automata in a single tutorial. The current tutorial presents some results on timed automata, focusing on the decidability of reachability properties and on implementation issues for verifying such properties.

References

[1] L. Aceto and F. Laroussinie. Is your model-checker on time ? on the complexity of model-checking for timed modal logics. *Journal of Logic and Algebraic Programming*, 52–53:7–51, 2002.

[2] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[3] R. Alur, C. Courcoubetis, D. Dill, N. Halbwachs, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proc. 13th IEEE Real-Time Systems Symp. (RTSS'92)*, pages 157–166. IEEE Comp. Soc. Press, 1992.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In *Proc. 3rd Int. Conf. Concurrency Theory (CONCUR'92)*, volume 630 of *LNCS*, pages 340–354. Springer, 1992.

[5] R. Alur, C. Courcoubetis, and T. A. Henzinger. The observational power of clocks. In *Proc. 5th Int. Conf. Concurrency Theory (CONCUR'94)*, volume 836 of *LNCS*, pages 162–177. Springer, 1994.

[6] R. Alur and D. Dill. Automata for modeling real-time systems. In *Proc. 17th Int. Coll. Automata, Languages and Programming (ICALP'90)*, volume 443 of *LNCS*, pages 322–335. Springer, 1990.

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

[8] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[9] R. Alur, L. Fix, and T. A. Henzinger. A determinizable class of timed automata. In *Proc. 6th Int. Conf. Computer Aided Verification (CAV'94)*, volume 818 of *LNCS*, pages 1–13. Springer, 1994.

[10] R. Alur and P. Madhusudan. Decision problems for timed automata. In *Proc. 4th Int. School Formal Methods for the Design of Computer, Communication and Software Systems: Real Time (SFM-04:RT)*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004.

[11] E. Asarin. Challenges in timed languages: From applied theory to basic theory. *The Bulletin of the European Association for Theoretical Computer Science*, 83, 2004.

[12] G. Behrmann, P. Bouyer, E. Fleury, and K. G. Larsen. Static guard analysis in timed automata verification. In *Proc. 9th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *LNCS*, pages 254–277. Springer, 2003.

[13] G. Behrmann, A. Fehnker, T. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th Int. Work. Hybrid Systems: Computation and Control (HSCC'01)*, volume 2034 of *LNCS*, pages 147–161. Springer, 2001.

[14] J. Bengtsson. *Clocks, DBMs ans States in Timed Systems*. PhD thesis, Department of Information

Technology, Uppsala University, Uppsala, Sweden, 2002.

[15] B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3):145–182, 1998.

[16] B. Bérard and C. Dufourd. Timed automata and additive clock constraints. *Information Processing Letters*, 75(1–2):1–7, 2000.

[17] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time Petri nets. In *Proc. IFIP 9th World Computer Congress*, volume 83 of *Information Processing*, pages 41–46. North-Holland/ IFIP, 1983.

[18] P. Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

[19] P. Bouyer and F. Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 2005. To appear.

[20] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2–3):291–345, 2004.

[21] P. Bouyer, F. Laroussinie, and P.-A. Reynier. Diagonal constraints in timed automata — Forward analysis of timed systems. In *Proc. 3rd Int. Work. Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, LNCS. Springer, 2005. To appear.

[22] T. Brihaye, V. Bruyère, and J.-F. Raskin. Model-checking for weighted timed automata. In *Proc. Joint Conf. Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, volume 3253 of *LNCS*, pages 277–292. Springer, 2004.

[23] C. Choffrut and M. Goldwurm. Timed automata with periodic clock constraints. *Journal of Automata, Languages and Combinatorics*, 5(4):371–404, 2000.

[24] E. Clarke, O. Grumberg, and D. Peled. *Model-Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[25] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Proc. Hybrid Systems III: Verification and Control (1995)*, volume 1066 of *LNCS*, pages 208–219. Springer, 1996.

[26] C. Daws and S. Tripakis. Model-checking of real-time reachability properties using abstractions. In *Proc. 4th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *LNCS*, pages 313–329. Springer, 1998.

[27] F. Demichelis and W. Zielonka. Controlled timed automata. In *Proc. 9th Int. Conf. Concurrency Theory (CONCUR'98)*, volume 1466 of *LNCS*, pages 455–469. Springer, 1998.

[28] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. of the Work. Automatic Verification Methods for Finite State Systems (1989)*, volume 407 of *LNCS*, pages 197–212. Springer, 1990.

[29] T. A. Henzinger. Hybrid automata with finite bisimulations. In *Proc. 22nd Int. Coll. Automata, Languages and Programming (ICALP'95)*, volume 944 of *LNCS*, pages 324–335. Springer, 1995.

[30] T. A. Henzinger. The theory of hybrid automata. In *Proc. 11th Ann. Symp. Logic in Computer Science (LICS'96)*, pages 278–292. IEEE Comp. Soc. Press, 1996.

[31] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model-checker for hybrid systems. *Journal on Software Tools for Technology Transfer*, 1(1–2):110–122, 1997.

[32] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

[33] T. A. Henzinger, P. W. Kopke, and H. Wong-Toi. The expressive power of clocks. In *Proc. 22nd Int. Coll. Automata, Languages and Programming (ICALP'95)*, volume 944 of *LNCS*, pages 417–428. Springer, 1995.

[34] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[35] T. A. Henzinger, J.-F. Raskin, and P.-Y. Schobbens. The regular real-time languages. In *Proc. 25th Int. Coll. Automata, Languages and Programming (ICALP'98)*, volume 1443 of *LNCS*, pages 580–591. Springer, 1998.

[36] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[37] K. G. Larsen, J. Pearson, C. Weise, and W. Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.

[38] K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *Journal of Software Tools for Technology Transfer*, 1(1–2):134–152, 1997.

[39] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall Int., 1967.

[40] J.-F. Raskin. *An Introduction to Hybrid Automata*, chapter Handbook of Networked and Embedded Control Systems, pages 491–518. Springer, 2005.

[41] A. Robin. Aux frontières de la décidabilité... Master's thesis, DEA Algorithmique, Paris, 2004.

[42] S. Tripakis. Folk theorems on the determinization and minimization of timed automata. In *Proc. 1st Int. Work. Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *LNCS*, pages 182–188. Springer, 2003.

[43] S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *Formal Methods in System Design*, 18(1):25–68, 2001.