Verification of real-time systems

Patricia Bouyer-Decitre

MPRI, Academic year 2023–2024

1 Introduction

1.1 The growing importance of computerized systems and their verification

In thirty years computerized systems have widely spread in our society, from ubiquitous electronic appliances (communication devices, automotive equipment, *etc*), to internet transactions (e-banking, e-business, *etc*), to new technologies (like wireless communications), and to critical systems (medical devices, industrial plants, *etc*). Due to their rapid development, such systems have become more and more complex, and unfortunately this development has come with many bugs, from arithmetic overflow (which caused the crash of the Ariane 5 rocket in 1996) to race conditions (which caused the lethal dysfunction of the Therac-25 radiotherapy machine in the late 80's) or infinite loops (for instance the leap-year bug turning all Zune MP3 devices off on 31 December 2008, see Algorithm 1). Many of those bugs could have been avoided if implemented softwares had been formally verified prior to their use. The need for formal methods for verifying and certifying computer-driven systems is therefore blatant.

Algorithm 1: Zune3.0 leap-year code

```
begin
    year = ORIGINYEAR ;                                  /* = 1980 */
    while (days > 365) do
        if (IsLeapYear(year)) then
            if (days > 366) then
                days -= 366;
                year += 1;
            end
        else
            days -= 365;
            year += 1;
        end
    end
end
```

1.2 The model-checking approach to verification

As part of the effort that has been made for the development of reliable computerized systems, several verification approaches have been developed, among which the so-called model-checking technique. Model-checking is a model-based approach to verification, which goes back to the late 70's [Pnu77,CE82,QS82]. It has been successfully applied to numerous industrial cases, see the webpages of tools like SPIN [Spi], PRISM [Pri] or UPPAAL [Upp]. This approach to verification has been distinguished by two Turing awards: Amir Pnueli received the award in 1996 for "seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification",[1] and Edmund M. Clarke, E. Allen Emerson and Joseph Sifakis jointly received the award in 2007 for making model-checking "a highly effective verification technology".[2]

Given a system $\mathcal{S}$ and a property P, the model-checking approach consists in constructing a mathematical model $\mathcal{M}_{\mathcal{S}}$ for the system and a mathematical model φ_P for the property, for which we will be able to *automatically* check that $\mathcal{M}_{\mathcal{S}}$ satisfies φ_P. If the models $\mathcal{M}_{\mathcal{S}}$ and φ_P are accurate enough with respect to $\mathcal{S}$ and P respectively, we will deduce with confidence that the system $\mathcal{S}$ satisfies the property P. This approach requires the development of expressive modelling formalisms (to increase faithfulness of models) and efficient algorithms.

There are several books on model-checking: [CGP99,SBB$^+$01,AILS08,BK08].

1.3 Expressive classes of models

Basic models for verification are finite-state systems. Those might not be expressive enough. These last twenty years a huge effort has been made to design expressive models for representing computerized systems. As part of this effort the model of timed automata has been proposed in the early nineties [AD94] by Alur and Dill, as a suitable model for representing systems with real-time constraints.[3] Numerous works have focused on that model, and it has received an important tool support, with for instance the development of tools like UPPAAL, KRONOS, RED or IF. The success of timed-automata-based technology is witnessed not only by numerous success stories but also by the award given in 2008[4] by the computer-aided verification community to Alur and Dill for their "fundamental contributions to the theory of real-time systems verification".

Given the success of the timed-automata-based technology for verifying real-time systems tremendously many extensions have been proposed, with the aim of representing the systems more faithfully. They include timed games, which can model control problems, priced timed automata, which can e.g. model energy consumption, stochastic extensions of timed automata, which can model randomized aspects of systems and uncertainties.

[1] `http://awards.acm.org/citation.cfm?id=4725172&srt=year&year=1996&aw=140&ao=AMTURING&yr=1996`

[2] `http://www.acm.org/press-room/news-releases/turing-award-07/`

[3] Time naturally appears in real systems (autonomous systems, GPS, communication protocols, processors, schedulers, *etc*) and in properties (e.g. bounded-response time).

[4] `http://www.princeton.edu/cav2008/cav_award_announce.html`

1.4 Discrete *vs* dense time

How can we add time information to behaviours? An untimed behaviour is a finite or infinite sequence of events:

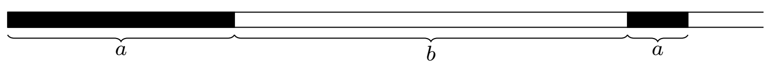
$$(ab)^n \text{ or } (ab)^\omega \text{ with } a, b \in \Sigma \text{ (finite alphabet)}$$

Time can be integrated as:

- explicit delays, that is as an alternating sequence of events and delays:

$$1.5\ a\ 2\ b\ 3.6\ a \cdots \text{ with } a, b, c \in \Sigma$$

- durations of events/atomic propositions:



- events with dates:

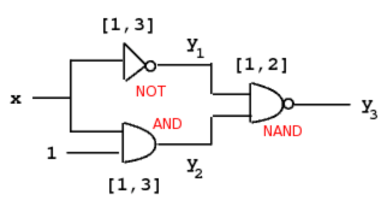
$$(a, 1.5)(b, 3.6)(a, 4.8)$$

We will choose this last formalism.

There are also several possible semantics for the time: the time domain can either be discrete (like $\mathbb{N}$) or dense (like $\mathbb{Q}_+$ or $\mathbb{R}_+$).

Why will we consider dense time [Alu91]? Dense time is more general than discrete time. But can we not always discretize? We will discuss this issue in the context of asynchronous circuits [BS91].

Example 1. We consider the following circuit:



Intervals labelling gates represent sets of possible delays before stabilization of the gate. For instance, if $x = 0$ and $y_1 = 1$ (which is a stable configuration), when x is set to 1, gate y_1 will be updated to 0, but this will take a delay from 1 up to 3 time units.

We start with input $x = 0$. The corresponding stable configuration is $y = [101]$. The input x changes to 1. The corresponding stable configuration is $y = [011]$. One of the possible behaviours to reach that stable configuration is (values represent absolut time):

$$[\underline{10}1] \xrightarrow[1.2]{y_2} [\underline{1}1\underline{1}] \xrightarrow[2.5]{y_3} [\underline{1}10] \xrightarrow[2.8]{y_1} [01\underline{0}] \xrightarrow[4.5]{y_3} [011]$$

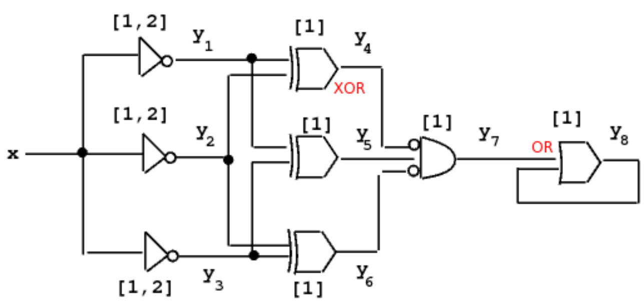
(an underlined value is not correct)

In this circuit, the set of reachable configurations is

$$\{[101],[111],[110],[010],[011],[001]\}$$

┘

Example 2. Consider now the following circuit:



We will see that this circuit **is not** 1-discretizable.

We assume that initially, $x = 0$, and $y = [11100000]$ (stable configuration). The input x is set to 1.

– In dense time, the following behaviour is possible:

$$[\underline{111}00000] \xrightarrow[1]{y_1} [0\underline{1100}000] \xrightarrow[1.5]{y_2} [00\underline{1}0\underline{00}000] \xrightarrow[2]{y_3,y_5} [0000\underline{1}0\underline{0}0] \xrightarrow[3]{y_5,y_7} [000000\underline{10}] \xrightarrow[4]{y_7,y_8} [00000001]$$

– In discrete time (granularity 1), the possible behaviours are (up to permutation):
  - $[\underline{111}00000] \xrightarrow[1]{y_1,y_2,y_3} [00000000]$
  - $[\underline{111}00000] \xrightarrow[1]{y_1} [0\underline{1100}000] \xrightarrow[2]{y_2,y_3,y_4,y_5} [000\underline{11}000] \xrightarrow[3]{y_4,y_5} [0\underline{1111}000] \xrightarrow[2]{y_2,y_3,y_4,y_5} [00000000]$
  - $[\underline{111}00000] \xrightarrow[1]{y_1,y_2} [00\underline{1}0\underline{00}000] \xrightarrow[2]{y_3,y_5,y_6} [0000\underline{11}00] \xrightarrow[3]{y_5,y_6} [00000000]$
  - $[\underline{111}00000] \xrightarrow[1]{y_1,y_3} [0\underline{1}0\underline{1}0\underline{1}00] \xrightarrow[2]{y_2,y_4,y_6} [00000000]$

  Thus, the state [00000001] is not reachable in discrete time.

We will admit the following theorem.

Theorem 1 ([BS91]). *For every $k \geq 1$, there exists a digital circuit such that the reachability set of states in dense-time is strictly larger than the one in discrete time with granularity $\frac{1}{k}$.*

Claim. Finding a correct granularity is as difficult as analysing the system in dense time.

We will now define one of the most prominent models for real-time systems, the model of timed automata. There exist other models, like time(d) Petri nets.

2 The timed automaton model

2.1 Preliminary notations

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* $w = (a_i, t_i)_{1 \le i \le p}$ is an element of $(\Sigma \times \mathbb{T})^*$, also written as a pair $w = (\sigma, \tau)$, 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. In the following, we denote by $\mathsf{Untime}(w)$ the finite word σ over the alphabet Σ.

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 $\mathbb{T}^X$, and $\mathbf{0}_X$ denotes the valuation assigning 0 to every clock $x \in X$.

Let $v \in \mathbb{T}^X$ be a valuation and $t \in \mathbb{T}$, the valuation $v+t$ is defined by $(v+t)(x) = v(x)+t$ for every $x \in X$. For a subset Y of X, we denote by $[Y \leftarrow 0]v$ the valuation such that for every $x \in Y$, $([Y \leftarrow 0]v)(x) = 0$ and for every $x \in X \setminus Y$, $([Y \leftarrow 0]v)(x) = v(x)$.

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:

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

Remark 1. We could allow rational constants in clock constraints (*i.e.*, have $c \in \mathbb{Q}$ in the above grammar), everything that follows would still hold, but developments would then be a bit more technical.

A clock constraint of the form $x - y \bowtie c$ is said *diagonal*. Next we also use the proper subset of *diagonal-free* clock constraints where the diagonal constraints are not allowed. This set, denoted $\mathcal{C}_{\mathit{df}}(X)$, is defined by the grammar:

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

A *k-bounded* clock constraint is a clock constraint which involves only (integral) 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. $\mathcal{C}^k_{\mathit{df}}(X)$).

If $v \in \mathbb{T}^X$ 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 write $[\![g]\!]_X$ the set of clock valuations $\{v \in \mathbb{T}^X \mid v \models g\}$.

Example 3. The valuation v over $\{x, y\}$ such that $v(x) = 4.1$ and $v(y) = 0$ satisfies the constraint $(x \le 5) \wedge (x - y > 3)$. ⌟

2.2 The model of timed automata

A *timed automaton* over $\mathbb{T}$ is a tuple $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T)$, where L is a finite set of locations, $L_0 \subseteq L$ is the set of initial locations, $L_F \subseteq L$ is the set of final locations, X is a finite set of clocks, Σ is a finite alphabet of actions, and $T \subseteq L \times \mathcal{C}(X) \times \Sigma \times 2^X \times L$ is a finite set of transitions.[5] If all constraints appearing in $\mathcal{A}$ are diagonal-free (*i.e.* are in $\mathcal{C}_{df}(X)$), we say that $\mathcal{A}$ is a *diagonal-free* timed automaton.

For modelling purpose some definitions assume invariants in the model. Invariants are clock constraints assigned to locations which have to be satisfied while the system is in the location: they restrict time elapsing and may enforce the system be live. A *timed automaton with invariants* is a tuple $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T, \mathsf{Inv})$ where $(L, L_0, L_F, X, \Sigma, T)$ is a timed automaton in the previous sense, and $\mathsf{Inv}\colon L \to \mathcal{C}_{df}(X)$, the invariant, assigns a clock constraint to every location. A timed automaton as defined initially is a special case of a timed automaton with invariants where the invariant assigns true to every location. In the sequel when we speak of timed automata we will equivalently mean timed automata with or without invariants.

Several semantics can be given to timed automata. We first give an operational semantics to timed automata, and then give a language-based semantics.

Semantics as a timed transition system. We first give an operational semantics as a timed transition system. Let $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T, \mathsf{Inv})$ be a timed automaton. Its operational semantics is given as the *timed transition system* $\mathcal{T}_\mathcal{A} = (S, S_0, S_F, \to)$ over alphabet Σ, where the set of states is $S = \{(\ell, v) \in L \times \mathbb{T}^X \mid v \models \mathsf{Inv}(\ell)\}$, the set of initial states is $S_0 = \{(\ell_0, \mathbf{0}_X) \in S \mid \ell_0 \in L_0\}$, the set of final states is $S_F = S \cap (L_F \times \mathbb{T}^X)$,[6] and $\to \subseteq S \times (\mathbb{T} \cup \Sigma) \times S$ is the set of moves defined as follows:

- *delay moves*: if $(\ell, v) \in S$ and $d \in \mathbb{T}$, there is a move $(\ell, v) \xrightarrow{d} (\ell, v + d)$ whenever for every $0 \leq d' \leq d$, $(\ell, v + d') \in S$ (*i.e.* $v + d' \models \mathsf{Inv}(\ell)$);
- *action moves*: if $(\ell \xrightarrow{g,a,Y} \ell') \in T$ and $(\ell, v) \in S$, there is a move $(\ell, v) \xrightarrow{a} (\ell', [Y \leftarrow 0]v)$ whenever $v \models g$ and $([Y \leftarrow 0]v) \models \mathsf{Inv}(\ell')$. In that case we say that the action move is associated with the transition $\ell \xrightarrow{g,a,Y} \ell'$.

Next it will be more convenient to consider *mixed moves*, and hence to consider the timed transition system $\mathcal{T}^m_\mathcal{A} = (S, S_0, \to)$ where S and S_0 are defined as in $\mathcal{T}_\mathcal{A}$, and $\to \subseteq S \times (\mathbb{T} \times \Sigma) \times S$. In that case a move is composed of a delay move directly followed by an action move: $(\ell, v) \xrightarrow{d,a} (\ell', v')$ is a mixed move if $(\ell, v) \xrightarrow{d} (\ell, v + d)$ is a delay move and $(\ell, v + d) \xrightarrow{a} (\ell', v')$ is an action move.

Remark 2. We will later use this semantics for defining behavioural equivalences (and preorders).

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

[6] Note that S_F is optional, but it will be useful to have it later.

Semantics as a timed language. We now give a language-based semantics to a timed automaton. Let $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T, \mathsf{Inv})$ be a timed automaton. A *path* in $\mathcal{A}$ is a finite sequence of consecutive transitions:

$$P = \ell_0 \xrightarrow{g_1,a_1,Y_1} \ell_1 \xrightarrow{g_2,a_2,Y_2} \ \dots \ \ell_{p-1} \xrightarrow{g_p,a_p,Y_p} \ell_p$$

where $\ell_{i-1} \xrightarrow{g_i,a_i,Y_i} \ell_i \in T$ for every $1 \le i \le p$. The path is said to be *accepting* if it starts in an initial location ($\ell_0 \in L_0$) and ends in a final location ($\ell_p \in L_F$). A *run* of the automaton along the path P is a sequence of the form:

$$\varrho = (\ell_0, v_0) \xrightarrow{d_1,a_1} (\ell_1, v_1) \xrightarrow{d_2,a_2} \ \dots \ (\ell_{p-1}, v_{p-1}) \xrightarrow{d_p,a_p} (\ell_p, v_p)$$

where $v_0 \models \mathsf{Inv}(\ell_0)$, and for each $1 \le i \le p$, $(\ell_{i-1}, v_{i-1}) \xrightarrow{d_i,a_i} (\ell_i, v_i)$ is a mixed move of the timed transition system $\mathcal{T}^m_{\mathcal{A}}$ (see Section 2.2) associated with the transition $\ell_{i-1} \xrightarrow{g_i,a_i,Y_i} \ell_i$. The run is *accepting* if the underlying path is accepting and if $v_0 = \mathbf{0}_X$.

The label of the run ϱ is the timed word $w = (a_1, t_1) \dots (a_p, t_p)$ where for each $1 \le i \le p$, $t_i = \sum_{j=1}^{i} d_j$ is the absolute time at which the i-th action a_i occurs (we also say that run ϱ *reads* the timed word w). If the run ϱ 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 $L(\mathcal{A})$ and is the *timed language accepted (or equivalently recognized) by* $\mathcal{A}$.

Remark 3. In these notes, we mostly consider finite paths, finite runs and finite timed words, but as for classical finite automata, one could consider infinite timed words and ω-regular accepting conditions (Büchi, Muller, etc...), see [AD94].

2.3 An example of a timed automaton

We consider the (diagonal-free) timed automaton in Figure 1 with two clocks x and y and over the alphabet $\{\texttt{problem}, \texttt{delayed}, \texttt{repair}, \texttt{done}\}$. It has four locations ('safe', 'alarm', 'failsafe', 'repairing'), and location 'safe' is both initial and final. Transitions are depicted in a standard way as labelled edges between locations. The transition between 'alarm' and 'repairing' has constraint $x < 15$, action label `repair` and resets clock y. A state of this automaton is a location, and a valuation for the two clocks x and y. The following is a sequence of delay and action moves in the timed transition system of the above timed automaton (we write a state vertically, with the name of the location on top, then the value of clock x and the value of clock y at the bottom):

$$\begin{array}{ccccccccccc} & \text{safe} & \xrightarrow{23} & \text{safe} & \xrightarrow{\texttt{problem}} & \text{alarm} & \xrightarrow{15.6} & \text{alarm} & \xrightarrow{\texttt{delayed}} & \text{failsafe} & \\ x & 0 & & 23 & & 0 & & 15.6 & & 15.6 & \cdots \\ y & 0 & & 23 & & 23 & & 38.6 & & 0 & \end{array}$$

$$\begin{array}{cccccccccc} & \text{failsafe} & \xrightarrow{2.3} & \text{failsafe} & \xrightarrow{\texttt{repair}} & \text{repairing} & \xrightarrow{22.1} & \text{repairing} & \xrightarrow{\texttt{done}} & \text{safe} \\ \cdots & 15.6 & & 17.9 & & 17.9 & & 40 & & 40 \\ & 0 & & 2.3 & & 0 & & 22.1 & & 22.1 \end{array}$$

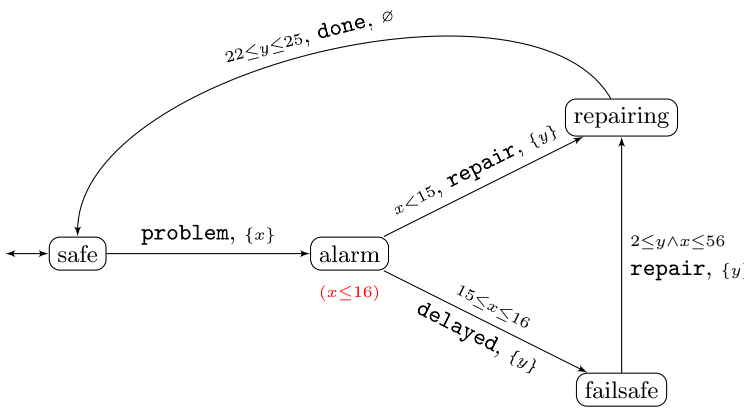


Fig. 1: *An example of a timed automaton*

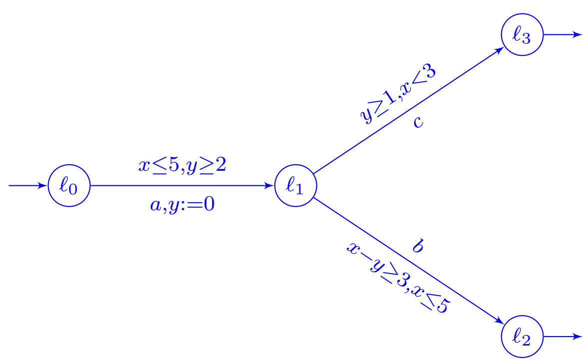
It corresponds to the following accepting run of the timed automaton (with the same convention for the representation of states):

$$\begin{array}{cccccccccc} & \text{safe} & \xrightarrow{23,\texttt{problem}} & \text{alarm} & \xrightarrow{15.6,\texttt{delayed}} & \text{failsafe} & \xrightarrow{2.3,\texttt{repair}} & \text{repairing} & \xrightarrow{22.1,\texttt{done}} & \text{safe} \\ x & 0 & & 0 & & 15.6 & & 17.9 & & 40 \\ y & 0 & & 23 & & 0 & & 0 & & 22.1 \end{array}$$

This run reads (and accepts) the timed word

$$(\texttt{problem}, 23)(\texttt{delayed}, 38.6)(\texttt{repair}, 40.9), (\texttt{done}, 63)$$

Exercise 1. Consider the following timed automata $\mathcal{A}$:



Do the timed words $w = (a, 4.7)(b, 4.9)$ and $w' = (a, 2.1)(c, 3.1)$ belong to $L(\mathcal{A})$? Justify your answer. ⌟

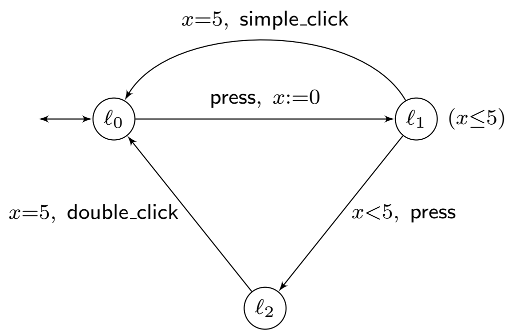
Exercise 2. Build a timed automaton which recognizes the following timed language over the single-letter alphabet $\Sigma = \{a\}$

$$L = \{(a, t_1) \ldots (a, t_n) \mid n > 0 \text{ and } \exists 1 \leq i < j \leq n \text{ s.t. } t_j = t_i + 1\}$$

What do you think of the complement of this timed language? ⌟

Exercise 3. Build a model for the computer mouse (click and double click), assuming double click should happen within 5 time units. ⌟

A possible solution:



2.4 Parallel composition of timed systems

Modelling a system with a unique timed automaton is not always convenient. A solution is to define small components of the system, and show how they communicate and interact. This is the role of the parallel composition.

Let $(\mathcal{A}_i)_{1\le i\le n}$ be n timed automata, with $\mathcal{A}_i = (L_i, L_{i,0}, L_{i,F}, X_i, \Sigma_i, T_i, \mathsf{Inv}_i)$. We assume that all X_i's are disjoint. If Σ is a new alphabet, given a (partial) synchronization function $f : \prod_{i=1}^n (\Sigma_i \cup \{\bullet\}) \setminus \{(\bullet, \dots, \bullet)\} \to \Sigma$, the *synchronized product* (or *parallel composition*) $(\mathcal{A}_1 \parallel \cdots \parallel \mathcal{A}_n)_f$ is the timed automaton $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T, \mathsf{Inv})$ where $L = L_1 \times \dots L_n$, $L_0 = L_{1,0} \times \dots L_{n,0}$, $L_F = L_{1,F} \times \dots L_{n,F}$, for every $(\ell_1, \dots, \ell_n) \in L_1 \times \dots L_n$, $\mathsf{Inv}((\ell_1, \dots, \ell_n)) = \bigwedge_{i=1}^{n} \mathsf{Inv}_i(\ell_i)$, $X = \bigcup_{i=1}^n X_i$, and the set T is composed of the transitions $(\ell_1, \dots, \ell_n) \xrightarrow{g,a,Y} (\ell'_1, \dots, \ell'_n)$ such that there exists $(a_1, \dots, a_n) \in f^{-1}(a)$ where:

- $a_i = \bullet$ implies $\ell_i = \ell'_i$;
- $a_i \neq \bullet$ implies that there exists $\ell_i \xrightarrow{g_i,a_i,Y_i} \ell'_i$ in T_i;
- $g = \bigwedge\{g_i \mid a_i \neq \bullet\}$ and $Y = \bigcup\{Y_i \mid a_i \neq \bullet\}$;

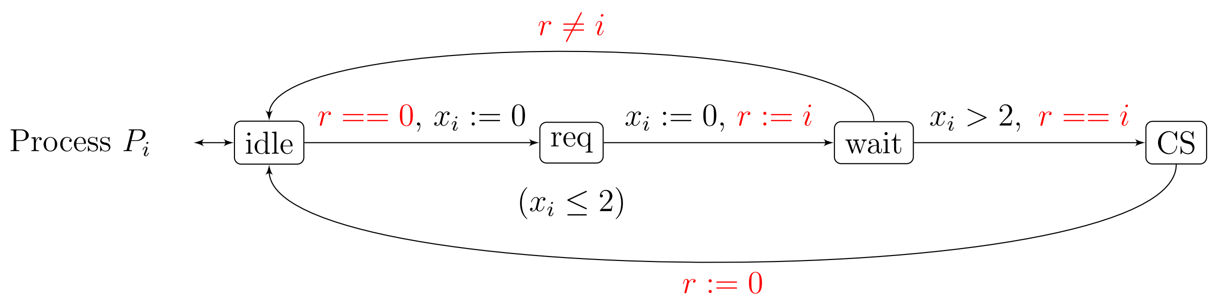
Remark 4. For modelling purpose we may use finitely-valued variables, and use shared variables. We however do not enter into more details here.

Example 4 (Fischer's protocol). This involves n participants. Each participant wants to access a shared resource. This is a mutual exclusion protocol, that is, no two participants should access the shared resource at the same time. The protocol is described as Algorithm 2.

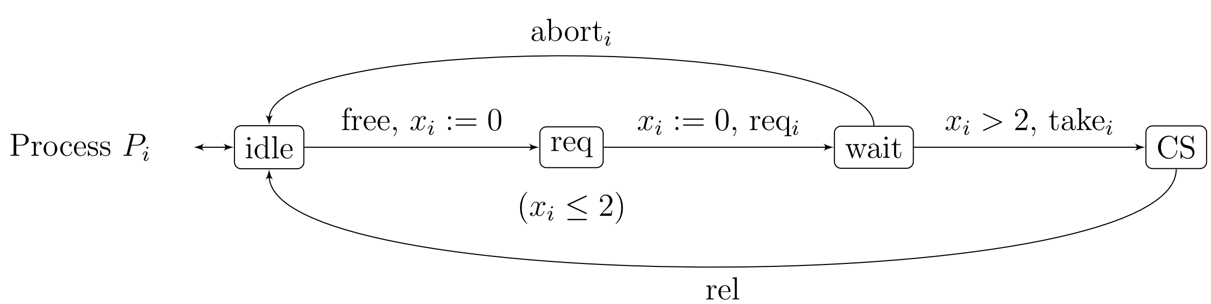
We propose a first modelization using *shared variable* r (which is not allowed in our syntax of timed automata).

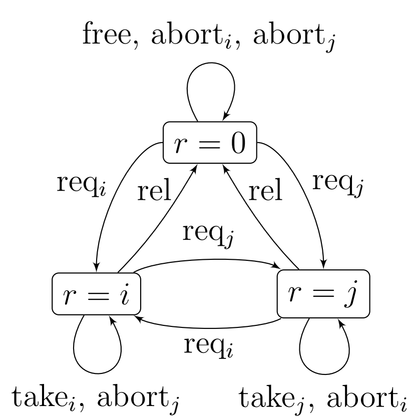
Algorithm 2: Fischer's protocol

```
begin
    r: shared variable;
    repeat
        repeat
            await(r == 0) ;                          /* takes at most 2 time units */
            r := i;
            delay at least 2 time units;
        until (r == i);
        access the critical section;
        r := 0;
    until forever;
end
```



The shared variable r can be replaced by an automaton representing the resource, which communicates (*via* a synchronization function) with the processes.





Exercise 4 (The train crossing example). A gate is controlled by a controller, which sends signals (GoDown and GoUp) for closing or opening the barrier of the gate. Once such a signal is received by the gate, 10 time units are required for processing the action. The trains communicate with the controller. A train sends a signal Approach to the controller when it is approaching the gate and Exit when it is exiting the gate. After having sent signal Approach, the train will be on the gate within 20 and 30 time units. It will exit the gate within 10 and 20 time units later.

The controller receives the Approach and Exit signals from the train, and sends the GoUp and GoDown instructions to the gate. After having received signal Approach the controller sends quickly (in no more than 10 time units) the signal GoDown, and whenever it receives the signal Exit it sends instruction GoUp to the gate precisely 20 time units later.

Give a model of the system, with a single train.

Here are some properties that we could check on that model:

(Safety) is the gate closed when the train crosses the road?
(Liveness) is the gate always closed for less than 50 time units? ⌟

Exercise 5. Build a model for a 2-floor (or 3-floor!) lift as a product of several small automata. It will be made of the cabin, one door at each floor, one button at each floor (or two directional buttons), 3 buttons in the cabin, and a controller. ⌟

Exercise 6. Four adventurers want to cross a damaged bridge in the middle of the night. The bridge can only carry two of the adventurers at a time, and to find the way over the bridge, the adventurers need to bring a torch. The adventurers need respectively 5, 10, 20 and 25 minutes (one-way) to cross the bridge. Does a schedule exist which allows the four adventurers to get the four of them on the other side of the river, within 60 minutes?

Model this problem using (a product of) timed automata. ⌟

3 Reachability analysis, why and how?

For verification purposes, the most fundamental properties that one should be able to verify on a model are reachability properties: basic safety properties (like 'avoid some bad states of the system') can be expressed as (non-)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 we will focus on (location) reachability properties of the form: "Is the set of final locations of a timed automaton $\mathcal{A}$ reachable from an initial state?" *I.e.*, "is there a run starting in an initial state leading to some final location ℓ?" There is no requirement as for the values of the clocks when reaching location ℓ (note however that any clock constraint could be added as a requirement). This problem is equivalent to the so-called *emptiness problem* (from a language-theoretical perspective), where the question is whether the language accepted by a timed automaton is empty or not.

Several classes of systems are known to be decidable, and the simplest such class of systems is that of finite automata, whose emptiness problem is known to be NLOGSPACE-complete [HU79]. The case of timed automata is much harder as there are potentially infinitely many states (recall that a state is a pair (ℓ, v) where ℓ is a location and v is a valuation). Techniques used for finite automata can thus not be applied to timed automata, and specific symbolic techniques and abstractions have to be developed.

In the next section we present the basic technics for solving the decidability of the emptiness problem in timed automata. The idea will be to build a finite abstraction of the timed automaton (actually a finite automaton), which preserves time-abstract properties (and in particular reachability properties).

4 The region abstraction: A key for decidability

The region automaton construction is due to Rajeev Alur and David Dill [AD90,AD94]. It is an abstraction of a timed automaton into a finite automaton which preserves many interesting properties of the system. In this section we present a more abstract version of this construction, which has been introduced in a wider context in [BDFP04]. This approach decouples the main arguments from the technicalities. As we focus on reachability properties we assume w.l.o.g. that timed automata have no invariants (they can be transferred to constraints on incoming and outgoing transitions while preserving reachability properties).

4.1 A tool: time-abstract bisimulation

Viewing the semantics of timed automata as timed transition systems allows to define behavioural equivalence relations (or pre-orders).

Standard bisimulation on finite transition systems. We first recall the notion of bisimulation. Let $\mathcal{T} = (S, S_0, S_F, \rightarrow)$ and $\mathcal{T}' = (S', S'_0, S'_F, \rightarrow')$ be two finite transition systems (corresponding *eg* to finite automata) over the same alphabet Σ (meaning that $\rightarrow \subseteq S \times \Sigma \times S$ and $\rightarrow' \subseteq S' \times \Sigma \times S'$). Let $\mathfrak{R} \subseteq S \times S'$ be a relation. It is a *bisimulation relation* whenever the following condition holds:

- if $s_1 \xrightarrow{a} s_2$ and $(s_1, s'_1) \in \mathfrak{R}$, then there exists $s'_2 \in S'$ such that $s'_1 \xrightarrow{a}{}' s'_2$ and $(s_2, s'_2) \in \mathfrak{R}$;
- if $s'_1 \xrightarrow{a}{}' s'_2$ and $(s_1, s'_1) \in \mathfrak{R}$, then there exists $s_2 \in S$ such that $s_1 \xrightarrow{a} s_2$ and $(s_2, s'_2) \in \mathfrak{R}$.

We can schematize the first condition as follows:

$$\forall \quad \begin{array}{ccc} s_1 & \xrightarrow{\quad a \quad} & s_2 \\ \mathfrak{R} & & \\ s'_1 & & \end{array} \qquad \Rightarrow \qquad \exists \quad \begin{array}{ccc} s_1 & \xrightarrow{\quad a \quad} & s_2 \\ \mathfrak{R} & & \mathfrak{R} \\ s'_1 & \xrightarrow{\quad a \quad} & s'_2 \end{array}$$

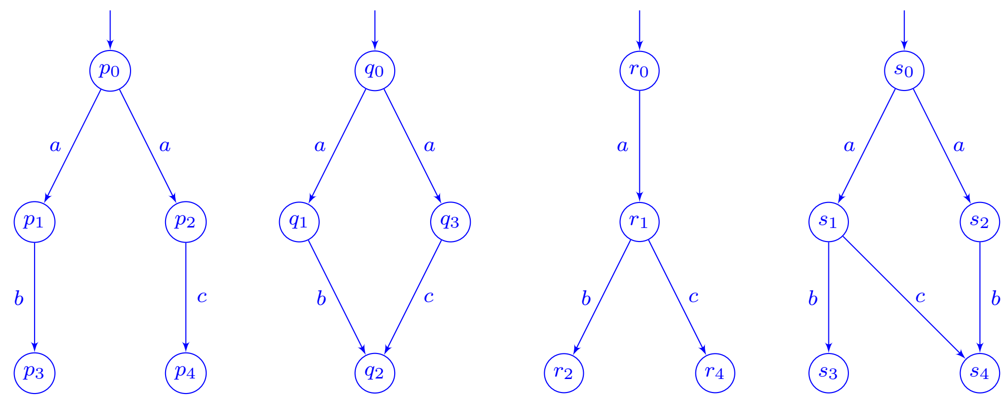
Such a relation *respects the final states* whenever $(s, s') \in \mathfrak{R}$ implies ($s \in S_F$ iff $s' \in S'_F$).

We say that the two transition systems $\mathcal{T}$ and $\mathcal{T}'$ are *bisimilar* whenever there exists a bisimulation relation $\mathfrak{R}$ such that for every $s_0 \in S_0$ there exists $s'_0 \in S'_0$ with $(s_0, s'_0) \in \mathfrak{R}$, and vice-versa.

Lemma 1. *If $\mathfrak{R}$ is a bisimulation relation which respects the final states, such that for every $s_0 \in S_0$, there exists $s'_0 \in S'_0$ with $(s_0, s'_0) \in \mathfrak{R}$ and vice-versa, then $L(\mathcal{T}) = L(\mathcal{T}')$.*

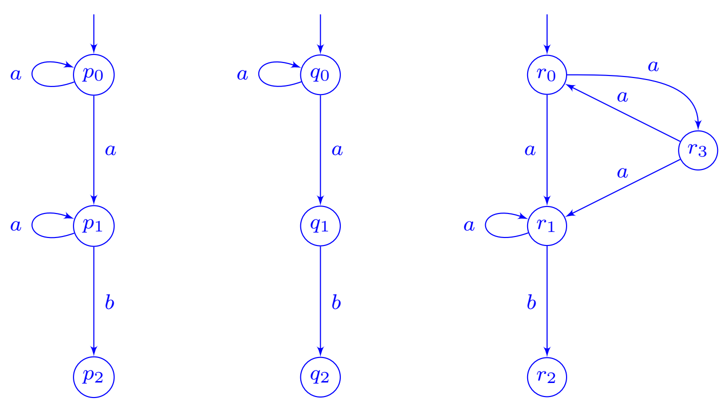
Exercise 7. Write the proof of the this lemma. ⌟

Exercise 8. Consider the four following automata. Which of these automata are bisimilar? Justify.

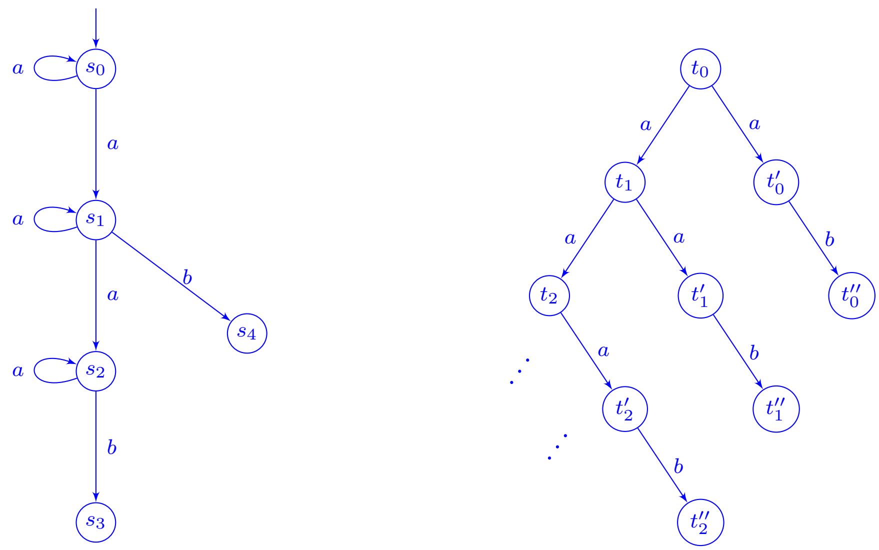


⌟

Exercise 9. 1. Consider the three following automata. Which of these automata are bisimilar? Justify.



2. How do the following transition systems relate (in a bisimulation sense) with the previous automata?



Exercise 10. Consider a finite automaton $\mathcal{A}_1$, and its complete determinization obtained by subset construction $\mathcal{A}_2$. Are $\mathcal{A}_1$ and $\mathcal{A}_2$ bisimilar?

Consider a deterministic complete finite automaton $\mathcal{B}_1$ and $\mathcal{B}_2$ its corresponding deterministic minimal automaton. Are $\mathcal{B}_1$ and $\mathcal{B}_2$ bisimilar?

Strong timed bisimulation. We extend the previous notion in an obvious way to timed transition systems. We choose to define it with mixed moves. Let $\mathcal{T} = (S, S_0, S_F, \rightarrow)$ and $\mathcal{T}' = (S', S'_0, S'_F, \rightarrow')$ be two timed transition systems with mixed moves over the same alphabet Σ. A relation $\mathfrak{R} \subseteq S \times S'$ is a *strong timed bisimulation* whenever the following properties hold:

- if $(s_1, s'_1) \in\mathfrak{R}$ and $s_1 \xrightarrow{d,a} s_2$ for some $d \in \mathbb{T}$ and some $a \in \Sigma$, then $s'_1 \xrightarrow{d,a}{}' s'_2$ and $(s_2, s'_2) \in\mathfrak{R}$;
- *vice-versa* (swap mixed moves in $\mathcal{T}$ and in $\mathcal{T}'$).

This equivalence is really strong and will seldom be used.

Time-abstract bisimulation. Let $\mathcal{T} = (S, S_0, S_F, \rightarrow)$ and $\mathcal{T}' = (S', S'_0, S'_F, \rightarrow')$ be two timed transition systems with mixed moves over the same alphabet Σ. A relation $\mathfrak{R} \subseteq S \times S'$ is a *time-abstract bisimulation* whenever the following properties hold:

- if $(s_1, s'_1) \in\mathfrak{R}$ and $s_1 \xrightarrow{d_1,a} s_2$ for some $d \in \mathbb{T}$ and some $a \in \Sigma$, then there exists $d' \in \mathbb{T}$ such that $s'_1 \xrightarrow{d',a}{}' s'_2$ and $(s_2, s'_2) \in\mathfrak{R}$;
- *vice-versa* (swap mixed moves in $\mathcal{T}$ and in $\mathcal{T}'$).

This relation is called time-abstract because it existentially quantifies over possible delays.

Unlike bisimulation, time-abstract bisimulation does not preserve languages, but it does preserve emptiness of languages.

Lemma 2. *If $\mathfrak{R}$ is a time-abstract bisimulation relation which respects final states, such that for every $s_0 \in S_0$, there exists $s'_0 \in S'_0$ with $(s_0, s'_0) \in \mathfrak{R}$ and vice-versa, then $L(\mathcal{T}) = \varnothing$ iff $L(\mathcal{T}') = \varnothing$.*

4.2 The region automaton construction

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. The construction relies on the definition of an equivalence relation over the set of valuations, which has finite index, and is called the *region equivalence.*

The region equivalence. We fix a finite set of clocks X, and we let $\mathcal{C}$ be a finite set of constraints over X. Let $\mathcal{R}$ be a partition of the set of (clock) valuations $\mathbb{T}^X$. We define three compatibility conditions as follows:

① $\mathcal{R}$ is *compatible with the set of constraints* $\mathcal{C}$ if for every constraint g in $\mathcal{C}$, for every R in $\mathcal{R}$, either $R \subseteq [\![g]\!]_X$ or $[\![g]\!]_X \cap R = \varnothing$;

② $\mathcal{R}$ is *compatible with the 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 $\mathcal{R}$ is *compatible with the resets* whenever for all R and R' in $\mathcal{R}$, for every subset $Y \subseteq X$, if $([Y \leftarrow 0]R) \cap R' \neq \varnothing$, then $([Y \leftarrow 0]R) \subseteq R'$.[7]

If $\mathcal{R}$ is finite and satisfies these three conditions, we say that $\mathcal{R}$ is a *set of regions* for the set of clocks X and the set of constraints $\mathcal{C}$ or simply a set of regions (if X and $\mathcal{C}$ are clear in the context). An element $R \in \mathcal{R}$ is then called a *region.* The set $\mathcal{R}$ defines in a natural way an equivalence relation $\equiv_{\mathcal{R}}$ over valuations:

$$v \equiv_{\mathcal{R}} v' \text{ iff (for each region } R \text{ of } \mathcal{R},\ v \in R \Leftrightarrow v' \in R)$$

If v is a valuation we note $[v]_{\mathcal{R}}$ the (unique) region to which v belongs.

The intuition behind these conditions is the following: we want to finitely abstract behaviours of timed automata. To that aim, we finitely abstract the (infinite) set of valuations (or equivalently the state space of the system): a valuation v will be abstracted into the region $[v]_{\mathcal{R}}$ (and a state (ℓ, v) will be abstracted into the pair $(\ell, [v]_{\mathcal{R}})$). In order for the abstraction to preserve (at least) reachability properties, it must be the case that if two states are equivalent, then future behaviours from those states should be more or less the same. The three conditions above express this property: condition ① says that two equivalent valuations satisfy the same clock constraints, condition ② says that time elapsing does not distinguish between two equivalent valuations, whereas condition ③ says that resetting clocks does not distinguish between two equivalent valuations. Note that condition ② is a *time-abstract* property: the precise values of delays needs not be the same

[7] $[Y \leftarrow 0]R$ is the set of valuations $\{[Y \leftarrow 0]v \mid v \in R\}$.

from two equivalent valuations. This 'abstraction of time' is the main reason which will make the region equivalence finite index for timed automata.

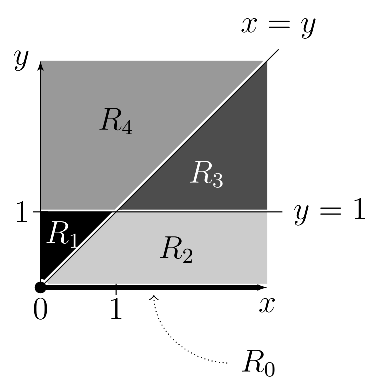
Example 5. Let us consider the partition of $\mathbb{R}_+^{\{x,y\}}$ made of five pieces

$$\mathcal{R} = \{R_0, R_1, R_2, R_3, R_4\}$$

which are defined by the following constraints:

$$\begin{matrix} R_0 & R_1 & R_2 & R_3 & R_4 \\ \begin{pmatrix} x \geq 0 \\ y = 0 \end{pmatrix} & \begin{pmatrix} 0 \leq x < 1 \\ 0 \leq y \leq 1 \\ x < y \end{pmatrix} & \begin{pmatrix} x \geq 0 \\ 0 < y \leq 1 \\ x \geq y \end{pmatrix} & \begin{pmatrix} x > 1 \\ y > 1 \\ x \geq y \end{pmatrix} & \begin{pmatrix} x \geq 0 \\ y > 1 \\ x < y \end{pmatrix} \end{matrix}$$

This partition is represented below:



It is easy to verify that $\mathcal{R}$ is a set of regions for the set of clocks $\{x, y\}$ and for the set of constraints $\{y \leq 1, y > 1, x \geq y, x < y\}$. ⌟

The main property of the region equivalence is the following:

Proposition 1. *Let $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T)$ be a timed automaton with set of constraints $\mathcal{C}$. Assume that $\mathcal{R}$ is a set of regions for X and $\mathcal{C}$. Then the equivalence relation $\cong_{\mathcal{R}}$ defined on configurations of $\mathcal{A}$ by:*

$$(\ell, v) \cong_{\mathcal{R}} (\ell', v') \text{ iff } \ell = \ell' \text{ and } v \equiv_{\mathcal{R}} v'$$

is a time-abstract bisimulation, which is called the region equivalence.

The proof of this proposition can be written as an exercise.

Exercise 11. Write the proof of the this proposition. ⌟

This property of the region equivalence will be used to construct a finite automaton, which will somehow be the quotient of the original timed automaton with the region equivalence. We start by constructing the region graph, which represents the (abstract) evolution of time and clock valuations. Then we construct the region automaton, which is a finite automaton, representing (in an abstract way) the behaviours of the timed automaton.

The region graph. From a set of regions $\mathcal{R}$ as characterized in the previous paragraph, one can define the so-called *region graph*, which represents the possible time evolutions of the system: the region graph is a finite automaton whose set of states is $\mathcal{R}$ and whose transitions are:

$$\begin{cases} R \xrightarrow{\varepsilon} R' \text{ if } R' \text{ is a time successor of } R,^{8} \\ R \xrightarrow{Y} R' \text{ if } [Y \leftarrow 0]R \subseteq R'. \end{cases}$$

Intuitively, the region graph records possible evolutions of time in the system: there is a transition $R \xrightarrow{\varepsilon} R'$ if, from some (or equivalently every, by condition ②) valuation of region R, it is possible to let some time elapse and reach region R'. There is a transition $R \xrightarrow{Y} R'$ if, from region R, the region R' can be reached by resetting clocks in Y. Note that this graph is closed by reflexivity and transitivity for ε-transitions, *i.e.*, $R \xrightarrow{\varepsilon} R$, and if $R \xrightarrow{\varepsilon} R'$ and $R' \xrightarrow{\varepsilon} R''$, then $R \xrightarrow{\varepsilon} R''$.

Example 6. The region graph associated with the set of regions $\mathcal{R}$ mentioned in Example 5 is represented on Figure 2. Plain edges are $\xrightarrow{\varepsilon}$ transitions of the region graph, whereas dashed (resp. dotted) edges are $\xrightarrow{\{x\}}$ (resp. $\xrightarrow{\{y\}}$) transitions. We have omitted transitions $\xrightarrow{\{x,y\}}$ (that reset both clocks) for readability reasons: there should be such a transition from any state to region R_0. ⌟

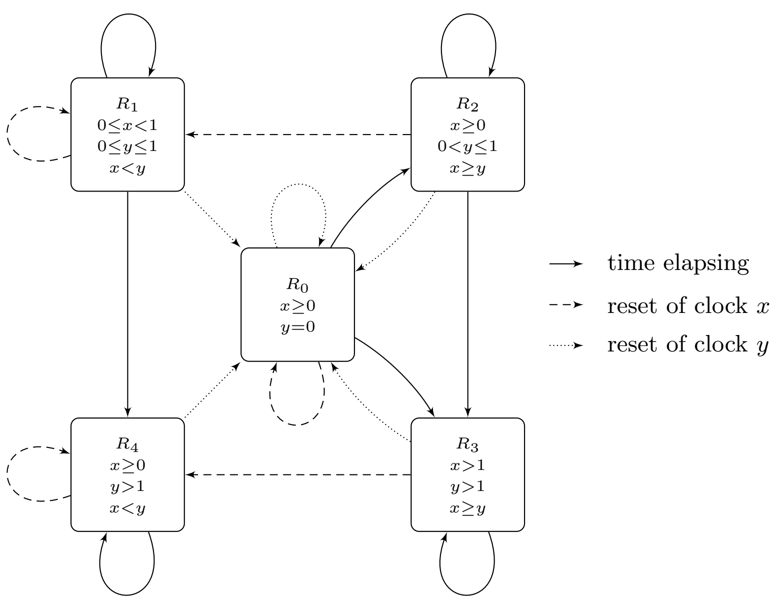


Fig. 2: *A simple example of a region graph*

[7] R' is a time successor of R whenever there is some $v \in R$ and some $t \in \mathbb{T}$ such that $v + t \in R'$. Note that, due to the compatibility condition ②, if R' is a time successor of R, then for every $v \in R$, there is some $t \in \mathbb{T}$ such that $v + t \in R'$.

The region automaton. Let $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T)$ be a timed automaton, and assume that the set of constraints occurring in $\mathcal{A}$ is $\mathcal{C}$. Let $\mathcal{R}$ be a finite set of regions for X and $\mathcal{C}$ (*i.e.*, a partition of $\mathbb{T}^X$ satisfying conditions ①, ② and ③). The *region automaton* $\Gamma_{\mathcal{R}}(\mathcal{A})$ is the finite automaton $(Q, Q_0, Q_F, \Sigma, T')$ where $Q = L \times \mathcal{R}$ is the set of states, $Q_0 = L_0 \times \{[\mathbf{0}_X]_{\mathcal{R}}\}$ is the set of initial states, $Q_F = L_F \times \mathcal{R}$ is the set of final states, Σ is the same alphabet as that of $\mathcal{A}$, and T' is the set of transitions defined as follows: there is a transition $(\ell, R) \xrightarrow{a} (\ell', R')$ in T' whenever there exists some region $R'' \in \mathcal{R}$ and some transition $\ell \xrightarrow{g,a,Y} \ell'$ in $\mathcal{A}$ such that

$$\begin{cases} R \xrightarrow{\varepsilon} R'' \text{ is a transition in the region graph,} \\ R'' \subseteq [\![g]\!]_X, \\ R'' \xrightarrow{Y} R' \text{ is a transition in the region graph.} \end{cases}$$

A transition of the region automaton abstracts a delay followed by an action in the original timed automaton. More precisely, whenever in $\mathcal{A}$ we can delay some time and make an a, then in $\Gamma_{\mathcal{R}}(\mathcal{A})$, we can make some a, and *vice-versa*. This is formalized by the following lemma, which can be derived from the definition of $\Gamma_{\mathcal{R}}(\mathcal{A})$ and from Proposition 1:

Lemma 3. *The three following sentences are equivalent:*

(*a*) *there is a transition* $(\ell, R) \xrightarrow{a} (\ell', R')$ *in* $\Gamma_{\mathcal{R}}(\mathcal{A})$*;*

(*b*) *there exists* $v \in R$*, there exists* $d \in \mathbb{T}$ *and there exists* $v' \in R'$ *such that* $(\ell, v) \xrightarrow{d,a} (\ell', v')$ *is a mixed move in* $\mathcal{A}$*;*

(*c*) *for every* $v \in R$*, there exists* $d \in \mathbb{T}$ *and there exists* $v' \in R'$ *such that* $(\ell, v) \xrightarrow{d,a} (\ell', v')$ *is a mixed move in* $\mathcal{A}$*.*

We are now ready to state the main language-related property of the region automaton construction.

Proposition 2. *Let* $\mathcal{A}$ *be a timed automaton with set of clocks* X *and set of constraints* $\mathcal{C}$*. We assume we can construct a set of regions* $\mathcal{R}$ *for* X *and* $\mathcal{C}$*. Then,*

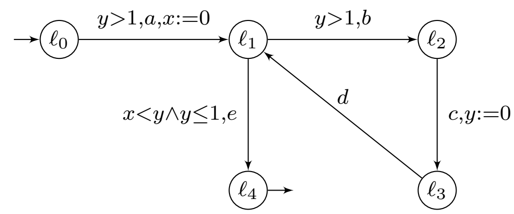
$$\mathsf{Untime}(L(\mathcal{A})) = L(\Gamma_{\mathcal{R}}(\mathcal{A}))$$

where $L(\Gamma_{\mathcal{R}}(\mathcal{A}))$ *is the (untimed) language accepted by the finite automaton* $\Gamma_{\mathcal{R}}(\mathcal{A})$ *and* $\mathsf{Untime}((a_1, t_1) \dots (a_p, t_p)) = a_1 \dots a_p$ *assigns to a timed word (and by extension to a timed language) a finite word (and by extension a classical language).*

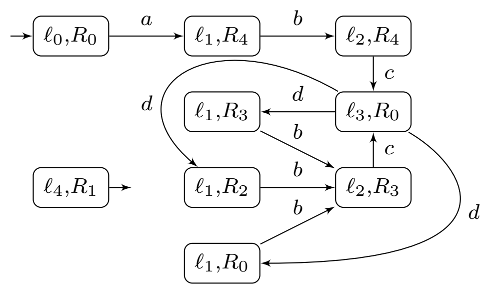
Proof. A standard induction on the length of runs allows to build runs in $\mathcal{A}$ and paths in $\Gamma_{\mathcal{R}}(\mathcal{A})$ which coincide in the following sense: there is a run $(\ell_0, v_0) \xrightarrow{d_1,a_1} \dots \xrightarrow{d_p,a_p} (\ell_p, v_p)$ in $\mathcal{A}$ iff there is a path $(\ell_0, [v_0]_{\mathcal{R}}) \xrightarrow{a_1} \dots \xrightarrow{a_p} (\ell_p, [v_p]_{\mathcal{R}})$. This implies the expected result. ⌟

Remark 5. In terms of behavioural equivalence, the finite automaton $\Gamma_{\mathcal{R}}(\mathcal{A})$ is the quotient of the timed transition system $\mathcal{T}^m_{\mathcal{A}}$ w.r.t. the time-abstract bisimulation $\cong_{\mathcal{R}}$.

Example 7. We consider the following small timed automaton, and we would like to know whether location ℓ_4 is reachable from the initial state $(\ell_0, \mathbf{0}_{\{x,y\}})$.



A possible set of regions for that automaton has been given in Example 5, and the corresponding region automaton is the following finite automaton.



Applying Proposition 2, we have that location ℓ_4 is reachable in the original timed automaton iff the state (ℓ_4, R_1) is reachable in the region automaton (actually it should be iff any of the (ℓ_4, R_i) is reachable, but due to the constraint labelling the transition from ℓ_1 to ℓ_4, we can focus on (ℓ_4, R_1)). We see that it is not the case, and hence that there is no run which starts in the initial state $(\ell_0, \mathbf{0}_{\{x,y\}})$ and ends in ℓ_4. ⌟

For every timed automaton $\mathcal{A}$ for which we can *effectively* construct a finite set of regions $\mathcal{R}$ (satisfying conditions ①, ② and ③), we can transfer the checking of reachability properties in $\mathcal{A}$ to the finite automaton $\Gamma_{\mathcal{R}}(\mathcal{A})$. It remains to see how we can effectively build sets of regions for timed automata.

4.3 Effective construction of sets of regions

In the previous section, we have presented an abstract construction which allows to reduce for instance the model-checking of reachability properties in timed automata to the model-checking of reachability properties in finite automata (under the condition that there is a finite set of regions for the set of constraints used in the timed automaton). However, we did not explain how we construct a set of regions for timed automata, which is the basis to the whole construction.

In this section, we fix a finite set of clocks X.

Regions for sets of diagonal-free constraints. Let $M \in \mathbb{N}$ be an integer. We define a set of regions for X and the set of M-bounded diagonal-free clock constraints $\mathcal{C}_{df}^{M}(X)$. A natural partition would be to take the partition induced by the set of constraints itself, see Figure 3(a) for an illustration with two clocks. But this is actually not fine enough because compatibility condition ③ is not satisfied (as illustrated on the figure by the two gray valuations). A correct partition is then the one of Figure 3(b).

(a) Partition compatible with the 2-bounded clock constraints and the resets (conditions ① and ③), but not with time elapsing (condition ②): the two gray points are not equivalent

(b) Partition $\mathcal{R}_{df}^{2}(\{x,y\})$ satisfying all the compatibility constraints ①, ② and ③ for the set of constraints $\mathcal{C}_{df}^{2}(\{x,y\})$

Fig. 3: *Region construction for set of diagonal-free constraints* $\mathcal{C}_{df}^{2}(\{x,y\})$

We formalize this idea and we define the equivalence relation $\equiv_{df}^{X,M}$ over $\mathbb{T}^X$. We will give three different (equivalent) definitions.

(i) The first definition formalizes the 2-dimensional intuition that we have seen earlier. Let v and v' be two valuations of $\mathbb{T}^X$, we say that $v \equiv_{df}^{X,M} v'$ if all three following conditions are satisfied:
   - (a) $v(x) > M$ iff $v'(x) > M$ for each $x \in X$,
   - (b) if $v(x) \leq M$, then $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ and $\Big(\{v(x)\} = 0$ iff $\{v'(x)\} = 0\Big)$ for each $x \in X$,[9] and

[9] $\lfloor\cdot\rfloor$ (resp. $\{\cdot\}$) represents the integral (resp. fractional) part.

(c) 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 $\equiv_{df}^{X,M}$ is an equivalence relation of finite index, and it naturally induces a finite partition $\mathcal{R}_{df}^{M}(X)$ of $\mathbb{T}^X$ (defined as the set of equivalence classes of $\mathbb{T}^X{}_{/\equiv_{df}^{X,M}}$). The construction for two clocks is precisely that illustrated on Figure 3(b).

(ii) We give a second description of the regions in $\mathcal{R}_{df}^{M}(X)$. An interval of $\mathbb{T}$ with integral bounds is said M*-simple* if it is of one of the following forms: $(c, c{+}1)$ with $0 \leq c < M$, or $[c, c]$ with $0 \leq c \leq M$, or $(M, +\infty)$. It is said bounded if it is one of the two first forms, and singular in the second form. Each region of $\mathcal{R}_{df}^{M}(X)$ can then be characterized uniquely by:
- an M-simple interval I_x for every clock $x \in X$, and
- a preorder $\prec$ on the set of clocks

$$Z_{(I_x)_{x \in X}} = \{x \in X \mid I_x \text{ bounded and non-singular}\}.$$

Intuitively the interval I_x is the interval to which x belongs, and the preorder is given by the preorder on the fractional parts of all clocks bounded by M with non-integral values.

Assume that region R is given by $(I_x)_{x \in X}$ and $\prec$; then:

$$v \in R \text{ iff } \begin{cases} \forall x \in X,\ v(x) \in I_x \text{ and} \\ \forall x, y \in Z_{(I_x)_{x \in X}},\ (x \prec y \Leftrightarrow \{v(x)\} \leq \{v(y)\}) \end{cases}$$

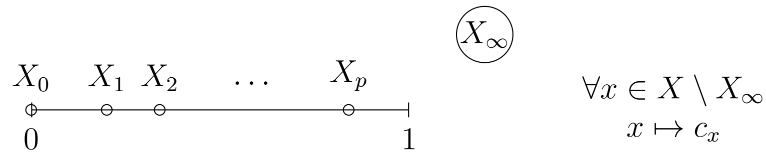
(iii) We give a third description of the regions, which give an interesting one-dimensional understanding of the regions. It reuses elements of the characterization above. Each region of $\mathcal{R}_{df}^{M}(X)$ can be characterized uniquely by:
- the set $X_\infty = \{x \in X \mid I_x = (M, +\infty)\}$
- the set $X_0 = \{x \in X \setminus X_\infty \mid I_x = [c, c] \text{ for some } c \leq M\}$
- a partition $(X_i)_{1 \leq i \leq p}$ of $X \setminus (X_0 \cup X_\infty)$ such that:
  - for every $1 \leq i \leq p$, $X_i \neq \varnothing$
  - for $x \in X \setminus (X_0 \cup X_\infty)$, writing $i(x)$ for the unique index such that $x \in X_{i(x)}$, for every $x, y \in X \setminus (X_0 \cup X_\infty)$

$$x \prec y \Leftrightarrow i(x) \leq i(y)$$

  That is: clocks in the same X_i have the same fractional part, whereas fractional parts of clocks in two different X_i's have their fractional parts ordered accordingly.
- for every $x \in X \setminus X_\infty$, c_x is an integer bounded by M, and if $x \notin X_0$, $c_x < M$.

Note: sets X_0 and X_∞ can be empty (contrary to the other X_i's).

The generic representation of such a region is as follows (it represents the interval $[0, 1)$ and shows the repartition of the clocks within that interval (according to their fractional part)):



Assume that region R is given by X_0, $(X_i)_{1\le i\le p}$, X_∞ and $(c_x)_{x\in X\setminus X_\infty}$; for every $x \in X \setminus X_\infty$, write $i(x) \in \{0, 1, \dots, p\}$ such that $x \in X_i$. Then:

$$v \in R \text{ iff } \exists(\gamma_i)_{0\le i\le p} \in [0,1)^{p+1} \text{ s.t. } \begin{cases} \gamma_0 = 0,\ i < j \Rightarrow \gamma_i < \gamma_j \\ \forall x \in X \setminus X_\infty,\ v(x) = c_x + \gamma_{i(x)} \\ \forall x \in X_\infty,\ v(x) > M \end{cases}$$

For instance, the lightgray region depicted in Figure 3(b) has the following second characterization:

$$\begin{cases} x \in (1,2) \\ y \in (0,1) \\ x \prec y,\ y \not\prec x \text{ (meaning } \{x\} < \{y\}) \end{cases}$$

and the following third characterization:

$$\begin{cases} X_0 = X_\infty = \varnothing \\ X_1 = \{x\},\ X_2 = \{y\} \\ c_x = 1,\ c_y = 0 \end{cases}$$

Exercise 12. Prove the equivalence of these definitions. ⌟

The following lemma states the correctness of the above partition.

Proposition 3. *The partition $\mathcal{R}^M_{df}(X)$ is a finite set of regions for clocks X and M-bounded diagonal-free clock constraints $\mathcal{C}^M_{df}(X)$.*

Proof. Conditions ① and ③ are rather easy to check, we thus omit the details. The case of condition ② requires more careful developments.

We will base the proof on the third characterization of the regions, and first define a "next successor" operation on the regions: the region r_∞ defined by $X_\infty = X$ has no next successor; given a region r characterized by $(X_i)_{0\le i\le p, i=\infty}$ and $(c_x)_{x\in X\setminus X_\infty}$, its next successor $\mathsf{succ}(r)$, characterized by $(X'_i)_{0\le i\le p', i=\infty}$ and $(c'_x)_{x\in X\setminus X'_\infty}$, is defined as follows:

– if $X_0 \neq \varnothing$ and $\{x \in X_0 \mid c_x < M\} \neq \varnothing$,

$$\begin{cases} X'_0 = \varnothing \\ X'_\infty = X_\infty \cup \{x \in X_0 \mid c_x = M\} \\ X'_1 = \{x \in X_0 \mid c_x < M\} \\ X'_{i+1} = X_i,\ \forall 1 \le i \le p \\ c'_x = c_x,\ \forall x \in X \setminus X'_\infty \end{cases}$$

– if $X_0 \neq \varnothing$ and $\{x \in X_0 \mid c_x < M\} = \varnothing$,

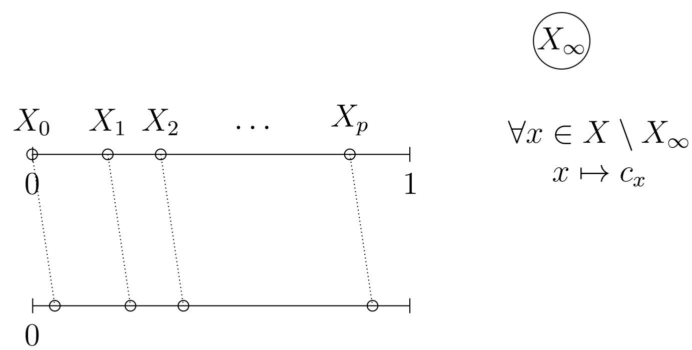
$$\begin{cases} X'_0 = \varnothing \\ X'_\infty = X_\infty \cup X_0 \\ X'_i = X_i, \ \forall 1 \leq i \leq p \\ c'_x = c_x, \ \forall x \in X \setminus X'_\infty \end{cases}$$

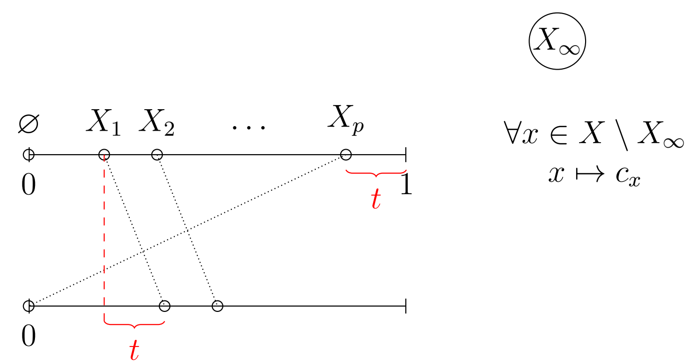
– if $X_0 = \varnothing$,

$$\begin{cases} X'_\infty = X_\infty \\ X'_0 = X_p \\ X'_i = X_i, \ \forall 1 \leq i < p \\ c'_x = c_x, \ \forall x \in \bigcup_{1 \leq i < p} X_i \\ c'_x = c_x + 1, \ \forall x \in X'_0 \end{cases}$$

Indeed, time elapsing on that representation is obtained by circular translation of the sets of clocks, with an absorbing set X_∞ when clocks go above the maximal constant.

Find below a (rough) representation of the two first cases, and then of the third case:





Formally:

Lemma 4. *Let $r \neq r_\infty$ be a region such that $\mathsf{succ}(r)$ is defined. For every $v \in r$, there exists $t \in \mathbb{T}$ such that $v + t \in \mathsf{succ}(r)$, and for all $0 \leq t' \leq t$, $v + t' \in r \cup \mathsf{succ}(r)$.*

Proof. Let $v \in r$, and let γ_i be the fractional part of the clocks in X_i, assuming $\gamma_0 = 0$: for every $x \in X \setminus X_\infty$, $v(x) = c_x + \gamma_{i(x)}$.

If we are in one of the two first cases above, we let $t = \frac{1-\gamma_p}{2}$, and set $v' = v + t$. It is easy to check that $v' \in \mathsf{succ}(r)$, and that is is also the case for all $v + t'$ with $0 < t' \leq t$.

Then, assume we are in the third case, and let $t = 1 - \gamma_p$. Setting $v' = v + t$, we get the expected result as well. ⌟

Let r be a region. We define $\mathsf{succ}^0(r) = r$, and $\mathsf{succ}^{i+1}(r) = \mathsf{succ}(\mathsf{succ}^i(r))$ if it is defined. Then $\mathsf{succ}^*(r) = \bigcup_{\substack{i \geq 0 \\ \mathsf{succ}^i(r) \text{ defined}}} \mathsf{succ}^i(r)$. We then prove the following lemma:

Lemma 5. *Let r be a region and let $v \in r$. For every $t \in \mathbb{T}$, $v + t \in \mathsf{succ}^*(r)$.*

Proof. We show the result by induction on the number of regions included in $\mathsf{succ}^*(r)$. The basis case is when $\mathsf{succ}^*(r)$ only includes r, which can only happen when $r = r_\infty$: this is obvious since for every $v \in r_\infty$, for every $t \in \mathbb{T}$, $v + r \in r_\infty$.

Pick $n \in \mathbb{N}_{\geq 1}$, and assume we have proven the result for every region r such that $\mathsf{succ}^*(r)$ contains at most n regions. Pick now a region r such that $\mathsf{succ}^*(r)$ contains $n + 1$ regions. Towards a contradiction assume that the expected property does not hold for r, and choose $v \in r$ and $t \in \mathbb{T}$ such that $v + t \notin \mathsf{succ}^*(r)$. Notice first that $r \neq r_\infty$, hence $\mathsf{succ}(r)$ is well-defined. Let t_1 be such that $v + t_1 \in \mathsf{succ}(r)$. Because of the previous lemma, it must be the case that $t > t_1$. Letting $r_1 = \mathsf{succ}(r)$ $v_1 = v + t_1 \in r_1$, with delay $t - t_1$, we can apply the induction hypothesis, and obtain a contradiction. ⌟

We are now ready to prove property ②. Let r be a region. Applying Lemma 4, we show by induction on i that for every $i \in \mathbb{N}$, for every $v \in r$, there is t_i such that $v + t_i \in \mathsf{succ}^i(r)$. Lemma 5 shows the completeness, since for every $v \in r$, for every $t \in \mathtt{T}$, there is i such that $v + t \in \mathsf{succ}^i(r)$.

Hence, condition ② is satisfied by the partition $\mathcal{R}_{df}^M(X)$. ⌟

Lemma 6. *The number of regions in $\mathcal{R}_{df}^M(X)$ is bounded by $(M+1)^{|X|}.|X|!.2^{|X|+2}$ where $|X|$ is the cardinal of X.*

Proof. We use the third characterization of regions to get the expected upper bound. In this representation, we can assume that $c_x = M$ whenever $x \in X_\infty$. The factor $(M+1)^{|X|}$ is then for the choice of each constant c_x (for $x \in X$). Now, to understand the rest of the formula, let us write down the list of the clocks by starting with X_0, then X_1, …, X_p and finally X_∞. There are $|X|!$ such lists. Finally we distinguish all first elements of each set (plus two Booleans for checking whether X_0 or X_∞ is empty or not), yielding the factor $2^{|X|+2}$.

Note: this bound is obviously not optimal; indeed, the order in which we list the various sets X_i is not important. ⌟

Note that a tighter bound can be obtained, but the computation is much more involved, see [Kop96].[10]

[10] Thanks to Luca Aceto who pointed out this reference.

Regions for sets of general constraints. Let $M \in \mathbb{N}$ be an integer. The aim is to define a set of regions for X and the set of M-bounded clock constraints $\mathcal{C}^M(X)$. The partition we have defined in the previous paragraph is no more compatible with the set of constraints $\mathcal{C}^M(X)$, we thus need to refine it.

We define an equivalence relation $\equiv^{X,M}$ over $\mathbb{T}^X$ as follows: let v and v' be two valuations of $\mathbb{T}^X$, we say that $v \equiv^{X,M} v'$ if all two following conditions hold:

- $v \equiv_{df}^{X,M} v'$, and
- $v \models (x - y \sim c)$ iff $v' \models (x - y \sim c)$ for every $(x - y \sim c) \in \mathcal{C}^M(X)$.

This equivalence relation refines $\equiv_{df}^{X,M}$ in that two equivalent valuations satisfy in addition the same diagonal constraints (bounded by M). This new partition is denoted $\mathcal{R}^M(X)$ and is illustrated for two clocks on Figure 4.

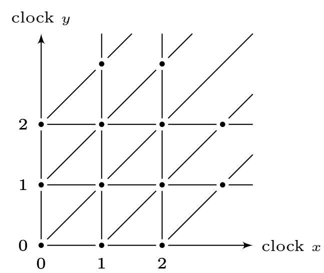


Fig. 4: *Set of regions $\mathcal{R}^2(X)$ for 2-bounded clock constraints with two clocks*

We omit the proof of the following proposition, which states the correctness of the refinement.

Proposition 4. *The partition $\mathcal{R}^M(X)$ is a set of regions for X and the set of M-bounded clock constraints $\mathcal{C}^M(X)$.*

As previously, each region of $\mathcal{R}^M(X)$ can be characterized by:

- an M-simple interval I_x for every clock $x \in X$, and
- an M-simple interval, or minus an M-simple interval, $J_{x,y}$ for all clocks $x, y \in X$.

Intuitively the interval I_x gives the constraint on clock x, and $J_{x,y}$ gives the constraint on the difference $x - y$. The order between fractional parts of two clocks x and y can now be infered from the intervals I_x, I_y and $J_{x,y}$.

As a direct consequence of this characterization, we get the following upper bound on the number of regions.

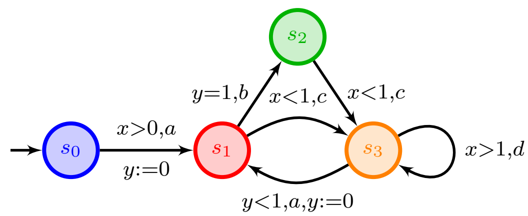
Lemma 7. *The number of regions in $\mathcal{R}^M(X)$ is bounded by $(4M+3)^{(|X|+1)^2}$ where $|X|$ is the cardinal of X.*

Remark 6. Note that $\mathcal{R}^M(X)$ is also a set of regions for the set of constraints $\mathcal{C}_{df}^M(X)$.

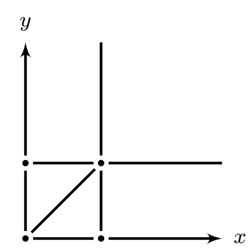
Remark 7. Note that sets of regions we have described could be made smaller: there is no need to have the same maximal constant for all clocks, one maximal constant for each clock could be used. However, for the purpose of these notes, there is no need for such a refinement.

4.4 An example of region automaton [AD94]

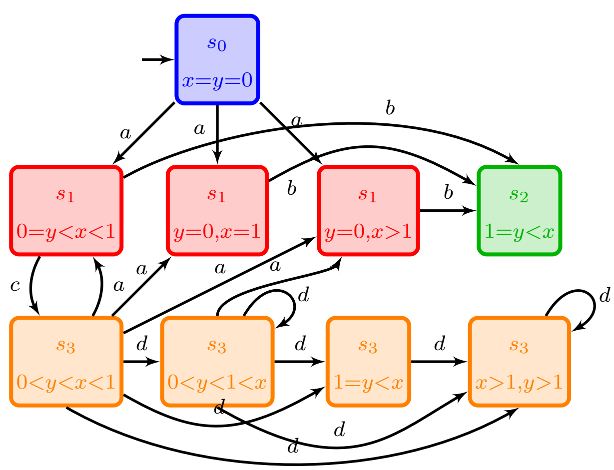
Consider the following timed automaton:



It has two clocks and maximal constant 1. The set of regions can be described as follows:



The corresponding region automaton is therefore:



Many things can be read on the region automaton... e.g. Zeno cycles.

4.5 Some exercices

Exercise 13. We assume we can update clocks with more involve operations than resets to zero. For this exercise, we write $\mathcal{U}$ for a finite set of functions $up : \mathbb{T}^X \to \mathbb{T}^X$. Those updates can be put on transitions in place of resets, and when firing the transition, the valuation is updated according to this function.

How should we strengthen conditions ①, ② and ③ in the construction of a finite set of regions to take into account those new updates?

Can you build a (finite) set of regions in the following cases? Explain.

– diagonal-free clock constraints, and updates

$$\mathcal{U} = \{x := 0, x := x + 1 \mid x \in X\}$$

where, writing *up* for '$x := x + 1$' $up(v)$ is the valuation assigning $v(x) + 1$ to x and $v(y)$ to $y \neq x$;
– general clock constraints, and updates

$$\mathcal{U} = \{x := 0, x := x + 1 \mid x \in X\}$$

– diagonal-free clock constraints, and updates

$$\mathcal{U} = \{x := 0, x := x - 1 \mid x \in X\}$$

– general clock constraints, and updates

$$\mathcal{U} = \{x := 0, x := x - 1 \mid x \in X\}$$

⌟

Exercise 14. Prove that for any timed automaton $\mathcal{A}$, we can construct a diagonal-free timed automaton $\mathcal{B}$ that recognizes the same timed language. What is the size of $\mathcal{B}$? Do you think we can avoid that blowup? ⌟

Exercise 15. If we add linear constraints to the model, *i.e.*, constraints of the form $\sum_{x \in X} \alpha_x x \sim c$ with $\alpha_x \in \mathbb{Z}$ for every $x \in X$ and $c \in \mathbb{Z}$, prove that the reachability problem becomes undecidable. ⌟

5 Complexity issues

5.1 Application to the reachability problem

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 $\mathcal{R}^M(X)$ (or even $\mathcal{R}^M_{df}(X)$ in case $\mathcal{A}$ is a diagonal-free timed automaton) is a set of regions for $\mathcal{A}$. Hence the region abstraction and all the developments made in the previous section can be used. The following result, due to Alur and Dill [AD90,AD94], is the core of the verification of timed systems.

Theorem 2. *The reachability problem is decidable for timed automata. It is a* PSPACE*-complete problem (for both diagonal-free and general timed automata).*

Although this theorem has been first stated in [AD90,AD94], the proof we present here is taken from [AL02].

Proof. To prove PSPACE membership, we use the region automaton construction, and check reachability properties in this finite abstraction (see Proposition 2). Applying Lemmas 6 and 7, we know that the size of this finite automaton is exponential in the size of the original timed automaton (in both diagonal-free and general cases). Moreover, using the characterizations of the regions we have given we know that each state of the finite automaton can be stored in polynomial space, and, given a state, we can guess a successor in polynomial space. Hence, we apply the classical NLOGSPACE algorithm for checking reachability properties in finite automata, and get a PSPACE algorithm for checking reachability properties in the original timed automaton.

The PSPACE-hardness can be proved by reducing the termination of a (non-deterministic) linearly bounded Turing machine (LBTM for short) on some given input to the reachability problem in timed automata. We present the proof for general timed automata, and then explain informally how it can be extended to diagonal-free timed automata. The complete reduction can be found in the appendices of [AL02].

Let $\mathcal{M}$ be a LBTM and w_0 an input for $\mathcal{M}$. We write N for a (linear) bound on the length of the tape which is used when $\mathcal{M}$ executes on w_0. We encode the behaviour of $\mathcal{M}$ on w_0 as the behaviour of a timed automaton. The encoding is as follows: assuming the alphabet of $\mathcal{M}$ is $\{a, b\}$ (this can be done w.l.o.g.), and writing # for the blank symbol, the content of cell C_i of the tape of the LBTM is encoded by a constraint on two clocks x_i and y_i. Cell C_i contains a symbol a when the constraint $x_i = y_j$ holds, and cell C_j contains a symbol b when the constraint $x_j < y_j$ holds. If the cell is empty (or equivalently contains the blank symbol #), then the constraint $x_j > y_j$ holds. Note that these three constraints are invariant by time elapsing. This is illustrated on Figure 5.

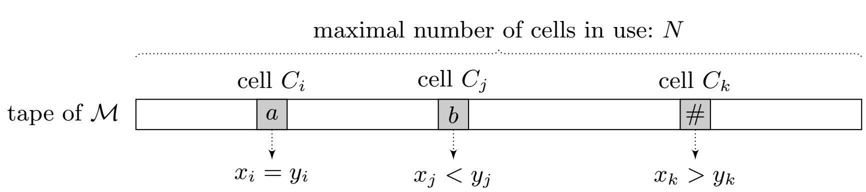


Fig. 5: *Encoding of the tape of the LBTM $\mathcal{M}$*

We assume that the set of states of $\mathcal{M}$ is Q, its initial state is q_0, and its halting state is q_F. We construct a timed automaton $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T)$ as follows:

- $L = \big(Q \times \{1, \ldots, N\}\big) \cup \{\mathsf{init}\}$;
- $L_0 = \{\mathsf{init}\}$;

- $L_F = \{(q_F, i) \mid i \in \{1, \ldots, N\}\}$;
- $X = \{x_i, y_i \mid i \in \{1, \ldots, N\}\} \cup \{u\}$;
- $\Sigma = \{a\}$;
- For every rule of $\mathcal{M}$, we will have several transitions in T. We define the constraint $g_{a,i}$ by $(u > 0 \wedge x_i < y_i)$, the constraint $g_{b,i}$ by $(u > 0 \wedge x_i = y_i)$, and the constraint $g_{\#,i}$ by $(u > 0 \wedge x_i > y_i)$. Those constraints express that cell i contains an a, respectively a b and a blank character. We define the resetting sets $Y_{a,i} = \{u, x_i, y_i\}$ and $Y_{b,i} = \{u, x_i\}$. Those are sets of clocks to be reset for expressing the fact that we write an a, respectively a b, in cell i.
  - Consider a rule $(q, \mathsf{Read}_\alpha, \mathsf{Write}_\beta, \mathsf{Right}, q')$ [11] in $\mathcal{M}$. For every $i \in \{1, \ldots, N-1\}$, there is a transition $(q, i) \xrightarrow{g_{\alpha,i}, a, Y_{\beta,i}} (q', i+1)$ in T.
  - Consider a move $(q, \mathsf{Read}_\alpha, \mathsf{Write}_\beta, \mathsf{Left}, q')$ in $\mathcal{M}$. For every $i \in \{2, \ldots, N\}$, there is a transition $(q, i) \xrightarrow{g_{\alpha,i}, a, Y_{\beta,i}} (q', i-1)$ in T.

  There is an extra transition from state init to initialize the input word w_0 on the tape: $\mathsf{init} \xrightarrow{u>0, a, Y_{w_0}} (q_0, 1)$ where $Y_{w_0} = \{u\} \cup \{x_i, y_i \mid w_0(i) = a\} \cup \{x_i \mid w_0(i) = b\} \cup \{y_i \mid i > |w_0|\}$ ($w_0(i)$ denotes the i-th letter of w_0).

We claim that there is a halting computation in $\mathcal{M}$ iff $L(\mathcal{A}) \neq \varnothing$, and we let the careful reader get convinced of this equivalence. As the halting problem for LBTMs is PSPACE-hard, we get the expected lower bound.

See next exercise for the case of diagonal-free clock constraints. ⌟

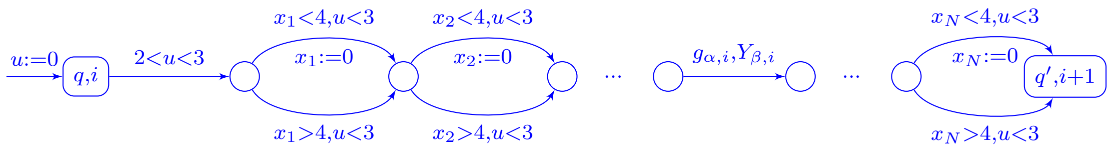
Remark 8. In [CY92], a proof of PSPACE-hardness is given for diagonal-free timed automata with only three clocks, it is rather technical, hence we have chosen not to present it here.

Exercise 16. Think of a PSPACE lower bound reduction for diagonal-free timed automata. ⌟

Proof (Solution to this exercise). We use a proof which is close to the previous one. We can nevertheless not use diagonal constraints, which were very convenient for encoding the content of the cells (since diagonal constraints are invariant by time elapsing).

Let N be the bound on the tape of $\mathcal{M}$ when simulating on input word w_0. We assume the alphabet is $\{a, b\}$ and we encode the content of j-th cell C_j using a clock x_j with the following convention: when we enter a module, cell C_j contains an a whenever $x_j < 1$ and it contains a b whenever $x_j > 2$. The simulation of a transition $(q, \mathsf{Read}_\alpha, \mathsf{Write}_\beta, \mathsf{Right}, q')$ is given on the figure below (we assume $i + 1 \leq N$, so that this transition is meaningful). Guard $g_{a,i}$ is $x_i < 4, u < 3$ whereas guard $g_{b,i}$ is $x_i > 4, u < 3$. Set $Y_{a,i}$ is $\{x_i\}$ and set $Y_{b,i}$ is $\varnothing$.

[11] This rule reads as follows: from state q, if we read an α in the current cell of the tape, then we write a β onto the current cell, move the head of the tape to the right and go to state q'.



The case of a transition $(q, \mathsf{Read}_\alpha, \mathsf{Write}_\beta, \mathsf{Right}, q')$ is very similar, but requires $i-1 \geq 0$, and then goes to $(q', i-1)$ instead of $(q', i+1)$.

5.2 The case of 'simply-timed' timed automata

The previous complexity result holds for timed automata with three clocks or more (the proof of three clocks is rather involved, though). For some simpler systems, for instance for systems with a single clock, this result can be improved. Of course, the same set of regions cannot be used, because even though the number of clocks is one, the number of regions given in Lemma 6 remains exponential, due to the binary encoding of constants in the timed automaton. However we can choose a smaller and rougher set of regions which yields the following result, due to [LMS04].

Proposition 5. *The reachability problem for single-clock timed automata is* NLOGSPACE*-complete.*

Proof. NLOGSPACE-hardness follows from that of reachability in finite graphs [HU79].

The NLOGSPACE membership can be obtained using a rougher set of regions than that presented in Section 4.3. Given a finite set $\mathcal{C}$ of constraints over a single clock x, we define the set of constants $C = \{c \in \mathbb{N} \mid \exists (x \bowtie c) \in \mathcal{C}\} \cup \{0\}$, and we assume that this set is ordered: $\mathcal{C} = \{c_0 < c_1 < c_2 < \cdots < c_p\}$. We define the partition $\mathcal{R}_\mathcal{C}$ as the (finite) set of intervals of one of the forms: (i) $\{c_i\}$ with $0 \leq i \leq p$, or (ii) (c_i, c_{i+1}) with $0 \leq i < p$, or (iii) $(c_p, +\infty)$. This is not hard to prove that $\mathcal{R}_\mathcal{C}$ is a set of regions for $\{x\}$ and $\mathcal{C}$. The size of $\mathcal{R}_\mathcal{C}$ is polynomial in the size of $\mathcal{C}$, which yields a polynomial-size region automaton, hence the expected result. ⌟

The case of two-clock timed automata has recently been solved in the literature [FJ15] and proved to be PSPACE-complete. For a long time, this was known to be NP-hard only. The proof is long and technical, therefore it is omitted here.

Exercise 17. Prove an NP lower bound for the reachability problem in two-clocks timed automata. ⌟

6 Timed logics

The region automaton abstraction is sound for verifying reachability properties. This is due to the time-abstract bisimulation property which links the region automaton and the original timed automaton. As a consequence, the region automaton abstraction can be used to verify all properties that are invariant by time-abstract bisimulation. This is for instance the case of safety properties, of ω-regular properties, or of untimed properties expressed in LTL [Pnu77] or in CTL [CE81]. However, this construction cannot be directly used to verify properties expressed in a timed temporal logic like TCTL [ACD90,ACD93] because a property like "reaching a state in exactly 5 units of time" is not invariant by time-abstract bisimulation. However we show that it can nevertheless be used.

6.1 Timed extensions of CTL

We let AP be a finite set of atomic propositions. In this section, timed automata are tuples $\mathcal{A} = (L, \ell_0, X, T, \mathcal{L})$ such that (L, ℓ_0, X, T) is a standard timed automaton with a single initial location, no final locations, no alphabet, and $\mathcal{L} : L \to 2^{\mathsf{AP}}$ is a labelling function.

Syntax and semantics. The branching-time logic TCTL,[12] which extends the classical untimed branching-time logic CTL with time constraints on modalities, has been defined

The syntax of TCTL is given by the following grammar:

$$\mathsf{TCTL} \ni \phi \ ::= \ a \ \mid \ \neg\phi \ \mid \ \phi \vee \phi \ \mid \ \mathbf{E}\,\phi\,\mathbf{U}_I\,\phi \ \mid \ \mathbf{A}\,\phi\,\mathbf{U}_I\,\phi$$

where $a \in \mathsf{AP}$, and I is an interval of $\mathbb{R}_+$ with integral bounds.

There are two possible semantics for TCTL, one which is said *'continuous'*, and the other one which is more discrete and is said *'pointwise'*. These two semantics share rules for basic modalities, and only differ in the interpretation of the term *'position'*:

$$\begin{array}{rcl}
(\ell, v) \models a & \Leftrightarrow & a \in \mathcal{L}(\ell) \\
(\ell, v) \models \neg\phi & \Leftrightarrow & (\ell, v) \not\models \phi \\
(\ell, v) \models \phi \vee \psi & \Leftrightarrow & (\ell, v) \models \phi \text{ or } (\ell, v) \models \psi \\
(\ell, v) \models \mathbf{E}\,\phi\,\mathbf{U}_I\,\psi & \Leftrightarrow & \text{there is an infinite run } \varrho \text{ in } \mathcal{A} \\
 & & \text{from } (\ell, v) \text{ such that } \varrho \models \phi\,\mathbf{U}_I\psi \\
(\ell, v) \models \mathbf{A}\,\phi\,\mathbf{U}_I\,\psi & \Leftrightarrow & \text{any infinite run } \varrho \text{ in } \mathcal{A} \text{ from } (\ell, v) \\
 & & \text{is such that } \varrho \models \phi\,\mathbf{U}_I\,\psi \\
\varrho \models \phi\,\mathbf{U}_I\,\psi & \Leftrightarrow & \text{there exists a position } \pi > 0 \text{ along } \varrho \text{ such that} \\
 & & \varrho[\pi] \models \psi, \text{ for every position } 0 < \pi' < \pi, \\
 & & \varrho[\pi'] \models \phi, \text{ and } \mathsf{duration}(\varrho_{\leq \pi}) \in I
\end{array}$$

where $\varrho[\pi]$ is the state of ρ at position π, $\varrho_{\leq \pi}$ is the finite prefix of ϱ ending at position π, and $\mathsf{duration}(\varrho_{\leq \pi})$ is the sum of all delays along ϱ up to position π. The $\mathbf{U}$-modality is called the 'Until' operator.

[12] TCTL stands for "Timed Computation Tree Logic" and has been defined in [ACD90].

In the continuous semantics, a position in a run ρ is any state appearing along ρ.[13] For instance if there is a transition $(\ell, v) \xrightarrow{\tau,e} (\ell', v')$ in ρ, then any state $(\ell, v+t)$ with $0 \leq t \leq \tau$ is a position of ρ, and obviously, so is (ℓ', v'). This semantics is very strong because for ρ to satisfy $\phi \, \mathbf{U}_{\sim c} \, \psi$, all intermediary states of ρ need to satisfy ϕ before ψ holds.

In the pointwise semantics, a position in a run $\rho = s_0 \xrightarrow{\tau_1,e_1} s_1 \xrightarrow{\tau_2,e_2} s_2 \cdots s_{n-1} \xrightarrow{\tau_n,e_n} s_n \cdots$ is an integer i and the corresponding state s_i. In this semantics, formulas are checked only right after a discrete action has been done. Sometimes, the pointwise semantics is given in terms of actions and timed words, but it does not change anything. Later, we may sometimes use the timed words terminology.

As usually in CTL, we define syntactic sugar to TCTL: $\mathtt{tt} \equiv a \vee \neg a$ standing for true, $\mathtt{ff} \equiv \neg \mathtt{tt}$ standing for false, the implication $\varphi \to \psi \equiv (\neg \varphi \vee \psi)$, the eventuality operator $\mathbf{F}_I \, \phi \equiv \mathtt{tt} \, \mathbf{U}_I \, \phi$, and the globally operator $\mathbf{G}_I \, \phi \equiv \neg(\mathbf{F}_I \, \neg \phi)$.

Example 8. In TCTL, we can write many kinds of properties, for instance, bounded-response time properties like

$$\mathbf{AG}(a \to \mathbf{AF}_{\leq 56} \, b) \tag{1}$$

expressing that each time a holds, along all possible runs, b has to hold within 56 time units.

Remark 9. In [HNSY94], TCTL is given with external clock variables. That is, we can use variables to express timing constraints. We will not give the precise grammar of that version of TCTL, but just give the equivalent of formula (1) in this framework:

$$\mathbf{AG} \, (a \to x.\mathbf{AF} \, (b \wedge x \leq 56))$$

The interpretation of that formula is the following: each time an a is encountered, we reset a clock x, and check that along all possible runs, later, b holds and the value of the clock x (which has increased at the same speed as the universal time) is not more than 56.

In [BCM05], it has been proved that TCTL with external clock variables is strictly more expressive than TCTL with intervals constraining the modalities.

Decidability. The model-checking problem asks, given a timed automaton $\mathcal{A}$ and a TCTL formula ϕ, whether $\mathcal{A}$ satisfies ϕ from its initial configuration $(\ell_0, \mathbf{0}_X)$.

Theorem 3. *For the two semantics, the model-checking problem for* TCTL *is PSPACE-complete.*

Proof. The hardness comes from the complexity of deciding reachability properties in timed automata (which can be expressed using formula $\mathbf{EF} \, q_f$, where q_f is an atomic proposition labelling the final state).

For the upper bound, we fix a diagonal-free timed automaton $\mathcal{A} = (L, \ell_0, X, T, \mathcal{L})$, and we consider its corresponding region equivalence, which we denote $\equiv$. We do the proof in the framework of the pointwise semantics, but it can be extended to the continuous semantics. We show the following lemma:

[13] That can be formalized, see for instance [BBBL05].

Lemma 8. *Let ϕ be a TCTL formula. For every $\ell \in L$, for all $v, v' \in \mathbb{T}^X$ such that $v \equiv v'$,*

$$(\ell, v) \models \phi \qquad \Leftrightarrow \qquad (\ell, v') \models \phi$$

Proof. We do the proof by a structural induction on ϕ. We therefore assume that $\phi = \mathbf{E}\phi_1\mathbf{U}_I\phi_2$, where the ϕ_i's are equally satisfied within a region.

We fix a constant M, which is larger than the finite bounds defining I. We define a new set of clocks $\widetilde{X} = X \cup \{u\}$, where u is a fresh clock. This clock will be used to measure the duration of the execution, which needs to lie in I.

We then consider the region equivalence over $\widetilde{X}$, denoted $\widetilde{\equiv}$, which refines $\equiv$ by adding clock u with maximal constant M. That is, if r is a region of $\widetilde{\equiv}$, then $r_{|X}$ is a region of $\equiv$.

We now assume that $(\ell, v) \models \mathbf{E}\phi_1\mathbf{U}_I\phi_2$. This means that there is an execution in $\mathcal{A}$ $\rho = (\ell, v_0) \xrightarrow{d_1,e_1} (\ell_1, v_1) \xrightarrow{d_2,e_2} \ldots \xrightarrow{d_k,e_k} (\ell_k, v_k) \to \ldots$ such that $v_0 = v$ and $\rho \models \phi_1\mathbf{U}_I\phi_2$. There exists k such that (a) $(\ell_k, v_k) \models \phi_2$, (b) for every $1 \leq j < k$, $(\ell_j, v_j) \models \phi_1$, and (c) $\mathsf{duration}(\rho_{\leq k}) \in I$.

Expanding v_i into $\widetilde{v}_i$ over $\widetilde{X}$ such that $\widetilde{v}_i(x) = v_i(x)$ for every $x \in X$ and $\widetilde{v}_i(u) = \sum_{j \leq i} d_j$, we get an execution $\widetilde{\rho} = (\ell, \widetilde{v}_0) \xrightarrow{d_1,e_1} (\ell_1, \widetilde{v}_1) \xrightarrow{d_2,e_2} \ldots \xrightarrow{d_k,e_k} (\ell_k, \widetilde{v}_k) \to \ldots$ such that $\widetilde{\rho} \models \phi_1\mathbf{U}_I\phi_2$. Indeed the same k holds as a witness of the formula. Note furthermore that $\mathsf{duration}(\rho_{\leq k}) = \widetilde{v}_k(u)$, hence $\widetilde{v}_k(u) \in I$ (clock u is not constrained in $\mathcal{A}$.

Pick v' such that $v \equiv v'$, write $v'_0 = v'$, and define $\widetilde{v}'_0$ such that $\widetilde{v}'_0(x) = v'_0(x)$ if $x \in X$ and $\widetilde{v}'_0(u) = 0$. Then $\widetilde{v}_0 \widetilde{\equiv} \widetilde{v}'_0$. Hence one can build an execution $\widetilde{\rho}' = (\ell, \widetilde{v}'_0) \xrightarrow{d'_1,e_1} (\ell_1, \widetilde{v}'_1) \xrightarrow{d'_2,e_2} \ldots \xrightarrow{d'_k,e_k} (\ell_k, \widetilde{v}'_k) \to \ldots$ such that for every $i \geq 0$, $\widetilde{v}'_i \widetilde{\equiv} \widetilde{v}_i$. Define v'_i the projection of $\widetilde{v}'_i$ over X. Then, $v_i \equiv v'_i$. It holds that $(\ell_k, v'_k) \models \phi_2$, and for every $1 \leq j < k$, $(\ell_j, v'_j) \models \phi_1$. Now, since $\widetilde{v}_k \widetilde{\equiv} \widetilde{v}'_k$, we also have that $\widetilde{v}'_k(u) \in I$, which implies that ρ' defined as the projection of $\widetilde{\rho}'$ over X, satisfies $\phi_1\mathbf{U}_I\phi_2$. This implies the expected result that $\mathcal{A}, (\ell, v') \models \mathbf{E}\,\phi_1\,\mathbf{U}_I\,\phi_2$. ⌟

Using a labelling algorithm as for CTL, we can therefore decide the model-checking problem for TCTL. By guessing and recomputing the truth for subformulas on-demand, we can improve the algorithm and make it use polynomial space only (witness paths have at most exponential length). This concludes the proof. ⌟

6.2 Timed extensions of LTL

Like classical temporal logics, linear-time property languages have also been studied in the framework of timed systems. Two main temporal formalisms have been defined: (i) MTL[14] is the counterpart of TCTL without external clock variables, and extends LTL [Pnu77] by adding timing constraints on modalities; (ii) TPTL[15] extends LTL by adding timing constraints to specifications using external variables and constraints thereon.

[14] MTL stands for "Metric Temporal Logic" and has been first proposed by Koymans [Koy90].
[15] TPTL stands for "Timed Propositional Temporal Logic", and has been first proposed by Alur and Henzinger [AH89,AH94].

Syntax and semantics of MTL. The syntax of MTL is given by the following grammar:

$$\mathsf{MTL} \ni \phi \ ::= \ a \ \mid \ \neg\phi \ \mid \ \phi \vee \phi \ \mid \ \phi \, \mathbf{U}_I \, \phi$$

where $a \in \mathsf{AP}$, and I is an interval of $\mathbb{R}_+$ with integral bounds.

It will even be more important in the context of linear-time timed temporal logics: as for TCTL, we distinguish between the two semantics, pointwise and continuous. Let $\rho = s_0 \xrightarrow{\tau_1, e_1} s_1 \xrightarrow{\tau_2, e_2} s_2 \cdots s_{n-1} \xrightarrow{\tau_n, e_n} s_n \cdots$ with $s_0 = (\ell_0, v_0)$ be a finite or infinite run. Then:

$$\begin{aligned}
\rho \models a \ &\Leftrightarrow \ a \in \mathcal{L}(\ell_0) \\
\rho \models \neg\phi \ &\Leftrightarrow \ \rho \not\models \phi \\
\rho \models \phi \vee \psi \ &\Leftrightarrow \ \rho \models \phi \text{ or } \rho \models \psi \\
\rho \models \phi \, \mathbf{U}_I \, \psi \ &\Leftrightarrow \ \text{there exists a position } \pi > 0 \text{ along } \varrho \text{ s.t.} \\
& \quad\ \varrho_{\geq \pi} \models \psi, \ \text{for every position } 0 < \pi' < \pi, \\
& \quad\ \varrho_{\geq \pi'} \models \phi, \ \text{and } \mathsf{duration}(\varrho_{\leq \pi}) \in I
\end{aligned}$$

with the same distinctions for the term 'position', depending on the choice of the semantics.

As for LTL, we define some syntactic sugar for MTL: $\mathtt{tt} \equiv (a \vee \neg a)$ stands for true, $\mathtt{ff} \equiv (\neg \mathtt{tt})$ stands for false, $(\varphi \rightarrow \psi) \equiv (\neg\varphi \vee \psi)$, $\mathbf{F}_I \, \varphi \equiv (\mathtt{tt} \, \mathbf{U}_I \, \varphi)$ (eventually, φ will hold within interval I from now), $\mathbf{G}_I \varphi \equiv \neg(\mathbf{F}_I \, \neg\varphi)$ (for all positions within I, φ holds), and $\mathbf{X}_I \varphi \equiv (\mathtt{ff} \, \mathbf{U}_I \, \varphi)$ (next position is within I from now and satisfies φ). Moreover, we use pseudo-arithmetical espressions to represent intervals. For instance, '$= 1$' stands for the singleton interval $[1, 1]$, and '≥ 2' stands for the interval $[2, +\infty)$.

Example 9. Using MTL, we can write properties like

$$\mathbf{G} \, (\mathsf{problem} \rightarrow \mathbf{F}_{\leq 56} \, \mathsf{alarm}) \tag{2}$$

expressing that each time a $\mathsf{problem}$ occurs, within 56 time units, an alarm rings.

We can also express more involved properties, like

$$\mathbf{G} \, (\mathsf{problem} \rightarrow (\mathbf{F}_{\leq 15} \, \mathsf{repair} \vee \mathbf{F}_{[12,15]} \, \mathsf{alarm}))$$

which expresses that each time a problem occurs, then either it is repaired in no more than 15 time units, or an alarm rings for 3 time units 12 time units after the problem. There is no direct and obvious way to express this kind of property in TCTL.

Remark 10. The choice of the interpretation of MTL in terms of the pointwise or of the continuous semantics has a large impact on the meaning of the formulas, and as we will see later, also on their applicability in model-checking. The formula $\mathbf{F}_{=2} \, a$ expresses that an a will happen two time units later. In the continuous semantics, this formula is equivalent to $\mathbf{F}_{=1} \mathbf{F}_{=1} \, a$ (in one time unit, it will be the case that in one time unit, an a occurs). However, it is not the case in the pointwise semantics, as there may be no action one time unit later, hence any formula $\mathbf{F}_{=1} \, \psi$ would be evaluated as wrong from the initial point.

Two extensions: TPTL and $\mathsf{MTL+Past}$ In the following, we will also consider two extensions of MTL. First, as for TCTL, external clock variables can be used to express timing constraints. For instance, property (2) can be written as

$$\mathbf{G}\,(\mathsf{problem} \rightarrow x.\mathbf{F}(\mathsf{alarm} \wedge x \leq 56))$$

where x is a fresh variable which is reset when a problem occurs, and whose value is checked to be within $[0,56]$ when the alarm rings. The value of x is supposed to evolve at the same speed as the universal time (similar to a clock in a timed automaton). This logic with external clock variables is called TPTL, and has been first proposed in [AH89]. We give another example of formulas that can be expressed in TPTL:

$$\mathbf{G}\,(\mathsf{problem} \rightarrow x.\mathbf{F}(\mathsf{alarm} \wedge \mathbf{F}(\mathsf{failsafe} \wedge x \leq 56))) \tag{3}$$

This formula says that whenever a problem occurs, then within 56 time units, an alarm rings and later (but still within 56 time units since the problem occurred), the system enters a failsafe mode. It has been proved in [BCM05] that TPTL is strictly more expressive than MTL. For the pointwise semantics, formula (3) is a witness to that expressiveness result, meaning that formula (3) cannot be expressed in MTL. Surprisingly, this formula has an equivalent formula in MTL,[16] and a more involved formula has been proposed to distinguish between MTL and TPTL.

Following the classical untimed framework [Kam68,LPZ85], we also extend MTL with past-time modalities, *i.e.*, with the 'Since' modality, somewhat the dual of the 'Until' modality: formula $\varphi\,\mathbf{S}_I\,\psi$ expresses that φ holds since ψ was true (within I in the past). In the following, we will only use the simple formula $\mathbf{F}_I^{-1}\varphi$ which is the dual of $\mathbf{F}_I\,\varphi$ for the past: it expresses that φ was true in the past, within a delay belonging to the interval I. For instance, the formula

$$\mathbf{G}\,(a \rightarrow \mathbf{F}_{=1}^{-1}b)$$

expresses that every a is preceded one time unit earlier by a b. This logic is denoted $\mathsf{MTL+Past}$.

6.3 The model-checking problem

The model-checking problem asks, given $\mathcal{A}$ a timed automaton and φ a formula, whether $\mathcal{A}$ satisfies φ, written $\mathcal{A} \models \varphi$, and meaning that all (accepting) runs of $\mathcal{A}$ satisfy the formula φ. We then give hints for understanding the following (un)decidability results:

Theorem 4. *Over finite runs, the model-checking problem for:*

[16] Indeed, we can prove (*cf* [BCM05]) that formula $x.\mathbf{F}(\mathsf{alarm} \wedge \mathbf{F}(\mathsf{failsafe} \wedge x \leq 56))$ is equivalent to

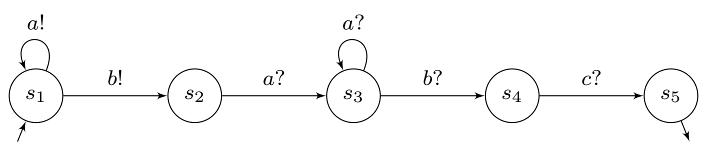
$$\begin{aligned} &\mathbf{F}_{\leq 28}\mathsf{alarm} \wedge \mathbf{F}_{[28,56]}\mathsf{failsafe} \\ \vee\ &\mathbf{F}_{\leq 28}(\mathsf{alarm} \wedge \mathbf{F}_{\leq 28}\mathsf{failsafe}) \\ \vee\ &\mathbf{F}_{\leq 28}(\mathbf{F}_{\leq 28}\mathsf{alarm} \wedge \mathbf{F}_{=28}\mathsf{failsafe}) \end{aligned}$$

- MTL *under the pointwise semantics is decidable, and non-primitive recursive [OW07];*
- MTL *under the continuous semantics is undecidable [AFH96];*
- MTL+Past *under the pointwise or the continuous semantics is undecidable;*
- TPTL *under the pointwise or the continuous semantics is undecidable [AH94].*

From these (un)decidability results, we learn that model-checking linear-time timed temporal logic is hard! And much harder than branching-time timed temporal logic. This is already the case in the untimed framework, but the gap dramatically increases in the timed framework. We first explain why it is so hard to model-check linear-time timed temporal logics. For that, we follow ideas developed in [Che07].

Model-checking linear-time timed properties is hard We first explain the non-primitive recursive lower bound for the MTL model-checking problem, which relies on the halting problem for channel machines with insertion errors. A *channel machine* is a finite automaton which can write on a channel and read from it following a FIFO policy. We note '$a!$' for writing a at the tail of the channel and '$a?$' for reading an a at the head of the channel. A channel machine has insertion errors if any letter can be written at any time anywhere in the channel. A channel machine without insertion errors is said perfect, or insertion-free.

Example 10. Consider the channel machine depicted on the next figure:



A configuration of this system is a pair (s, w) where s is a discrete state of the machine and w is a word representing the content of the channel. We give an error-free computation example for that machine:

$$(s_1, \varepsilon) \xrightarrow{a!} (s_1, a) \xrightarrow{a!} (s_1, aa) \xrightarrow{b!} (s_2, aab) \xrightarrow{a?} (s_3, ab) \xrightarrow{a?} (s_3, b) \xrightarrow{b?} (s_4, \varepsilon)$$

We can see that no error-free computation allows to reach state s_5 (because no c is written on the channel). If we assume that this machine has insertion errors, then the following move is allowed:

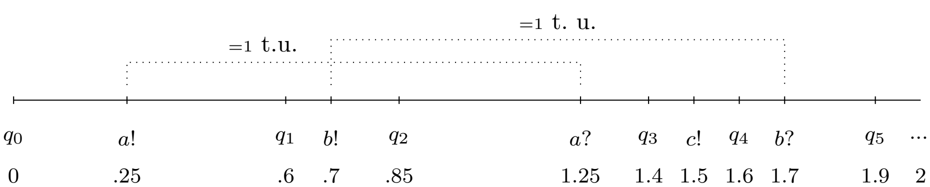
$$(s_4, \varepsilon) \xrightarrow{c?} (s_5, \varepsilon)$$

(we assume implicitely that a c has been inserted on the channel, so that the last transition labelled by '$c?$' can now be fired).

Given a channel machine $\mathcal{C}$ with a distinguished final state, the *halting problem* asks whether the machine $\mathcal{C}$ has an execution halting in the final state, with empty channel. The hardness results stated in Theorem 4 will be proved by reduction to the following problems about channel machines.

Proposition 6. – *The halting problem is undecidable for channel systems [BZ83].*
– *The halting problem with empty channel is non-primitive recursive for channel machines with insertion errors [Sch02].*[17]

We now explain how MTL (and variants) can capture the behaviours of channel machines. The idea is to encode a computation of a channel machine as a timed word. In this encoding, the underlying untimed word is the trace of the computation, that is, an alternating sequence of states and actions. We use timing constraints to enforce the channel be FIFO: we require that any write action '$a!$' is followed one time unit later by a corresponding read action '$a?$'. This is not difficult to be convinced that this enforces the channel be FIFO. We illustrate this encoding on the next figure, which represents a timed word (actions and time stamps).



The above timed word encodes the following computation of the channel machine:

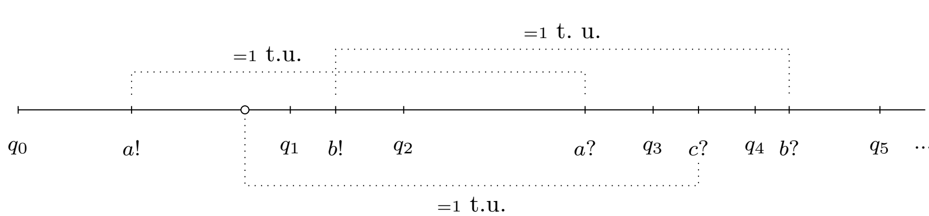
$$(q_0, \varepsilon) \xrightarrow{a!} (q_1, a) \xrightarrow{b!} (q_2, ab) \xrightarrow{a?} (q_3, b) \xrightarrow{c!} (q_4, bc) \xrightarrow{b?} (q_5, c) \cdots$$

To properly encode a behaviour of a channel machine, a timed word must satisfy the following constraints:

– states and actions alternate. This can be checked using an LTL formula.
– the untimed projection of the timed words follows the rules of the channel machine. This can also be encoded with an LTL formula.
– the channel is FIFO: to do that we express that every write action is followed one time unit later by a corresponding read action. This can be expressed in MTL using formulas of the form:

$$\mathbf{G}\ (a! \rightarrow \mathbf{F}_{=1} a?)$$

However, this formula does not encode the property that the channel behaves properly. Indeed, nothing prevents a read event '$a?$' to happen, even though there is no corresponding write event '$a!$' one time unit earlier. For instance, consider the following timed word:



17 Formally, in [Sch02], that's the halting problem for lossy channel machines which is proved non-primitive recursive, but this is indeed equivalent.

This timed word satisfies the propagating formulas $\mathbf{G}\ (a! \to \mathbf{F}_{=1} a?)$ (for every letter a), even though the event '$c?$' is not preceded by any action one unit earlier. The above formula hence only encodes the behaviour of a channel machine *with* insertion errors. However, from that study, we already learn that the model-checking of MTL (over finite words) is non-primitive recursive. To encode a perfect channel machine, we need to be able to express the property that every '$a?$' is preceded one time unit earlier by an '$a!$'. We call this property the 'backward matching property'.

We now discuss how we can express the backward matching property in timed temporal logics. Indeed, we would like to know whether MTL can express or not the behaviour of a perfect channel machine. We will present here natural ideas, which will happen to be wrong for MTL, but sufficient to prove undecidability of several variants or extensions of MTL.

- A first simple idea is to express this 'backward matching property' using the following formula:

$$\mathbf{G}\ ((\mathbf{F}_{=1} a?) \to a!)$$

  which expresses the fact that if there is a read event '$a?$' one time unit later, then there must be right now a corresponding write event '$a!$'. It is not hard to see that in the pointwise semantics, this does not express what we want. Indeed this formula is still satisfied by the above-mentioned timed word, because there is no action one time unit before the action '$c?$'. However, in the continuous semantics, this formula really enforces a perfect behaviour of the FIFO channel. That is why MTL in the continuous semantics has an undecidable model-checking problem.
- A second idea is to express this 'backward matching property' using a past-time modality (hence in MTL+Past). The formula

$$\mathbf{G}\ (a? \to \mathbf{F}^{-1}_{=1} a!)$$

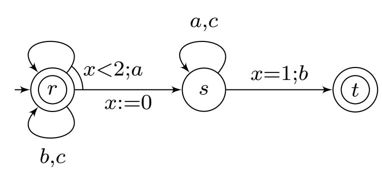
  precisely expresses that every read event '$a?$' is preceded one time unit earlier by a matching write event '$a!$'. That is why MTL+Past is undecidable, even in the pointwise semantics.
- Finally, the 'backward matching property' can be expressed in TPTL using the following more involved property:

$$\neg\Big(\mathbf{F} x \cdot \mathbf{X}\big(y \cdot \mathbf{F}(x > 1 \wedge y < 1 \wedge a?)\big)\Big)$$

  Informally, this formula negates the fact that there are two consecutive positions (in the pointwise sense) such that an a is read more than one time unit after the first position, and less than time unit after the second position. This precisely negates the fact that there is an '$a?$' not preceded one time unit earlier by an action. This implies that TPTL is undecidable, already in the pointwise semantics (when at least two clock variables are used).

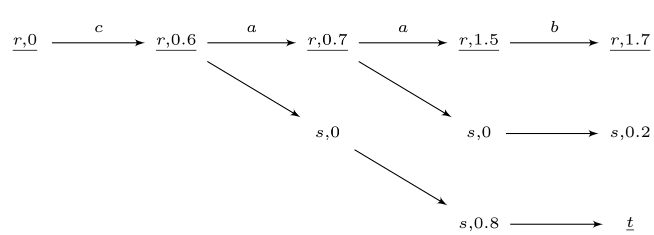
From all these considerations, we get that in the pointwise semantics, over finite runs, we can only express channel machines with insertion errors with MTL, whereas perfect channel machines can be expressed either using MTL in the continuous semantics, or using MTL+Past or TPTL in the pointwise semantics (both over finite words). This concludes the hardness results stated in Theorem 4.

MTL model-checking over finite words is decidable We now explain how we can prove the decidability of MTL over finite words in the pointwise semantics. We know that LTL formulas can be transformed into alternating finite automata [MSS88,Var96]. In a similar way, we can transform any formula of MTL into an alternating timed automaton [LW05] with a single clock [OW05]. For instance, the formula $\mathbf{G}_{<2}\Big(a \to \mathbf{F}_{=1} b\Big)$ can be transformed into the following alternating timed automaton:



with the obvious interpretation that any time an a is done (within the two first time units), we fork a new thread which will check that a b appears one time unit later. A behaviour of this alternating timed automaton is an unbounded tree, and it is not obvious that it is possible to check for emptiness of such a system. Indeed, checking emptiness of alternating timed automata is decidable only for one clock over finite timed words, any slight extension (infinite timed words, two clocks, silent moves[18]) leading to undecidability [LW08,OW07].

We explain how we can however understand the decidability of this model [OW05]. Consider the timed word $(c, 0.6)(a, 0.7)(a, 1.5)(b, 1.7)$. The execution of the above alternating timed automaton on that timed word can be depicted as the following tree, which is not accepting as one of the branches (the second one on the picture) is not accepting (accepting states are underlined).



A configuration of the alternating timed automaton is a slice of the tree, for instance, $\{(r, 1.5), (s, 0), (s, 0.8)\}$ is a configuration. Because we consider finite words, there is no need to consider the tree structure of the execution, but we can reason globally on configurations of the automaton. There are infinitely many such configurations, but as for the region automaton construction for timed automata [AD94], the precise values of the clocks is not

[18] Or ε-transitions, if we follow the classical terminology in formal language theory.

really relevant, and the things which are important in a configuration are the integral parts of the clocks and the relative order of the fractional parts of the clocks. For instance, for the above-mentioned configuration, we only need to know that there is a state $(s, 0)$ with fractional part 0, and two other states $(s, 0)$ and $(r, 1)$ such that the fractional part for $(s, 0)$ is greater than the fractional part for $(r, 1)$. For all configurations with the same abstraction, the possible future behaviours are the same, in a time-abstract bisimulation sense [OW07,LW08]. Unfortunately, the set of abstractions of possible configurations of the alternating timed automaton is also infinite. The most important property is then that there is a *well-quasi-order* on the set of abstractions of configurations, and that we can use it to provide an algorithm to decide emptiness [FS01]! This briefly sketches an algorithm for deciding the MTL model-checking problem over finite timed words (in the pointwise semantics).

Note that we can prove the decidability of TPTL with a single internal variable applying the same method.

7 The language-theoretic perspective

In the previous section we have presented the region automaton abstraction, which can be used to model-check several kinds of simple properties, like reachability properties. From a language perspective, this means that the emptiness problem is decidable for timed automata. In this section we study further language-theoretic properties of timed languages accepted by timed automata, and show in particular some negative results.

7.1 Boolean operations

Closure under Boolean operations is a basic property which is interesting for modelling and verification reasons.

Proposition 7. *The class of timed languages accepted by timed automata is closed under finite union and finite intersection.*

Proof (Sketch of proof). Closure under finite union is rather straightforward by taking the disjoint union of all timed automata.

The closure under finite intersection follows the lines of the standard product construction used in the case of finite automata. Only clock constraints, invariants and resets of clocks need be carefully handled. We illustrate the general construction with the intersection of two timed automata $\mathcal{A}_1 = (L_1, L_{1,0}, L_{1,F}, X_1, \Sigma, T_1, \mathsf{Inv}_1)$ and $\mathcal{A}_2 = (L_2, L_{2,0}, L_{2,F}, X_2, \Sigma, T_2, \mathsf{Inv}_2)$ over a single alphabet Σ. We assume that the two sets of clocks X_1 and X_2 are disjoint (otherwise we rename clocks so that it is actually the case). Then we define the timed automaton $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T, \mathsf{Inv})$ by:

- $L = L_1 \times L_2$, $L_0 = L_{1,0} \times L_{2,0}$, $L_F = L_{1,F} \times L_{2,F}$;
- $X = X_1 \cup X_2$ (disjoint union);

– the set T is composed of transitions of the form $(\ell_1, \ell_2) \xrightarrow{g,a,Y} (\ell_1', \ell_2')$ whenever there exist two transitions $\ell_1 \xrightarrow{g_1,a,Y_1} \ell_1'$ in T_1 and $\ell_2 \xrightarrow{g_2,a,Y_2} \ell_2'$ in T_2 such that:
  - $g = g_1 \wedge g_2$;
  - $Y = Y_1 \cup Y_2$;

– $\mathsf{Inv}((\ell_1, \ell_2)) = \mathsf{Inv}_1(\ell_1) \wedge \mathsf{Inv}_2(\ell_2)$.

This is straightforward to prove that a timed word is accepted by $\mathcal{A}$ iff it is both accepted by $\mathcal{A}_1$ and $\mathcal{A}_2$.

The following proposition is on the contrary rather bad news.

Proposition 8. *The class of timed languages accepted by timed automata is not closed under complementation.*

The most well-known timed automaton, already given in [AD94], which cannot be complemented is given in Figure 6. This automaton, over the single-letter alphabet $\{a\}$,

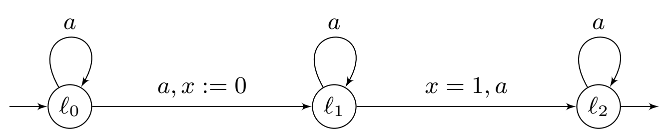


Fig. 6: *A non-complementable timed automaton*

recognizes the timed language:

$$\{(a, t_1)(a, t_2) \dots (a, t_n) \mid n \in \mathbb{N},\ n \geq 2 \text{ and there exist } 1 \leq i < j \leq n \text{ with } t_j - t_i = 1\}$$

Intuitively, to be recognized by a timed automaton, the complement of this timed language would require an unbounded number of clocks, because for any action a, we need to check that there is no a-action one time unit later, so a fresh clock is intuitively required. However the complete proof is rather technical and harassing [Bou98], and we do not provide it here.

An alternative and elegant proof of the above proposition has been proposed in [AM04], and this is the one we have decided to present here.

Proof. We consider the timed automaton of Figure 7. It accepts the following timed lan-

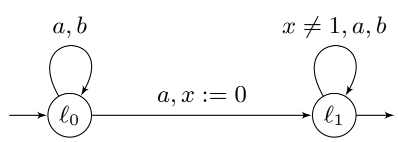


Fig. 7: *Another non-complementable timed automaton*

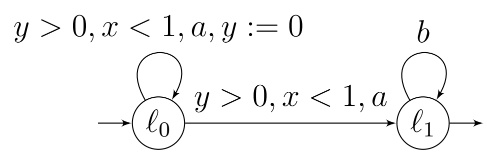
guage over the alphabet $\{a, b\}$:

$$L = \{(\alpha_1, t_1) \dots (\alpha_n, t_n) \mid n \in \mathbb{N},\ n \geq 1, \exists 1 \leq i \leq n \text{ s.t. } \alpha_i = a \text{ and } \forall i < j \leq n,\ t_j - t_i \neq 1\}$$

We assume towards a contradiction that $\overline{L}$ (the complement of L) can be recognized by a timed automaton. It is not hard to get convinced that the timed language over the alphabet $\{a, b\}$

$$L' = \{(a^+b^*, \tau) \mid \text{all } a's \text{ happen before 1 and no two } a's \text{ simultaneously}\}$$

is accepted by the timed automaton:



Hence by Proposition 7, $\overline{L} \cap L'$ is accepted by some timed automaton. The following lemma is just a matter of expanding and manipulating the definition of $\overline{L} \cap L'$.

Lemma 9. *The untiming of $\overline{L} \cap L'$ is the non-regular language*

$$\{a^n b^m \mid n \in \mathbb{N},\ n \geq 1 \text{ and } m \geq n\}.$$

This lemma yields a contradiction with the fact that $\overline{L} \cap L'$ is accepted by some timed automaton, say $\mathcal{B}$, because the untiming of $\overline{L} \cap L'$ should then be recognized by the region automaton of $\mathcal{B}$. Hence we conclude that the complement of L is not recognized by any timed automaton.

7.2 The universality and inclusion problems

The *universality problem* asks, given a timed automaton $\mathcal{A}$, whether $\mathcal{A}$ accepts all (finite) timed words. The *inclusion problem* asks, given two timed automata $\mathcal{A}$ and $\mathcal{B}$, whether all timed words accepted by $\mathcal{B}$ are also accepted by $\mathcal{A}$, that is whether $L(\mathcal{B}) \subseteq L(\mathcal{A})$. Note that the universality problem is a special instance of the inclusion problem, where $\mathcal{B}$ is universal, *i.e.* accepts all (finite) timed words. The following result is bad news in the verification context, as argued in Section 3.

Theorem 5 ([AD90,AD94]). *The universality problem is undecidable for timed automata.*

Proof. We encode the halting problem for *perfect* channel machines as a universality problem of a timed automaton.

Let $\mathcal{M}$ be a channel machine. We use the same encoding as for the logics MTL, that is, a (finite) feasible execution[19]

$$(q_0, \epsilon) \xrightarrow{\alpha_1} (q_1, w_1) \dots \xrightarrow{\alpha_n} (q_n, w_n)$$

[19] If $\alpha_i = a?$, then the first letter of w_{i-1} is a, and $w_{i-1} = aw_i$; if $\alpha_i = a!$, then $w_i = w_{i-1}a$.

of $\mathcal{M}$ will be encoded by the (finite) timed word

$$(q_0, t_0)(\alpha_1, t_1)(q_1, t_2) \dots (\alpha_n, t_{2n-1})(q_n, t_{2n})$$

such that:

① $(t_i)_{0 \le i \le 2n}$ is (stricly) increasing;
② for every i such that $\alpha_i = a!$, there exists $j > i$ such that $\alpha_j = a?$ and $t_j = t_i + 1$
③ for every j such that $\alpha_j = a?$, there exists $i < j$ such that $\alpha_i = a!$ and $t_i = t_j - 1$

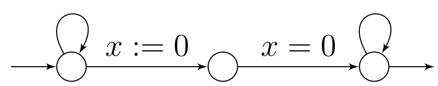
We build a timed automaton $\mathcal{A}$ (over finite words) which will accept all (finite) timed words which are not encodings of a halting computation of $\mathcal{M}$. The alphabet of $\mathcal{A}$ is:

$$Q \cup \{a?, a! \mid a \in \Sigma\}$$

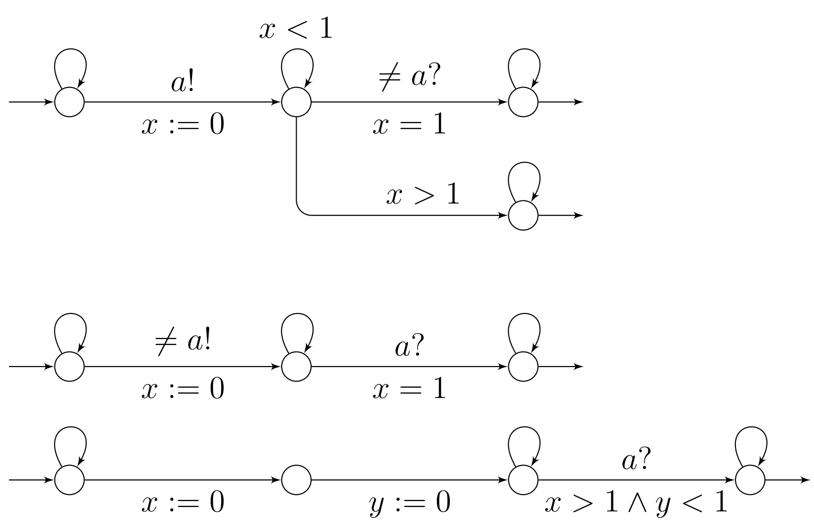
where Q is the set of states of $\mathcal{M}$ and Σ is the alphabet of $\mathcal{M}$. We write $\widetilde{\Sigma}$ for the set $\{a?, a! \mid a \in \Sigma\}$, $\Sigma_? = \{a? \mid a \in \Sigma\}$ and $\Sigma_! = \{a! \mid a \in \Sigma\}$.

This will be done using a highly non-deterministic timed automaton, which will deny one-by-one the various conditions:

- deny "correct alternations between states and actions, and follow the rules of $\mathcal{M}$": complement the automaton of the machine $\mathcal{M}$.
- deny "the sequence of timestamps is increasing"



- deny the timing constraints implementing the fifo rules



We can show that $\mathcal{A}$ is universal if and only if $\mathcal{M}$ does not halt. Indeed, assume $\mathcal{M}$ halts, and consider an accepting execution, and one of its corresponding encodings as a timed word. It satisfies all the rules, hence it cannot be accepted by $\mathcal{A}$. Conversely pick a timed word which is not accepted by $\mathcal{A}$. Then this word is a proper encoding of an halting execution in $\mathcal{M}$, otherwise one of the conditions would be denied and the word would be accepted by $\mathcal{A}$ □

The following is a straightforward corollary of the initial observation that the universality problem is a special instance of the inclusion problem.

Corollary 1. *The inclusion problem is undecidable for timed automata.*

It is interesting to notice that the reduction used in the above proof builds a timed automaton with two clocks. And actually, the universality problem (and also the inclusion problem) is decidable (but non-primitive recursive) for single-clock timed automata, see [ADOW05]. Recent developments have considered alternating timed automata (a natural extension of timed automata with alternations) [LW05,OW05,LW08,OW07], but Theorem 5 implies that the emptiness problem is undecidable for alternating timed automata.

7.3 Timed automata and determinism

In the context of formal languages, determinism is a standard and central notion which expresses that for a word there is at most one execution which reads that word. For regular languages determinism does not restrict recognition of languages, but for context-free languages this is not the case [HMU01]. We discuss in this section the issue of determinism in the context of timed automata, which gives some explanation to the previous negative results.

The class of deterministic timed automata. We give a syntactical definition of determinism in timed automata (with no invariants, for simplicity). A timed automaton $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T)$ is *deterministic* whenever L_0 is a singleton, and for every $\ell \in L$, for every $a \in \Sigma$, $(\ell, g_1, a, Y_1, \ell_1) \in T$ and $(\ell, g_2, a, Y_2, \ell_2) \in T$ imply $[\![g_1 \wedge g_2]\!]_X = \varnothing$. This notion extends in a natural way the standard notion of determinism in finite automata. In a deterministic timed automaton, for every timed word, there is at most one run that reads that timed word from a given configuration.

Example 11. The timed automaton of Figure 1 (see page 8) is deterministic. From location 'alarm', there are two outgoing transitions, but the constraints labelling those two transitions are disjoint. From the other locations, there is only one outgoing transition.

On the other hand, the timed automata of Figures 6 and 7 are not deterministic. In the first one, there is a non-deterministic choice from location ℓ_1, but it can be removed by strengthening the constraint on the self-loop (adding one self-loop with the constraint $x < 1$ and another one with the constraint $x > 1$). There is another non-determinisc choice from location ℓ_0, and this one cannot be removed (note that this is in general not obvious to see whether a non-deterministic choice can be removed or not!): it is not possible to predict when will be the occurrence of an a that will be followed one time unit later by another a.

Deterministic timed automata form a strict subclass of timed automata.[20] Using a product construction, as done in the proof of Proposition 7 for the intersection, it is easy

[20] The strictness is obvious at the syntactical level, and also holds at the semantical level, as will be argued later: there exists a timed automaton such that no deterministic timed automata accepts the same timed language.

to get convinced that this subclass is closed under finite union and finite intersection. On the other hand the two timed automata we have given to illustrate the non-closure under complementation of the class of standard timed automata (Proposition 8) are not deterministic. And actually it is not very hard to get convinced that the class of deterministic timed automata is closed under complementation: add a sink location, add transitions to that sink from every location, with constraints complementing the union of all the constraints labelling the outgoing transitions from that location, and finally swap final and non-final locations. As a consequence, this is not possible to construct deterministic timed automata which accept the same languages as the two timed automata of Figures 6 and 7. And it is even possible to prove that this is not possible to decide whether a timed automaton is determinizable or not [Tri03,Fin06].

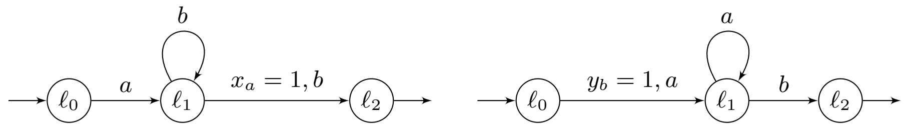
Finally it is interesting to mention that the reduction to prove the undecidability of the universality problem (proof of Theorem 5) builds a non deterministic timed automaton. And indeed the universality problem (and the inclusion problem) is decidable for the class of deterministic timed automata: to check for the universality of a given deterministic timed automaton $\mathcal{A}$, first build a (deterministic) timed automaton which accepts the complement of $L(\mathcal{A})$, and then check for emptiness of this automaton.

Determinizable classes of timed automata. As mentioned in the previous paragraph, not all timed automata can be determinized (*i.e.* there exist timed automata that accept timed languages which cannot be recognized by any deterministic timed automaton). However, deterministic (and hence effectively determinizable) timed automata enjoy nice closure (complementation) and decidability (universality, inclusion) properties. Verification can thus benefit of such properties.

One of the first determinizable classes of timed automata which have been investigated is the class of *event-clock timed automata* [AFH94]. In such an automaton every letter of the alphabet is associated two clocks, one which measures delays since the last occurrence of this action (those are called event-recording clock), and one which measures delays to the next occurrence of this action (those are called event-predicting clock). In the syntax of event-clock timed automata. resets of clocks are ommitted as they are implicitely given by actions.

Example 12. In Figure 8 we give two event-clock timed automata (we take the convention that x_a is the event-recording clock associated with a whereas y_b is the event-predicting clock associated with b). In the first automaton, the time between the last b and the unique initial a is precisely one time unit (specified with the constraint on the last transition $x_a = 1$): when we do the last b, we know that the last a was precisely one time unit earlier. In the second automaton, the time between the first a and the unique final b is precisely one time unit as well (specified with the constraint on the first transition $y_b = 1$): when the first a is done, we know that the next b has to be one time unit later.

We give the intuition why an event-clock timed automaton with only event-recording clocks can be determinized (the case of event-predicting clocks is more involved and we



x_a: event-recording clock for a $\quad$ y_b: event-predicting clock for b

Fig. 8: *Two event-clock timed automata*

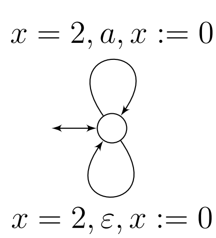
refer the reader to [AFH94] for more details). The reason is that the timed behaviour of those automata is input-determined: given a timed word, the value of the clocks after each prefix of the timed word is determined by that prefix (and not by the run followed in the timed automaton). For that reason a subset construction can be done. This kind of arguments has later been used for more complex classes of timed systems [DT04].

Recently more determinizable classes of timed automata have been investigated [BBBB09], among which we can find the class of so-called strongly non-Zeno timed automata (we omit the definition of this class here, but basically it enforces time elapsing in a rather strong way) or more dedicated classes, e.g. corresponding to logical formalisms [NP10] or to simpler classes of timed systems [SP09].

7.4 What about ε-transitions?

Following classical automata theory, we assume some transitions are silent, and we denote them ε-transitions.

Exercise 18. Consider the following timed automaton.

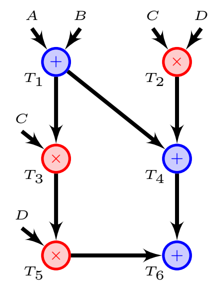


Prove that the timed language recognized by the above timed automaton cannot be recognized by any classical timed automaton with no ε-transitions. ⌟

8 The task-graph example

8.1 Description of the problem

Compute $D{\times}(C{\times}(A{+}B)){+}(A{+}B){+}(C{\times}D)$ using two processors:

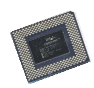


P_1 (fast):

	time
+	2 picoseconds
×	3 picoseconds

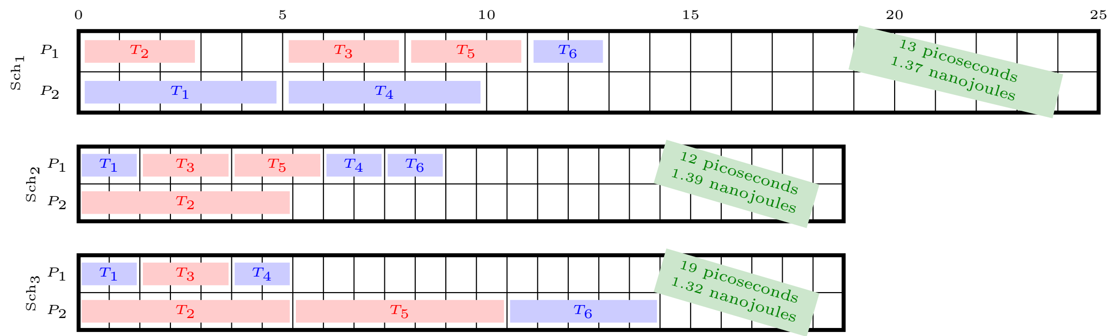
energy	
idle	10 Watt
in use	90 Watts

P_2 (slow):



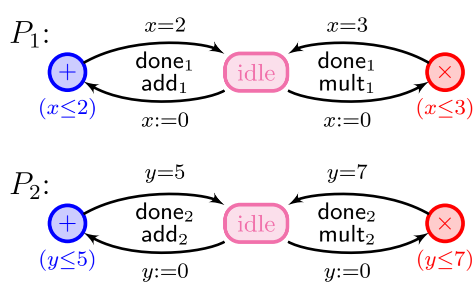
	time
+	5 picoseconds
×	7 picoseconds

energy	
idle	20 Watts
in use	30 Watts

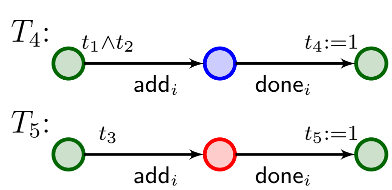


8.2 Modelization of the problem

– Processors



– Tasks



8.3 How to model more?

How should we take into account:

– uncertainties on delays? (not mentioned in the lecture)
– power consumption?
– job preemption?

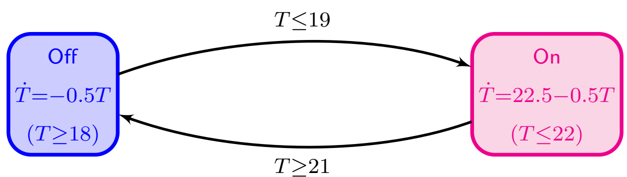
9 Hybrid systems

Hybrid systems are convenient models for representing general continuous behaviours. They have application in control theory, biology, etc.

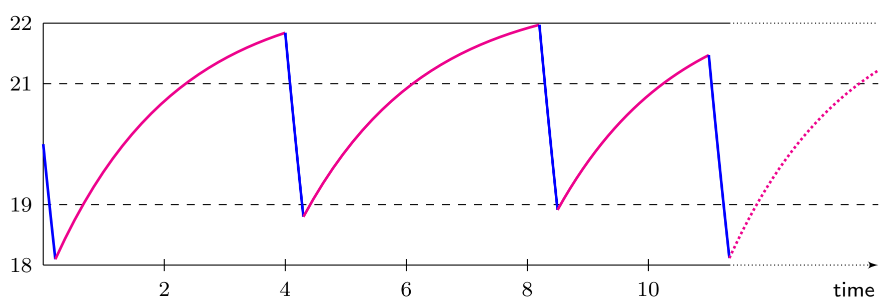
Example 13. We want to model a thermostat. If T is the temperature, it can written as a list of rules:

- when the heater is Off, the room cools down: $\dot{T} = -0.5T$;
- when the heater is On, the room heats: $\dot{T} = 22.5 - 0.5T$;
- when $21 \leq T \leq 22$, it switches Off;
- when $18 \leq T \leq 19$, it switches On.

This thermostat can be modelled by the following automaton:



A possible behaviour for that system is as follows:



A behaviour can also be seen as a sequence of delays and jumps.

Theorem 6. *The reachability problem is undecidable in hybrid automata.*

Therefore, to be able to analyze such systems, one needs to consider subclasses:

- dimension (number of variables), continuous- or discrete time
- what kind of dynamics
- what kind of guards, invariants, jumps

A zoology of subclasses:

- Linear hybrid automata: dynamics described by $\dot{x} = c$, polyhedral guards and invariants, linear resets
  - memory cell: $\dot{x} = 0$ (but x can be reset)
  - clock: $\dot{x} = 1$

        - stopwatch: in some locations, $\dot{x} = 1$, in others $\dot{x} = 0$
        - skewed clock: $\dot{x} = c$ in all locations
- Rectangular hybrid automata: dynamics described by $\dot{x} \in [c, d]$, guards and invariants described by $x \in [c, d]$, resets described by $x \in [c, d]$
    - Initialized rectangular hybrid automata: each time a variable changes its slope, it should be reset
    - Linear hybrid automata are rectangular hybrid automata
- O-minimal hybrid automata: strong resets between locations, everything described in an o-minimal theory
- Piecewise affine maps
- Piecewise constant derivatives (PCD):
- Polygonal differential inclusion systems (PDIS):
- *etc...*

Exercise 19. Back to the task-graph scheduling problem, how would you model the power consumption? And how would you model job preemption?

In this lecture we will study:

① Piecewise affine maps: rich jumps, poor dynamics
② (Initialized) Rectangular hybrid automata: decorrelated variables (rather poor jumps)
③ Weighted timed automata, a timed-automata based model with observer variables

9.1 Piecewise affine maps (PAMs)

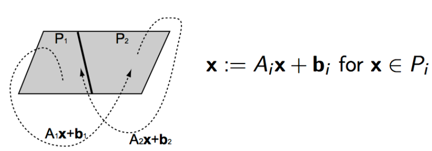
(Pictures borrowed from Eugene Asarin)

A d-dimensional *piecewise affine map* (PAM) is a discrete-time dynamical system defined as a tuple $(\mathcal{P}, (P_i)_{1 \leq i \leq n}, f)$ where:

- $\mathcal{P}$ is a bounded polyhedron of $\mathbb{R}^d$ (e.g. $\mathcal{P} = [0, 1]^d$);
- $(P_i)_{1 \leq i \leq n}$ is a polyhedral partition of $\mathcal{P}$;
- $f : \mathcal{P} \to \mathcal{P}$ is such that for every $1 \leq i \leq n$, $f_{|P_i}$ is affine.

The semantics of a PAM is given as a sequence $(f^i(x))_{i \geq 0}$, where $x \in \mathcal{P}$.

The reachability problem asks whether, given two points $x_1, x_2 \in \mathcal{P}$, there exists an index n such that $f^n(x_1) = x_2$.



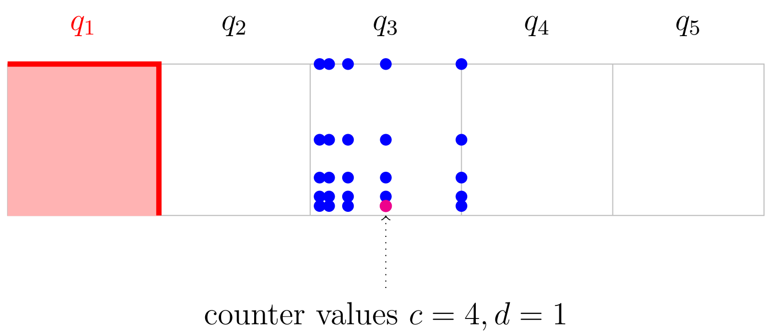
Theorem 7 ([KCG94]). *Two-dimensional PAMs are undecidable.*

Proof. Note that we cannot directly express a counter machine using a PAM since in a PAM, the domain is bounded.

However we can simulate a deterministic two-counter machine as follows: A state (q_i, c, d) will be represented by point $(i-1+2^{-c}, 2^{-d})$. The domain is therefore $(0, N] \times (0, 1]$ where N is the number of states of the two-counter machine. State q_i is encoded by the polyhedron $P_i = (i-1, i] \times (0, 1]$ (we may have to split this polyhedron for defining the affine functions). We now define the affine functions, depending on the instruction starting at q_i:

- if q_i : $c++$; `goto` q_j, then we should go from point $(i-1+2^{-c}, 2^{-d})$ to point $(j-1+2^{-c-1}, 2^{-d})$, that is $y := y$ and $x := j-1+\frac{x-(i-1)}{2}$
- if q_i : $c--$; `goto` q_j, then we should go from point $(i-1+2^{-c}, 2^{-d})$ to point $(j-1+2^{-c+1}, 2^{-d})$ in case $c > 0$, that is from $P_i \cap (x < i)$. The affine function is then defined as: $x := j-1+2(x-(i-1))$ and $y := y$
- if q_i : `if` $c > 0$ `then goto` q_j `else goto` q_k, then we should go from $P_i \cap (x < i)$ to P_j using $x := j-i+x$ and $y := y$ and from $P_i \cap (x = i)$ to P_k using $x := k-i+x$ and $y := y$
- if q_i : $d++$; `goto` q_j, then we should go from point $(i-1+2^{-c}, 2^{-d})$ to point $(j-1+2^{-c}, 2^{-d-1})$, that is $x := x+j-i$ and $y := \frac{y}{2}$
- if q_i : $d--$; `goto` q_j, then we should go from point $(i-1+2^{-c}, 2^{-d})$ to point $(j-1+2^{-c}, 2^{-d+1})$ on $P_i \cap (y < 1)$, that is $x := x+j-i$ and $y := 2y$
- if q_i : `if` $d > 0$ `then goto` q_j `else goto` q_k, then we should go from $P_i \cap y < 1$ to P_j using $x := j-i+x$ and $y := y$, and from $P_i \cap y = 1$ to P_k using $x := k-i+x$ and $y := y$.

The encoding is illustrated on the following picture. Blue dots represent counter values.



If q_n is the halting state, and q_1 is the initial state of the counter machine, the two-counter machine halts iff there is a path in the PAM from $(1, 1)$ (all counters set to 0 in q_1) to the area $(n-1, n] \times (0, 1]$. By adding instructions decreasing the two counters when reaching q_n, one can reduce to checking whether there is a path in the PAM from $(1, 1)$ to $(n, 1)$. □

Remark 11 (An example open problem). It is unknown whether we can decide the reachability problem in one-dimensional PAMs.

9.2 Rectangular hybrid automata

We write $\mathcal{I}$ for the set of intervals of $\mathbb{R}$ with rational bounds.

A *rectangular (hybrid) automaton* is a tuple $\mathcal{H} = (L, \ell_0, X, E, \mathsf{Inv}, \mathsf{Act}, \mathsf{Pre}, \mathsf{Post}, \mathsf{Upd})$ where:

- L is a finite set of locations;
- $\ell_0 \in L$ is the initial location;
- $X = \{x_1, \ldots, x_n\}$ is a finite set of variables;
- $E \subseteq L \times L$ is a finite set of edges, each edge $e = (\ell, \ell') \in E$ is characterized by its source, $\mathsf{src}(e) = \ell \in L$, and its target $\mathsf{target}(e) = \ell' \in L$;
- $\mathsf{Inv} : L \to \mathcal{I}^X$ assigns an invariant to every location;
- $\mathsf{Act} : L \to \mathcal{I}^X$ indicates the evolution (aka activity) of the variables in each location;
- $\mathsf{Pre} : E \to \mathcal{I}^X$ and $\mathsf{Post} : E \to \mathcal{I}^X$ are the pre- and post-conditions to be fulfilled when firing transitions;
- $\mathsf{Upd} : E \to 2^X$ indicates which variables should be updated.

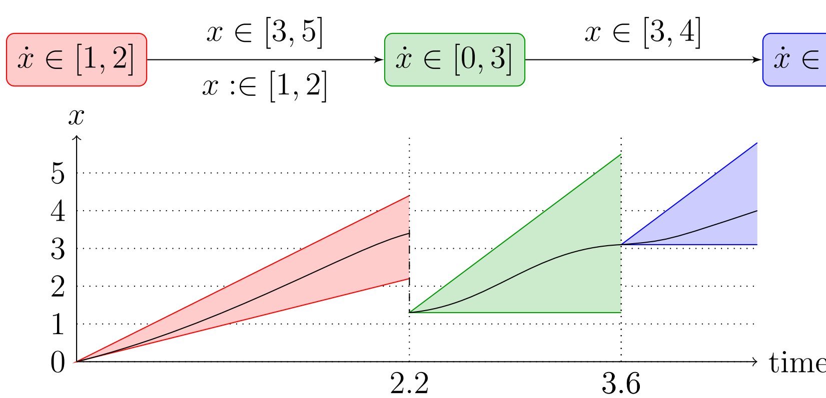
It is said *initialized* whenever for every $e \in E$ for every $x \in X$, either $x \in \mathsf{Upd}(e)$, or $\mathsf{Act}(\mathsf{src}(e))(x) = \mathsf{Act}(\mathsf{target}(e))(x)$.

We assume $\mathcal{H}$ is equipped with an extra variable t which is a clock.

The semantics of $\mathcal{H}$ is given as a timed transition system $(S, s_0, \to)$ defined by $S = \{(\ell, \nu) \in L \times \mathbb{R}^X \mid \forall x \in X,\ \nu(x) \in \mathsf{Inv}(\ell)(x)\}$, $s_0 = (\ell_0, \mathbf{0})$, and $\to$ is defined as follows: $(\ell, \nu) \to (\ell', \nu')$ iff one of the two following conditions are met:

- [**continuous step**] $\ell = \ell'$, and either $\nu = \nu'$, or $\nu'(t) > \nu(t)$, and for every $x \in X$,
$$\frac{\nu'(x) - \nu(x)}{\nu'(t) - \nu(t)} \in \mathsf{Act}(\ell)(x)$$
- [**discrete step (or jump)**] there is a transition $e \in E$ such that $\mathsf{src}(e) = \ell$, $\mathsf{target}(e) = \ell'$, $\nu \models \mathsf{Pre}(e)$, $\nu' \models \mathsf{Post}(e)$, $\nu'(t) = \nu(t)$, for all $x \in X$, $\nu'(x) = \nu(x)$ unless $x \in \mathsf{Upd}(e)$.

Example 14. The following is an example of an initialized rectangular automaton:



Remark 12. Obviously any timed automaton is an initialized rectangular automaton.

Undecidability of rectangular automata. A rectangular automaton is said *simple* whenever:

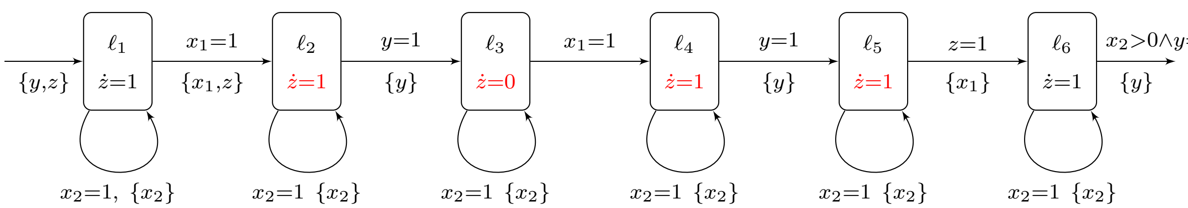
- only one variable, say z, is not a clock;
- the activity of z is singular in each location;
- Inv, Pre, Post have compact values;
- $\mathsf{Post}(e)(x) = [0, 0]$ if $x \in \mathsf{Upd}(e)$, and $\mathsf{Post}(e)(x) = \mathsf{Pre}(e)(x)$ otherwise.

Such a simple automaton is said 2-slope whenever the only non-clock variable can take two values only.

Theorem 8. *The reachability problem is undecidable for simple* 2*-slope rectangular automata.*

Proof. The result holds for any two different rationals serving as the slopes of the 2-slope variable, but we only prove it for the special case where the slopes are 0 and 1.

We begin with the case of stopwatches. We encode the halting problem for two-counter machines. Counters c_1 and c_2 are encoded by clocks x_1 and x_2, with $x_i = 2^{-c_i}$. We now have to be able to double and halve the value of a clock, preserving the value of the other clock at the same time. Doubling the value of a clock is achieved by the automaton depicted below:



Notice that, in this figure, we omit invariants in each state (but they can be easily inferred). Then it can be checked that the value of c_1 is doubled between the entry and exit of this automaton, while the value of c_2 is preserved. Halving the clock is achieved in a similar way, which can be done as an exercise. □

Decidability of initialized rectangular automata.

Theorem 9. *The reachability problem is decidable in initialized rectangular automata. It is even* PSPACE*-complete.*

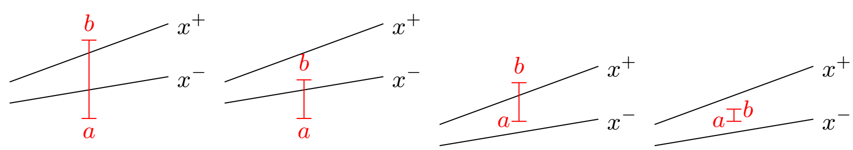
Proof. We assume $\mathcal{H}$ is a rectangular automataton. W.l.o.g. we assume that:

- $\mathsf{Pre}(e) \subseteq \mathsf{Inv}(\mathsf{src}(e))$;
- $\mathsf{Post}(e) \subseteq \mathsf{Inv}(\mathsf{target}(e))$;
- $\forall x \notin \mathsf{Upd}(e)$, $\mathsf{Pre}(e)(x) = \mathsf{Post}(e)(x)$.

W.l.o.g. we assume that Inv is always true (this does not affect reachability properties, because of the above assumptions). We furthermore assume that all intervals are compact. Obviously, this restriction can be relaxed.

We first construct a multi-rate automaton $\mathcal{M}$ (activities are singular). Locations are the same as in $\mathcal{H}$. We have that:

- its set of variables is $X' = \{x^-, x^+ \mid x \in X\}$;
  [variable x will take any value $x^- \leq x \leq x^+$]
- activites are defined by $\mathsf{Act}'(\ell)(x^+) = [b, b]$ and $\mathsf{Act}'(\ell)(x^-) = [a, a]$ whenever $\mathsf{Act}(\ell)(x) = [a, b]$;
- We will have several copies of every edge in $\mathcal{H}$: $E' = E \times 4^X$. They are defined as follows.
  - Assume $x \in \mathsf{Upd}(e)$. We write $\mathsf{Pre}(e)(x) = [a, b]$ and $\mathsf{Post}(e)(x) = [m, M]$. Then any transition $e' = (e, \sigma) \in E'$ will update x^- and x^+ ($\{x^-, x^+\} \subseteq \mathsf{Upd}'(e)$). Furthermore:
$$\begin{cases} \mathsf{Pre}'(e')(x^-) = (-\infty, b] \\ \mathsf{Pre}'(e')(x^+) = [a, +\infty) \end{cases} \qquad \begin{cases} \mathsf{Post}'(e')(x^-) = [m, m] \\ \mathsf{Post}'(e')(x^+) = [M, M] \end{cases}$$
  - Assume $x \notin \mathsf{Upd}(e)$. We write $\mathsf{Pre}(e)(x) = [a, b]$ and $\mathsf{Post}(e)(x) = [m, M]$. Note that by hypothesis, $[a, b] = [m, M]$. These constraints on x must be transferred to x^- and x^+. The way it will be handled depends on the relative positions of x^- and x^+ with regards to constants a and b.



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4

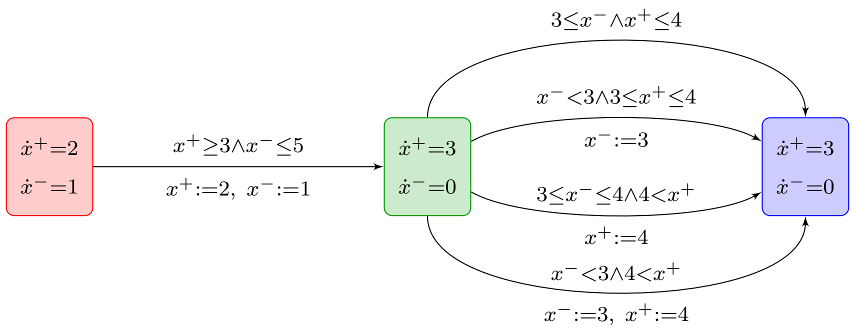
    * Case 1: We fix σ such that $\sigma(x) = 1$, and define $e' = (e, \sigma) \in E'$.
$$\begin{cases} \mathsf{Pre}'(e')(x^-) = [a, b] \\ \mathsf{Pre}'(e')(x^+) = [a, b] \end{cases} \qquad \begin{cases} \mathsf{Post}'(e')(x^-) = [a, b] \\ \mathsf{Post}'(e')(x^+) = [a, b] \end{cases}$$
      and variables x^- and x^+ are unchanged: $x^-, x^+ \notin \mathsf{Upd}'(e')$.
    * Case 2: We fix σ such that $\sigma(x) = 2$, and define $e' = (e, \sigma) \in E'$.
$$\begin{cases} \mathsf{Pre}'(e')(x^-) = [a, b] \\ \mathsf{Pre}'(e')(x^+) = (b, +\infty) \end{cases} \qquad \begin{cases} \mathsf{Post}'(e')(x^-) = [a, b] \\ \mathsf{Post}'(e')(x^+) = [b, b] \end{cases}$$
      variable x^- is unchanged whereas variable x^+ is changed: $x^- \notin \mathsf{Upd}'(e')$ but $x^+ \in \mathsf{Upd}'(e')$.
    * Case 3: We fix σ such that $\sigma(x) = 3$, and define $e' = (e, \sigma) \in E'$.
$$\begin{cases} \mathsf{Pre}'(e')(x^-) = (-\infty, a) \\ \mathsf{Pre}'(e')(x^+) = [a, b] \end{cases} \qquad \begin{cases} \mathsf{Post}'(e')(x^-) = [a, a] \\ \mathsf{Post}'(e')(x^+) = [a, b] \end{cases}$$
      variable x^+ is unchanged whereas variable x^- is changed: $x^+ \notin \mathsf{Upd}'(e')$ but $x^- \in \mathsf{Upd}'(e')$.

* Case 4: We fix σ such that $\sigma(x) = 4$, and define $e' = (e, \sigma) \in E'$.

$$\begin{cases} \mathsf{Pre}'(e')(x^-) = (-\infty, a) \\ \mathsf{Pre}'(e')(x^+) = (b, +\infty) \end{cases} \qquad \begin{cases} \mathsf{Post}'(e')(x^-) = [a, a] \\ \mathsf{Post}'(e')(x^+) = [b, b] \end{cases}$$

variables x^+ and x^- are changed: $\{x^-, x^+\} \subseteq \mathsf{Upd}'(e')$.

Example 15. The transformation applied to the previous example yields:



We now discuss the correctness of the construction of $\mathcal{M}$. For every configuration (ℓ, ν) of $\mathcal{M}$ we define the following set of configurations of $\mathcal{H}$:

$$\chi(\ell, \nu) = \{(\ell, v) \mid \forall x \in X,\ \nu(x^-) \leq v(x) \leq \nu(x^+)\}$$

We define the following set of states of $\mathcal{M}$: $U = \{(\ell, \nu) \mid \forall x \in X,\ \nu(x^-) \leq \nu(x^+)\}$. Obviously, $\chi(\ell, \nu) \neq \varnothing$ iff $(\ell, \nu) \in U$.

We can easily prove:

Lemma 10. *Let $(\ell, \nu) \in U$, $t \in \mathbb{R}_+$ and $e \in E$. Then:*

- $\mathsf{timePost}^t_{\mathcal{H}}\Big(\chi(\ell, \nu)\Big) = \chi\Big(\mathsf{timePost}^t_{\mathcal{M}}(\ell, \nu)\Big)$;
- $\mathsf{discPost}^e_{\mathcal{H}}\Big(\chi(\ell, \nu)\Big) = \chi\Big(\mathsf{discPost}^{\{(e,\sigma)\in E' \mid \sigma \in 4^X\}}_{\mathcal{M}}(\ell, \nu)\Big)$.

where $\mathsf{timePost}^t$ *is the time-successor after a delay of t time units, and* $\mathsf{discPost}^e$ *is the discrete successor after edge e.*

As a consequence, the set of reachable states in $\mathcal{H}$ is the image by χ of the set of reachable states in $\mathcal{M}$.

Remark 13. Let us summarize the characteristics of multi-rate automaton $\mathcal{M}$:

- for each variable y, for each location ℓ, $\mathsf{Act}'(\ell)(y)$ is a singleton;
- for every edge e, if $y \in \mathsf{Upd}'(e)$, then $\mathsf{Post}'(e)(y)$ is a singleton;
- for every edge e, if $y \notin \mathsf{Upd}'(e)$, then $\mathsf{Pre}'(e)(y) = \mathsf{Post}'(e)(y)$;
- if $\mathcal{H}$ is initialized, then so is $\mathcal{M}$.

We now transform the (initialized) multi-rate automaton $\mathcal{M}$ into a(n initialized) stopwatch automaton[21] $\mathcal{S}$. The idea is to scale the variables by a correct factor. We assume the set of variable of $\mathcal{M}$ is $X' = \{y_1, \ldots, y_p\}$. For every location ℓ of $\mathcal{M}$, we define m_i^ℓ by $\mathsf{Act}'(\ell)(y_i)$ if $\neq 0$, and 1 otherwise. We define $\zeta_\ell : \mathbb{R}^p \to \mathbb{R}^p$ such that $\zeta_\ell(y_1, \ldots, y_p) = (y_1/m_1^\ell, \ldots, y_p/m_p^\ell)$. The stopwatch automaton is obtained by applying ζ_ℓ to $\mathsf{Act}'(\ell)$ (therefore defining Act'') and $\mathsf{Pre}'(e')$ when $\ell = \mathsf{src}(e')$ (therefore defining Pre''), and $\mathsf{Post}'(e')$ when $\ell = \mathsf{target}(e')$ (therefore defining Post''). Updates are unchanged: $\mathsf{Upd}'' = \mathsf{Upd}'$. We then define $\zeta(\ell, \nu) = (\ell, \zeta_\ell(\nu))$ when (ℓ, ν) is a state of $\mathcal{M}$.

Lemma 11. *The relation $\equiv$ defined between $\mathcal{M}$ and $\mathcal{S}$ by $(\ell, \nu) \equiv \zeta(\ell, \nu)$ is a strong-timed bisimulation. In particular, if (ℓ, ν) is a configuration of $\mathcal{M}$, $t \in \mathbb{R}_+$ and $e \in E'$, then:*

- $\mathsf{timePost}_{\mathcal{S}}^t\Big(\zeta(\ell, \nu)\Big) = \zeta\Big(\mathsf{timePost}_{\mathcal{M}}^t(\ell, \nu)\Big)$;
- $\mathsf{discPost}_{\mathcal{S}}^e\Big(\zeta(\ell, \nu)\Big) = \zeta\Big(\mathsf{discPost}_{\mathcal{M}}^e(\ell, \nu)\Big)$.

Remark 14. Let us summarize the characteristics of the stopwatch automaton $\mathcal{S}$:

- for each variable y, for each location ℓ, $\mathsf{Act}''(\ell)(y) \in \{0, 1\}$;
- for every edge e, if $y \in \mathsf{Upd}''(e)$, then $\mathsf{Post}''(e)(y)$ is a singleton;
- for every edge e, if $y \notin \mathsf{Upd}''(e)$, then $\mathsf{Pre}''(e)(y) = \mathsf{Post}''(e)(y)$;
- if $\mathcal{M}$ is initialized, then so is $\mathcal{S}$.

We finally transform $\mathcal{S}$ into a timed automaton $\mathcal{A}$, under the assumption that $\mathcal{S}$ is initialized. We let $K = \{\text{finite endpoints of intervals in } \mathsf{Post}''\}$. Assume L'' is the set of locations of $\mathcal{S}$. The set of locations of $\mathcal{A}$ is then:

$$L''' = \{(\ell, \alpha) \in L'' \times (K \cup \{\top\})^{X''} \mid \forall y \in X'',\ \mathsf{Act}''(\ell)(y) \neq 0 \text{ iff } \alpha(y) = \top\}$$

[$\top$ represents active clocks, whereas value k represents stopped clocks at value k]

If $e = \ell \xrightarrow{\mathsf{Pre},\mathsf{Post},\mathsf{Upd}} \ell'$ is an edge of $\mathcal{S}$, then, for every α such that $(\ell, \alpha) \in L'''$, we will have a transition $(\ell, \alpha) \xrightarrow{g, up} (\ell', \alpha')$ where:

- g encodes Pre (which is simplified by assuming that the value of each y with $\alpha(y) \in K$ is $\alpha(y)$);
- for each $y \in \mathsf{Upd}(e)$ such that $\mathsf{Act}(\mathsf{target}(e))(y) = 1$, up_y is an update $y := cte$, if $\mathsf{Post}(e)(y) = [cte, cte]$; also, $\alpha'(y) = \top$;
- for each $y \in \mathsf{Upd}(e)$ such that $\mathsf{Act}(\mathsf{target}(e))(y) = 0$, $\alpha'(y) = cte$, if $\mathsf{Post}(e)(y) = [cte, cte]$; also, $\alpha'(y) = \top$;
- for each $y \notin \mathsf{Upd}(e)$ such that $\mathsf{Act}(\mathsf{target}(e))(y) = 0$, as the system is initialized, it holds that $\alpha(y) \in K$, and we therefore set $\alpha'(y) = \alpha(y)$.

The resulting system is a diagonal-free timed automaton, which uses updates to constants (instead of just resets to 0). However (see exercise below), those are harmless in diagonal-free timed automata.

Reachability is decidable in PSPACE in timed automata and in space linear in the number of states of the automaton. We thus get a PSPACE algorithm. ◻

[21] That is, in each location, each variable has either slope 0 or 1.

Exercise 20. Show that we can remove updates of the form $x := c$ in timed automata. However notice that this yields an exponential blowup in the size of the system. Therefore, show that standard region automata construction can be used to prove decidability of this class of systems (adapt the compatibility of the regions with resets).

9.3 Weighted timed automata

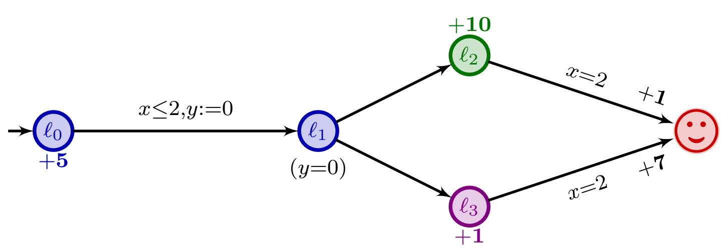
Weighted timed automaton. A *weighted timed automaton* is a timed automaton with an observer weight variable. It is defined as a tuple $\mathcal{A} = (L, \ell_0, L_F, X, \Sigma, T, \mathsf{weight})$, where:

- $(L, \ell_0, L_F, X, \Sigma, T)$ is a standard (diagonal-free) timed automaton, and
- $\mathsf{weight} : L \cup T \to \mathbb{Z}$ assigns a value to each location and to each transition.

We assume $\Sigma = T$, that is each transition can be identified with its label.

The weight of a delay move $(\ell, v) \xrightarrow{d} (\ell, v + d)$ is given by $d \cdot \mathsf{weight}(\ell)$: $\mathsf{weight}(\ell)$ is the rate in ℓ. The weight of a discrete move $(\ell, v) \xrightarrow{e} (\ell', v')$ is given by $\mathsf{weight}(e)$. The weight of a finite run ϱ is the accumulated weight of all moves that compose the run ϱ, we write $\mathsf{cost}(\varrho)$ for that value, and we call it the cost of ϱ.

Example 16. We consider the following weighted timed automaton:

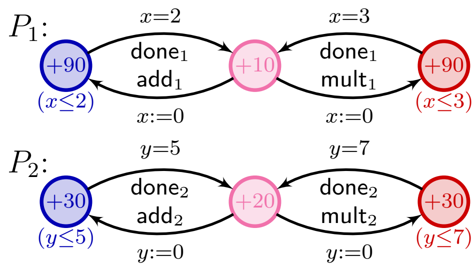


A possible execution for that system is:

	ℓ_0	$\xrightarrow{1.3}$	ℓ_0	$\xrightarrow{c}$	ℓ_1	$\xrightarrow{u}$	ℓ_3	$\xrightarrow{0.7}$	ℓ_3	$\xrightarrow{c}$	☺		
x	0		1.3		1.3		1.3		2				
y	0		1.3		0		0		0.7				
cost :		6.5	+	0	+	0	+	0.7	+	7			$= 14.2$

Remark 15. Note that weighted timed automata can be seen as linear hybrid automata, or as (non-initialized) rectangular automata. However the weight variable is never constrained, that is, the behaviour of the weighted timed automaton is that of the underlying timed automaton.

Example 17. Back to the taskgraph scheduling problem, we can refine the models for the processors and get:



Optimal reachability in weighted timed automata We focus on the optimal reachability question: given a weighted timed automaton $\mathcal{A}$, can we compute the optimal cost that allows to reach the set of target states. The optimal cost is defined as

$$\mathsf{optcost}_{\mathcal{A}} = \inf\{\mathsf{cost}(\varrho) \mid \varrho \text{ run from } (\ell_0, \mathbf{0}) \text{ to } L_F\}$$

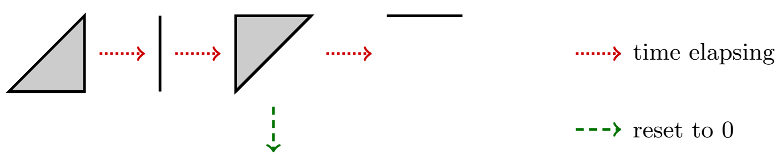
Example 18. If we consider the weighted timed automaton of Example 16, this optimal cost can be computed as:

$$\inf_{0 \leq t \leq 2} \Big(5t + \min(10(2-t)+1, (2-t)+7)\Big) = 9$$

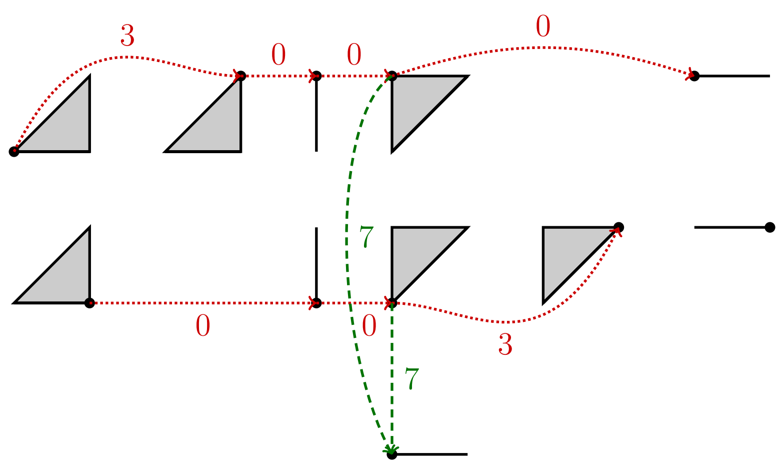
and the "strategy" is to take the first transition when $x = 0$.

Theorem 10. *We can compute the optimal cost for reaching the target location in weighted timed automata in polynomial space.*

First notice that the region abstraction is too rough to correctly take into account the cost information:



Therefore the proof of this theorem relies on a refinement of the region automaton construction, called the *corner-point abstraction*, which is a finite weighted graph:



We fix a weighted timed automaton $\mathcal{A}$, and w.l.o.g. we assume all clocks are bounded by maximal constant M. We fix its set of diagonal-free regions R. A corner-point is a pair (r, α) where $r \in R$, and $\alpha \in \overline{r}$ with integral coordinates. Note that if r is given by the following order on fractional part $X_0 < X_1 < \cdots < X_p$ where (X_i) is a partition of the set of clocks, X_0 contains all clocks with integral values, for every i, the fractional parts of all clocks in X_i are equal, and all clocks in X_i have fractional part strictly larger than clocks in X_{i-1}, then there are $p+1$ corner-points to r, each one is characterized by $0 \leq i \leq p$ such that the value of clocks in $X_0 \cup X_1 \cup \cdots \cup X_i$ is the integral part whereas the value of clocks in $X_{i+1} \cup \cdots \cup X_p$ is the integral part plus one.

We write R_{cp} for the set $\{(r, \alpha) \mid r \in R \text{ and } \alpha \text{ is a corner-point of } r\}$. We write r_0 for the region where all clocks have value 0, and α_0 for its unique corner-point.

Example 19. Region $0 < x < 1 \wedge 1 < y < 2 \wedge y - x < 1$ has three corner-points:

$$(x = 0, y = 1) \qquad (x = 1, y = 1) \qquad (x = 0, y = 2)$$

The *corner-point abstraction* $\mathcal{R}_{cp}(\mathcal{A})$ of $\mathcal{A}$ is a finite weighted graph $(Q, q_0, Q_F, \rightarrow)$ where $Q = L \times R_{cp}$, $q_0 = (\ell_0, r_0, \alpha_0)$, $Q_F = L \times R_{cp}$, and $\rightarrow \subseteq Q \times \mathbb{Z} \times Q$ is defined as follows:

- there is a transition $(\ell, r, \alpha) \xrightarrow{\mathsf{weight}(\ell)} (\ell, r, \alpha')$ if α and α' are corner-points of r (this is only possible if $\alpha' = \alpha + 1$)
- there is a transition $(\ell, r, \alpha) \xrightarrow{0} (\ell, r', \alpha)$ if r' is the immediate successor of r, and α is a corner-point of both r and r'
- there is a transition $(\ell, r, \alpha) \xrightarrow{\mathsf{weight}(e)} (\ell', r', \alpha')$ if edge $e = (\ell, g, Y, \ell')$ is such that $r \subseteq [\![g]\!]$, $r' = [Y \leftarrow 0]r$ and $\alpha' = [Y \leftarrow 0]\alpha$.[22]

We will show that this abstraction is sound and complete w.r.t. optimal reachability, that is $\mathsf{optcost}_{\mathcal{A}} = \mathsf{optcost}_{\mathcal{R}_{cp}(\mathcal{A})}$. More precisely we will prove the following proposition:

Proposition 9. *For every accepted finite run ϱ in $\mathcal{A}$, there is an accepted finite path π in $\mathcal{R}_{cp}(\mathcal{A})$ such that $\mathsf{cost}(\pi) \leq \mathsf{cost}(\varrho)$.*

For every accepted finite path π in $\mathcal{R}_{cp}(\mathcal{A})$, for every $\varepsilon > 0$, there is an accepted finite run ϱ in $\mathcal{A}$ such that $\mathsf{cost}(\varrho) \leq \mathsf{cost}(\pi) + \varepsilon$.

Proof (of Proposition 9). We first prove the correctness, and then the soundness.

Correctness. Let $\varrho = (\ell_0, u_0) \rightarrow (\ell_0, u_0 + d_0) \rightarrow (\ell_1, u_1) \rightarrow (\ell_1, u_1 + d_1) \cdots \rightarrow (\ell_n, u_n)$ be a finite run in $\mathcal{A}$ (with alternating delay and discrete transitions). We moreover assume that this execution is read on the sequence of transitions $\ell_0 \xrightarrow{g_1, Y_1} \ell_1 \ldots \xrightarrow{g_n, Y_n} \ell_n$ in $\mathcal{A}$. The cost of ϱ is given by:

$$f(d_0, d_1, ..., d_{n-1}) = \sum_{i=0}^{n-1} c_i \cdot d_i + c$$

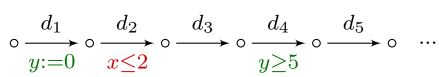
[22] Note that $[Y \leftarrow 0]\alpha$ is a corner-point of $[Y \leftarrow 0]r$ if α is a corner-point of r.

where c_i's are the weights of the locations ℓ_i's, and c is the sum of all the discrete weights of transitions along ϱ.

We want to minimize this function f with the constraints that $\widetilde{\varrho} = (\ell_0, v_0) \to (\ell_0, v_0 + \widetilde{d}_0) \to (\ell_1, v_1) \to (\ell_1, v_1 + \widetilde{d}_1) \cdots \to (\ell_n, v_n)$ is a run which is region-equivalent to ϱ, that is, if $\widetilde{u}_i(x) = \sum_{h=j}^{i-1} \widetilde{d}_h$ and $\widetilde{u}'_i(x) = \sum_{h=j}^{i} \widetilde{d}_h$, where $j = \max\{k \leq i \mid x \in Y_k\}$, then $\widetilde{u}_i \in r_i$, $\widetilde{u}'_i \in r'_i$ where r_i (resp. r'_i) is the region of u_i (resp. $u_i + d_i$).

The (topological) closure of this set of constraints on variables $D = (\widetilde{d}_i)_{0 \leq i \leq n-1}$ can be represented by a linear constraint $M \cdot D \leq A$, where each line of M is of the form $0 \dots 0\ 1 \dots 1\ 0 \dots 0$ or $0 \dots 0\ -1 \cdots -1\ 0 \dots 0$. One can show that such a matrix is totally unimodular.[23] This implies that minimizing f along the region path of ϱ yields an integer solution $(\alpha_i)_{0 \leq i \leq n-1}$.

Example 20 (Optimal reachability as a linear programming problem). We illustrate the above construction:



$$\begin{cases} d_1 + d_2 \leq 2 \\ d_2 + d_3 + d4 \geq 5 \end{cases}$$

We define the valuations $(\sigma_i)_{0 \leq i \leq n}$ by $\sigma_i(x) = \sum_{h=j}^{-i-1} \alpha_h$ where $j = \max\{k \leq i \mid x \in Y_k\}$. Each valuation σ_i is in $\overline{r_i}$ and has integer coordinates. It is thus a corner-point of r_i. Moreover, the sequence of valuations $(\sigma_i)_i$ would be an accepted sequence if we replace the constraints r_i by $\overline{r_i}$. In addition, the time elapsed in each state ℓ_i would then be α_i. It is technical but easy to build a corresponding path π in the corner-point abstraction $\mathcal{R}_{cp}(\mathcal{A})$.

As $(\alpha_i)_{i=1\dots n}$ minimizes f over the closure of the constraint defined by the region path of ϱ, we get that $\mathsf{cost}(\pi) \leq \mathsf{cost}(\varrho)$ and we are done.

Completeness. Note that if all guards are closed, then this is straightforward. In the general case, it relies on the following lemma:

Lemma 12. *Let $\pi = (\ell_0, r_0, \alpha_0) \to (\ell_1, r_1, \alpha_1) \to \dots$ be a path in $\mathcal{A}_{cp}$. For every $\varepsilon > 0$, there exists a real run $\varrho_\varepsilon = (\ell_0, v_0) \to (\ell_1, v_1) \to \dots$ such that for every i, $v_i \in r_i$ and $\|\alpha_i - v_i\|_\infty < \varepsilon$.*

To prove this lemma we show the following: for every $(\ell, r, \alpha) \to (\ell', r', \alpha')$, for every $\varepsilon > 0$, for every $v \in r$ such that $\delta_\alpha(v) < \varepsilon$, there exists $(\ell, v) \to (\ell', v')$ in $\mathcal{A}$ such that $v' \in r'$ and $\delta_{\alpha'}(v') < \varepsilon$, where the *diameter of v w.r.t. α* is defined as $\delta_\alpha(v) = \max\Big(\max_x(|v(x) - \alpha_x|), \max_{x,y}(|(v(x) - \alpha_x) - (v(y) - \alpha_y)|)\Big)$.

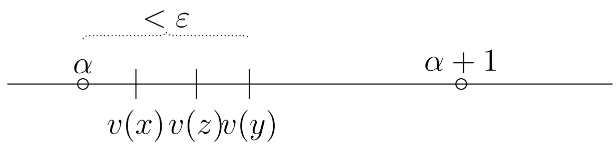
To prove this we distinguish between all cases for the transition $(\ell, r, \alpha) \to (\ell', r', \alpha')$.

- Assume $(\ell, r, \alpha) \to (\ell', r', \alpha')$ is a discrete transition and $\delta_\alpha(v) < \varepsilon$ with $v \in r$. Then let v' be such that $(\ell, v) \to (\ell', v')$. Let Y be the set of clocks which are reset along that move. Then $v'(y) = 0$ and $\alpha'_y = 0$ for every $y \in Y$, and $v'(y) = v(y)$ and $\alpha'_y = \alpha_y$ for every $y \notin Y$. Then obviously $\delta_{\alpha'}(v') \leq \delta_\alpha(v) < \varepsilon$.

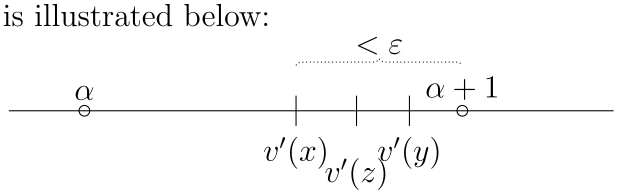
[23] Which means that every square matrix extracted from M has determinant -1, 0 or 1.

– Assume $(\ell, r, \alpha) \to (\ell', r', \alpha')$ is a delay transition. There are several cases:

- $(\ell, r, \alpha) \to (\ell, r, \alpha + 1)$

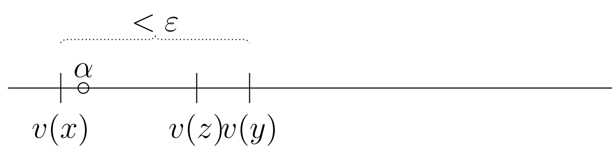


  In that case, write $\tau = \min\{v(x) - \alpha_x \mid x \in X\}$ and $\tau' = \max\{v(x) - \alpha_x \mid x \in X\}$, and define $v' = v + 1 - \tau' - \tau$. With this value we easily get $\delta_{\alpha+1}(v') = \delta_\alpha(v) < \varepsilon$. The new situation is illustrated below:

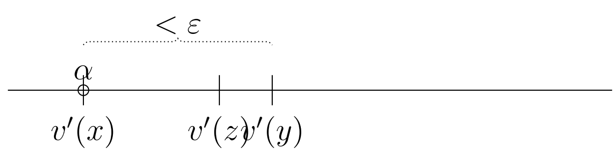


- $(\ell, r, \alpha) \to (\ell, r', \alpha)$ where r' is the immediate successor of r. There are several cases, which depend on the position of v relative to α.

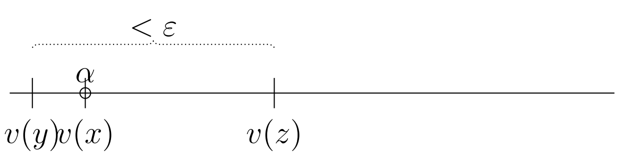
  * There are some clocks x, such that $v(x) < \alpha_x$, and there is no clock y such that $v(y) = \alpha_y$. This case is illustrated below.



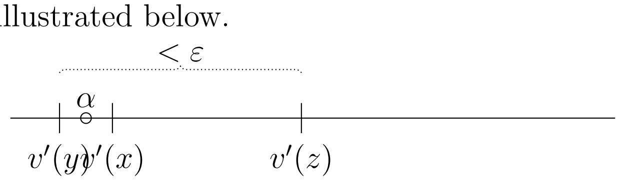
    In that case, we define $\tau = \min\{\alpha_x - v(x) \mid v(x) < \alpha_x\}$, and then $v' = v + \tau$. We get that $v' \in r'$, and that $\delta_\alpha(v') = \delta_\alpha(v) < \varepsilon$. The new situation is illustrated below.



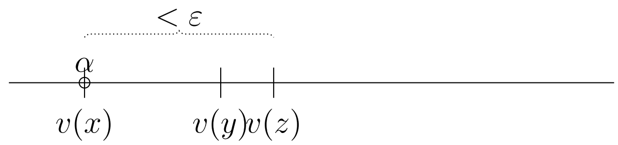
  * There is some clock x such that $v(x) = \alpha_x$, and there is some clock y such that $v(y) < \alpha_y$.



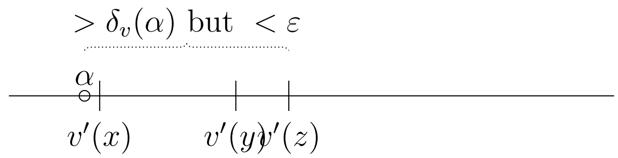
    We let $\tau = \min\{\alpha_y - v(y) \mid v(y) < \alpha_y\}$, and we define $v' = v + \frac{\tau}{2}$. We get the expected result, illustrated below.



  * This is the "most complex" case. There is some clock x such that $v(x) = \alpha_x$, and there is no clock y such that $v(y) < \alpha_y$.



In that case, we let $\tau = \varepsilon - \delta_\alpha(v)$ (which we know is positive), and we let $v' = v + \frac{\tau}{2}$. We then get the expected result.



This concludes the proof. $\square$

Now, along a given sequence of transitions, the cost is continuous, which means that for every $\nu > 0$, we can find some $\varepsilon > 0$ such that the distance between two runs is smaller than ε implies that the difference of costs for the two runs is smaller than ν. This concludes the completeness of the construction.

THE END

10 Implementation and data structures

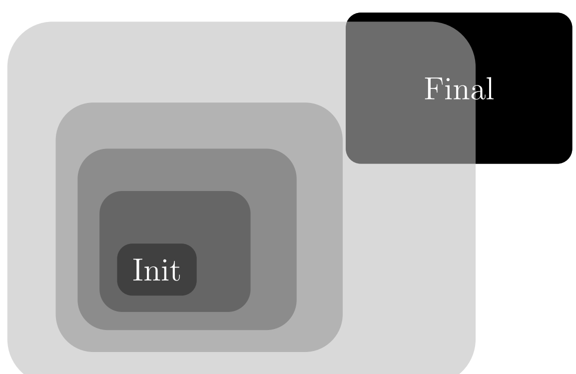
In the original work by Alur and Dill, the so-called region automaton construction is an abstraction which proves the decidability of the model. However, whereas well-suited for establishing decidability of problems related to timed automata, the region automaton is highly impractical from a tool implementation point-of-view. Instead, most real-time verification tools (like CMC[24] [LL98], Kronos[25] [DOTY96], and Uppaal[26] [BDL+06]) apply abstractions based on so-called zones, which in practice provide much coarser abstractions.

In this section, we describe methods that can be used for analysing reachability (or equivalently simple safety) properties in timed automata.

10.1 Checking reachability properties: two general methods

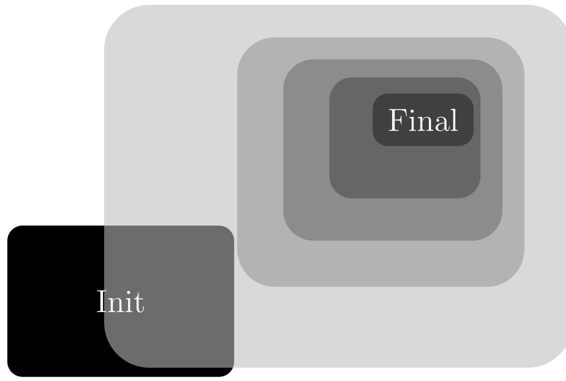
There are two main approaches for checking reachability (or safety) properties in systems (not only timed systems, but all kinds of systems). We describe these two approaches shortly and apply them to timed automata.

- *Forward analysis.* The general idea is to compute configurations which are reachable from the initial configuration within 1 step, 2 steps, *etc.* until final (or goal) configurations are computed, or until the computation terminates. The forward analysis computation can be schematized as below.



- *Backward analysis.* The general idea is to compute configurations from which we can reach final configurations within 1 step, 2 steps, *etc.* until the initial configuration is computed, or until the computation terminates. The backward analysis computation can be represented as below.

24 `http://www.lsv.ens-cachan.fr/~fl/cmcweb.html/`
25 `http://www-verimag.imag.fr/TEMPORISE/kronos/`
26 `http://www.uppaal.com/`



These two generic approaches are used in many contexts, including the analysis of models like counter machines, hybrid systems, *etc.* Of course, given a class of systems, specific techniques (*eg.* abstractions, widening operations, *etc.*) can be used for improving the computations. We will now focus on timed automata and explain how these two approaches can be implemented in that framework.

10.2 Reachability analysis in timed automata: the zone symbolic representation

Timed automata have infinitely (and even uncountably) many configurations, it is thus necessary to use symbolic representations for doing the computations. For the discussion which follows we fix a timed automaton $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T)$. Given an edge $e = (\ell, g, a, Y, \ell') \in T$ of $\mathcal{A}$, we need to be able to compute its effect on a set of valuations. More precisely, if W is a set of valuations, we define the two following sets of valuations:

$$\begin{cases} \mathsf{Post}_e(W) = \{v' \in \mathbb{T}^X \mid \exists v \in W\ \exists t \in \mathbb{T} \text{ such that } v + t \models g \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{and } v' = [Y \leftarrow 0](v+t)\} \\ \mathsf{Pre}_e(W)\ \ = \{v \in \mathbb{T}^X \mid \exists v' \in W\ \exists t \in \mathbb{T} \text{ such that } v + t \models g \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{and } [Y \leftarrow 0](v+t) = v'\} \end{cases}$$

A valuation v' is in $\mathsf{Post}_e(W)$ whenever there exists some valuation $v \in W$ and some $t \in \mathbb{R}_{\geq 0}$ such that $(\ell, v) \xrightarrow{t,e} (\ell', v')$ is a mixed move in $\mathcal{T}_\mathcal{A}$. Similarly a valuation v is in $\mathsf{Pre}_e(W)$ whenever there exists some valuation $v' \in W$ and some $t \in \mathbb{R}_{\geq 0}$ such that $(\ell, v) \xrightarrow{t,e} (\ell', v')$ is a mixed move in $\mathcal{T}_\mathcal{A}$.

It is worth noticing that if W is a *zone*, *i.e.*, a set of valuations defined by a general clock constraint, then for every transition e of $\mathcal{A}$, $\mathsf{Post}_e(W)$ and $\mathsf{Pre}_e(W)$ are both zones. For analysing timed automata, zones are the most basic and commonly used *symbolic representation*.

In the following, we will need to decompose the computation of Post_e and Pre_e in several simpler steps, hence we define the following operations on zones (or more generally on sets of valuations):

- Future of W: $\overrightarrow{W} = \{v + t \mid v \in W \text{ and } t \in \mathbb{R}_{\geq 0}\}$

- Past of W: $\overleftarrow{W} = \{v - t \mid v \in W \text{ and } t \in \mathbb{R}_{\geq 0}\}$
- Intersection of W and W': $W \cap W' = \{v \mid v \in W \text{ and } v \in W'\}$
- Reset to zero of W with respect to the set of clocks Y:
$$[Y \leftarrow 0]W = \{[Y \leftarrow 0]v \mid v \in W\}$$
- Inverse reset to zero of W with respect to the set of clocks Y:
$$[Y \leftarrow 0]^{-1}W = \{v \mid [Y \leftarrow 0]v \in W\}$$

Exercise 21. Prove that those elementary operations preserve zones. ⌟

These operations allow to express the Post_e and Pre_e operators:

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

10.3 The DBM data structure

The most common data structure for representing zones is the so-called DBM data structure. This data structure has been first introduced in [BM83] and then set in the framework of timed automata in [Dil90]. Several presentations of this data structure can be found in the literature, for example in [CGP99,Ben02,Bou04].

A *difference bound matrix*, we shall write DBM for short, for a set $X = \{x_1, \ldots, x_n\}$ of n clocks is an $(n+1)$-square matrix of pairs

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

A DBM $M = (\prec_{i,j}, m_{i,j})_{0 \leq i,j \leq n}$ defines the following subset of $\mathbb{R}^X_{\geq 0}$ (if $v \in \mathbb{R}^X_{\geq 0}$, $\overline{v}$ is the 'canonical' valuation over $X \cup \{x_0\}$ — where x_0 is a fresh clock — such that $\overline{v}(x) = v(x)$ for every $x \in X$, and $\overline{v}(x_0) = 0$; In the following we may write v instead of $\overline{v}$):

$$\{v : X \to \mathbb{R}_{\geq 0} \mid \forall\ 0 \leq i,j \leq n,\ \overline{v}(x_i) - \overline{v}(x_j) \prec_{i,j} m_{i,j}\}$$

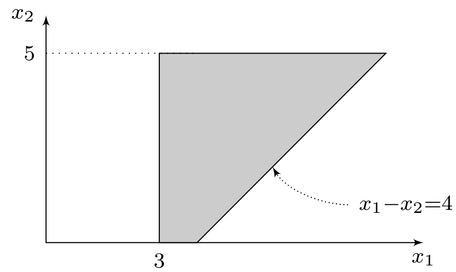
where $\gamma < \infty$ simply means that $\gamma \in \mathbb{R}_{\geq 0}$ (without any constraint on γ). This subset of $\mathbb{R}^X_{\geq 0}$ is a zone and will be denoted by $[\![M]\!]$. To simplify the notations, we now assume that all constraints are non-strict, so that coefficients of DBMs will simply be elements of $\mathbb{Z} \cup \{\infty\}$.

Example 21. We consider the zone over the set of clocks $X = \{x_1, x_2\}$ defined by the general clock constraint

$$(x_1 \geq 3)\ \wedge\ (x_2 \leq 5)\ \wedge\ (x_1 - x_2 \leq 4).$$

This zone, depicted on the next picture on the right, can be represented by the DBM on the left.

$$
\begin{array}{c|ccc}
 & x_0 & x_1 & x_2 \\
\hline
x_0 & \infty & -3 & \infty \\
x_1 & \infty & \infty & 4 \\
x_2 & 5 & \infty & \infty
\end{array}
$$



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

$$
\begin{array}{c|ccc}
 & x_0 & x_1 & x_2 \\
\hline
x_0 & 0 & -3 & 0 \\
x_1 & 9 & 0 & 4 \\
x_2 & 5 & 2 & 0
\end{array}
$$

Normal forms of DBMs. We define a total order on $\mathbb{V}$ in the following way: if $(\prec, m), (\prec', m') \in \mathbb{V}$, then

$$(\prec, m) \leq (\prec', m') \Leftrightarrow \begin{cases} m < m' \\ \text{or} \\ m = m' \text{ and either } \prec \, = \, \prec' \text{ or } \prec' \, = \, \leq . \end{cases}$$

Of course, for each $m \in \mathbb{Z}$, we have $m < \infty$. We define $>$, $\geq$ and $<$ in a natural way. These orders are extended to DBMs $M = (\prec_{i,j}, m_{i,j})_{i,j=0\ldots n}$ and $M' = (\prec'_{i,j}, m'_{i,j})_{i,j=0\ldots n}$ by

$$M \leq M' \Leftrightarrow \text{for every } i, j = 0 \ldots n, \ (\prec_{i,j}, m_{i,j}) \leq (\prec'_{i,j}, m'_{i,j}).$$

We also define an addition on the set $\mathbb{V}$ as follows:

$$(\prec, m) + (\prec', m') = (\prec'', m'')$$

where $m'' = m + m'$ and $\prec''$ is $\leq$ if both $\prec$ and $\prec'$ are $\leq$ and $\prec''$ is $<$ otherwise.

Let M be a DBM. We write G_M for the graph corresponding to that DBM. Then the following property follows:

Property 1 (Emptiness checking).

$$[\![M]\!] = \varnothing \text{ iff there is a cycle with weight strictly smaller than } (\leq, 0) \text{ in } G_M$$

Proof. The right-to-left implication is obvious.

We focus on the left-to-right implication. The proof relies on shortest paths properties [CLR90].

We first assume that all $\prec_{i,j}$ are $\leq$. In the graph G_M, there is a path from vertex x_0 to any other vertex. We can therefore define $\delta(i)$ the shortest path in G_M from 0 to i (forgetting the comparison operator, since it is $\leq$). We have that for every j, $\delta(i) \leq \delta(j) + m_{j,i}$, that is $(-\delta(j)) - (-\delta(i)) \leq m_{j,i}$. Hence, $(-\delta(i))_i$ is a solution to the system of equations defined by M.

In the general case, we fix some $\epsilon_{i,j} \geq 0$ such that the following conditions hold:

- $\prec_{i,j} = <$ iff $\epsilon_{i,j} > 0$
- for every cycle $i_1 i_2 \dots i_k$ of length at most $n+1$,

$$\sum_{j=1}^{k-1} \left(m_{i_j,i_{j+1}} - \epsilon_{i_j,i_{j+1}} \right) \geq 0$$

This is possible to do so since there is no negative cycle in M. We define $M_\epsilon = ((m_{i,j} - \epsilon_{i,j}); \leq)_{i,j}$. Applying the previous result, we get that $[\![M_\epsilon]\!] \neq \varnothing$. Furthermore, $M_\epsilon \leq M$, which implies $[\![M]\!] \neq \varnothing$. ⌟

We apply the Floyd-Warshall algorithm to DBM M (seen as the adjacency matrix of G_M), yielding the DBM $\phi(M)$. The DBM $\phi(M)$ is the *normal form* of M, which will be justified later. Notice that $\phi(\phi(M)) = \phi(M)$ if $[\![M]\!]$ is not empty.

We can now state some (very useful) properties of normal forms of DBMs.

Property 2. If M and M' are DBMs, then:

(i) $[\![M]\!] = [\![\phi(M)]\!]$ and $\phi(M) \leq M$,
(ii) $[\![M]\!] \subseteq [\![M']\!] \Leftrightarrow \phi(M) \leq M' \Leftrightarrow \phi(M) \leq \phi(M')$.

The last point expresses the fact that the test for inclusion of zones can be checked syntactically on the normal forms of the DBMs (representing the zones).

Normal forms of DBMs can be characterized in a natural way.

Property 3. If $M = (\prec_{i,j}, m_{i,j})_{i,j=0\dots n}$ is a DBM such that $[\![M]\!] \neq \varnothing$, then the two following properties are equivalent:

(i) M is in normal form,
(ii) for every $i, j = 0 \dots n$, for every real $-m_{j,i} \prec_{j,i} r \prec_{i,j} m_{i,j}$, there exists a valuation $v \in [\![M]\!]$ such that $v(x_j) - v(x_i) = r$ (still assuming that $v(x_0) = 0$).

This property expresses the fact that if a DBM is in normal form, then no constraint of this DBM can be tightened using Floyd algorithm.

Computation of Some Operations on DBMs. As we argued at the beginning of the section, the data structure used to represent zones must also be appropriate to compute several operations.

Intersection. Let $M = (\prec_{i,j}, m_{i,j};)_{i,j=1\dots n}$ and $M' = (\prec'_{i,j}, m'_{i,j})_{i,j=1\dots n}$ be two DBMs and define $M'' = (\prec''_{i,j}, m''_{i,j})_{i,j=1\dots n}$ by

$$(\prec''_{i,j}, m''_{i,j}) = \min((\prec_{i,j}, m_{i,j}), (\prec'_{i,j}, m'_{i,j})) \text{ for all indices } i, j = 1 \dots n.$$

Then $[\![M'']\!] = [\![M]\!] \cap [\![M']\!]$. Note that it can be the case that M'' is not in normal form, even if M and M' are in normal form.

Future. Assume that $M = (\prec_{i,j}, m_{i,j})_{i,j=1\dots n}$ is a DBM in normal form. Define the DBM $\overrightarrow{M} = (\prec'_{i,j}, m'_{i,j})_{i,j=1\dots n}$ by:

$$\begin{cases} (\prec'_{i,j}, m'_{i,j}) = (\prec_{i,j}, m_{i,j}) & \text{if } j \neq 0 \\ (\prec'_{i,0}, m'_{i,0}) = (<, \infty) \end{cases}.$$

Then $[\![\overrightarrow{M}]\!] = \overrightarrow{[\![M]\!]}$ and the DBM $\overrightarrow{M}$ is in normal form.

Past. Assume that $M = (\prec_{i,j}, m_{i,j})_{i,j=1\dots n}$ is a DBM in normal form. Define the DBM $\overleftarrow{M} = (\prec'_{i,j}, m'_{i,j})_{i,j=1\dots n}$ by:

$$\begin{cases} (\prec'_{i,j}, m'_{i,j}) = (\prec_{i,j}, m_{i,j}) & \text{if } i \neq 0 \\ (\prec'_{0,j}, m'_{0,j}) = (\leq, 0) \end{cases}.$$

Then $[\![\overleftarrow{M}]\!] = \overleftarrow{[\![M]\!]}$ and the DBM $\overleftarrow{M}$ is in normal form.

Image by resets. Assume that $M = (\prec_{i,j}, m_{i,j})_{i,j=1\dots n}$ is a DBM in normal form and define the DBM $M_{x_k:=0} = (\prec'_{i,j}, m'_{i,j})_{i,j=1\dots n}$ by:

$$\begin{cases} (\prec'_{i,j}, m'_{i,j}) = (\prec_{i,j}, m_{i,j}) & \text{if } i,j \neq k \\ (\prec'_{k,k}, m'_{k,k}) = (\prec'_{k,0}, m'_{k,0}) = (\prec'_{0,k}, m'_{0,k}) = (\leq, 0) \\ (\prec'_{i,k}, m'_{i,k}) = (\prec_{i,0}, m_{i,0}) & \text{if } i \neq k \\ (\prec'_{k,i}, m'_{k,i}) = (\prec_{0,i}, m_{0,i}) & \text{if } i \neq k \end{cases}$$

Then $[\![M_{x_k:=0}]\!] = [x_k \leftarrow 0][\![M]\!]$ and the DBM $M_{x_k:=0}$ is in normal form.

Let us just mention that the DBM data structure is the most basic data structure which is used for analysing timed systems, some more involved BDD-like data structures can also be used, for example CDDs (which stands for 'Clock Difference Diagrams') [LPWY99], or more recently federations [DHGP04,Dav05].

10.4 Backward analysis

We first focus on the backward analysis computation, which will surprisingly turn out to be the simplest to analyse. We fix a timed automaton $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T)$.

Backward symbolic transition system. The *backward symbolic transition system* associated with $\mathcal{A}$ is denoted by '$\Leftarrow$' and is defined inductively as follows:

$$\frac{e = \left(\ell_1 \xrightarrow{g,a,Y} \ell_2\right) \in E \qquad W_1 = \mathsf{Pre}_e(W_2)}{(\ell_2, W_2) \Leftarrow (\ell_1, W_1)}$$

Obviously, if we write $\Leftarrow^*$ for the reflexive and transitive closure of $\Leftarrow$, we have that $(\ell', W') \Leftarrow^* (\ell, W)$ if and only if for every $v \in W$, there exists $v' \in W'$ and a run in $\mathcal{A}$ from (ℓ, v) to (ℓ', v').

The backward analysis algorithm then consists in computing iteratively the following sets of symbolic configurations:

$$\begin{aligned}
\mathcal{S}_0^{\mathsf{b}} &= \qquad \{(\ell, \mathbb{R}_{\geq 0}^X) \mid \ell \in L_F\} \\
\mathcal{S}_1^{\mathsf{b}} &= \mathcal{S}_0^{\mathsf{b}} \ \cup\ \{(\ell, W) \mid \exists (\ell', W') \in \mathcal{S}_0^{\mathsf{b}} \text{ such that } (\ell', W') \Leftarrow (\ell, W)\} \\
&\ \vdots \\
\mathcal{S}_{p+1}^{\mathsf{b}} &= \mathcal{S}_p^{\mathsf{b}} \ \cup\ \{(\ell, W) \mid \exists (\ell', W') \in \mathcal{S}_p^{\mathsf{b}} \text{ such that } (\ell', W') \Leftarrow (\ell', W')\} \\
&\ \vdots
\end{aligned}$$

until either (i) the computation stabilizes, or (ii) a symbolic state is computed, which contains the initial configuration of $\mathcal{A}$. To help event (i) happen, it is possible to add the following inclusion check: if $(\ell, W) \in \mathcal{S}_{p+1}^{\mathsf{b}}$ and if there exists $(\ell, W') \in \mathcal{S}_p^{\mathsf{b}}$ such that $W \subseteq W'$ (or even if there exist $(\ell, W_i) \in \mathcal{S}_p^{\mathsf{b}}$ for finitely many i's such that $W \subseteq \bigcup_i W_i$), then do not include (ℓ, W) in $\mathcal{S}_{p+1}^{\mathsf{b}}$. The procedure answers 'Yes' in case (ii) and 'No' in case $(i) \wedge \neg(ii)$.

Termination and correctness. The backward analysis computation enjoys the following nice property, which can be seen as a consequence of the correctness of the backward analysis algorithm for TCTL [HNSY94]. However, we will give below a simple and direct proof of that result.

Theorem 11. *The backward computation terminates and is correct with respect to reachability properties.*[27]

Proof. Correctness is immediate as the computation is *exact* (as opposed to over-(or under-)approximate). The inclusion check does not cause any trouble as any state reachable from a configuration belonging to a symbolic state that is not added due to that test is actually already reachable from the previous symbolic state that was computed.

Termination needs some additional argument, that we sketch here. Assume that R_i's (for $1 \leq i \leq p$) are regions of $\mathcal{A}$, then:

- $\overleftarrow{\bigcup_{i=1}^{p} R_i}$ is a finite union of regions;
- $[Y \leftarrow 0]^{-1} \left(\bigcup_{i=1}^{p} R_i\right)$ is a finite union of regions (for any set of clocks Y);
- $[\![g]\!] \cap \left(\bigcup_{i=1}^{p} R_i\right)$ is a finite union of regions if g is a clock constraint of $\mathcal{A}$.

This is because of the three compatibility constraints which are satisfied by the set of regions.

[27] *I.e.* L_F is reachable if and only if it is declared as reachable by the backward computation.

These properties altogether imply that each of the symbolic configurations (ℓ, W) that are added to $\mathcal{S}_i^{\mathsf{b}}$ is such that W is a finite union of regions. As there are finitely many regions, the sequence $(\mathcal{S}_i^{\mathsf{b}})_{i\geq 0}$ stabilizes, hence the termination of the backward computation. $\lrcorner$

Backward analysis may appear as an accurate method for analysing timed automata, but in practice, some tools (like Uppaal) prefer using a forward analysis computation. One of the reasons comes from the use of (bounded) integer variables that are really helpful for modelling real systems. Backward analysis is then not suitable for dealing with 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.

10.5 Forward analysis

In this section we focus on the forward analysis computation, which will require the development of abstractions, and more effort for proving its correctness. We fix a timed automaton $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T)$.

Forward symbolic transition system. The *forward symbolic transition system* associated with $\mathcal{A}$ is denoted by '$\Rightarrow$' and is defined inductively as follows:

$$\frac{e = \left(\ell_1 \xrightarrow{g,a,Y:=0} \ell_2\right) \in E \qquad W_2 = \mathsf{Post}_e(W_1)}{(\ell_1, W_1) \Rightarrow (\ell_2, W_2)}$$

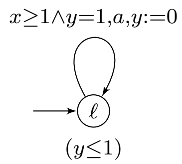
Obviously, if we write $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$, we have that $(\ell, W) \Rightarrow^* (\ell', W')$ if and only if for every $v' \in W'$, there exists $v \in W$ and a run in $\mathcal{A}$ from (ℓ, v) to (ℓ', v').

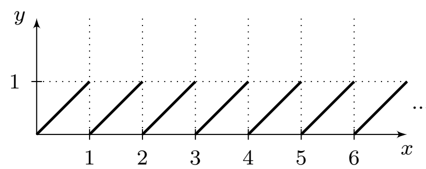
The forward analysis computation then consists in computing iteratively the following sets of symbolic configurations:

$$\begin{aligned}
\mathcal{S}_0^{\mathsf{f}} &= \qquad\quad \{(\ell_0, \mathbf{0}_X)\} \\
\mathcal{S}_1^{\mathsf{f}} &= \mathcal{S}_0^{\mathsf{f}} \;\cup\; \{(\ell', W') \mid \exists (\ell, W) \in \mathcal{S}_0^{\mathsf{f}} \text{ such that } (\ell, W) \Rightarrow (\ell', W')\} \\
&\;\vdots \\
\mathcal{S}_{p+1}^{\mathsf{f}} &= \mathcal{S}_p^{\mathsf{f}} \;\cup\; \{(\ell', W') \mid \exists (\ell, W) \in \mathcal{S}_p^{\mathsf{f}} \text{ such that } (\ell, W) \Rightarrow (\ell', W')\} \\
&\;\vdots
\end{aligned}$$

until either (i) the computation stabilizes, or (ii) a symbolic state is computed, which contains a final configuration of the timed automaton (*i.e.*, a configuration of the form (ℓ_f, v) with $\ell_f \in L_F$). To help event (i) happen, it is possible to add an inclusion check, as in the backward analysis computation. The procedure answers 'Yes' in case (ii) and 'No' in case $(i) \wedge \neg(ii)$.

Discussion on the termination and correctness. The forward analysis gives a correct answer, but it may not terminate. An example of automaton in which the forward computation does not terminate is given below. The zones that are computed by the above procedure are represented on the right part of the figure, and it is easy to check that the computation will never terminate.





To overcome this problem, it is necessary to use some abstraction operators. Several have been proposed in [DT98]: for instance, if Z and Z' are computed for the location ℓ, they are replaced by the smallest zone containing both Z and Z'; this approximation is called the *convex-hull* abstraction[28], it does not ensure termination and is only semi-correct, in the sense that a location announced as reachable might not be reachable (the convex-hull abstraction is an over-approximation). The most interesting abstraction studied in this paper is the *extrapolation* operator. We will present it now, but we first need to formalize a little more the forward analysis procedure. We follow the lines of [BBFL03,BBLP04] and define (abstract) symbolic transition systems.

Abstract forward symbolic transition systems. Let $\mathfrak{a}$ be an abstraction operator (possibly partially) defined on the sets of valuations ($\mathfrak{a}$ associates to sets of valuations sets of valuations). We define the *abstract forward symbolic transition system* '$\Rightarrow_{\mathfrak{a}}$' in the following way:

$$\frac{(\ell, W) \Rightarrow (\ell', W') \qquad W = \mathfrak{a}(W)}{(\ell, W) \Rightarrow_{\mathfrak{a}} (\ell', \mathfrak{a}(W'))}$$

This transition system gives naturally rise to the following forward computation in $\mathcal{A}$.

$$\begin{aligned}
\mathcal{S}_0^{\mathsf{f},\mathfrak{a}} &= \qquad\quad \{(\ell_0, \mathfrak{a}(\{\mathbf{0}_X\}))\} \\
\mathcal{S}_1^{\mathsf{f},\mathfrak{a}} &= \mathcal{S}_0^{\mathsf{f},\mathfrak{a}} \ \cup \ \{(\ell', W') \mid \exists (\ell, W) \in \mathcal{S}_0^{\mathsf{f},\mathfrak{a}} \text{ such that } (\ell, W) \Rightarrow_{\mathfrak{a}} (\ell', W')\} \\
&\ \,\vdots \\
\mathcal{S}_{p+1}^{\mathsf{f},\mathfrak{a}} &= \mathcal{S}_p^{\mathsf{f},\mathfrak{a}} \ \cup \ \{(\ell', W') \mid \exists (\ell, W) \in \mathcal{S}_p^{\mathsf{f},\mathfrak{a}} \text{ such that } (\ell, W) \Rightarrow_{\mathfrak{a}} (\ell', W')\} \\
&\ \,\vdots
\end{aligned}$$

with the same halting conditions (and inclusion checks) as previously.

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

Soundness criteria. The abstraction operator $\mathfrak{a}$ is said *correct* with respect to reachability properties in $\mathcal{A}$ whenever the following holds:

$$\text{if } (\ell_0, \mathfrak{a}(\{\mathbf{0}_X\})) \Rightarrow_{\mathfrak{a}}^* (\ell, W) \text{ then there exists a run} \\ (\ell_0, \mathbf{0}_X) \rightarrow^* (\ell, v) \text{ with } v \in W \text{ in } \mathcal{A}$$

The abstraction operator $\mathfrak{a}$ is said *complete* with respect to reachability properties whenever the following holds in $\mathcal{A}$:

$$\text{if } (\ell_0, \mathbf{0}_X) \rightarrow^* (\ell, v) \text{ is a run in } \mathcal{A} \text{ then} \\ (\ell_0, \mathfrak{a}(\{\mathbf{0}_X\})) \Rightarrow_{\mathfrak{a}}^* (\ell, W) \text{ for some } W \text{ with } v \in W$$

Remark 16. Note that these two notions could be generalized to more general properties than reachability properties, but we follow our lines and concentrate on reachability properties.

⌟

Our aim is to define abstraction operators $\mathfrak{a}$ such that the four following properties hold:

(Finiteness) $\{\mathfrak{a}(W) \mid \mathfrak{a} \text{ defined on } W\}$ is finite
(this ensures termination of the "abstract" forward computation)
(Correctness) $\mathfrak{a}$ is correct with respect to reachability
(Completeness) $\mathfrak{a}$ is complete with respect to reachability
(Effectiveness) $\mathfrak{a}$ is "effective"

The three first properties are properly defined, the last one is more informal. The effectiveness criterion expresses that the abstraction has to be easily computable. In timed automata literature this is most of the time interpreted as "$\mathfrak{a}$ has to be defined for all zones and $\mathfrak{a}(Z)$ has to be a zone when Z is a zone". Note that other effectiveness criteria could be proposed, but that is the one we choose here.

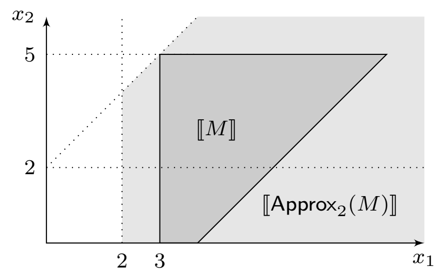
The extrapolation operator. The abstraction operator which is commonly used is called *extrapolation*, and sometimes *normalization* [Ben02] or *approximation* [Bou04]. We will note it here Approx_K, it is defined up to a constant K as follows: if Z is a zone, $\mathsf{Approx}_K(Z)$ is the smallest K-bounded zone[29] 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 obtained from M where each coefficient $(\prec; m)$ with $m < -K$ is replaced by $(<; -K)$ and all coefficients $(\prec; m)$ with $m > K$ is replaced by $(<; \infty)$, all other coefficients are unchanged. We write $\mathsf{Approx}_K(M)$ for this transformed DBM: it holds that $[\![\mathsf{Approx}_K(M)]\!] = \mathsf{Approx}_K([\![M]\!])$.

Example 22. Consider again the zone introduced in example 21. As we have already mentioned, it can be represented by the DBM in normal form on the left and its 2-extrapolation is the DBM on the right (where we again do not mention the comparison operators):

$$M = \begin{pmatrix} 0 & -3 & 0 \\ 9 & 0 & 4 \\ 5 & 2 & 0 \end{pmatrix} \quad \text{and} \quad \mathsf{Approx}_2(M) = \begin{pmatrix} 0 & -2 & 0 \\ +\infty & 0 & +\infty \\ +\infty & 2 & 0 \end{pmatrix}$$

[29] A *K-bounded zone* is a zone defined by a K-bounded clock constraint.

They are both represented on the picture below.



Obviously,

- Approx_K is a finite abstraction operator because there are finitely many DBMs whose coefficients are either $(<;\infty)$ or some $(\prec;m)$ with $\prec\,\in\{<;\leq\}$ and $-K \leq m \leq K$;
- the computation of Approx_K is effective and can easily be done using DBMs;
- Approx_K is a complete abstraction with respect to reachability because for every zone Z, $Z \subseteq \mathsf{Approx}_K(Z)$.

The only point that needs to be carefully studied is the correctness of Approx_K: of course, not all constants K yield correctness, but we have to choose such a constant K carefully, so that the abstraction operator be correct with respect to reachability properties. We now discuss in details this important aspect.

Correctness of the extrapolation operator. It is far from being obvious! We can state it as follows.

Theorem 12 ([Bou04]). *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 a correct abstraction with respect to reachability properties in $\mathcal{A}$.*

Proof. We prove the following property: for every region R (with maximal constant K), for every zone Z, $R \cap Z = \varnothing$ implies $R \cap \mathsf{Approx}_K(Z) = \varnothing$.

Let $R = ((I_{x_i})_{x_i \in X}; \prec)$ be a region and let $M_R = ((r_{i,j}; \prec_{i,j})_{i,j=0\ldots n})$ be a DBM representing R such that:

$$\begin{cases}
(r_{i,0}; \prec_{i,0}) = \begin{cases} (c+1; <) & \text{if } I_{x_i} =]c; c+1[\\ (c; \leq) & \text{if } I_{x_i} = \{c\} \\ (+\infty; <) & \text{if } I_{x_i} =]K; +\infty[\end{cases} \\
(r_{0,i}; \prec_{0,i}) = \begin{cases} (-c; <) & \text{if } I_{x_i} =]c; c+1[\\ (-c; \leq) & \text{if } I_{x_i} = \{c\} \\ (-K; <) & \text{if } I_{x_i} =]K; +\infty[\end{cases} \\
(r_{i,j}; \prec_{i,j}) = \begin{cases} (c-d+1; <) & \text{if } I_{x_i} =]c; c+1[,\ I_{x_j} =]d; d+1[\text{ and } x_j \prec x_i,\ x_i \not\prec x_j \\ (c-d; <) & \text{if } I_{x_i} =]c; c+1[,\ I_{x_j} =]d; d+1[\text{ and } x_i \prec x_j,\ x_j \not\prec x_i \\ (c-d; \leq) & \text{if } I_{x_i} =]c; c+1[,\ I_{x_j} =]d; d+1[\text{ and } x_i \prec x_j,\ x_j \prec x_i \\ (c-d; \leq) & \text{if } I_{x_i} = \{c\} \text{ and } I_{x_j} = \{d\} \\ (c+1-d; <) & \text{if } I_{x_i} =]c; c+1[,\ I_{x_j} = \{d\} \\ (c-d-1; <) & \text{if } I_{x_i} = \{c\} \text{ and } I_{x_j} =]d; d+1[\\ (+\infty; <) & \text{in all other cases} \end{cases}
\end{cases}$$

Note that the diagonal-free hypothesis has the following important consequence on the above representation of R: if $(r_{i,0}; \prec_{i,0}) = (+\infty; <)$, then for every $j \neq 0$, $(r_{i,j}; \prec_{i,j}) = (r_{j,i}; \prec_{j,i}) = (+\infty; <)$. *This property will be the core of the proof.*

Let Z be a zone and $M = ((m_{i,j}; <_{i,j})_{i,j})$ a DBM in normal form representing zone Z. Assume that $Z \cap R = \varnothing$. It means that there exists a sequence of distinct indices $(i_1, i_2, \ldots, i_l = i_1)$ such that

$$\alpha_{i_1,i_2} + \alpha_{i_2,i_3} + \cdots + \alpha_{i_{l-1},i_l} < (0; \leq) \tag{4}$$

where $\alpha_{j,k} = \min((m_{j,k}; <_{j,k}); (r_{j,k}; \prec_{j,k}))$. Since M is in normal form, we can assume that two successive $\alpha_{h,l}$ do not come from M, otherwise we could simplify the sum (4).

- Assume that $(m_{h,l}, <_{h,l}) < (r_{h,l}, \prec_{h,l})$ for some h and some l such that I_{x_h} and I_{x_l} are bounded intervals.
  We distinguish between two cases:
  - $\prec_{h,l} = \leq$, which implies that $(m_{h,l}, <_{h,l}) \leq (r_{h,l}, <)$: $Z \subseteq (x_h - x_l < r_{h,l})$. Zone defined by $(x_h - x_l < r_{h,l})$ is K-bounded, hence $\mathsf{Approx}_K(Z) \subseteq (x_h - x_l < r_{h,l})$. However, in that case, $R \subseteq (x_h - x_l = r_{h,l})$, which implies that $\mathsf{Approx}_K(Z) \cap R = \varnothing$.

- $\prec_{h,l}=<$, which implies that $(m_{h,l};<_{h,l}) \leq (r_{h,l}-1;\leq) = (-r_{l,h},\leq)$: $Z \subseteq (x_l - x_h \geq r_{l,h})$. This is also K-bounded, hence $\mathsf{Approx}_K(Z) \subseteq (x_h - x_l \geq r_{l,h})$. However, $R \subseteq (x_l - x_h < r_{l,h})$, which implies that $\mathsf{Approx}_K(Z) \cap R = \varnothing$.

- Let us now assume that the previous case does not happen: in the above sum, if $(m_{i_j,i_{j+1}},<_{i_j,i_{j+1}}) < (r_{i_j,i_{j+1}},\prec_{i_j,i_{j+1}})$, then either $I_{x_{i_j}} =]K,+\infty[$ or $I_{x_{i_{j+1}}} =]K,+\infty[$.
  - Assume that $I_{x_{i_j}} =]K;+\infty[$. Then, either $i_{j-1}=0$ or $(r_{i_{j-1},i_j};\prec_{i_{j-1},i_j}) = (+\infty;<)$ (by definition of M_R). The second term cannot appear in the sum. Hence, $i_{j-1}=0$.
  - Assume that $I_{x_{i_{j+1}}} =]K;+\infty[$. By definition of M_R we get that every coefficient $r_{i_{j+1},*}$ is infinite, which is not possible in the sum. This case cannot happen.

After this analysis, we know that in the sum (4), we cannot have two coefficients coming from M (otherwise index 0 would appear several times, which is not allowed). Thus (since R is not empty) we have exactly one coefficient coming from M in the sum (4). Collecting all these informations, we obtain a sequence of indices $(k_1,\dots,k_p)$ such that

$$(r_{0,k_1};\prec_{0,k_1}) + (m_{k_1,k_2};<_{k_1,k_2}) + (r_{k_2,k_3};\prec_{k_2,k_3}) + \cdots + (r_{k_p,0};\prec_{k_p,0}) < (0;\leq)$$

(and $I_{x_{k_1}} =]K;+\infty[$).
Now, we can notice that all coefficients $(r_{k_i,k_{i+1}},\prec_{k_i,k_{i+1}})$ (with $2 \leq i < p$) of M_R are normalized (they cannot be tightened). Therefore, $(r_{k_2,k_3};\prec_{k_2,k_3})+\cdots+(r_{k_{p-1},k_p};\prec_{k_{p-1},k_p}) \geq (r_{k_2,k_p};\prec_{k_2,k_p})$. The sum can therefore be simplified into:

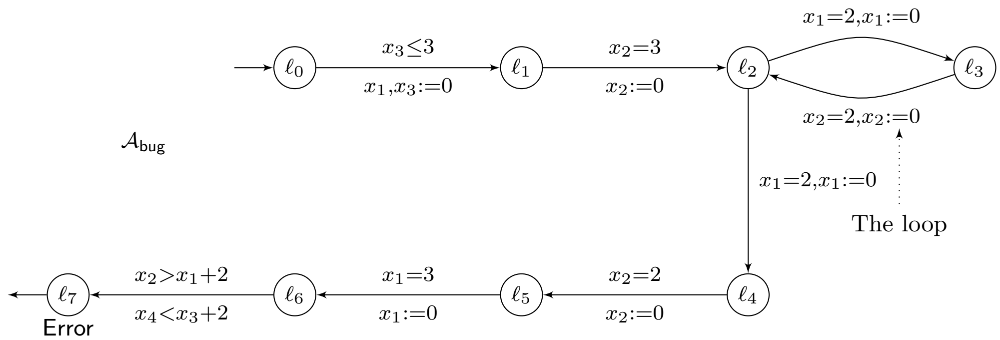
$$(r_{0,k_1};\prec_{0,k_1}) + (m_{k_1,k_2};<_{k_1,k_2}) + (r_{k_2,0};\prec_{k_2,0}) < (0;\leq)$$

with $I_{x_{k-1}} =]K;+infty[$. In particular, $(r_{0,k_1};\prec_{0,k_1}) = (-K;<)$.
We set $(c;\prec) = (r_{k_2,0};\prec_{k_2,0}) + (r_{0,k_1};\prec_{0,k_1}) = (-K;<) + (r_{k_2,0};\prec_{k_2,0})$. Since $0 \leq r_{k_2} \leq K$ we get that $-K \leq c \leq K$. We obtain that:

$$\begin{cases} Z \subseteq (x_{k_1} - x_{k_2} \prec -c) \\ R \cap (x_{k_1} - x_{k_2} \prec -c) = \varnothing \end{cases}$$

Then $-K \leq c \leq K$ implies that the zone $(x_{k_1} - x_{k_2} \prec -c)$ is K-bounded. As previously, we conclude that $\mathsf{Approx}_K(Z) \subseteq (x_{k_1} - x_{k_2} \prec -c)$ and thus $R \cap \mathsf{Approx}_K(Z) = \varnothing$. ⌟

Surprisingly this theorem does not extend to timed automata with general clock constraints. Indeed, consider the timed automaton $\mathcal{A}_{\mathsf{bug}}$ depicted below. For every integer k, the extrapolation operator Approx_k is not correct with respect to reachability properties for $\mathcal{A}_{\mathsf{bug}}$. One can even also prove that, for automaton $\mathcal{A}_{\mathsf{bug}}$, there is no abstraction operator Abs satisfying the four above-mentioned criteria (finiteness, correctness, completeness and effectiveness).



Proposition 10. *Consider the timed automaton $\mathcal{A}_{\mathsf{bug}}$ defined before. For every integer k, Approx_k is not a correct abstraction with respect to reachability properties in $\mathcal{A}_{\mathsf{bug}}$. Furthermore, there is no abstraction operator $\mathfrak{a}$ that over-approximates zones, and that can be finite, effective and correct with respect to reachability properties in $\mathcal{A}_{\mathsf{bug}}$.*

Proof. We explain the problem with automaton $\mathcal{A}_{\mathsf{bug}}$. The zone Z_α which is computed by a forward analysis when reaching the location ℓ_6 after having taken α times the loop is defined by the constraints below (on the left). Fixing an integer k, taking α large enough the extrapolated zone is also described below (on the right).

$$Z_\alpha : \begin{cases} 1 \le x_2 - x_1 \le 3 \\ 1 \le x_4 - x_3 \le 3 \\ x_3 - x_1 = 2\alpha + 5 \\ x_4 - x_2 = 2\alpha + 5 \end{cases} \qquad \mathsf{Approx}_k(Z_\alpha) : \begin{cases} 1 \le x_2 - x_1 \le 3 \\ 1 \le x_4 - x_3 \le 3 \\ x_3 - x_1 > k \\ x_4 - x_2 > k \end{cases}$$

Note that if $v \in Z_\alpha$, then in particular $v(x_2) - v(x_1) = v(x_4) - v(x_3)$. On the other hand, there are valuations $v \in \mathsf{Approx}_k(Z_\alpha)$ such that $v(x_2) - v(x_1) \neq v(x_4) - v(x_3)$. Obviously, the zone Z_α does not intersect the constraint $x_2 - x_1 > 2 \wedge x_4 - x_3 < 2$, which implies that the location 'Error' is not reachable. However, $\mathsf{Approx}_k(Z_\alpha)$ intersects the constraint $x_2 - x_1 > 2 \wedge x_4 - x_3 < 2$ (for α large enough), which implies that the location 'Error' is computed as reachable by the abstract forward analysis that uses the abstraction operator Approx_k (for any integer k). The problem with automaton $\mathcal{A}_{\mathsf{bug}}$ comes from the use of diagonal constraints on the transition leading to location 'Error'. ⌟

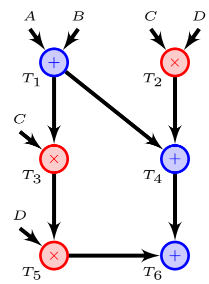
Note however that for timed automata with three clocks (and possibly diagonal constraints), it is possible to find a constant K such that Approx_K is correct with respect to reachability properties (the constant K may however be larger than the maximal constant appearing in a constraint of the automaton) [Bou04]. In the general case (more than three clocks), a way of handling diagonal constraints is to remove first (or on-the-fly) diagonal constraints as we know they can be removed (see [BDGP98]). However this leads to an unavoidable exponential blowup in the size of the model [BC05].

Remark 17. Several improvements on that extrapolation operator have been made, like distinguish locations, distinguish lower- and upper-bounds.

11 The task-graph example

11.1 Description of the problem

Compute $D\times(C\times(A+B))+(A+B)+(C\times D)$ using two processors:

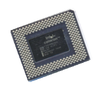


P_1 (fast):

time	
+	2 picoseconds
×	3 picoseconds

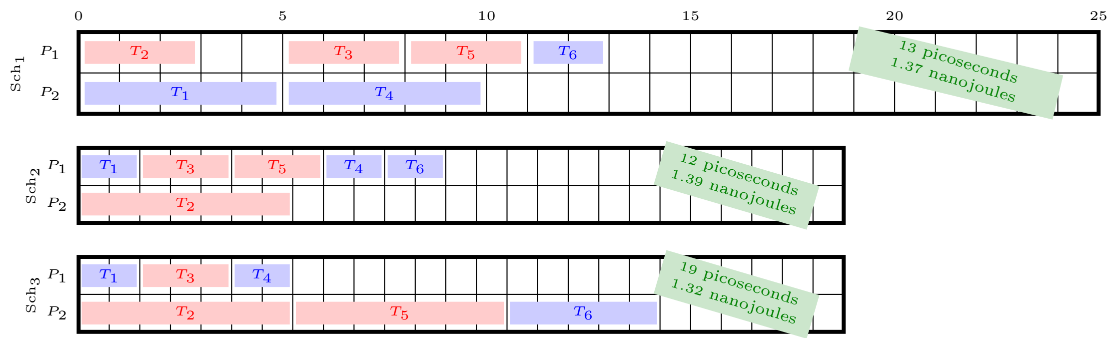
energy	
idle	10 Watt
in use	90 Watts

P_2 (slow):



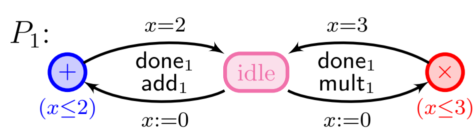
time	
+	5 picoseconds
×	7 picoseconds

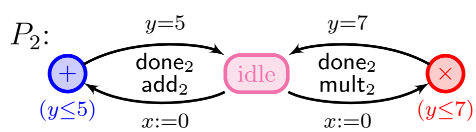
energy	
idle	20 Watts
in use	30 Watts



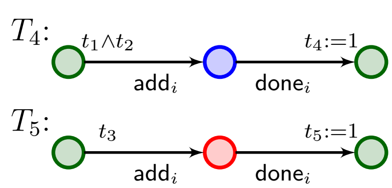
11.2 Modelization of the problem

– Processors





– Tasks



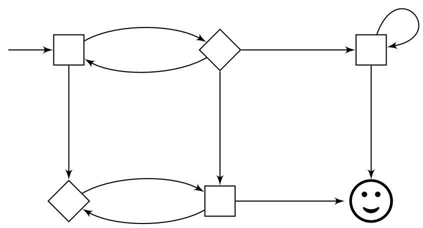
11.3 How to model more?

How should we take into account:

– uncertainties on delays?
– power consumption?
– job preemption?

12 Timed games: controller synthesis

Example 23 (Finite untimed games). Recall the bases.



Motivation for timed games. Uncertainty over delays, controllers. We choose here the model of control games, which is not used in a game-theoretic perspective.

A timed game $\mathcal{G}$ is a timed automaton $(L, L_0, L_F, X, \Sigma, T)$ in which we assume that edges are partitioned into controllable and uncontrollable edges, $T = T_c \cup T_u$. We assume that L_F are sink locations, and we will consider *reachability games*. Its dual, *safety games*, are useful to model *eg* control problems.

12.1 Reachability games

A *strategy* for the controller from state $s = (\ell, v)$ is a partially defined function $f : \mathsf{Runs}(\mathcal{G}, s) \to (\mathbb{R}_+ \times T_c)$ (where $\mathsf{Runs}(\mathcal{G}, s)$ is the set of finite runs from configuration s) such that for every run ϱ on which f is defined, if $f(\varrho) = (t, e)$, then (t, e) is a valid mixed move after ϱ. The set of possible *outcomes* (or *plays*) of f from s in $\mathcal{G}$ is denoted $\mathsf{Out}_f(\mathcal{G}, s)$ and is defined inductively as follows:

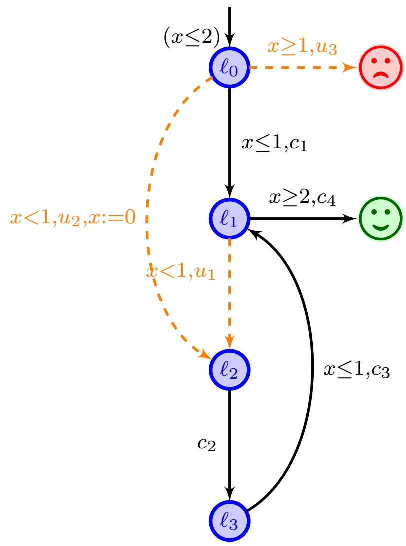
- the run s is in $\mathsf{Out}_f(\mathcal{G}, s)$;
- if ϱ is in $\mathsf{Out}_f(\mathcal{G}, s)$ and if $(t, e) = f(\varrho)$, then $\varrho \xrightarrow{t,c} s'$ is in $\mathsf{Out}_f(\mathcal{G}, s)$, and for every $0 \le t' \le t$ and every $e' \in T_u$, if $\varrho \xrightarrow{t',e'} s'$ is a possible extension of ϱ, then $\varrho \xrightarrow{t',e'} s'$ is in $\mathsf{Out}_f(\mathcal{G}, s)$;
- if ϱ is an infinite run such that every prefix ϱ' of ϱ is in $\mathsf{Out}_f(\mathcal{G}, s)$, then ϱ is in $\mathsf{Out}_f(\mathcal{G}, s)$.

An outcome $\varrho \in \mathsf{Out}_f(\mathcal{G}, s)$ is *f-maximal* if it is infinite or if it cannot be extended by the strategy f (its image by f is not defined). A run is *winning* if it visits in $L_F \times \mathbb{R}_+^X$. A strategy f is *winning* if all f-maximal outcomes of f are winning.

Remark 18. Any reasonable change in this definition (only infinite runs, uncontrollable edges should be taken, *etc*) can be taken into account. *Explain turn-based games, and changes that should be made.*

Given a timed game, we want to synthesize winning strategies, if possible, or assert it is not possible to do so.

Example 24. We consider the following timed game:



Is the configuration $(\ell_0, 0)$ winning? [objective: reach the happy location] ⌟

Theorem 13. *The controller synthesis problem is computable/decidable.*

We define the following operators, defined on sets of configurations:

$$\mathsf{Pred_t}(W, W') = \{(\ell, v) \in L \times \mathbb{R}_+^X \mid \exists t \in \mathbb{R}_+ \text{ s.t. } (\ell, v+t) \in W \text{ and } \forall 0 \le t' \le t,\ (\ell, v+t') \in W'\}$$

Intuitively, W' are safe states for the controller.

We also define

$$\mathsf{cPred}(W) = \{(\ell, v) \in L \times \mathbb{R}_+^X \mid \exists e \in T_c \text{ s.t. } (\ell, v) \xrightarrow{0,e} (\ell', v') \text{ and } (\ell', v') \in W\}$$

$$\mathsf{uPred}(W) = \{(\ell, v) \in L \times \mathbb{R}_+^X \mid \exists e' \in T_u \text{ s.t. } (\ell, v) \xrightarrow{0,e'} (\ell', v') \text{ and } (\ell', v') \in W\}$$

Finally, we define

$$\pi(W) = W \cup \mathsf{Pred_t}(\mathsf{cPred}(W), (\mathsf{uPred}(W^c))^c)$$

This is the set of controllable predecessor of W: from $\pi(W)$, one can ensure a move to W, without being kicked out by the environment.

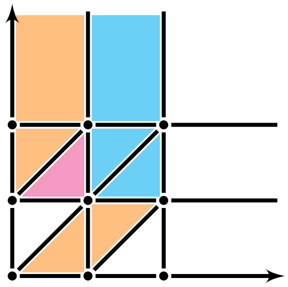
Lemma 13. *A configuration (ℓ, v) is in $\pi(W) \setminus W$ iff there exists $t \in \mathbb{R}_+$ and $e \in T_c$ such that $(\ell, v) \xrightarrow{t,e} W$, and for every $0 \le t' \le t$, for every $e' \in T_u$, $(\ell, v) \xrightarrow{t',e'} W$.*

Proof. This is a direct application of the definitions. ⌟

Lemma 14. *Let $\mathsf{Goal} = \{(\ell, v) \in L \times \mathbb{R}_+^X \mid \ell \in L_F\}$. The non-decreasing sequence $(\pi^i(\mathsf{Goal}))_{i \in \mathbb{N}}$ stabilizes in a finite number of steps.*

Proof. Operators cPred, uPred and Pred_t preserve finite unions of extended regions (an extended region is a pair (ℓ, R) where ℓ is a location of the automaton, and R is a region). This can be done as an exercise. ⌟

Remark 19. This fixpoint operator π does not preserve zones in general:



The limit of the sequence $(\pi^i(\mathsf{Goal}))_{i\in\mathbb{N}}$, denoted $\pi^*(\mathsf{Goal})$, is the *attractor* of Goal. Note that, due to the convergence of the iterative sequence, $\pi(\pi^*(\mathsf{Goal})) = \pi^*(\mathsf{Goal})$.

Proposition 11. *A configuration (ℓ, v) is in $\pi^*(\mathsf{Goal})$ iff the controller has a winning strategy for the corresponding reachability game.*

Proof. We will define a memoryless strategy for the controller, which will be winning for all configurations in $\pi^*(\mathsf{Goal})$. For every configuration $s \in \pi^*(\mathsf{Goal})$, there is a unique index $i \in \mathbb{N}$ such that $s \in \pi^i(\mathsf{Goal}) \setminus \pi^{i-1}(\mathsf{Goal})$. We will call this index the *rank* of s in the game. We construct the strategy f, inductively on the rank of the configurations, as follows.

For every $i \in \mathbb{N}$ we construct a memoryless strategy f_i defined over all configurations in $\pi^i(\mathsf{Goal}) \setminus \mathsf{Goal}$, and such that:

- f_i restricted to $\pi^{i-1}(\mathsf{Goal})$ coincides with f_{i-1}
- for every configuration $s_0 \in \pi^i(\mathsf{Goal})$, if $\varrho \in \mathsf{Out}_{f_i}(\mathcal{G}, s_0)$, $\varrho = s_0 \xrightarrow{t_1,a_1} s_1 \xrightarrow{t_2,a_2} s_2 \cdots \xrightarrow{t_p,a_p} s_p$ implies $\mathsf{rank}(s_{j+1}) < \mathsf{rank}(s_j)$ for every $0 \le j < p$, and furthermore, if ϱ is f_i-maximal, then $\mathsf{rank}(s_p) = 0$ (the run is winning).

Only configurations in Goal have rank 0, hence we define f_0 as being undefined everywhere. It satisfies the above conditions over $\pi^0(\mathsf{Goal}) \setminus \mathsf{Goal} = \varnothing$.

Assume now the induction hypothesis for all configurations of rank at most $i-1$. Pick $(\ell, v) \in \pi^i(\mathsf{Goal}) \setminus \pi^{i-1}(\mathsf{Goal})$. There exists $t \in \mathbb{R}_+$, $e \in T_c$ such that $(\ell, v) \xrightarrow{t,e} (\ell', v')$ implies (ℓ', v') has at most rank $i-1$, and for all $0 \le t' \le t$ and $e' \in T_u$, $(\ell, v) \xrightarrow{t',e'} (\ell'', v'')$ implies that (ℓ'', v'') has rank at most $i-1$. So if we assume that we have already defined the strategy f_i as f_{i-1} from configurations of rank no more than $i-1$, we define the strategy in (ℓ, v) as the move (t, e). Now if we take a (maximal) run ϱ generated by the strategy f_i, $\varrho = (\ell, v) \xrightarrow{t_1,a_1} (\ell_1, v_1) \xrightarrow{t_2,a_2} (\ell_2, v_2) \cdots$, we have that the sub-run $(\ell_1, v_1) \xrightarrow{t_2,a_2} (\ell_2, v_2) \cdots$ is generated by the strategy f_{i-1} and is thus finite and winning.

Conversely, assume that $(\ell, v) \notin \pi^*(\mathsf{Goal}) = \pi(\pi^*(\mathsf{Goal}))$, but that the controller has a strategy f (possibly with memory) to win the game from (ℓ, v). We will build a run starting in (ℓ, v), generated by the strategy f, and which only visits configurations outside $\pi^*(\mathsf{Goal})$ (hence never winning). Assume we have constructed such a run, say ϱ, of length i. We will extend this run into a run of length $i+1$. Assume that ϱ ends in configuration

$(\ell', v') \notin \pi^*(\mathsf{Goal})$ and that $f(\varrho) = (t, e)$. There are two cases: either $(\ell', v') \xrightarrow{t,e} (\ell'', v'')$ is such that $(\ell'', v'') \notin \pi^*(\mathsf{Goal})$, in which case we are done, or there exists $0 \leq t' \leq t$ and $e' \in T_u$ such that $(\ell', v') \xrightarrow{t,e'} (\ell''', v''')$ implies $(\ell''', v''') \notin \pi^*(\mathsf{Goal})$, in which case the extended run is $\varrho \xrightarrow{t',e'} (\ell''', v''')$. That way, we build an infinite run which is generated by f but is not winning... contradiction. ⌟

Remark 20. Memoryless strategies are sufficient to win!

Exercise 22. Apply the algorithm to the game of Example 24.

Exercise 23. Compute the complexity of the above algorithm. Think of (and prove) a matching lower bound. Conclude.

Proof. This problem is EXPTIME-complete, and EXPTIME-hardness follows from a reduction from the halting problem of alternating linearly-bounded Turing machines. ⌟

12.2 Safety games

In that case, a run is winning if it avoids L_F. A strategy f is winning if all f-maximal outcomes of f are winning.

Those safety games can be solved using a fixpoint operator as well:

$$\pi'(W) = W \cap \mathsf{Pred}_t(\mathsf{cPred}(W), (\mathsf{uPred}(W^c))^c)$$

starting from $\mathsf{Safe} = \{(\ell, v) \in L \times \mathbb{R}_+^X \mid \ell \notin L_F\}$.

Those games are also EXPTIME-complete.

12.3 Imperfect information in reachability games

Instead of observing the runs, we observe timed words, that is the strategies only depend on the underlying timed word, not on the real run. If ϱ is a run, we write $tw(\varrho)$ the underlying timed word of ϱ. A strategy f respects the imperfection information whenever $tw(\varrho) = tw(\varrho')$ implies $f(\varrho) = f(\varrho')$. This allows to have a partial observation of the system.

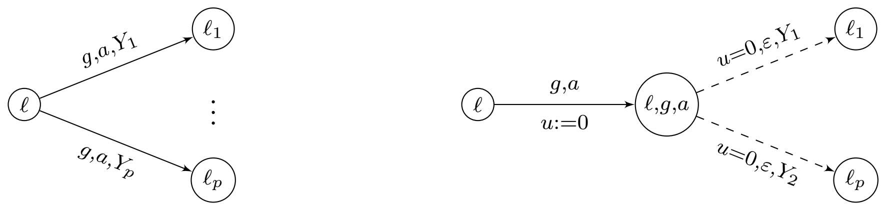
Theorem 14. *Reachability timed games under the imperfect information hypothesis are undecidable.*

Proof (Sketch). By reduction from the universality problem of timed automata. Let $\mathcal{A} = (L, L_0, L_F, X, \Sigma, T)$ be a timed automaton. W.l.o.g. we assume that if $\ell \xrightarrow{g',a,Y'} \ell'$ and $\ell \xrightarrow{g'',a,Y''} \ell''$ then either $[\![g']\!] = [\![g'']\!]$ or $[\![g']\!] \cap [\![g'']\!] = \varnothing$. We also assume the automaton is complete. We construct the game $\mathcal{G} = (L', L'_0, L'_F, X \cup \{u\}, \Sigma', T')$ where:

- $L' = L \cup (L \times \times \mathcal{C}^k(X) \times \Sigma) \cup \{\mathsf{win}, \mathsf{lose}\}$;[30]

[30] $\mathcal{C}^k(X)$ is the set of k-bounded contraints, and k is the maximal constant in $\mathcal{A}$.

- $L'_0 = L_0$;
- $L'_F = \{\mathsf{win}\}$;
- $\Sigma' = \Sigma \cup \{\varepsilon, \#\}$;
- T' is defined as follows: for every ℓ, g and a,



in $\mathcal{A}$ in $\mathcal{G}$

We also add transitions $\ell \xrightarrow{u=0,\#} \mathsf{win}$ if $\ell \in L \setminus L_F$, and $\ell \xrightarrow{u=0,\#} \mathsf{lose}$ if $\ell \in L_F$.

It is not difficult to prove that the controller has a strategy to ensure $\mathsf{Goal} = L'_F \times \mathbb{R}_+^X$ iff the timed automaton $\mathcal{A}$ is not universal. The idea is to play a timed word w which is not accepted by $\mathcal{A}$: in $\mathcal{A}$, all runs that read w leads to a location in $L \setminus L_F$, from which we can play the action $\#$ and reach location win. ⌟

12.4 Application of games to checking strong timed bisimulation

Theorem 15. *Strong timed bisimulation between timed automata can be decided in EXPTIME.*

Proof. The proof of this theorem can be done using timed games!

Assume $\mathcal{A}_1 = (L_1, L_{1,0}, L_{1,F}, X_1, \Sigma, T_1)$ and $\mathcal{A}_2 = (L_2, L_{2,0}, L_{2,F}, X_2, \Sigma, T_2)$ are two timed automata over the same alphabet.

We construct a turn-based[31] timed game $\mathcal{G} = (Q, Q_0, Q_{\text{tester}}, Q_{\text{prover}}, Q_F, X, \Sigma, T)$ as follows:

- $Q_{\text{tester}} = L_1 \times L_2$
- $Q_{\text{prover}} = \{(\ell_1, e_2) \mid \ell_1 \in L_1,\ e_2 \in T_2\} \cup \{(\ell_2, e_1) \mid \ell_2 \in L_2,\ e_1 \in T_1\}$
- $Q = Q_{\text{tester}} \cup Q_{\text{prover}} \cup \{\mathsf{Bad}\}$
- $Q_0 = L_{1,0} \times L_{2,0}$
- $Q_F = \{\mathsf{Bad}\}$
- $X = X_1 \cup X_2 \cup \{z\}$
- T is composed of the following transitions:
  - $(\ell_1, \ell_2) \xrightarrow{g_1, a, Y_1 \cup \{z\}} (\ell_2, e_1)$ where $(\ell_1, \ell_2) \in L_1 \times L_2$, $e_1 = (\ell_1, g_1, a, Y_1, \ell'_1) \in T_1(\ell_1)$
  - $(\ell_1, \ell_2) \xrightarrow{g_2, a, Y_2 \cup \{z\}} (\ell_1, e_2)$ where $(\ell_1, \ell_2) \in L_1 \times L_2$, $e_2 = (\ell_2, g_2, a, Y_2, \ell'_2) \in T_2(\ell_2)$
  - $(\ell_2, e_1) \xrightarrow{g_2 \wedge (z=0), a, Y_2} (\ell'_1, \ell'_2)$ where $(\ell'_1, \ell_2) \in L_1 \times L_2$, $e_1 = (\ell_1, g_1, a, Y_1, \ell'_1) \in T_1(\ell_1)$, and $e_2 = (\ell_2, g_2, a, Y_2, \ell'_2) \in T_2(\ell_2)$

[31] To be explained.

- $(\ell_1, e_2) \xrightarrow{g_1 \wedge (z=0), a, Y_1} (\ell_1', \ell_2')$ where $(\ell_1, \ell_2') \in L_1 \times L_2$, $e_2 = (\ell_2, g_2, a, Y_2, \ell_2') \in T_2(\ell_2)$, and $e_1 = (\ell_1, g_1, a, Y_1, \ell_1') \in T_1(\ell_1)$
- $(\ell_2, e_1) \xrightarrow{(z=0) \wedge \neg g, a}$ Bad where $g = \bigvee_{(\ell_2, g_2, a, Y_2, \ell_2') \in T_2(\ell_2)} g_2$
- $(\ell_1, \ell_2', e_2) \xrightarrow{(z=0) \wedge \neg g, a}$ Bad where $g = \bigvee_{(\ell_1, g_1, a, Y_1, \ell_1') \in T_1(\ell_1)} g_1$

This is depicted on Figure 9. Then, (ℓ_1, v_1) and (ℓ_2, v_2) are timed bisimilar iff the prover

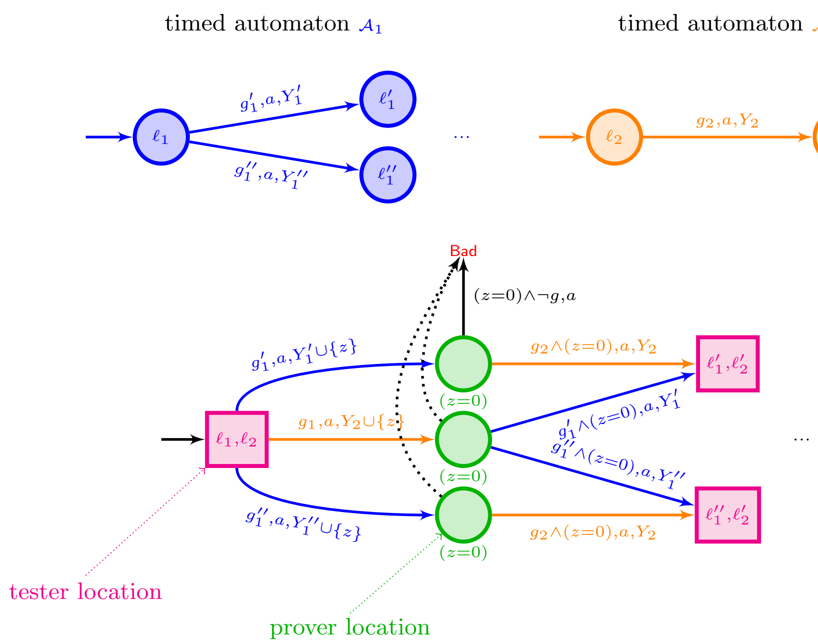


Fig. 9: *Bisimulation game*

has a winning strategy to avoid the Bad state from $((\ell_1, \ell_2), v_1 v_2)$.

Define $\mathcal{R}$ the relation defined by $(\ell_1, v_1) \mathcal{R} (\ell_2, v_2)$ iff the prover has a winning strategy to avoid the Bad state from $((\ell_1, \ell_2), v_1 v_2)$. It is easy to prove that $\mathcal{R}$ is a bisimulation relation by playing a memoryless winning strategy.

Assume now that there is a bisimulation relation $\mathcal{R}$ such that $(\ell_1, v_1) \mathcal{R} (\ell_2, v_2)$. We define a winning memoryless strategy for the player as follows: for all states $(\ell_1, v_1) \mathcal{R} (\ell_2, v_2)$, for all t, from states $((\ell_2, e_1), (v_1 v_2) + t)$, play the corresponding matching move witnessing the fact that there is an edge e_2 labelled with a such that $(\ell_2, v_2) \xrightarrow{t, a} (\ell_2', v_2')$. This yields a winning strategy... $\lrcorner$

13 Going further: current challenges

References

[ACD90] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking for real-time systems. In *Proc. 5th Annual Symposium on Logic in Computer Science (LICS'90)*, pages 414–425. IEEE Computer Society Press, 1990.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.

[AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990.

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

[ADOW05] Parosh Aziz Abdulla, Johann Deneux, Joël Ouaknine, and James Worrell. Decidability and complexity results for timed automata via channel machines. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1089–1101. Springer, 2005.

[AFH94] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In *Proc. 6th International Conference on Computer Aided Verification (CAV'94)*, volume 818 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1994.

[AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43(1):116–146, 1996.

[AH89] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. In *Proc. 30th Annual Symposium on Foundations of Computer Science (FOCS'89)*, pages 164–169. IEEE Computer Society Press, 1989.

[AH94] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *Journal of the ACM*, 41(1):181–204, 1994.

[AILS08] Luca Aceto, Anna Ingólfsdóttir, Kim G. Larsen, and Jiří Srba, editors. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, 1008. `http://www.cambridge.org/uk/catalogue/catalogue.asp?isbn=9780521875462`.

[AL02] Luca Aceto and François 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.

[Alu91] Rajeev Alur. *Techniques for Automatic Verification of Real-Time Systems*. PhD thesis, Stanford University, Stanford, CA, USA, 1991.

[AM04] Rajeev Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *Proc. 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Real Time (SFM-04:RT)*, volume 3185 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2004.

[AMP95] Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1):35–65, 1995.

[BBBB09] Christel Baier, Nathalie Bertrand, Patricia Bouyer, and Thomas Brihaye. When are timed automata determinizable? In *Proc. 36th International Colloquium on Automata, Languages and Programming (ICALP'09)*, volume 5556 of *Lecture Notes in Computer Science*, pages 43–54. Springer, 2009.

[BBBL05] Houda Bel Mokadem, Béatrice Bérard, Patricia Bouyer, and François Laroussinie. A new modality for almost everywhere properties in timed automata. In *Proc. 16th International Conference on Concurrency Theory (CONCUR'05)*, volume 3653 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2005.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim G. Larsen. Static guard analysis in timed automata verification. In *Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 254–277. Springer, 2003.

[BBL08] Patricia Bouyer, Ed Brinksma, and Kim G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):2–23, 2008.

[BBLP04] Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and upper bounds in zone based abstractions of timed automata. In *Proc. 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, volume 2988 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 2004.

[BC05] Patricia Bouyer and Fabrice Chevalier. On conciseness of extensions of timed automata. *Journal of Automata, Languages and Combinatorics*, 10(4):393–405, 2005.

[BCM05] Patricia Bouyer, Fabrice Chevalier, and Nicolas Markey. On the expressiveness of TPTL and MTL. In *Proc. 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 432–443. Springer, 2005.

[BDFP04] Patricia Bouyer, Catherine Dufourd, Emmanuel Fleury, and Antoine Petit. Updatable timed automata. *Theoretical Computer Science*, 321(2–3):291–345, 2004.

[BDGP98] Béatrice Bérard, Volker Diekert, Paul Gastin, and Antoine Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2–3):145–182, 1998.

[BDL$^+$06] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 125–126. IEEE Computer Society Press, 2006.

[Ben02] Johan Bengtsson. *Clocks, DBMs ans States in Timed Systems*. PhD thesis, Department of Information Technology, Uppsala University, Uppsala, Sweden, 2002.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008. `http://mitpress.mit.edu/catalog/item/default.asp?ttype=2\&tid=11481`.

[BLMR06] Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob I. Rasmussen. Almost optimal strategies in one-clock priced timed automata. In *Proc. 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 345–356. Springer, 2006.

[BM83] Bernard Berthomieu and Miguel 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.

[Bou98] Patricia Bouyer. Automates temporisés et modularité. Master's thesis, DEA Algorithmique, Paris, 1998.

[Bou04] Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods in System Design*, 24(3):281–320, 2004.

[BS91] Janusz A. Brzozowski and Carl-Johan H. Seger. Advances in asynchronous circuit theory. Bulletin of the European Association of Theoretical Computer Science (EATCS), 1991.

[BZ83] Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[CE81] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronous skeletons using branching-time temporal logic. In *Proc. 3rd Workshop on Logics of Programs (LOP'81)*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[CE82] Edmund M. Clarke and E. Allen Emersen. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2(3):241–266, 1982.

[CGP99] Edmund Clarke, Orna Grumberg, and Doron Peled. *Model-Checking*. MIT Press, 1999.

[Che07] Fabrice Chevalier. *Logiques pour les systèmes temporisés : contrôle et expressivité*. PhD thesis, École Normale Supérieure de Cachan, France, 2007.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, 1990.

[CY92] Costas Courcoubetis and Mihalis Yannakakis. Minimum and maximum delay problems in real-time systems. *Formal Methods in System Design*, 1(4):385–415, 1992.

[Dav05] Alexandre David. Merging DBMs efficiently. In *Proc. 17th Nordic Workshop on Programming Theory (NWPT'05)*, 2005. DIKU, University of Copenhagen.

[DHGP04] Alexandre David, John Håkansson, Larsen Kim G., and Paul Pettersson. Minimal DBM subtraction. In *Proc. 16th Nordic Workshop on Programming Theory (NWPT'04)*, pages 17–20, 2004. Uppsala University IT Technical Report 2004-041.

[Dil90] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. of the Workshop on Automatic Verification Methods for Finite State Systems (1989)*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1990.

[DOTY96] Conrado Daws, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. The tool KRONOS. In *Proc. Hybrid Systems III: Verification and Control (1995)*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer, 1996.

[DT98] Conrado Daws and Stavros Tripakis. Model-checking of real-time reachability properties using abstractions. In *Proc. 4th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98)*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

[DT04] Deepak D'Souza and Nicolas Tabareau. On timed automata with input-determined guards. In *Proc. Joint Conference on Formal Modelling and Analysis of Timed Systems and Formal Techniques in Real-Time and Fault Tolerant System (FORMATS+FTRTFT'04)*, volume 3253 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2004.

[Fin06] Olivier Finkel. Undecidable problems about timed automata. In *Proc. 4th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'06)*, volume 4202 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2006.

[FJ15] John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Information and Computation*, 243:26–36, 2015.

[FL08] Uli Fahrenberg and Kim G. Larsen. Discount-optimal infinite runs in priced timed automata. In *Proc. 10th International Workshop on Verification of Infinite-State Systems (INFINITY'08)*, 2008. To appear.

[FS01] Alain Finkel and Philippe Schnoebelen. Well structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[HMU01] John Hopcroft, Rajeev Motwani, and Jeffrey Ullman. *Introduction to Automata Theory, Languages and Computation (2nd edition)*. Addison-Wesley, 2001.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model-checking for real-time systems. *Information and Computation*, 111(2):193–244, 1994.

[HU79] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[Kam68] Johan A.W. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, UCLA, Los Angeles, CA, USA, 1968.

[KCG94] Pascal Koiran, Michel Cosnard, and Max Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, 132(1–2):113–128, 1994.

[Kop96] Peter W. Kopke. *The Theory of Rectangular Hybrid Automata.* PhD thesis, Cornell University, Ithaca, NY, USA, 1996.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[LL98] François Laroussinie and Kim G. Larsen. CMC: A tool for compositional model-checking of real-time systems. In *Proc. IFIP Joint International Conference on Formal Description Techniques & Protocol Specification, Testing, and Verification (FORTE-PSTV'98)*, pages 439–456. Kluwer Academic, 1998.

[LMS04] François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In *Proc. 15th International Conference on Concurrency Theory (CONCUR'04)*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer, 2004.

[LPWY99] Kim G. Larsen, Justin Pearson, Carsten Weise, and Wang Yi. Clock difference diagrams. *Nordic Journal of Computing*, 6(3):271–298, 1999.

[LPZ85] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer, 1985.

[LW05] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. In *Proc. 8th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS'05)*, volume 3441 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 2005.

[LW08] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Transactions on Computational Logic*, 9(2:10), 2008.

[MSS88] David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proc. 3rd Annual Symposium on Logic in Computer Science (LICS'88)*, pages 422–427. IEEE Computer Society Press, 1988.

[NP10] Dejan Nickovic and Nir Piterman. From MTL to deterministic timed automata. In *Proc. 8th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'10)*, Lecture Notes in Computer Science. Springer, 2010. To appear.

[OW05] Joël Ouaknine and James Worrell. On the decidability of Metric Temporal Logic. In *Proc. 20th Annual Symposium on Logic in Computer Science (LICS'05)*, pages 188–197. IEEE Computer Society Press, 2005.

[OW07] Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. *Logical Methods in Computer Science*, 3(1:8), 2007.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *Proc. 18th Annual Symposium on Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, 1977.

[Pri] Webpage of PRISM. `http://www.prismmodelchecker.org/`.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *Proc. 5th International Symposium on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer, 1982.

[SBB+01] Philippe Schnoebelen, Béatrice Bérard, Michel Bidoit, François Laroussinie, and Antoine Petit. *Systems and Software Verification - Model-Checking Techniques and Tools.* Springer, 2001. `http://www.springer.com/computer/programming/book/978-3-540-41523-7`.

[Sch02] Philippe Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Information Processing Letters*, 83(5):251–261, 2002.

[SP09] P. Vijay Suman and Paritosh K. Pandya. Determinization and expressiveness of integer reset timed automata with silent transitions. In *Proc. 3rd International Conference on Language and Automata Theory and Applications (LATA'09)*, volume 5457 of *Lecture Notes in Computer Science*, pages 728–739. Springer, 2009.

[Spi] Webpage of SPIN. `http://spinroot.com/`.

[Tri03] Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. In *Proc. 1st International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'03)*, volume 2791 of *Lecture Notes in Computer Science*, pages 182–188. Springer, 2003.

[Upp] Webpage of UPPAAL. `http://www.uppaal.org/`.

[Var96] Moshe Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Proc. Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer, 1996.